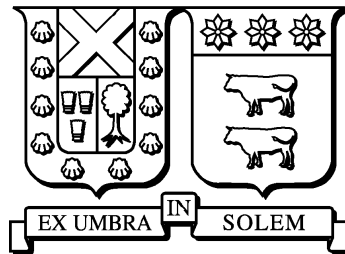


UNIVERSIDAD TÉCNICA FEDERICO SANTA
MARÍA

DEPARTAMENTO DE INFORMÁTICA
SANTIAGO – CHILE



“TÉCNICAS DE ADAPTACIÓN ENTRE
DOMINIOS PARA APRENDIZAJE
AUTOMÁTICO”

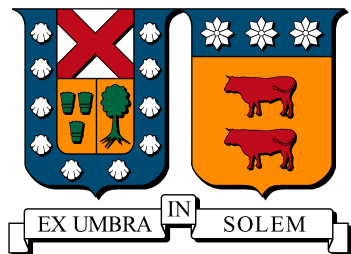
FRANCISCO IGNACIO CASTILLO DINAMARCA

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL INFORMÁTICO

PROFESOR GUÍA: RICARDO ÑANCULEF

MAYO 2018

**UNIVERSIDAD TÉCNICA FEDERICO SANTA
MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO – CHILE**



**“TÉCNICAS DE ADAPTACIÓN ENTRE
DOMINIOS PARA APRENDIZAJE
AUTOMÁTICO”**

FRANCISCO IGNACIO CASTILLO DINAMARCA

**MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL INFORMÁTICO**

PROFESOR GUÍA: RICARDO ÑANCULEF

PROFESOR CORREFERENTE: CARLOS VALLE

MAYO 2018

**MATERIAL DE REFERENCIA, SU USO NO INVOLUCRA RESPONSABILIDAD DEL AUTOR O DE LA
INSTITUCIÓN**

Resumen

En términos muy generales, el problema de adaptación entre dominios consiste en diseñar técnicas que permitan utilizar datos de un determinado contexto o dominio (por ejemplo opiniones clasificadas en *Amazon*) para resolver un problema de aprendizaje en otro contexto o dominio (por ejemplo clasificar opiniones en una tienda de *retail* nacional). Estas técnicas son útiles en escenarios en que no se cuenta con un conjunto de datos suficientemente grande para aplicar con éxito un método de aprendizaje automático o se desea mejorar los resultados actuales mediante conocimiento adquirido en problemas similares. En esta memoria se presenta un estudio de métodos representativos del estado del arte actual con énfasis en el problema de clasificación de sentimiento, es decir el reconocimiento del estado emocional subyacente a un texto corto que representa la opinión de su autor con respecto a una cosa. Además, se presenta una comparación experimental de un conjunto de técnicas seleccionadas usando *benchmarks* públicamente disponibles y también un *dataset* nuevo construido durante una práctica profesional en una empresa dedicada entre otras al análisis de sentimiento. Finalmente, se presentan conclusiones que, junto a los programas construidos durante la ejecución de este trabajo, constituyen una base para guiar investigaciones en el área.

Palabras Clave: Aprendizaje Automático, Adaptación entre dominios, Análisis de sentimiento.

Abstract

In general terms, the domain adaptation problem consists in designing techniques that allow to use data in a determined context or domain (for instance, opinions classified in *Amazon*) to solve a learning problem in another context or domain (for instance, classify opinions on a national *retail* store). These techniques are useful on scenarios on which there isn't a group of data large enough to apply with success a *machine learning* method or there's a need to improve actual results through acquired knowledge on similar problems. In this report, a study of the most representative methods of the actual state of the art is presented with emphasis on the sentiment classification problem, that is to say the identification of the underlying emotional state of a short text that represents the opinion of its author with respect to a subject. Furthermore, it is presented an experimental comparison of a set of selected techniques using publicly available *benchmarks* and also a new *dataset* built during an internship on an enterprise engaged amongst other things to sentiment analysis. Finally, conclusions are presented that, together with software built during the making of this work, make a base to guide further research on the area.

Keywords: Machine Learning, Domain Adaptation, *transfer-loss*.

Índice de Contenidos

Resumen	III
Abstract	IV
Índice de Contenidos	V
Lista de Tablas	XI
Lista de Figuras	XIV
Glosario	XV
1. Introducción	1
1.1. Definición del Problema	1
1.2. Objetivos	3
1.2.1. Objetivos Generales	3
1.2.2. Objetivos Específicos	3
1.3. Estructura del Documento	4
2. Estado del Arte	6
2.1. Aprendizaje Automático	7
2.1.1. Aprendizaje Supervisado	8

2.1.2.	Aprendizaje No Supervisado	9
2.1.3.	Aprendizaje Semi-supervisado	9
2.2.	Análisis de Sentimientos	9
2.3.	Transfer Learning	10
2.4.	Publicaciones sobre adaptación entre dominios	15
2.4.1.	Cross-Domain Sentiment Classification via Spectral Feature Alignment	15
2.4.2.	Domain Adaptation for Large-Scale Sentiment Classification: A Deep Learning Approach	21
2.4.3.	Domain Adaptation for Object Recognition: An Unsupervised Approach	25
2.4.4.	Geodesic flow kernel for unsupervised domain adaptation	28
2.4.5.	Marginalized Denoising Autoencoders for Domain Adaptation	34
2.5.	Representación matemática de los datos	37
2.6.	Medición de similitud entre dominios	38
3.	Implementación	41
3.1.	Métodos implementados	41
3.1.1.	Estructura de los datasets	42
3.2.	Configuración del entorno de trabajo	44
3.2.1.	Amazon Elastic Compute Cloud	45
3.2.2.	Entorno de Python	46
4.	Método Experimental	49
4.1.	Datasets	50
4.1.1.	Amazon	50
4.1.2.	Twitter	51
4.2.	Métricas	52

4.3. Pruebas	52
4.3.1. Pruebas de adaptación	52
4.3.2. Pruebas de tiempo	58
5. Análisis y Resultados	59
5.1. Pruebas de Adaptación	60
5.1.1. Amazon	61
5.1.2. Twitter	85
5.2. Pruebas de Tiempo	107
Conclusiones	110
Conclusiones Generales	110
Conclusiones Específicas	112
Trabajo a Futuro	118
Bibliografía	120
A. Tablas	122
B. Principal Components Analysis	125
C. Partial Least Squares	127
D. Marginalized Denoising Autoencoders	128
E. Transfer Loss	131

Índice de tablas

4.1. Cantidad de instancias en el <i>dataset</i> de <i>Amazon</i>	51
4.2. Cantidad de instancias en el <i>dataset</i> <i>Twitter</i>	52
5.1. Baseline In-domain error de los dominios del <i>dataset</i> <i>Amazon</i>	62
5.2. <i>Proxy A-distance</i> de los dominios del <i>dataset</i> <i>Amazon</i> con 3000 di- mensiones	64
5.3. <i>Proxy A-distance</i> de los dominios del <i>dataset</i> <i>Amazon</i> con 1000 di- mensiones	65
5.4. Mejores valores de <i>Proxy A-distance</i> para cada dominio del <i>dataset</i> <i>Amazon</i> en ambas dimensiones.	65
5.5. Comparación de los resultados obtenidos por cada algoritmo en com- paración al método <i>Baseline</i> en el marco experimental 1.	68
5.6. Conteo de la cantidad de veces que cada método obtiene los mejores resultados con el marco experimental 1.	69
5.7. Menores valores de <i>transfer error</i> por cada tarea de adaptación con el <i>dataset</i> <i>Amazon</i> en el marco experimental 1.	72
5.8. Comparación de los resultados obtenidos por cada algoritmo en com- paración al método <i>Baseline</i> en el marco experimental 2.	74

5.9. Conteo de la cantidad de veces que cada método obtiene los mejores resultados con el marco experimental 2.	74
5.10. Menores valores de <i>transfer error</i> por cada tarea de adaptación con el <i>dataset Amazon</i> en el marco experimental 2.	76
5.11. Comparación de los resultados obtenidos por ambos marcos experimentales en el <i>dataset Amazon</i>	81
5.12. Mejores resultados obtenidos por cada algoritmo para adaptar cada dominio objetivo en el <i>dataset Amazon</i> utilizando 3000 dimensiones. . .	83
5.13. Mejores resultados obtenidos por cada algoritmo para adaptar cada dominio objetivo en el <i>dataset Amazon</i> utilizando 1000 dimensiones. . .	84
5.14. Mejores adaptaciones obtenidas en ambos marcos experimentales, utilizando 3000 dimensiones con el <i>dataset Amazon</i>	85
5.15. Mejores adaptaciones obtenidas en ambos marcos experimentales, utilizando 1000 dimensiones con el <i>dataset Amazon</i>	86
5.16. Conteo de la cantidad de veces que cada método obtiene los mejores resultados en ambos marcos experimentales en el <i>dataset Amazon</i> . . .	86
5.17. <i>Baseline In-domain error</i> de los dominios del <i>dataset Twitter</i>	87
5.18. <i>Proxy A-distance</i> de los dominios del <i>dataset Twitter</i> con 2000 dimensiones	88
5.19. <i>Proxy A-distance</i> de los dominios del <i>dataset Twitter</i> con 1000 dimensiones	89
5.20. Mejores valores de <i>Proxy A-distance</i> para cada dominio del <i>dataset Twitter</i> en ambas dimensiones.	89
5.21. Comparación de los resultados obtenidos por cada algoritmo en comparación al método <i>Baseline</i> en el marco experimental 1.	91

5.22. Conteo de la cantidad veces que cada método obtiene los mejores resultados con el marco experimental 1.	92
5.23. Menores valores de <i>transfer error</i> por cada tarea de adaptación con el <i>dataset Twitter</i> en el marco experimental 1.	92
5.24. Comparación de los resultados obtenidos por cada algoritmo en comparación al método <i>Baseline</i> en el marco experimental 2.	95
5.25. Conteo de la cantidad veces que cada método obtiene los mejores resultados con el marco experimental 2.	96
5.26. Menores valores de <i>transfer error</i> por cada tarea de adaptación con el <i>dataset Twitter</i> en el marco experimental 2.	97
5.27. Comparación de los resultados obtenidos por ambos marcos experimentales en el <i>dataset Twitter</i>	103
5.28. Mejores resultados obtenidos por cada algoritmo para adaptar cada dominio objetivo en el <i>dataset Twitter</i> utilizando 2000 dimensiones. . . .	104
5.29. Mejores resultados obtenidos por cada algoritmo para adaptar cada dominio objetivo en el <i>dataset Twitter</i> utilizando 1000 dimensiones. . . .	105
5.30. Mejores adaptaciones obtenidas en ambos marcos experimentales, utilizando 2000 dimensiones con el <i>dataset Twitter</i>	106
5.31. Mejores adaptaciones obtenidas en ambos marcos experimentales, utilizando 1000 dimensiones con el <i>dataset Twitter</i>	106
5.32. Conteo de la cantidad veces que cada método obtiene los mejores resultados en ambos marcos experimentales en el <i>dataset Twitter</i>	107
A.1. Comparación de los valores de <i>transfer error</i> entre las distintas dimensiones utilizadas en el <i>dataset Amazon</i> en el marco experimental 1. . . .	122

A.2. Comparación de los valores de <i>transfer error</i> entre las distintas dimensiones utilizadas en el <i>dataset Amazon</i> en el marco experimental 2. . .	123
A.3. Comparación de los valores de <i>transfer error</i> entre las distintas dimensiones utilizadas en el <i>dataset Twitter</i> en el marco experimental 1. . .	123
A.4. Comparación de los valores de <i>transfer error</i> entre las distintas dimensiones utilizadas en el <i>dataset Twitter</i> en el marco experimental 2. . .	124

Índice de figuras

2.1. Clasificación de <i>transfer learning</i>	13
2.2. Representación de un <i>Autoencoder</i>	22
5.1. Baseline In-domain error para el dataset Amazon en ambas dimensiones.	62
5.2. <i>Transfer loss</i> para el dataset Amazon utilizando 3000 dimensiones en el marco experimental 1.	67
5.3. <i>Transfer loss</i> para el dataset Amazon utilizando 1000 dimensiones en el marco experimental 1.	67
5.4. Promedio de los valores de <i>transfer ratio</i> para el <i>dataset Amazon</i> en el marco experimental 1.	71
5.5. <i>Transfer loss</i> para el dataset Amazon utilizando 3000 dimensiones en el marco experimental 2.	73
5.6. <i>Transfer loss</i> para el dataset Amazon utilizando 1000 dimensiones en el marco experimental 2.	73
5.7. Promedio de los valores de <i>transfer ratio</i> para el <i>dataset Amazon</i> en el marco experimental 2.	75
5.8. <i>Transfer loss</i> para el algoritmo <i>GFK</i> utilizando 3000 dimensiones en ambos marcos experimentales.	76

5.9. <i>Transfer loss</i> para el algoritmo <i>GFK</i> utilizando 1000 dimensiones en ambos marcos experimentales.	77
5.10. <i>Transfer loss</i> para el algoritmo <i>PCA</i> utilizando 3000 dimensiones en ambos marcos experimentales.	78
5.11. <i>Transfer loss</i> para el algoritmo <i>PCA</i> utilizando 1000 dimensiones en ambos marcos experimentales.	78
5.12. <i>Transfer loss</i> para el algoritmo <i>SDA</i> utilizando 3000 dimensiones en ambos marcos experimentales.	79
5.13. <i>Transfer loss</i> para el algoritmo <i>SDA</i> utilizando 1000 dimensiones en ambos marcos experimentales.	79
5.14. <i>Transfer loss</i> para el algoritmo <i>mSDA</i> utilizando 3000 dimensiones en ambos marcos experimentales.	80
5.15. <i>Transfer loss</i> para el algoritmo <i>mSDA</i> utilizando 1000 dimensiones en ambos marcos experimentales.	81
5.16. <i>Baseline In-domain error</i> para el <i>dataset Twitter</i> en ambas dimensiones.	88
5.17. <i>Transfer loss</i> para el <i>dataset Twitter</i> utilizando 2000 dimensiones en el marco experimental 1.	90
5.18. <i>Transfer loss</i> para el <i>dataset Twitter</i> utilizando 1000 dimensiones en el marco experimental 1.	91
5.19. Promedio de los valores de <i>transfer ratio</i> para el <i>dataset Twitter</i> en el marco experimental 1.	93
5.20. <i>Transfer loss</i> para el <i>dataset Twitter</i> utilizando 2000 dimensiones en el marco experimental 2.	94
5.21. <i>Transfer loss</i> para el <i>dataset Twitter</i> utilizando 1000 dimensiones en el marco experimental 2.	95

5.22. Promedio de los valores de <i>transfer ratio</i> para el <i>dataset Twitter</i> en el marco experimental 2.	97
5.23. <i>Transfer loss</i> para el algoritmo GFK utilizando 2000 dimensiones en ambos marcos experimentales.	98
5.24. <i>Transfer loss</i> para el algoritmo GFK utilizando 1000 dimensiones en ambos marcos experimentales.	99
5.25. <i>Transfer loss</i> para el algoritmo PCA utilizando 2000 dimensiones en ambos marcos experimentales.	99
5.26. <i>Transfer loss</i> para el algoritmo PCA utilizando 1000 dimensiones en ambos marcos experimentales.	100
5.27. <i>Transfer loss</i> para el algoritmo SDA utilizando 2000 dimensiones en ambos marcos experimentales.	100
5.28. <i>Transfer loss</i> para el algoritmo SDA utilizando 1000 dimensiones en ambos marcos experimentales.	101
5.29. <i>Transfer loss</i> para el algoritmo mSDA utilizando 2000 dimensiones en ambos marcos experimentales.	102
5.30. <i>Transfer loss</i> para el algoritmo mSDA utilizando 1000 dimensiones en ambos marcos experimentales.	102
5.31. Tiempos de ejecución para el dataset Amazon utilizando 3000 dimensiones en el marco experimental 1.	108
5.32. Tiempos de ejecución para el dataset Amazon utilizando 3000 dimensiones en el marco experimental 2.	109

Glosario

algoritmo: Conjunto de instrucciones secuenciales para realizar una determinada tarea.

benchmark: Punto de referencia o estándar de desempeño con el cual las cosas pueden ser comparadas o medidas.

cross-validation: Técnica de evaluación de resultados, que busca probar la independencia entre estos resultados y las particiones de un conjunto de datos.

dataset: Conjunto de datos.

deep learning: Conjunto de algoritmos de aprendizaje automático inspirados en la estructura y funcionamiento del cerebro humano.

dimensionalidad: Cantidad de atributos que uno o varios datos poseen.

entorno virtual: En el área de la computación, corresponde a una representación de un entorno de trabajo existente que no modifica a uno real.

epoch: Ciclo de entrenamiento que se ejecuta sobre un conjunto de datos.

grafo bipartito: Grafo cuyos vértices pueden ser separados en dos grupos, sin que sus aristas tengan conexiones entre vértices del mismo grupo.

i.i.d: Sigla de *independiente e idénticamente distribuida*. En estadística esto se refiere a variables que presentan una misma distribución de probabilidad y que son independientes unas de otras.

métrica: Medida o conjunto de medidas que sirven para estimar el desempeño u otra característica de un *software*.

nube: Paradigma de computación que permite ejecutar servicios computacionales a través de *internet*.

paquete: En programación, un paquete o librería es un conjunto de clases e interfaces relacionadas que permiten realizar un trabajo particular.

tradeoff: Concepto que representa un balance a obtener entre dos condiciones incompatibles.

v.a.: Sigla de variables aleatorias.

Capítulo 1

Introducción

1.1. Definición del Problema

El aprendizaje automático (*machine learning*) consiste en crear programas para realizar tareas, optimizando el rendimiento de este programa a través de un criterio y de la utilización de datos de ejemplo. El “aprendizaje” o “entrenamiento” de un programa (también conocido como modelo) se da en casos en los que no es posible definir explícitamente las reglas que resuelven un problema, debido a la complejidad de estas o a la falta de conocimiento previo. Tomando por ejemplo la clasificación de una imagen en distintos tipos de objetos, se puede notar que definir las reglas para clasificar a esta dentro de una categoría u otra, tendría una enorme complejidad, tanto de tiempo como en los conocimientos requeridos para que el modelo creado tenga un buen desempeño, sin mencionar que esto debería realizarse cada vez que los parámetros a considerar puedan cambiar. Dado esto, sería ideal tener un método que se adapte a las distintas circunstancias del problema a resolver sin requerir que sus reglas sean definidas explícitamente. Con el uso del aprendizaje automático es posible realizar esto, utilizando grandes cantidades de datos de ejemplo (también considerado como experiencia previa) un modelo puede “aprender” a realizar una tarea con un buen desempeño en diversos tipos de tareas. Por ende, se hace evidente la importancia que tiene el uso y la disponibilidad de los datos para el aprendizaje automático.

Dentro de las técnicas de aprendizaje automático con mayor desarrollo y que han alcanzado mejores desempeños se encuentran las llamadas “técnicas supervisadas”, que para realizar el entrenamiento de un modelo utilizan, como parte de sus datos disponibles, un valor (o etiqueta) como objetivo de su tarea a realizar y también para medir el desempeño del modelo. Sin embargo, obtener las etiquetas suele ser un proceso lento y caro, ya que muchas veces no puede realizarse automáticamente y requiere de trabajo humano para obtener una cantidad suficiente y de buena calidad.

Actualmente las aplicaciones del aprendizaje automático han alcanzado gran éxito en el mundo. La cantidad de datos disponibles ha permitido que se obtengan resultados de mucha utilidad, existiendo datasets de gran tamaño, de los cuales se han adquirido conocimientos relevantes. Sin embargo, no siempre se cuenta con la cantidad de datos suficientes con la cual obtener el conocimiento necesario para entrenar un buen modelo y así obtener buenos resultados, o incluso teniendo la cantidad de datos suficientes, los datos con los cuales se utilizan los modelos generados no son del todo similares a los utilizados durante el entrenamiento de este, causando que la calidad de los resultados del modelo sea distinta a la esperada o que su desempeño empeore.

El caso de interés a tratar corresponde a lo siguiente: se tienen dos grupos de datos (*datasets*), el primero con la suficiente cantidad de datos etiquetados y del cual se ha obtenido conocimiento relevante a través de un modelo entrenado, mientras que el segundo grupo no posee datos etiquetados o la suficiente cantidad, por lo cual se espera utilizar el conocimiento adquirido del primer *dataset* para entrenar una máquina que cumpla una misma función con los datos del segundo *dataset*. El caso descrito anteriormente puede ser resuelto mediante lo que es conocido en la literatura como adaptación entre dominios (*domain adaptation*).

Variadas técnicas de adaptación entre dominios han sido propuestas, que sirven de forma general para diversas tareas o que son realizadas para tareas específicas, sin embargo, no existe una definición única que sea utilizada como base para definir a los problemas de adaptación entre dominios. En variados casos, la definición del problema a resolver se menciona sin muchos detalles, y por otro lado, existen propuestas para resolver el caso descrito anteriormente, pero bajo nombres distintos a los de adaptación

entre dominios, dificultando la búsqueda y el estudio de nuevas propuestas.

El trabajo presente pretende establecer una definición formal de lo que corresponde a un problema de adaptación entre dominios, y partiendo de esta definición luego se realizan estudios a diversas propuestas que cumplan con la definición establecida. Luego, aprovechando las herramientas disponibles para la programación de estas soluciones, se implementa una selección de las propuestas estudiadas.

Mediante el uso de *Python* y librerías como *scikit-learn*, *Theano* y *Keras*, cuyos usos en el aprendizaje automático han tenido gran popularidad, y plataformas como *Amazon Web Services* (AWS) es posible realizar las implementaciones y pruebas deseadas de forma tal que sean replicables y con una dificultad moderada.

El fin último de implementar dichas propuestas es el de realizar pruebas de comparación en distintos *datasets*, con el fin de descubrir las ventajas y desventajas de cada algoritmo, y tener un punto de comparación concreto entre estos algoritmos.

1.2. Objetivos

1.2.1. Objetivos Generales

- Realizar una revisión de los estudios más relevantes relacionados a la adaptación entre dominios.
- Probar y comparar experimentalmente una selección de los estudios revisados anteriormente.
- Generar una base de conocimientos, software y datos que permita acelerar investigaciones o aplicaciones futuras.

1.2.2. Objetivos Específicos

- Definir y formalizar matemáticamente la adaptación entre dominios.

- Revisar la literatura disponible sobre el problema, clasificando los métodos disponibles en un conjunto significativo de categorías.
- Seleccionar un conjunto relevante y expresivo de problemas que permitan comparar las técnicas de interés.
- Implementar en el lenguaje *Python* una selección de métodos y experimentar en un número definido de *datasets*.
- Comparar experimentalmente, en términos de precisión y eficiencia, las técnicas seleccionadas.
- Generar conclusiones que permitan orientar el trabajo futuro y la construcción de una propuesta original en el problema.

1.3. Estructura del Documento

El trabajo de Memoria presente continúa con los capítulos necesarios para obtener los objetivos planteados y mostrar de una manera estructurada las etapas necesarias para cumplirlos.

El Capítulo 2 presenta los principales tópicos a conocer para poder realizar una definición de la adaptación entre dominios. Entre estos tópicos se incluyen los conceptos de aprendizaje automático y *transfer learning*. También, habiendo definido la adaptación entre dominios, se presentan las publicaciones estudiadas en relación a esta.

En el Capítulo 3 se da a conocer la selección de métodos a implementar. También se describen las herramientas a utilizar para cada implementación y la instalación del entorno de trabajo que permite replicar los experimentos realizados.

En el Capítulo 4 se describe el método experimental a utilizar, la configuración que utiliza cada prueba. También se exponen los *datasets* a utilizar en las pruebas y las métricas que permiten medir el desempeño y comparar los resultados entre cada método.

En el Capítulo 5 se muestran y analizan los resultados de las distintas pruebas realizadas.

Finalmente se presentan las conclusiones obtenidas de la realización de este trabajo, el trabajo a futuro a realizar y las referencias y anexos pertinentes.

Capítulo 2

Estado del Arte

Para comprender de mejor manera lo que la adaptación entre dominios implica y la principal aplicación considerada en esta memoria, es necesario comprender primero lo que es el aprendizaje automático, el análisis de sentimientos (el cual surge de este) y su relación con el procesamiento de lenguaje natural (*natural language processing* o *NLP*). Luego, habiendo presentado estos temas, se da pie a uno de los problemas que surgen y que es resuelto mediante *transfer learning*, y del cual la adaptación entre dominios es uno de los casos.

El estado del arte tiene la siguiente estructura: Se realiza una introducción al aprendizaje automático. Luego se introduce el concepto de análisis de sentimientos y *NLP*, continuando con una definición de *transfer learning*, explicando su taxonomía y presentando el problema de interés en el trabajo presente. Una vez conocido el problema de interés, se realiza la exposición de las publicaciones estudiadas que abordan la adaptación entre dominios. Finalmente se muestra conceptualmente la forma en que se representan los datos de texto para poder ser utilizados por los distintos algoritmos de clasificación y adaptación, además de presentar medidas útiles para conocer la similitud existente entre distintos dominios.

2.1. Aprendizaje Automático

El aprendizaje automático, o *machine learning*, se enfoca en tratar de aprender a realizar una tarea de manera automática a través de la experiencia, sin definir explícitamente las instrucciones para realizarla. De acuerdo a la Definición 2.1 (dada en [11]) se pueden distinguir los elementos principales que definen el aprendizaje automático. En la mayoría de los casos, la experiencia E corresponde a un conjunto de datos conocidos como datos de entrenamiento, la tarea T puede ser expresada mediante una función matemática $f(\cdot)$, y la medida de desempeño mediante una función de pérdida $l(\cdot)$, la cual puede recibir un conjunto conocido como datos de prueba.

Complementando lo anterior, es necesario explicitar la diferencia existente entre los datos de entrenamiento y los datos de prueba. El primer conjunto de datos tiene un uso exclusivo para ajustar o entrenar un modelo, mientras que el segundo conjunto de datos se utiliza para medir el desempeño del modelo entrenado. Es importante no mezclar los datos de ambos conjuntos, ya que el uso de datos de prueba al entrenar un modelo puede dar resultados poco certeros al momento de probar su desempeño, y también puede ocurrir lo mismo al utilizar datos de entrenamiento para probar el desempeño del modelo.

Definición 2.1. “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .”¹

Dependiendo de los tipos de datos de entrenamiento que se asuman disponibles, se puede categorizar el aprendizaje automático en supervisado (*supervised learning*), no supervisado (*unsupervised learning*) y semi-supervisado (*semi-supervised learning*).

¹Un programa es dicho que aprende de una experiencia E con respecto a una clase de tareas T y medida de desempeño P , si su desempeño en las tareas T , tal como es medido por P , mejora con la experiencia E .

2.1.1. Aprendizaje Supervisado

En el aprendizaje supervisado la tarea a realizar tiene como propósito la predicción de un valor asociado a un determinado patrón de entrada (imagen, trozo de texto, trozo de audio, etc.), a partir de un conjunto de atributos o características que describen el patrón de entrada. Una máquina de aprendizaje es entrenada con datos en los cuales se conocen los valores a predecir. Estos valores son conocidos como etiquetas y los datos son conocidos como datos etiquetados. Los tipos de tarea que se pueden realizar son:

- **Clasificación:** De acuerdo a una instancia (un dato o ejemplo) se trata de predecir una salida correspondiente a un valor numérico discreto o una categoría. Un ejemplo de esto es la categorización de un texto en positivo o negativo dado el conjunto de palabras que componen a este texto.
- **Regresión:** De acuerdo a una instancia se trata de predecir una salida correspondiente a un valor numérico continuo. Un ejemplo de esto es la predicción del valor de una casa dado su tamaño.

Más formalmente hablando, un método de aprendizaje se dice supervisado cuando se intenta predecir la distribución condicional de una v.a. Y (típicamente univariada) a otra v.a. X (típicamente multivariada), o algunos de sus momentos (por ejemplo su valor esperado); y se asume que para realizar esta tarea se dispone de una muestra de casos de la forma (x_i, y_i) obtenidos muestreando la distribución de probabilidad conjunta de X e Y .

Un ejemplo de problema de clasificación, de gran interés en esta memoria, es el problema conocido como análisis de sentimientos (*sentiment analysis*).

2.1.2. Aprendizaje No Supervisado

El aprendizaje no-supervisado incluye técnicas capaces de descubrir estructuras y relaciones entre los datos sin necesidad de utilizar etiquetas. Los tipos de tarea que se pueden realizar son:

- **Clustering:** Se trata de encontrar agrupaciones inherentes entre los datos. Un ejemplo es agrupar compradores de acuerdo a los productos comprados en una tienda.
- **Asociación:** En el aprendizaje de reglas de asociación se trata de encontrar reglas que permitan describir grandes porciones de los datos. Un ejemplo de esto es obtener que las personas que vieron una determinada película también tienden a ver otra determinada película.
- **Reducción de dimensionalidad:** Se busca comprimir los datos dejando solamente la información más valiosa o representativa de un grupo de datos.

2.1.3. Aprendizaje Semi-supervisado

En el aprendizaje semi-supervisado se tienen grandes cantidades de datos de entrenamiento, sin embargo, la cantidad de datos etiquetados que se posee es baja, utilizando una mezcla de aprendizaje supervisado y no supervisado para resolver estos problemas.

2.2. Análisis de Sentimientos

El análisis de sentimientos, tal como fue mencionado en la sección anterior, corresponde a la tarea de clasificar un texto dentro de un conjunto de etiquetas que representen un sentimiento o una polaridad. Comúnmente las etiquetas designadas son positiva, negativa o neutral, y el texto a clasificar puede variar desde una palabra, a una

oración o un documento completo. Puede ser realizado utilizando técnicas de aprendizaje automático, modelos estadísticos y *Natural Language Processing* (o *NLP*). En donde este último permite realizar análisis de mayor profundidad a los textos utilizando información sobre el lenguaje tratado, realizando análisis léxicos, sintácticos y semánticos entre otros.

Sin embargo, a pesar de la disponibilidad de técnicas de **NLP**, el alcance del trabajo a realizar se enfoca mayormente en el aprendizaje automático y las soluciones con base en modelos estadísticos.

2.3. Transfer Learning

En las aplicaciones realizadas mediante aprendizaje automático se tiene la suposición de que los datos de entrenamiento y de prueba pertenecen al mismo espacio de características y tienen la misma distribución de probabilidades. Al cambiar alguno de estos, es necesario reentrenar los modelos anteriores con los datos de entrenamiento necesarios, sin embargo, obtener la cantidad de datos necesarios para obtener buenos resultados puede ser costoso, ya sea hablando de trabajo humano o de tiempo empleado. Es entonces donde surge *Transfer Learning*, el cual pretende utilizar los conocimientos adquiridos previamente para nuevos problemas, considerando que los dominios, tareas y distribuciones utilizadas en el entrenamiento y la prueba sean diferentes. De acuerdo al trabajo realizado por Pan y Yang en [13], el concepto de adaptación entre dominios corresponde a un caso de una de las categorías en las que se pueden agrupar las técnicas de *Transfer Learning*. Para entender mejor la adaptación entre dominios y la clasificación mencionada es necesario dar definiciones con las cuales se trabajan en [13].

Primero es necesario definir los conceptos de “dominio” y “tarea”. Un dominio $\mathcal{D} = \{\mathcal{X}, P(X)\}$ consiste de dos componentes: un espacio de características $\mathcal{X}$ y una distribución de probabilidad marginal $P(X)$, $X \in \mathcal{X}$. Luego, dado un dominio $\mathcal{D}$, una tarea $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$ consiste de dos componentes: un espacio de etiquetas $\mathcal{Y}$ y una

función predictiva $f(\cdot)$, la cual puede ser aprendida de los datos de entrenamiento, los cuales consisten en pares $\{x_i, y_i\}$, en donde $x_i \in \mathcal{X}$ e $y_i \in \mathcal{Y}$. La función $f(\cdot)$ puede ser utilizada para predecir la etiqueta de una instancia x . Desde un punto de vista probabilístico $f(x)$ puede ser entendida como $P(y|x)$ o bien, como una característica de esta (por ejemplo: un valor esperado en el caso de una regresión). También se considera que existe un dominio fuente $\mathcal{D}_S$ y un dominio objetivo $\mathcal{D}_T$, que pueden diferir en sus espacios de características o su distribución de probabilidades. De esta manera, *transfer learning* puede definirse de acuerdo de la Definición 2.2.

Definición 2.2. (Transfer Learning): Dado un dominio $\mathcal{D}_S$, una tarea de aprendizaje $\mathcal{T}_S$, un dominio objetivo $\mathcal{D}_T$ y una tarea de aprendizaje $\mathcal{T}_T$, *transfer learning* intenta mejorar el aprendizaje de la función predictiva objetivo $f_T(\cdot)$ en $\mathcal{D}_T$ utilizando el conocimiento sobre $\mathcal{D}_S$ o $\mathcal{T}_S$, en donde $\mathcal{D}_S \neq \mathcal{D}_T$ o $\mathcal{T}_S \neq \mathcal{T}_T$.

Dados los casos posibles que pueden existir con los distintos dominios y tareas, así como también según la disponibilidad de datos etiquetados en el dominio objetivo, se tienen 3 clasificaciones de *transfer learning*, además de acuerdo a qué es lo transferido para lograr un aprendizaje se tienen 4 clasificaciones para esto. Las clasificaciones de *transfer learning* son:

- **Inductive Transfer Learning:** La tarea objetivo es diferente de la tarea fuente, sin importar si los dominios son iguales. Además se considera que existe una poca cantidad de datos etiquetados en el dominio objetivo, y dependiendo de la disponibilidad de datos etiquetados en el dominio fuente se tienen los siguientes casos:
 - Se tiene una gran cantidad de datos etiquetados en el dominio fuente, en cuyo caso se trabaja de manera similar a *multitask learning*, sin embargo, en *inductive transfer learning* se trata de mejorar solamente los resultados en la tarea objetivo, mientras que en *multitask learning* se trata de mejorar los resultados en las tareas fuente y objetivo.
 - No se tienen datos etiquetados en el dominio fuente, en cuyo caso se trabaja de manera similar al aprendizaje semi-supervisado.

- ***Transductive Transfer Learning***: Las tareas fuente y objetivo son las mismas, mientras que los dominios fuente y objetivo son distintos. Además se considera que no hay datos etiquetados en el dominio objetivo, mientras que se tiene una gran cantidad de datos etiquetados en el dominio fuente. Los casos posibles que se pueden presentar son:
 - Los espacios de características de los dominios fuente y objetivo son distintos, $\mathcal{X}_S \neq \mathcal{X}_T$.
 - Los espacios de características de los dominios fuente y objetivo son iguales, pero las distribuciones de probabilidad marginal de los dominios fuente y objetivo son distintos, $P(X_S) \neq P(X_T)$. Este último caso es similar a los requerimientos que se piden en ***adaptación entre dominios*** y *sample selection bias*.
- **Unsupervised Transfer Learning**: La tarea objetivo es diferente, pero relacionada a la tarea fuente. También se considera que no hay datos etiquetados en ambos dominios. El enfoque de esta clasificación es para resolver problemas de aprendizaje no supervisado.

Una representación de la clasificación expuesta en base a la disponibilidad de los datos del dominio fuente y objetivo se puede observar en la Figura 2.1.

Las formas de resolver los problemas de *transfer learning* son basadas de acuerdo a qué es lo que se transfiere durante el entrenamiento. Estas son:

- Transferencia del conocimiento de las instancias: Se considera que algunas instancias del dominio fuente pueden ser útiles para el entrenamiento en el dominio objetivo, por lo que se les asigna un peso que determina su importancia en el entrenamiento.
- Transferencia del conocimiento de la representación de características: Se busca encontrar una buena representación de las características que ayuden a disminuir la diferencia entre los dominios fuente y objetivo, luego con este nuevo dominio se entrena un nuevo modelo.

- Transferencia del conocimiento de los parámetros: Similar a *multitask learning*, en donde se entrenan modelos simultáneamente utilizando un mismo conjunto de parámetros, en *transfer learning* se busca mejorar solo el rendimiento del modelo en el dominio objetivo, teniendo la posibilidad de asignar distintos valores a los parámetros según la importancia que se le da a cada dominio.
- Transferencia del conocimiento relacional: Se asume que los datos disponibles no siguen un comportamiento independiente e idénticamente distribuidos, por lo que trata de transferir las relaciones entre los datos desde el dominio fuente al dominio objetivo.

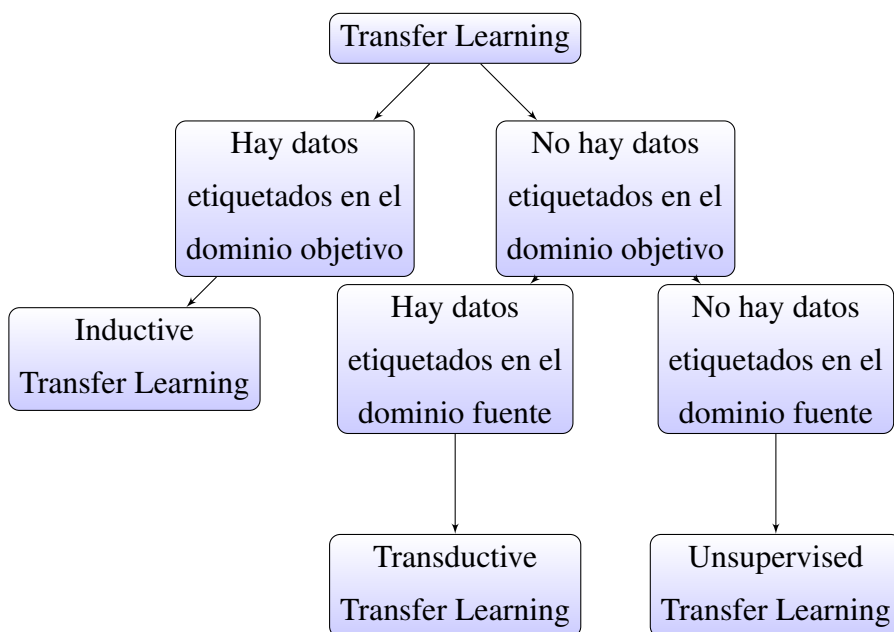


Figura 2.1: Clasificación de *transfer learning*

El caso de interés para este trabajo es aquel en donde se tiene una gran cantidad de datos etiquetados en el dominio fuente, mientras que para el dominio objetivo no se tienen datos etiquetados. De acuerdo a la clasificación según los datos disponibles, dada anteriormente, se tiene que el caso de interés cae dentro de las categorías de *transductive transfer learning*.

En [13] se considera que la adaptación entre dominios cae dentro de la categoría de *transductive transfer learning*, quedando establecida según la Definición 2.3. Sin

embargo, se considera que la Definición 2.3 restringe demasiado los casos posibles a estudiar, tanto en los problemas como en las soluciones, por lo que en este trabajo se considera la adaptación entre dominios como un sinónimo de *transfer learning*. Además, en este trabajo se toma en cuenta la clasificación de *transfer learning*, dada anteriormente, principalmente como una referencia para reconocer casos posibles de interés.

Definición 2.3. (Adaptación entre dominios): Dado un dominio $\mathcal{D}_S$, con muchos datos etiquetados, una tarea de aprendizaje $\mathcal{T}_S$, un dominio objetivo $\mathcal{D}_T$, sin datos etiquetados, y una tarea de aprendizaje $\mathcal{T}_T$. La adaptación entre dominios intenta mejorar el aprendizaje de la función predictiva objetivo $f_T(\cdot)$ en $\mathcal{D}_T$, utilizando el conocimiento en $\mathcal{D}_S$ y $\mathcal{T}_S$, en donde $\mathcal{D}_S \neq \mathcal{D}_T$ y $\mathcal{T}_S = \mathcal{T}_T$. En donde $\mathcal{X}_S = \mathcal{X}_T$ y $P(X_S) \neq P(X_T)$.

Por último, en este trabajo se considera a un *dataset* como a un conjunto de 1 o más dominios, en donde las distribuciones de probabilidad del dataset y de cada dominio pueden ser entendidas de las siguientes formas:

- Cada dominio tiene un espacio de características $\mathcal{X}$ distinto, y por consiguiente el *dataset* tiene un espacio de características que representa la unión de los espacios de características de todos los dominios.
- Cada dominio tiene el mismo espacio de características $\mathcal{X}$, al igual que el dataset, y estos se diferencian en sus distribuciones de probabilidad.

A pesar de que ambas descripciones cumplen con lo expresado en la definición 2.2, por razones prácticas se utiliza la última descripción dada. Considerando además que los valores $P(Y|X)$ de cada dominio pueden ser distintos.

2.4. Publicaciones sobre adaptación entre dominios

2.4.1. Cross-Domain Sentiment Classification via Spectral Feature Alignment

2.4.1.1. Resumen general

En [12] se aborda la adaptación entre dominios para la clasificación de sentimientos (nombrado aquí como *cross-domain sentiment classification*) utilizando métodos espectrales para reducir la diferencia entre distintos dominios y así encontrar una representación intermedia que pueda servir para entrenar un dominio sin datos etiquetados.

El algoritmo propuesto de *Spectral Feature Alignment (SFA)* pretende encontrar similitudes entre términos específicos de un par de dominios (palabras dependientes del dominio) mediante su relación con términos que son compartidos por ambos (palabras independientes del dominio). La suposición parte de que la co-ocurrencia de las palabras dependientes del dominio con las mismas palabras independientes del dominio sirve para establecer un nexo entre la funcionalidad que tienen estas últimas para la clasificación de un texto.

Las relaciones encontradas entre palabras independientes y dependientes del dominio son modeladas como un grafo bipartito, y para realizar la asociación entre las funcionalidades de las palabras dependientes del dominio se hace el uso de métodos espectrales.

El algoritmo **SFA** es comparado con otros métodos que realizan trabajos similares, demostrando resultados prometedores en el área de análisis de sentimientos.

2.4.1.2. Propuesta

El problema considerado para resolver se define de la forma: Dados dos dominios específicos D_{src} y D_{tar} , considerados como dominio fuente y objetivo respectivamente. Suponiendo que se tiene un conjunto de datos etiquetados $D_{src} = \{(x_{src_i}, y_{src_i})\}_{i=1}^{n_{src}}$ en el dominio fuente y una cantidad menor de datos no etiquetados $D_{tar} = \{x_{tar_j}\}_{j=1}^{n_{tar}}$ en el dominio objetivo. La tarea de *cross-domain sentiment classification* es aprender un clasificador preciso para predecir la polaridad de datos no vistos de D_{tar} .

El algoritmo propuesto para ejecutar la tarea, en grandes términos, realiza una serie de operaciones entre los datos de dos dominios para obtener un nexo entre las palabras dependientes de cada dominio. Estas operaciones son:

1. Elección de características (también entendidas como palabras) dependientes e independientes de los dominios.
2. Construcción de un grafo bipartito que separa las palabras dependientes e independientes, obtenidas anteriormente.
3. Aplicar *clustering* (mediante métodos espectrales) en el grafo bipartito para alinear (crear un nexo) las palabras dependientes del dominio, obteniendo con esto una función de alineamiento que permite obtener nuevas características.
4. Aumentar los datos originales con las nuevas características obtenidas del alineamiento, balanceando la importancia que tienen estas nuevas características.
5. Entregar un clasificador entrenado con los datos aumentados y las etiquetas originales, el cuál sirve para clasificar los datos del dominio objetivo.

1. Elección de características dependientes e independientes del dominio

Considerando que los datos tienen una cantidad m de características. Para la realización de esta tarea, se consta de las funciones:

- $\phi_{DI}(\cdot)$: que selecciona l características independientes del dominio, las cuales ocurren frecuentemente y actúan de manera similar a través de los dominios fuente (D_{src}) y objetivo (D_{tar}).
- $\phi_{DS}(\cdot)$: que selecciona las $m - l$ características dependientes del dominio. Las cuales pueden ser entendidas como el complemento de las características independientes del dominio.

Para la distinción de cuáles son las palabras dependientes e independientes de los dominios se utiliza la información mutua para medir la dependencia existente entre cada característica y dominio. La forma de medir esta dependencia se realiza mediante la fórmula 2.1, en donde D es una variable correspondiente a cada dominio y X^i corresponde a una característica específica.

$$I(X^i; D) = \sum_{d \in D} \sum_{x \in X^i, x \neq 0} p(x, d) \log_2 \left(\frac{p(x, d)}{p(x)p(d)} \right) \quad (2.1)$$

De la fórmula 2.1 se obtiene que una característica tiene un alto valor de información mutua cuando esta tiene una alta relación con un determinado dominio, por ende es considerada dependiente del dominio. Por otra parte, al tener un valor bajo se considera que tiene una baja relación con el dominio, y por ende, más se puede considerar que esta es independiente del dominio.

2. Construcción del grafo bipartito

Habiendo identificado las características dependientes e independientes del dominio se procede a construir el grafo bipartito, que representa la separación existente entre ambos tipos de características. El grafo bipartito está compuesto de la forma $G = (V_{DS} \cup V_{DI}, E)$, siendo V_{DS} y V_{DI} vértices que representan a las características dependientes e independientes del dominio respectivamente, y E un conjunto de arcos, cuyo peso m representa la frecuencia con la que co-ocurren las características a

través de las instancias. Cabe notar además, que no existe la posibilidad de que un arco conecte un par de vértices en donde ambos pertenezcan a V_{DS} o V_{DI} .

3. Clusterización mediante métodos espectrales

Mediante la clusterización se espera aprender una función $\varphi : \mathbb{R}^{(m-l)} \rightarrow \mathbb{R}^k$ que alinea las características específicas de ambos dominios en k *clusters*, los cuales pueden reducir la diferencia entre las características específicas de los dominios.

Lo que realiza el método espectral es, en términos generales, agrupar en k *clusters* un conjunto de características dependientes de ambos dominios con una cantidad menor de características independientes, en base a las co-ocurrencias modeladas en el grafo bipartito y los pesos de los arcos. Esto permite realizar la comparación entre las funcionalidades que tienen las características dependientes de ambos dominios.

El proceso completo para obtener una función que alinea las características tiene los siguientes pasos:

1. Construir una matriz de pesos $\mathbf{M} \in \mathbb{R}^{(m-l) \times l}$ que representa al grafo bipartito. En esta matriz las filas y columnas representan a las características dependientes e independientes, respectivamente. Luego, una entrada M_{ij} corresponde a la co-ocurrencia de una característica dependiente i y una característica independiente j .
2. Formar una matriz de afinidad $\begin{bmatrix} 0 & \mathbf{M} \\ \mathbf{M}^T & 0 \end{bmatrix} \in \mathbb{R}^{m \times m}$, en donde las primeras $m - l$ filas y columnas corresponden a las características dependientes del dominio, y las últimas l filas y columnas corresponden a las características independientes del dominio.
3. Construir una matriz diagonal $\mathbf{D}$, en donde $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$, y luego construir la matriz $\mathbf{L} = \mathbf{D}^{1/2} \mathbf{A} \mathbf{D}^{-1/2}$.

4. Encontrar los k mayores autovectores de $\mathbf{L}$, definidos como $u_1, u_2 \dots u_k$, y formar la matriz $\mathbf{U} = [u_1 u_2 \dots u_k] \in \mathbb{R}^{m \times k}$.
5. La función de mapeo se define como: $\varphi(x_i) = x_i \mathbf{U}_{[1:m-l,:]}$, en donde $\mathbf{U}_{[1:m-l,:]}$ denota las primeras $m - l$ filas de $\mathbf{U}$, y $x_i \in \mathbb{R}^{m-l}$.

4. Aumento de datos

Dado que es posible no tener identificadas correctamente las características independientes del dominio, causando esto un mal desempeño al alinear las características, se realiza un aumento a la características originales con las características aprendidas mediante la función $\varphi(\cdot)$, utilizando un parámetro γ de *tradeoff* para balancear el efecto de las características originales y las nuevas. La nueva representación de los datos queda de acuerdo a la ecuación 2.2.

$$\tilde{x}_i = [x_i, \gamma \varphi(\phi_{DS}(x_i))] \quad (2.2)$$

En donde $x_i \in \mathbb{R}^{1 \times m}$, $\tilde{x}_i \in \mathbb{R}^{1 \times m+k}$ y $0 \leq \gamma \leq 1$.

Finalmente, se entrega un clasificador entrenado con los datos transformados del dominio fuente y sus respectivas etiquetas (punto 5 de la lista).

2.4.1.3. Datasets

El primer *dataset* utilizado proviene de [2], el cual consiste en una colección de reseñas de productos de Amazon. Las reseñas son sobre cuatro dominios distintos: **libros, dvds, electronicos y artículos de cocina**. Cada reseña tiene una etiqueta positiva (+1) o negativa (-1), basada en el puntaje que le asigna el autor de la reseña al producto. Cada dominio consta de 1000 reseñas positivas y 1000 reseñas negativas. A este *dataset* se le denomina *RevDat*.

El segundo *dataset* utilizado es generado por los autores, con reseñas obtenidas

de los sitios Amazon, Yelp y CitySearch. Los dominios de las reseñas de Amazon son: **videojuegos (V)**, **electrónica (E)** y **software (S)**. El dominio de los sitios Yelp y CitySearch corresponde a **hoteles (H)**. En vez de asignar a cada reseña una etiqueta, se dividen las reseñas en oraciones y manualmente se le asigna una etiqueta positiva o negativa a cada una de estas. Cada dominio tiene seleccionados aleatoriamente 1500 oraciones positivas y 1500 oraciones negativas. Este *dataset* es denominado *SentDat*.

2.4.1.4. Método experimental

Se compara **SFA** con otros métodos de referencia:

- **NoTransf**: Es un clasificador entrenado directamente con los datos del dominio fuente.
- **LSA**: Es un clasificador entrenado con una representación aumentada de los datos, obtenida aplicando un análisis semántico latente (que también puede ser entendido como realizar *PCA*) a las características específicas del dominio.
- **FALSA**: Es un clasificador entrenado con una representación aumentada de los datos, obtenida aplicando un análisis semántico latente a la matriz de co-ocurrencia de características dependientes e independientes de ambos dominios.
- **SCL**: Algoritmo que utiliza datos no etiquetados de ambos dominios para encontrar representaciones conjuntas de baja dimensionalidad de los datos.

Como clasificador de sentimientos se utiliza regresión logística en **NoTransf**, **LSA**, **FALSA** y **SFA**. Para todos los experimentos en *RevDat* se divide cada dominio en un conjunto de entrenamiento de 1600 instancias y conjunto de prueba de 400 instancias. Para todos los experimentos en *SentDat* se divide cada dominio en un conjunto de entrenamiento de 2000 instancias y conjunto de prueba de 1000 instancias. Se reportan los resultados del promedio de 5 ejecuciones aleatorias.

2.4.1.5. Resultados

En los resultados se tiene que el método **SFA** propuesto da mejores resultados que los otros métodos, incluyendo a **SCL** en varias de las pruebas al utilizar el *dataset Rev-Dat*. De la misma manera, con el *dataset SentDat* se obtiene una mejoría del método propuesto en comparación a los otros métodos.

2.4.2. Domain Adaptation for Large-Scale Sentiment Classification:

A Deep Learning Approach

2.4.2.1. Resumen general

En [6] se propone una solución de adaptación entre dominios utilizando *Deep Learning* para extraer características de manera no supervisada entre ambos dominios. La aplicación de la propuesta se realiza en clasificadores de sentimientos, comparando experimentalmente los resultados con otros métodos base.

2.4.2.2. Propuesta

Los algoritmos de *Deep Learning* han sido capaces de capturar, hasta cierto punto, factores subyacentes que permiten explicar cambios en los datos. Además de esto, a partir de un conjunto de características, han permitido identificar propiedades invariantes entre los datos, lo cual es de gran interés para ser aplicado en adaptación entre dominios para análisis de sentimientos, ya que si estos pueden permitir descubrir propiedades invariantes entre ambos dominios, entonces permitirán realizar una mejor transferencia de conocimiento.

La herramienta que se utiliza en esta propuesta para aplicar *Deep Learning* es el *Stacked Denoising Autoencoder (SDA)*, el cual permite obtener representaciones de manera no supervisada de un conjunto de datos.

Stacked Denoising Autoencoders

Un *Autoencoder* es una red de 3 capas, la cual es entrenada con el propósito de reconstruir su entrada. La primera capa corresponde a una entrada x , que mediante una función de codificación $h(\cdot)$ es convertida a una representación oculta y en la segunda capa, y que en la tercera capa es convertida a una versión reconstruida z mediante una función $g(\cdot)$, de modo que $g(h(x)) = z$. Los *autoencoders* son típicamente entrenados para reducir un error de reconstrucción $l(x, z)$. La arquitectura descrita anteriormente puede observarse en la figura 2.2.

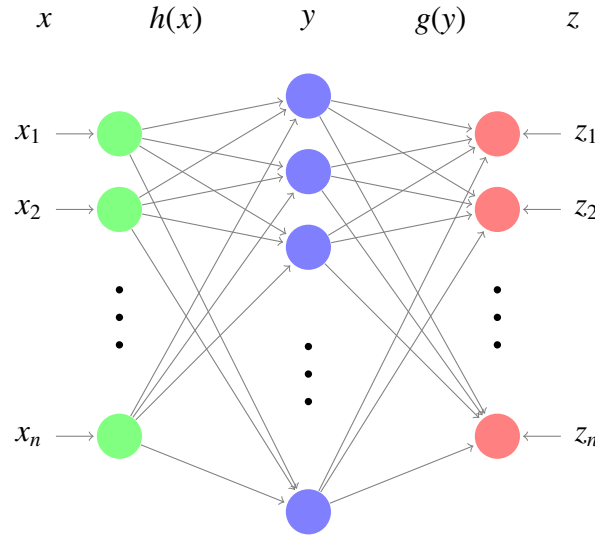


Figura 2.2: Representación de un *Autoencoder*.

De una forma similar, el vector de entrada x puede ser corrompido estocásticamente en un vector $\tilde{x}$ con probabilidad $p > 0$, de esta forma evitando que un *Autoencoder* simplemente haga la operación de copiar su entrada. Esto es conocido como un *Denoising Autoencoder (DA)*.

En el caso de un vector binario, las entradas de un *DA* son corrompidas estableciendo los valores de algunas dimensiones a 0, con probabilidad $p > 0$. Por otra parte, el error de reconstrucción a minimizar durante el entrenamiento se conoce como *cross-entropy loss*, definido según la ecuación 2.3. El entrenamiento de un *DA* se hace utilizando el algoritmo *stochastic gradient descent*, el cual calcula las variaciones que

deben tener los parámetros para disminuir los errores, utilizando información de una cantidad reducida de instancias (del total de los datos de entrenamiento) a la vez.

$$L(x, \tilde{x}) = - \sum_j [x_j \log \tilde{x}_j + (1 - x_j) \log(1 - \tilde{x}_j)] \quad (2.3)$$

Adicionalmente, varios *denoising autoencoders* pueden ser apilados para formar una red profunda conocida como *Stacked Denoising Autoencoder* o *SDA*. Esta red es formada de modo que la representación oculta de un *denoising autoencoder* pase a ser la capa de entrada del *denoising autoencoder* de la siguiente capa. El entrenamiento para minimizar el error de reconstrucción se hace una capa a la vez, en un proceso llamado *pre-entrenamiento* (o *pre-training*). Una vez que se tiene entrenada una capa, es posible realizar el entrenamiento de la capa siguiente con las representaciones ocultas de la capa recientemente entrenada.

Una vez que todas las capas han sido entrenadas, la red puede pasar por una segunda etapa conocida como *fine-tuning*. En esta etapa, la capa de salida de más alto nivel de la red (o bien, de la última capa) es utilizada como entrada para algún algoritmo de aprendizaje supervisado. Sin embargo, a pesar de existir esta etapa, para los efectos de la propuesta realizada, solamente se realiza la etapa de *pre-training*.

Al utilizar *SDA* con los datos de los dominios fuente y objetivo se espera obtener las propiedades que son invariantes entre estos dominios.

Dado lo anterior, la propuesta consiste de dos grandes pasos:

1. Realizar una extracción de alto nivel de las características de ambos dominios utilizando un *SDA*.
2. Entrenar un clasificador con datos etiquetados transformados del dominio fuente. El tipo de clasificador utilizado es una *SVM*.

2.4.2.3. Dataset

Se utiliza el dataset de [2], consistente de reseñas de productos de Amazon para cuatro categorías distintas: **libros**, **dvds**, **electronicos** y **artículos de cocina**, cada una de estas categorías corresponde a un dominio. Se utilizan 1000 reseñas positivas y 1000 reseñas negativas para cada dominio en los dominios fuente, asignando 1600 para entrenamiento y 400 para pruebas en cada dominio.

Cada reseña es tratada como un *bag-of-words*² y transformada en un vector binario que representa la presencia o no de una palabra. Solo las 5000 palabras más frecuentes son dejadas en el conjunto de características.

2.4.2.4. Método experimental

Se compara el método propuesto con otros 3 métodos: **SCL**, **SFA** y **MCT**. Realizando 12 experimentos de adaptación entre dominios para las distintas combinaciones de dominios posibles. Además se utilizan métricas distintas para medir la pérdida de exactitud al realizar la adaptación entre dominios. Primero, se define el *transfer error* $e(S, T)$ como el error obtenido por un clasificador entrenado con datos del dominio fuente S y probado con datos del dominio objetivo T , pudiendo estos datos haber sido o no adaptados, dependiendo de la prueba. Luego, se define el *baseline in-domain error* $e_b(T, T)$, como el error obtenido por un clasificador entrenado y probado con datos del dominio objetivo, sin realizar ningún tipo de adaptación. Con estas dos medidas se define la métrica a utilizar en los experimentos: *transfer loss*, la cual se define como la diferencia entre *transfer error* y *baseline in-domain error*

2.4.2.5. Resultados

De las 12 adaptaciones realizadas, el método propuesto presenta los mejores resultados en 11 de ellos, y en 8 de estos, los resultados son mejores significativamente.

²Representación de las palabras de un texto como una lista, sin considerar orden u otra información.

2.4.3. Domain Adaptation for Object Recognition: An Unsupervised Approach

2.4.3.1. Resumen general

En [8] el problema a resolver es el de reconocimiento de objetos en imágenes, y en menor medida clasificación de texto, aplicando *domain adaptation*. Inicialmente se hace la distinción entre *unsupervised domain adaptation* y *semi-supervised domain adaptation*, en donde en el primer caso no se tienen datos etiquetados en el dominio fuente, y en el segundo caso se tienen pocos datos etiquetados en el dominio fuente.

El método propuesto se basa en representar sub-espacios entre ambos dominios, los cuales son considerados como puntos en un *Grassmaniano*³, y generar puntos de muestra a través de la curva geodésica entre estos. Luego, utilizando la información de estos sub-espacios, se aprende un clasificador para predecir etiquetas en el dominio objetivo.

2.4.3.2. Propuesta

Para una mayor comprensión del método propuesto, es necesario conocer dos conceptos importantes: el *Grassmaniano*, y la curva geodésica. En términos simples, el *Grassmaniano* puede ser visto como una colección de sub-espacios formados a partir de otro espacio vectorial. Por ejemplo: al aplicar reducción de dimensionalidad mediante *PCA* a un conjunto de datos, el sub-espacio obtenido por esta nueva representación pasaría a ser parte del *Grassmaniano*, y el conjunto de todos los sub-espacios con la misma dimensionalidad conformarían el *Grassmaniano*. De esta estructura, es posible obtener información tanto geométrica (por ejemplo: distancias), como probabilística. Luego, pensando en el *Grassmaniano* con una esfera, en donde cada sub-espacio corresponde a un punto de esta, la curva geodésica corresponde a la serie de sub-espacios que conectan a dos puntos con una distancia mínima.

³Grassman Manifold. Para más información, visitar <https://www.math.hmc.edu/~ursula/teaching/math189/finalpapers/dhruv.pdf>

El método propuesto considera los dominios fuente y objetivo como puntos en un *Grassmanniano*, los cuales son unidos mediante una curva geodésica. Luego, al moverse por cada punto de la curva se pueden observar las características que cambian y las que se mantienen invariantes, por lo que estas características invariantes pueden ayudar a realizar una adaptación entre dominios.

La estrategia a seguir para realizar la adaptación entre dominios consiste en lo siguiente:

1. Construir la curva geodésica⁴, con los dominios fuente y objetivo como puntos de inicio y fin, respectivamente.
2. Obtener un conjunto de puntos de la curva geodésica.
3. Proyectar los datos de los dominios fuente y objetivo en los sub-espacios (puntos de la curva) obtenidos y concatenarlos, obteniendo un nuevo vector de sub-espacios.
4. Reducir la dimensionalidad de los sub-espacios del vector.
5. Utilizar las representaciones obtenidas como vectores de características para construir un clasificador⁵.

Para implementar el algoritmo propuesto, se necesita ajustar una serie de parámetros: la dimensionalidad N de los datos a utilizar, y la dimensionalidad d de los sub-espacios que conforman la curva geodésica. Además, para la reducción de dimensionalidad, se utiliza el algoritmo *PLS* (Véase Apéndice C), con el cual se debe escoger la cantidad p de dimensiones a reducir.

⁴La curva geodésica es construida mediante los pasos expuestos en [8].

⁵Para el dominio objetivo se puede dar el caso de que haya una cantidad baja de datos etiquetados, por lo que el clasificador se construye utilizando también estos datos etiquetados en conjunto con los datos del dominio fuente.

2.4.3.3. Dataset

Se utiliza el dataset de [14], el cual posee 31 categorías de objetos diferentes, agrupadas en 3 dominios: imágenes de *amazon*, cámaras *dslr*, y *webcam*. El dominio de *amazon* tiene un promedio de 90 instancias por categoría, mientras que los dominios *dslr* y *webcam* tienen aproximadamente 30 instancias por categoría, reuniendo 4652 imágenes en total. El cambio entre dominios es causado por factores tales como el cambio de resolución, pose, iluminación, etc.

Para realizar pruebas de clasificación de texto se utiliza el dataset presentado en [2], consistente de reseñas de Amazon para los dominios: **libros**, **DVD**, **electrónica** y **artículos de cocina**. Cada dominio tiene una total de 1000 instancias positivas y negativas, divididas en 1600 para entrenamiento y 400 para pruebas.

2.4.3.4. Método experimental

Se utiliza el procedimiento realizado en [14] para el tratamiento de las imágenes. En los experimentos se dejan los valores de $N = 800$, y d entre 185 y 200, la cantidad de sub-espacios intermedios es dejada en 8, mientras que la dimensionalidad p , a la cual el algoritmo *PLS* reduce los datos, es de 30.

Los marcos de experimentación son:

1. Datos etiquetados disponibles tanto en el dominio fuente como objetivo. En el dominio objetivo se tienen 3 instancias etiquetadas por categoría cuando los dominios son *amazon/webcam/dslr*. En el dominio fuente se tienen 8 instancias etiquetadas por categoría cuando los dominios son *webcam/dslr* y 20 cuando el dominio es *amazon*. La tarea consiste en determinar las categorías de los datos no etiquetados en el dominio objetivo.
2. Datos etiquetados disponibles para ambos dominios solamente para la primera mitad de las categorías, mientras que la otra mitad de las categorías solo tienen etiquetas en el dominio fuente. La tarea consiste en determinar las categorías de

los datos no etiquetados, correspondientes a la segunda mitad de las categorías del dominio objetivo.

Para ambos marcos se realizan pruebas de forma no supervisada (sin utilizar información de las etiquetas del dominio objetivo, incluso si hay disponibles) y semi-supervisada (utilizando la información de las etiquetas del dominio objetivo). La exactitud es calculada en base a 20 ejecuciones con diferentes datos en cada dominio, comparando los resultados obtenidos con los expuestos en [14].

Para la clasificación de texto se realiza el tratamiento de datos utilizado en [2], realizando pruebas de forma no supervisada y comparando con lo expuesto en [2] y [3].

2.4.3.5. Resultados

Se obtiene que los resultados de la adaptación en forma no supervisada son ligeramente menores que los expuestos en [14], mientras que en la forma semi-supervisada se obtiene un mejor desempeño.

Para la clasificación de texto se observa que en general los resultados son mejores que los expuestos en [2] y [3].

2.4.4. Geodesic flow kernel for unsupervised domain adaptation

2.4.4.1. Resumen general

En [7] se resuelve el problema de clasificación de imágenes utilizando adaptación entre dominios, realizando una distinción entre adaptación entre dominios no supervisada (*unsupervised domain adaptation*) y adaptación entre dominios semi-supervisada (*semi-supervised domain adaptation*), en donde no se tienen datos etiquetados en el dominio objetivo para el primer caso, y para el segundo caso se cuenta con una pequeña cantidad de datos etiquetados en el dominio objetivo. Para esto se busca identificar un

nuevo espacio de características en el cual el dominio fuente y objetivo manifiesten características compartidas. Partiendo del trabajo realizado en [8] (presentado anteriormente), se busca explotar los sub-espacios entre los dominios fuente y objetivo, los cuales pueden revelar información sobre las similitudes y disimilitudes entre estos, proponiendo un método que extrae el *kernel* del flujo geodésico. Este flujo representa cambios incrementales en las propiedades estadísticas y geométricas entre ambos espacios, y el método propuesto extrae las direcciones de los sub-espacios que se mantienen invariantes entre ambos dominios.

Una segunda tarea que se muestra es la de una métrica que permite ordenar un conjunto de dominios fuente basados en qué tan aptos son para realizar una buena adaptación entre dominios.

2.4.4.2. Propuesta

Tal como fue mencionado anteriormente, la propuesta hecha es un avance a lo realizado en [8]. En [8] se consideran los dominios fuente y objetivo como puntos en el *Grassmanniano*, la propuesta conocida como **SGF** consiste en:

1. Construir la curva geodésica, con los dominios fuente y objetivo como puntos de inicio y fin, respectivamente.
2. Obtener un conjunto de puntos de la curva geodésica.
3. Proyectar los datos de los dominios fuente y objetivo en los sub-espacios (puntos de la curva) obtenidos y concatenarlos, obteniendo un nuevo vector de sub-espacios.
4. Reducir la dimensionalidad de los sub-espacios del vector.
5. Utilizar las representaciones obtenidas como vectores de características para construir un clasificador.

Sin embargo **SGF** tiene varias limitaciones, tales como la poca claridad para el criterio de *sampleo* de sub-espacios desde la curva geodésica, y la cantidad de parámetros que necesitan ser ajustados para lograr un buen desempeño. Estas limitaciones se tratan de superar mediante la utilización de un *kernel*, además de una forma de obtener una dimensionalidad óptima de los sub-espacios.

Los pasos de la propuesta de [7] son:

1. Determinar la dimensionalidad óptima de los sub-espacios.
2. Construir la curva geodésica.
3. Calcular el *kernel* del flujo geodésico (GFK).
4. Utilizar el *kernel* para construir un clasificador con los datos etiquetados.

Para una mayor comprensión de las operaciones a realizar, el orden en el cual se explican los pasos son 2, 3 y 1. Además, se omite una explicación del punto 4, dado que este paso solo corresponde al entrenamiento de un clasificador. La explicación de los pasos mencionados anteriormente es:

2. Construir la curva geodésica:

Sean $\mathbf{P}_S, \mathbf{P}_T \in \mathbb{R}^{D \times d}$ dos bases de los sub-espacios de los dominios fuente y objetivo.

Sea $\mathbf{R}_S \in \mathbb{R}^{D \times (D-d)}$ un complemento ortogonal a $\mathbf{P}_S$, específicamente $\mathbf{R}_S^T \mathbf{P}_S = \mathbf{0}$.

El flujo geodésico es parametrizado de la forma $\Phi : t \in [0, 1] \rightarrow \Phi(t) \in G(d, D)$ bajo las condiciones $\Phi(0) = \mathbf{P}_S$ y $\Phi(1) = \mathbf{P}_T$. Para cualquier t , se define de acuerdo a la ecuación 2.4:

$$\Phi(t) = \mathbf{P}_S \mathbf{U}_1 \Gamma(t) - \mathbf{R}_S \mathbf{U}_2 \Sigma(t) \quad (2.4)$$

en donde $U_1 \in \mathbb{R}^{d \times d}$ y $U_2 \in \mathbb{R}^{(D-d) \times d}$ son matrices ortonormales dadas por las siguientes descomposiciones SVD:

$$P_S^T P_T = U_1 \Gamma V^T, \quad R_S^T P_T = -U_2 \Sigma V^T \quad (2.5)$$

Siendo Σ y Γ matrices diagonales de dimensión $d \times d$. Los elementos diagonales de ambas matrices son $\cos\theta_i$ y $\sin\theta_i$, respectivamente, para $i = 1, 2, \dots, d$. Los valores θ_i son llamados los angulos principales entre P_S y P_T :

$$0 \leq \theta_1 \leq \theta_2 \leq \dots \leq \theta_d \leq \pi/2 \quad (2.6)$$

Los cuales miden el grado en que los sub-espacios se “sobreponen”. Además $\Gamma(t)$ y $\Sigma(t)$ son matrices diagonales cuyos elementos son $\cos(t\theta_i)$ y $\sin(t\theta_i)$ respectivamente.

3. **Calcular el kernel del flujo geodésico (GFK).** El flujo geodésico permite parametrizar los cambios que ocurren desde el dominio fuente hacia el dominio objetivo, al ser considerados estos como dos puntos. Luego, considerando un flujo $\Phi(t)$ para algún $t \in (0, 1)$, y calculando la proyección de un vector x en este, de la forma $\Phi(t)^T x$, se puede entender que si t se encuentra cerca de 0, el vector proyectado va a parecer más si viniese del dominio fuente, mientras que si t se encuentra cerca de 1, el vector proyectado va a parecer más como si viniese del dominio objetivo. Entonces, para obtener una representación de los datos que se adapte de buena manera a ambos dominios y que permita entrenar un buen clasificador, se requiere realizar proyecciones con flujos parametrizados con todos los valores de t posibles.

Para dos vectores x_i y x_j , se calculan sus proyecciones de acuerdo a flujos $\Phi(t)$, calculados con valores continuos t entre 0 y 1, y se concatenan en vectores de dimensionalidad infinita z_i^∞ y z_j^∞ . Luego, el producto interno entre estos dos vectores define el *kernel* del flujo geodésico, según la ecuación 2.7:

$$\langle z_i^\infty, z_j^\infty \rangle = \int_0^1 (\Phi(t)^T x_i)^T (\Phi(t)^T x_j) dt = x_i^T G x_j \quad (2.7)$$

El *kernel* G luego se calcula mediante la fórmula cerrada:

$$G = [P_S U_1 \ R_S U_2] \begin{bmatrix} \Lambda_1 & \Lambda_2 \\ \Lambda_2 & \Lambda_3 \end{bmatrix} \begin{bmatrix} U_1^T P_S^T \\ U_2^T R_S^T \end{bmatrix} \quad (2.8)$$

En donde Λ_1, Λ_2 y Λ_3 son matrices diagonales, cuyos elementos diagonales son:

$$\lambda_{1i} = 1 + \frac{\sin(2\theta_i)}{2\theta_i}, \lambda_{2i} = \frac{\cos(2\theta_i)-1}{2\theta_i}, \lambda_{3i} = 1 - \frac{\sin(2\theta_i)}{2\theta_i}$$

El *kernel* G obtenido, en términos prácticos, es una matriz, con la cual los datos pueden ser proyectados, obteniendo una representación adaptada de estos. Estos datos pueden ser usados posteriormente en el entrenamiento de un clasificador.

1. Determinar la dimensionalidad óptima de los sub-espacios:

Para realizar esto se propone una medida de desacuerdo entre los sub-espacios (SDM), la cual permite escoger automáticamente la dimensionalidad óptima d .

Primero se debe contar con los sub-espacios generados por PCA en ambos dominios, $\mathbf{PCA}_S$ y $\mathbf{PCA}_T$. También se deben unir ambos dominios y calcular su sub-espacio PCA, $\mathbf{PCA}_{S+T}$. Intuitivamente, si los dos dominios son similares, entonces los tres sub-espacios deberían encontrarse cerca en el *Grassmaniano*. Entonces SDM se define de acuerdo a los ángulos principales:

$$\mathcal{D}(d) = 0,5[\sin\alpha_d + \sin\beta_d] \quad (2.9)$$

2.4.4.3. Datasets

Se utilizan 3 datasets que aparecen en [14]: *amazon*, *webcam* y *dslr* y además se utiliza el dataset *Caltech-256* [9], considerando cada dataset como un dominio distinto. Además se extraen 10 clases comunes en los cuatro datasets: *Video-projector*, *Touring-bike*, *Laptop-101*, *Calculator*, *Head-phones*, *Backpack*, *Computer-keyboard*, *Computer-monitor*, *Computer-mouse* y *Coffee-mug*. Se tienen desde 8 a 151 imágenes por clase por cada dominio, y un total de 2533 imágenes.

2.4.4.4. Método experimental

Los experimentos se dividen en dos partes: para medir el desempeño en adaptación no supervisada, y en adaptación semi-supervisada.

Para la adaptación no supervisada se comparan los resultados con un método de referencia llamado **OrigFeat**, en el cual se utilizan las características originales del dominio para realizar entrenamiento, sin utilizar alguna adaptación. Para realizar la adaptación con el método propuesto, los sub-espacios posibles de los datos en el dominio fuente son: el sub-espacio **PCA**, y el sub-espacio generado por el método *Partial Least Squares* **PLS** (Véase Apéndice C). Para el sub-espacio del dominio objetivo se tiene solamente el sub-espacio **PCA**. Por lo que los posibles experimentos a realizar con el método **GFK** son: **GFK(PCA,PCA)** y **GFK(PLS,PCA)**. Además se implementa el método propuesto en [8] y mostrado en la sub-sección anterior, el cual es referido como **SGF**.

Para la adaptación semi-supervisada los algoritmos tienen acceso a una pequeña cantidad de datos etiquetados, por lo que además se comparan los resultados con una versión de **GFK** que utiliza el sub-espacio generado por **PLS** para el dominio objetivo (**GFK(PLS,PLS)**). Además se utiliza el método **Metric** propuesto en [14], el cual determina la correspondencia entre los datos etiquetados del dominio fuente y objetivo para crear un nuevo espacio de características.

2.4.4.5. Resultados

En la adaptación no supervisada se muestra que el método **GFK(PLS,PCA)** supera a la clasificación normal realizada por **OrigFeat** y al método **SGF**, en donde se atribuyen dos motivos principales a estos resultados. El primero es debido a que el *kernel* generado integra todos los sub-espacios a través de la curva, permitiendo escoger el mejor cambio de dominio entre la fuente y el objetivo. El segundo motivo es debido a que al utilizar el sub-espacio generado por **PLS**, se evita realizar proyecciones a direcciones que no entregan mucha información o que pueden contener ruido, informando

que esto puede suceder al utilizar **PCA**. También se obtuvo que para un dominio objetivo dado, hay un dominio fuente que entrega mejores resultados para la mayoría de los métodos probados en comparación a utilizar otros dominios fuente.

Para la adaptación semi-supervisada se tiene que el método **GFK(PLS,PCA)** entrega los mejores resultados, seguido de **GFK(PCA,PCA)**. Se menciona que a pesar de que se cuenta con una pequeña cantidad de datos etiquetados para generar el método **GFK(PLS,PLS)**, esta cantidad no es la suficiente como para generar una estimación confiable de los sub-espacios generados por **PLS**. También se encuentra que el método **Metric** tiene un bajo desempeño, atribuido al mismo motivo mencionado anteriormente.

2.4.5. Marginalized Denoising Autoencoders for Domain Adaptation

2.4.5.1. Resumen General

La aplicación de *Stacked Denoising Autoencoders* para adaptación entre dominios ha entregado resultados favorables en conocidos *datasets* de prueba con tareas de análisis de sentimientos. Sin embargo, existen dos grandes limitaciones al utilizar *SDAs*: Un alto costo computacional, y falta de escalabilidad ante una gran cantidad de dimensiones.

En [5] se propone el uso de *marginalized Stacked Denoising Autoencoders (mS-DA)*, los cuales marginalizan el ruido introducido en los *SDA*, causando que no se necesite utilizar el método de gradiente descendente estocástico o algún otro algoritmo de optimización para aprender parámetros, aumentando la rapidez con la que se puede entrenar un *SDA* y obtener los parámetros de aprendizaje.

La aplicación del método propuesto sigue la configuración experimental utilizada en [6], en donde los *datasets* y métricas son iguales para las comparaciones.

2.4.5.2. Propuesta

Al igual que en *SDA*, el algoritmo *mSDA* se basa en la concatenación de varios *denoisers*, los cuales siguen la idea de reconstruir una entrada a partir de una versión corrompida. Sin embargo, la forma en que un *denoiser* aprende a reconstruir una instancia se realiza de una manera distinta.

Cada *mDA* (*marginalized Denoising Autoencoder*) recibe valores corrompidos de entrada $x_1, \dots, x_n$, denotados como $\tilde{x}_1, \dots, \tilde{x}_n$, y a diferencia de un *Denoising Autoencoder*, realiza la reconstrucción de la entrada mediante una sola operación $W\tilde{x}_i$ ⁶. En donde W representa una matriz de pesos de reconstrucción, la cual es calculada de forma cerrada (ver apéndice D).

Luego de que W es calculada, los valores de *mDA* pasan por una función de no-linealidad que genera una representación h . De forma similar a *SDA*, se genera una pila de *mDA*'s, en donde cada uno de estos recibe como entrada la representación h del *mDA* de la capa anterior, considerando que el primer *mDA* recibe como entrada los valores x . De esta forma se tiene el algoritmo *mSDA*.

2.4.5.3. Dataset

Se utiliza el *dataset* de *Amazon* propuesto en [2] descrito en la sección 2.4.2.3.

2.4.5.4. Método Experimental

Se realiza la adaptación entre dominios utilizando *mSDA*, con los datos transformados del dominio fuente obtenidos se entrena una *SVM* y se prueba con los datos etiquetados del dominio objetivo. Los resultados se comparan con los métodos *SCL*[3], *SDA* [6] de una sola capa, y *CODA* [4], además se establece como *baseline* el entrenamiento de una *SVM* utilizando una representación de *bag-of-words* de los datos

⁶Cabe notar que, a diferencia de un *Denoising Autoencoder*, la función de no-linealidad no se utiliza para obtener representaciones ocultas, o para la reconstrucción de estas. Con el fin de mantener el cálculo de la matriz W en forma cerrada, la no-linealidad se realiza luego de calcular esta matriz.

etiquetados del dominio fuente para entrenamiento, y los datos del dominio objetivo como prueba. Por último, se realiza una adaptación utilizando el sub-espacio *PCA* obtenido del *dataset* completo (datos etiquetados y no etiquetados de los dominios fuente y objetivo), para luego transformar los datos y entrenarlos en una *SVM*. Los hiperparámetros de los modelos entrenados fueron escogidos mediante utilizando *5-fold cross validation*.

Las métricas usadas como punto de comparación son *transfer loss* y *transfer ratio*, utilizadas en [6] y presentadas en la sección 2.4.2.4. Bajos valores de *transfer loss* y *transfer ratio* implican una mejor adaptación entre dominios. Además de esto, se compara el tiempo que demora realizar la adaptación entre dominios utilizando *mSDA* y *SDA* (sin tomar en cuenta el tiempo que demora en entrenar una *SVM*).

2.4.5.5. Resultados

Los resultados de las pruebas ejecutadas muestran que los métodos *SDA* y *mSDA* superan a los métodos comparados, lo que demuestra que las representaciones que se generan utilizando estos métodos permiten disminuir la diferencia existente entre distintos dominios. Entre *mSDA* y *SDA* se tiene que el primer método obtiene menores valores de *transfer loss* para 10 de las 12 adaptaciones posibles.

Las comparaciones de tiempo con *SDA* de una muestran que *mSDA* demora considerablemente menos tiempo. *SDA* toma alrededor de 5 horas en entrenar, mientras que *mSDA* puede demorar alrededor de 25 segundos cuando se utiliza una sola capa, y menos de 2 minutos cuando se utilizan 5 capas. Por otra parte, se tiene que *SCL* y *PCA* son relativamente rápidos, pero el desempeño en la adaptación realizada no compensa esta ventaja.

2.5. Representación matemática de los datos

Dado que en las publicaciones revisadas anteriormente no se encuentra un formato común para la representación de los datos, a continuación se dan a conocer opciones disponibles para su utilización en este trabajo.

Los datos utilizados para las pruebas, inicialmente se encuentran en formatos que son entendibles por personas, ya sean estos archivos de texto, documentos web, o en el caso aplicado para esta memoria, archivos *csv*. Sin embargo, para que estos datos puedan ser manipulados eficientemente por los distintos programas es necesario convertirlos a una nueva representación.

Entre las distintas representaciones existentes se consideran dos aspectos: Primero, la información a utilizar de los documentos disponibles, y segundo, la estructura de datos en la cual la información obtenida es almacenada. Con esto, para el primer aspecto se consideran los modelos *bag-of-words* y *N-gramas*, mientras que para el segundo aspecto se utiliza el modelo de espacio vectorial. El modelo *bag-of-words* consiste en representar las palabras de un documento como una lista de palabras, sin considerar el orden en que estas aparecen en el documento, la información semántica de estas o la sintaxis al estar acompañada de otras palabras. Haciendo con esto que cada término sea independiente de otro en esta representación.

Un *N-grama* consiste en representar las palabras de un documento como una lista de secuencias contiguas de N palabras. En términos simples, el modelo *bag-of-words* puede considerarse como un caso de un *1-grama* (o unigrama). Así mismo, al aumentar la cantidad de palabras que se agrupan se obtiene una mayor cantidad de información espacial respecto al documento, la cual no se obtiene con el modelo *bag-of-words*.

Una vez obtenida la representación en *N-gramas* desde un documento, se procede a crear un vector de características para el documento, en donde cada dimensión del vector representa un término del *N-grama* y el valor que este contiene representa el peso de este. Los valores más simples que puede tomar son:

- Un valor binario: en donde un 1 o un 0 representa la existencia o no, respectivamente, de un término dentro de una secuencia de texto.
- Una frecuencia: en donde un valor entero representa la cantidad de veces que ocurre un término dentro de una secuencia de texto.

Luego, la totalidad de los textos disponibles puede ser representada mediante un conjunto de vectores de características, lo cual facilita la manipulación de los datos para el entrenamiento de los distintos clasificadores.

2.6. Medición de similitud entre dominios

Para tener una idea previa de cómo se comportarán los distintos algoritmos de adaptación al utilizar distintos dominios, se tiene una serie de métricas disponibles para medir la similitud existente entre los distintos dominios, o bien, la distancia entre las distintas distribuciones de probabilidad que cada dominio posee. Entre las medidas que se han utilizado en las distintas publicaciones se encuentran:

- Divergencia de Kullback-Leibler.
- Rank of Domain.
- Proxy A-Distance.

Divergencia de Kullback-Leibler

La divergencia de *Kullback-leibler* es una medida de la divergencia existente entre dos distribuciones de probabilidad. Considerando dos distribuciones p y q , la divergencia de *Kullback-leibler* se define como:

$$D_{KL}(p||q) = \sum_{i=1}^N p(x_i) \cdot \log \frac{p(x_i)}{q(x_i)} \quad (2.10)$$

Entonces, de la ecuación 2.10 se obtiene que a menores valores, menos divergen ambas distribuciones. Por otra parte, cabe notar que esta medida no es simétrica, es decir, el valor de $D_{KL}(p||q)$ no siempre es igual al valor que se obtiene de $D_{KL}(q||p)$, por lo cual, a pesar de ser utilizada en publicaciones para medir la diferencia entre dominios, no se puede considerar como una medida de distancia entre distribuciones de probabilidad.

Rank of Domain

La medida *Rank of Domain (ROD)* fue propuesta en [7] para medir la diferencia entre pares de dominios. Para realizar el cálculo de esta medida se utilizan dos tipos de información: La alineación entre los sub-espacios que se utilizan al aplicar el algoritmo *GFK*, y las divergencias de *Kullback-Leibler* existentes entre las distribuciones de los datos al ser proyectadas en estos sub-espacios.

Dados un par de dominios S y T , y considerando los ángulos principales θ_i obtenidos según lo explicado en la sección 2.4.4, el cálculo de la medida *ROD* se realiza, de manera general, de acuerdo a los siguientes pasos:

1. Determinar la dimensionalidad óptima d^* de los sub-espacios.
2. Por cada dimensión $i \leq d^*$, se apróxima una distribución *gaussiana* de una dimensión por cada uno de los dominios, definidos como S_i y T_i .
3. Calcular un par de divergencias de *Kullback-Leibler* y sumarlas de la forma $D_{KL}(S_i||T_i) + D_{KL}(T_i||S_i)$.
4. Calcular una versión promediada, y proyectada a los ángulos principales, de las divergencias anteriores según la ecuación 2.11.

$$\frac{1}{d^*} \sum_i^{d^*} \theta_i [D_{KL}(S_i||T_i) + D_{KL}(T_i||S_i)] \quad (2.11)$$

Proxy A-Distance

A-Distance es propuesta en [10] como una medida de la similitud entre dos distribuciones de probabilidad. Esta medida se basa en la hipótesis de que debería ser difícil distinguir entre un par de dominios para que sea posible una buena adaptación, consecuencia de que sus distribuciones sean similares. Sin embargo, en la práctica es casi imposible calcular exactamente la distancia entre ambas distribuciones, por lo que en [1] se sugiere una versión *proxy* de esta medida.

Para calcular la medida *Proxy A-distance* de un par de dominios se entrena un clasificador, cuya tarea es clasificar cada instancia como perteneciente a uno de los dos dominios. Luego, con el error de clasificación ϵ obtenido se calcula la medida *Proxy A-distance* de acuerdo a la ecuación 2.12.

$$\hat{d}_{\mathcal{A}} = 2(1 - 2\epsilon) \quad (2.12)$$

Capítulo 3

Implementación

En este capítulo se presentan los métodos a implementar, así como las especificaciones de software utilizadas para implementarlos en conjunto con los clasificadores. También se presenta la estructura implementada para almacenar y manipular los datasets a utilizar en las distintas pruebas.

Por último, se muestran los pasos seguidos para instalar y configurar el entorno de trabajo utilizado.

3.1. Métodos implementados

De las publicaciones estudiadas en la sección [2.4](#) se realiza una selección de un conjunto de estas, con el fin de reproducir los resultados obtenidos por sus autores y también con el objetivo de probar las técnicas en casos reales y aplicables. Los criterios de selección de las publicaciones son en base a:

- Disponibilidad de código abierto para su ejecución provisto por los autores.

- Relevancia de la publicación realizada. Estimada en base a los resultados expuestos, el impacto de la publicación (cantidad de veces citada por otras publicaciones) y novedad de la técnica propuesta.
- Facilidad de implementación de la propuesta en caso de no existir código disponible para su uso.

Dado lo anterior, se realizan las implementaciones de los métodos:

- GFK: Implementación en *Python* disponible en *Bob*. Un paquete de herramientas para procesamiento de señales y *machine learning* ¹.
- SDA: Implementación en *Python* utilizando la librería *Keras* ² para el uso de *Stacked Denoising Autoencoders*.
- mSDA: Implementación realizada en *Python*, utilizando la librería *Theano* ³.

Adicionalmente, a modo de comparación y siguiendo algunas las pruebas realizadas en las publicaciones revisadas se hace una implementación de:

- PCA: Implementación en *Python* en la cual se obtiene el sub-espacio *PCA* de las instancias de ambos dominios para ser utilizado como en el espacio que hace posible la adaptación entre dominios.

3.1.1. Estructura de los datasets

Los *datasets* a utilizar se encuentran en archivos con formato *.pkl* ⁴, los cuales almacenan un objeto de la clase ***Dataset***, que inicialmente posee la siguiente estructura:

- Dataset:

¹Sitio web: <https://pythonhosted.org/bob/>

²Sitio web: <https://keras.io/>

³Sitio web: <http://www.deeplearning.net/software/theano/>

⁴Formato de archivo que puede almacenar objetos del lenguaje *Python*.

- domains: Lista con los nombres de los dominios que contiene el dataset.
- labeled: Diccionario cuyas claves son los nombres de los dominios, en donde cada valor corresponde a otro diccionario con los datos etiquetados, de la forma:
 - X: Matriz numérica de tipo *numpy.ndarray* con cada fila representando una instancia del dataset, y cada columna corresponde a una característica.
 - y: Vector numérico de tipo *numpy.ndarray* con cada fila representando a la etiqueta de una instancia.
- unlabeled: Diccionario cuyas claves son los nombres de los dominios, en donde cada valor corresponde a otro diccionario con los datos no etiquetados, de la forma:
 - X: Matriz numérica de tipo *numpy.ndarray* con cada fila representando una instancia del dataset, y cada columna corresponde a una característica.

Esta estructura ha sido diseñada para mantener un orden común entre los distintos *datasets*, disminuyendo así la cantidad de código a realizar para cada prueba, y además para generalizar el acceso que tiene cada método implementado a los distintos *datasets*. Sin embargo, con el fin de establecer una cantidad fija de datos de entrenamiento y prueba, y para lograr una mayor consistencia entre las distintas pruebas, se tiene un método '*split_dataset*', que divide la estructura anterior de la siguiente forma:

- Dataset:
 - domains: Lista con los nombres de los dominios que contiene el dataset.
 - labeled: Diccionario cuyas claves son los nombres de los dominios, en donde cada valor corresponde a otro diccionario con los datos etiquetados, de la forma:
 - X_tr: Matriz numérica de tipo *numpy.ndarray* con cada fila representando una instancia del dataset, y cada columna corresponde a una característica.

- `y_tr`: Vector numérico de tipo *numpy.ndarray* con cada fila representando a la etiqueta de una instancia.
- `X_ts`: Matriz numérica de tipo *numpy.ndarray* con cada fila representando una instancia del dataset, y cada columna corresponde a una característica.
- `y_ts`: Vector numérico de tipo *numpy.ndarray* con cada fila representando a la etiqueta de una instancia.
- `unlabeled`: Diccionario cuyas claves son los nombres de los dominios, en donde cada valor corresponde a otro diccionario con los datos no etiquetados, de la forma::
 - `X_tr`: Matriz numérica de tipo *numpy.ndarray* con cada fila representando una instancia del dataset, y cada columna corresponde a una característica.
 - `X_ts`: Matriz numérica de tipo *numpy.ndarray* con cada fila representando una instancia del dataset, y cada columna corresponde a una característica.

En la estructura anterior, los sufijos ‘*_tr*’ y ‘*_ts*’ indican que se trata de datos de entrenamiento y prueba, respectivamente. En el caso de no existir datos no etiquetados en alguno de los *datasets*, las claves *unlabeled* tendrán un valor igual a *None*.

3.2. Configuración del entorno de trabajo

Con el fin de que las pruebas a realizar sean replicables y que además la ejecución de estas no sea difícil de hacer o que requiera configuraciones y *hardware* específicos, se utiliza un conjunto de herramientas que dejan establecer un entorno de trabajo capaz de cumplir lo mencionado anteriormente. Estas herramientas se dividen en *Amazon Elastic Compute Cloud (Amazon EC2)*, que proveen servicios de virtualización en la nube, y en el entorno de *Python*, que permite la ejecución de las pruebas.

3.2.1. Amazon Elastic Compute Cloud

Amazon EC2 es un servicio web de pago que presta servicios de cómputo a través de la virtualización de servidores (máquinas virtuales) en la nube. Utilizando una interfaz, estos servidores tienen la capacidad de modificar sus características con poca dificultad, tales como la capacidad de almacenamiento y cómputo.

Amazon EC2 permite crear imágenes de las máquinas virtuales configuradas, llamadas instancias, permitiendo así crear, iniciar y eliminar la cantidad de instancias que se consideren necesarias, haciendo el cobro monetario de estas instancias de acuerdo a sus características y uso en horas. También se proporciona una cantidad de distintos tipos de instancias con fines diferentes, tales como su uso para almacenamiento, tareas con alta demanda de cálculos o aprendizaje automático.

Entre los tipos de instancias que más aptas se encuentran para su uso en las pruebas a realizar se encuentran las del tipo **C4** y **P2**. Las primeras cuentan con optimización de los procesadores que contienen, ofreciendo una buena relación entre precio y desempeño obtenido. Las segundas utilizan *GPU* para el procesamiento principal, que acelera el tiempo de ejecución de aplicaciones realizadas para este tipo de procesamiento, sin embargo su precio es considerablemente mayor al de las instancias del primer tipo.

Debido a que no todos los algoritmos a implementar están diseñados para ser utilizados con *GPU* y no pueden obtener beneficios de esto, se opta por utilizar una instancia del tipo *C4*, en específico la instancia *c4.xlarge*⁵, que viene con el sistema operativo *Ubuntu 14.04 LTS* e incluye 4 *CPU* virtuales.

La instancia utilizada es modificada para tener más almacenamiento, pasando a tener de 8 a 32 *GB*, y así poder guardar los modelos creados, además también se realizan cambios para poder aceptar conexiones entrantes y así controlar la instancia y ejecutar programas de forma remota.

Una vez lista la instancia, es posible configurar el entorno de *Python* para realizar

⁵Sitio web: <https://aws.amazon.com/es/ec2/instance-types/>

las pruebas.

3.2.2. Entorno de Python

Anaconda y Jupyter Notebooks

La implementación de los algoritmos y la ejecución de pruebas se hace utilizando *Anaconda* y *Jupyter Notebooks* ⁶. *Anaconda* es una distribución de *Python* utilizada para el procesamiento de datos, aprendizaje de máquinas y computación científica, entre otras tareas. Su uso simplifica en gran manera estas tareas al venir con una cantidad de librerías instaladas y proveer herramientas para la instalación de nuevas librerías y el manejo de varios entornos virtuales.

El uso de entornos virtuales permite tener un conjunto de librerías que no interactúan con las previamente instaladas en el sistema, creando así un espacio separado de trabajo que no afecta al resto del sistema y que puede ser compartido y replicado en otras máquinas.

En adición a *Anaconda*, también se utiliza *Jupyter Notebooks*, el cual es un ambiente interactivo para ejecutar secciones de código en *Python* y visualizar variables e imágenes. La posibilidad de ejecutar secciones de código, en contraste a ejecutar un script de *Python*, permite un desarrollo más rápido, permitiendo ejecutar y probar solamente las secciones necesarias. La interfaz web en la que se ejecutan los códigos también presenta ventajas, al permitir describir operaciones e ideas en distintos formatos de texto.

scikit-learn, Theano y Keras

Para implementar los algoritmos y realizar las pruebas en *Python* se utilizan principalmente tres librerías: *scikit-learn* ⁷, *Theano* y *Keras*. La primera de estas contiene

⁶Sitio web: <https://ipython.org/notebook.html>

⁷Sitio web: <http://scikit-learn.org/>

las funciones y clases necesarias para realizar la clasificación de sentimientos, la división de los *datasets* en conjuntos de entrenamiento y prueba y los métodos para realizar *cross-validation* y *grid search*.

La librería *Theano* provee una interfaz para realizar operaciones matemáticas y principalmente implementar algoritmos de *deep learning* de manera optimizada. Las expresiones matemáticas definidas en *Theano* pueden ser ejecutadas tanto en *CPU* como en *GPU*, si es que el *hardware* disponible lo permite.

Keras es una *API* que provee una manera rápida de implementar estructuras que son comunes en *deep learning*, permitiendo experimentar y crear prototipos de forma más acelerada. Es capaz de realizar operaciones utilizando *Theano* como base, por lo cual estas son optimizadas y capaces de ser ejecutadas tanto en *CPU* como *GPU*. Sin embargo, a pesar de la facilidad con la que se pueden implementar algoritmos, el nivel de control que se tiene sobre los parámetros a ajustar o características específicas de algún algoritmo es menor en comparación a realizar una implementación en *Theano*.

Instalación del entorno de Python

Para instalar y configurar el entorno de *Python*, se tiene a disposición un repositorio público con los códigos de las implementaciones y librerías a utilizar. Primero es necesario descargar e instalar la distribución de *Anaconda* correspondiente al sistema en donde se ejecutarán los códigos, y luego se debe clonar el repositorio a una ruta local:

```
1 $ git clone https://github.com/franciscoprimero/Memoria.git
```

Una vez clonado el repositorio, se procede a instalar el entorno virtual y las librerías necesarias:

```
1 $ conda env create -f env_memoria.yml
2 $ python -m nltk.downloader stopwords
```

Luego, para iniciar el entorno virtual y ejecutar las pruebas:

```
1  $ source activate env_memoria
2  (env_memoria) $ jupyter notebook
```

Con eso se abrirá una ventana del navegador predeterminado del sistema y será posible acceder a los archivos con las pruebas.

Capítulo 4

Método Experimental

Para obtener resultados comparables tanto entre las distintas técnicas implementadas como con los valores expuestos en sus correspondientes publicaciones, se hace el uso de *datasets* que han sido utilizados en un gran número de publicaciones relacionadas a *transfer learning* y *domain adaptation*, incluyendo a las revisadas en este documento. Además se tiene disponible un dataset creado a partir de *tweets* en español con el fin de probar las técnicas implementadas en un escenario práctico y real.

Se establecen marcos de trabajo, basados en las publicaciones estudiadas, que permitan replicar lo más posible a sus versiones originales, y del mismo modo que también aseguren realizar las pruebas correctas al utilizar *datasets* nuevos con las distintas técnicas.

Para poder comparar los resultados que entrega cada técnica también se hace uso de métricas estándares para evaluar el desempeño de los clasificadores en los *datasets* y también se utilizan métricas específicas para medir el desempeño de la adaptación entre dominios.

En la siguiente sección se realiza una descripción de los *datasets* utilizados, para luego presentar y explicar las métricas a utilizar con los distintos algoritmos. Finalizando con la presentación de los marcos de trabajo a utilizar en las pruebas de adaptación entre dominios.

4.1. Datasets

Los *datasets* a utilizar son: *Amazon* y *Twitter*. El primero es escogido debido al uso notorio que ha recibido en tareas de clasificación de sentimientos y adaptación entre dominios, mientras que el último ha sido escogido para representar un *dataset* cercano a lo aplicado comúnmente en la industria local.

En ambos *datasets* se considera que los dominios a utilizar se encuentran disponibles desde un inicio, estableciendo el espacio de características como el espacio que se obtiene al procesar todos los dominios. De esta forma se respeta lo establecido al final de la sección 2.3, en donde se considera que el espacio de características de un *dataset* corresponde a la unión de los espacios de todos los dominios que lo conforman. Del mismo modo, al utilizar datos pertenecientes a un dominio nuevo con un dataset ya creado, solo se tomarán en cuenta las características ya existentes, sin modificar el espacio de características.

4.1.1. Amazon

Este *dataset* consiste en reseñas a distintos tipos de productos vendidos en el sitio *Amazon*. Los productos se separan en distintas categorías, representando cada una de estas categorías un dominio distinto. Cada reseña en el sitio representa una instancia de clasificación, y cada una de estas es puntuada con una cantidad de estrellas que van desde 1 a 5, en donde una cantidad menor o igual a 3 estrellas pasa a ser una etiqueta de valor negativo, y una cantidad mayor a 3 pasar a ser una etiqueta positiva, siendo esta forma la establecida en [2] y utilizada en [2], [6] y [5] entre otros. Por otra parte, cada comentario es representado como un vector binario de unigramas y bigramas, y por razones computacionales se dejan disponibles solo los 3000 términos más frecuentes.

Los dominios considerados para este trabajo y las cantidades de datos etiquetados y no etiquetados, así como el porcentaje de datos etiquetados positivos, son los siguientes:

Tabla 4.1: Cantidad de instancias en el *dataset* de *Amazon*

Dominio	Entrenamiento	% Positivos	Prueba	% Positivos	No Etiquetados
Books	1600	50 %	400	50 %	4465
DVDs	1600	50 %	400	50 %	5945
Electronics	1600	50 %	400	50 %	5681
Kitchen	1600	50 %	400	50 %	3586

4.1.2. Twitter

El dataset de *Twitter* se compone de *tweets* en español etiquetados como positivos o negativos. Este *dataset* está conformado por 3 dominios, de los cuales 2 han sido etiquetados manualmente, son específicos a un tema y contienen un número reducido de datos, mientras que el tercero corresponde a un subconjunto de *tweets* obtenidos de un repositorio público¹, de temática general y con mayor cantidad de datos. La distribución de las instancias en el dataset creado corresponde a la mostrada en la Tabla 4.2.

Por cada *tweet* en el dataset se realiza un procesamiento, en el cual se le remueven los enlaces *web*, *hashtags* y menciones a otro usuarios. Además, los *retweets* son eliminados, debido a la posibilidad de tener instancias duplicadas en el *dataset*. Luego de estas operaciones, cada *tweet* es representado como un vector binario de unigramas y bigramas, y se dejan disponibles solo los 2000 términos más frecuentes.

¹Sitio Web: <https://github.com/garnachod/TwitterSentimentDataset>

Tabla 4.2: Cantidad de instancias en el *dataset Twitter*

Dominio	Entrenamiento	% Positivos	Prueba	% Positivos	Total
thevoice	519	51 %	130	49 %	649
rio2016	380	53 %	96	60 %	476
general	3736	46 %	935	47 %	4671

4.2. Métricas

En las pruebas se consideran dos métricas principales para realizar la comparación entre los distintos métodos: el tiempo y la métrica de *transfer-loss* [6].

- **Tiempo:** Esta métrica considera solamente el tiempo que requieren los distintos métodos para generar las representaciones que permiten realizar la adaptación entre dominios, sin considerar el tiempo que requiere entrenar los clasificadores y probarlos, por ende, solamente las pruebas que requieran hacer alguna adaptación utilizan esta métrica.
- ***Transfer-loss:*** Corresponde al error que se esperaría reducir para obtener una clasificación cercana a la ideal, y depende de la obtención de los valores de *transfer error* y *baseline in-domain error*, los cuales varían dependiendo de cada prueba. Para una mayor explicación de esta métrica véase el Anexo [E](#).

4.3. Pruebas

4.3.1. Pruebas de adaptación

Las publicaciones en donde se muestran los algoritmos escogidos a implementar presentan distintos marcos experimentales para realizar sus pruebas. Dadas las tareas principales para las cuales estos algoritmos fueron diseñados se obtienen diferencias

en los datos que se utilizan para entrenar cada uno de estos algoritmos. En consecuencia, los criterios para escoger un modelo de adaptación por sobre otro van a variar de acuerdo al algoritmo. Sin embargo, no se puede asumir si el marco experimental con el que fue probado un algoritmo va a presentar un beneficio o un perjuicio en otro algoritmo, por lo que se decide realizar las pruebas de los algoritmos en los distintos marcos experimentales encontrados.

En general, los pasos a seguir en las pruebas a realizar siguen una secuencia definida brevemente por los siguientes pasos:

- Obtener los datos del *dataset* correspondiente.
- Obtener un conjunto de modelos adecuados para realizar la adaptación de acuerdo a los dominios disponibles.
- Realizar pruebas de clasificación utilizando los modelos de adaptación obtenidos anteriormente.

En las publicaciones que conciernen a los algoritmos implementados se distinguen dos marcos experimentales que se diferencian mayormente en los datos con los que se entrena cada algoritmo y en el criterio por el cual se escogen los modelos entrenados para realizar la adaptación.

Dada la cantidad de valores que pueden tomar los distintos parámetros de cada algoritmo, se establece por cada uno de estos un conjunto de valores posibles a tomar, basando estos valores en los utilizados por sus autores en sus respectivas publicaciones.

Para todos los tipos de pruebas, el clasificador a utilizar es una *SVM* lineal, escogida debido a los buenos resultados que se han obtenido utilizándola en las publicaciones revisadas y la simpleza de utilizarla. Los parámetros de la *SVM* lineal son ajustados realizando *cross validation* y *grid search* con los datos de entrenamiento correspondientes.

Marco Experimental 1

El marco experimental 1 se basa en las pruebas realizadas en [6] y [5], los cuales implementan los métodos *SDA* y *mSDA* respectivamente. En este marco experimental los datos utilizados para entrenar los algoritmos de adaptación son todos los datos de entrenamiento disponibles del *dataset*, es decir, los datos etiquetados y no etiquetados de cada dominio disponible.

Al estar todos los dominios y sus datos involucrados en la adaptación no es necesario crear un conjunto de modelos por cada de dominios a adaptar, sino que solo se necesita un único conjunto de modelos con el que luego se puede obtener cuál realiza una mejor adaptación dependiendo del dominio a adaptar. La secuencia de pasos a seguir en este marco experimental, para obtener el mejor modelo de adaptación por cada dominio fuente, es:

- Obtener todos los datos de entrenamiento del *dataset* correspondiente.
- Entrenar un conjunto de modelos de adaptación utilizando todos los datos disponibles, variando los parámetros de cada algoritmo de adaptación².
- Para el dominio fuente al cual se le busca el mejor modelo, se realiza validación cruzada en un clasificador con los datos de entrenamiento adaptados del dominio correspondiente. Realizando esto con cada uno de los modelos de adaptación entrenados.
- El modelo que obtiene el mejor resultado de validación cruzada va a ser entonces el mejor modelo para el dominio fuente utilizado.

La forma de obtener el modelo que mejor puede hacer una adaptación por cada dominio se obtiene solamente utilizando los datos de un dominio fuente, sin considerar un dominio objetivo con el cuál se pueda realizar la adaptación. Esta medida es realizada debido al supuesto de la ausencia de datos etiquetados de un dominio objetivo que

²Para el caso del algoritmo *GFK*, que dado su planteamiento requiere de una distinción entre el dominio fuente y objetivo. Se establece un solo dominio como objetivo, y la unión de los dominios restantes como el dominio fuente.

habría en un caso real, por ende, se considera que el modelo que obtenga los mejores valores de clasificación con validación cruzada, utilizando solamente los datos transformados del dominio fuente y sus etiquetas, va a ser en consecuencia el modelo que mejor realice una adaptación cuando existan datos etiquetados del dominio objetivo.

Luego, habiendo obtenido los modelos de adaptación necesarios, los pasos para realizar la clasificación con adaptación entre dominios son:

- Por cada par de dominios distintos, uno se considera como fuente y el otro como objetivo.
- Se obtiene el mejor modelo de adaptación correspondiente al dominio fuente (obtenido de acuerdo a la secuencia de pasos detallada anteriormente).
- Se adaptan los datos de entrenamiento del dominio fuente y se entrena un clasificador con estos datos.
- Se adaptan los datos de prueba del dominio objetivo utilizando el modelo obtenido anteriormente.
- Con los datos adaptados del dominio objetivo se prueba el clasificador y se obtienen las métricas necesarias.

Marco Experimental 2

El marco experimental 2 se basa en las pruebas que se realizan en [7], en donde se implementa el método *GFK*. Aquí se utilizan exclusivamente los datos etiquetados del dominio fuente y objetivo involucrados en cada adaptación.

Al utilizar solamente los datos de los dominios concernientes a cada adaptación entonces se requiere un modelo entrenado por cada par posible dentro de un conjunto de dominios. Así mismo, para obtener cada modelo entrenado, es necesario crear un conjunto de modelos posibles por cada par de dominios. La secuencia de pasos a seguir para obtener los modelos de adaptación es:

- Por cada par de dominios distintos uno se considera como fuente y el otro como objetivo.
- Se obtienen los datos de entrenamiento de cada uno de estos dominios.
- Se genera un conjunto de modelos de adaptación con los parámetros posibles que permita cada algoritmo, utilizando solamente los datos etiquetados de entrenamiento de ambos dominios.
- Para obtener el mejor modelo del conjunto se utiliza validación cruzada en un clasificador con los datos de entrenamiento del dominio fuente³.

De la misma forma que en el marco experimental anterior se considera la ausencia de datos etiquetados en el dominio objetivo, por lo que el criterio para obtener el mejor modelo por cada par de dominios va a ser nuevamente la mejora que realiza un modelo dentro del propio dominio fuente.

Luego, habiendo obtenido los modelos de adaptación necesarios, los pasos para realizar la clasificación con adaptación entre dominios son:

- Por cada par de dominios distintos, uno se considera como fuente y el otro como objetivo.
- Se obtiene el mejor modelo de adaptación para el par correspondiente.
- Se adaptan los datos de entrenamiento del dominio fuente y los datos de prueba del dominio objetivo.
- Se entrena un clasificador con los datos adaptados del dominio fuente.
- Se prueba el clasificador con los datos adaptados del dominio objetivo y se obtienen las métricas necesarias.

³A pesar de tener disponibles los datos de entrenamiento del dominio objetivo, estos no se utilizan, ya que en el caso de estudio se considera que estos datos no están etiquetados, por lo que no se pueden utilizar para realizar validación cruzada.

Cabe notar que en ambos marcos experimentales, para obtener los valores de *transfer error* y *transfer loss* se utiliza la totalidad de los datos de prueba correspondientes al dominio objetivo.

Parámetros de los algoritmos

Los parámetros disponibles para cada algoritmo se mantienen fijos en los dos marcos experimentales y ambos *datasets*, estos son:

■ PCA

- Dimensiones a transformar entre $1/2$, $1/3$ y $1/4$ de la dimensionalidad original de los datos.

■ SDA

- 1 capa de tamaño $1/2$ de la dimensionalidad original, o 2 capas con tamaño $1/2$ y $1/4$ de la dimensionalidad original cada una.
- 50 o 25 *epochs*.
- Probabilidad de corrupción de las entradas entre 0,3, 0,5, 0,8.

■ mSDA

- Cantidad de capas entre 1, 3 o 5.
- Probabilidad de corrupción de las entradas entre 0,3, 0,5, 0,8.

■ GFK

- Cantidad de sub-espacios entre 10, 20, 50.
- Dimensionalidad de los sub-espacios entre $1/2$ y $1/4$ de la dimensionalidad original de los datos.

4.3.2. Pruebas de tiempo

Para estas pruebas se mide el tiempo que demora cada algoritmo en realizar la adaptación entre dominios. Por cada algoritmo se establece un conjunto de parámetros con los cuales es ejecutado, y por cada posible combinación de parámetros se realiza una cantidad de 5 ejecuciones, siendo el tiempo reportado entonces el promedio de estas 5 ejecuciones. Cabe recordar que todas las pruebas son realizadas utilizando el lenguaje *Python* en la máquina virtual descrita en la sección 3.2.

Las pruebas utilizan el dataset *Amazon* con 3000 dimensiones, debido a que este posee una mayor cantidad de datos para notar diferencias sustanciales entre los tiempos de ejecución. Las descripciones de las pruebas de cada algoritmo son:

- PCA: Ejecuciones cambiando la dimensionalidad a 1500 y 750.
- SDA: Ejecuciones variando la cantidad de capas entre 1, 3 y 5, con cada capa de tamaño 1500 y una cantidad de *epochs* entre 25 y 50.
- mSDA: Ejecuciones variando la cantidad de capas entre 1, 3 y 5.
- GFK: Ejecuciones con cantidad de sub-espacios entre 10, 20 y 50, ajustando la dimensionalidad de cada sub-espacio permanece constante con un valor de 1500.

Para las pruebas *SDA* y *mSDA* se deja una probabilidad de corrupción un valor fijo de 0,3, dado que no se considera que un cambio en este valor altere significativamente el tiempo de ejecución.

Capítulo 5

Análisis y Resultados

Para el análisis de los resultados se realiza una comparación entre los distintos marcos experimentales, dimensiones utilizadas y además entre los algoritmos por cada *dataset* utilizado. Además, se analiza el tiempo empleado por cada método en realizar la tarea de adaptación en el *dataset* de *Amazon*, realizando también una comparación entre los distintos métodos.

El análisis a realizar busca explicar las causas de los desempeños obtenidos, las diferencias con respecto a los valores de referencias, las limitantes y las ventajas y desventajas que se observan en cada método. En general, se busca analizar lo siguiente:

- La correcta implementación de los algoritmos estudiados.
- El comportamiento de los algoritmos en comparación al comportamiento mostrado en sus respectivas publicaciones.
- El comportamiento de los algoritmos al variar la dimensionalidad de los *datasets* en ambos marcos experimentales.
- El comportamiento de los algoritmos al variar los marcos experimentales.
- El comportamiento de los algoritmos al utilizar *datasets* distintos.

- La influencia que tiene sobre los resultados el uso de datos de más de dos dominios para entrenar un modelo.

5.1. Pruebas de Adaptación

Inicialmente, por cada *dataset* se muestran los resultados de *in-domain error* y la similitud que presenta cada par de dominios mediante la medida *proxy a-distance*. Eso se realiza para tener un idea previa de cómo debieran variar los resultados de *transfer loss* al disminuir las dimensiones y utilizar los distintos dominios para cada tarea de adaptación.

Las pruebas de adaptación discuten los resultados obtenidos según la métrica de *transfer loss* definida anteriormente, estos resultados se encuentran divididos de acuerdo al *dataset* y marco experimental correspondiente. Los análisis se realizan en base a los gráficos y tablas presentados, en donde por cada par de dominios adaptados se ven los resultados de cada algoritmo.

En adición a los métodos implementados, también se muestran los valores de *transfer loss* de un clasificador que no realiza adaptación alguna y que realiza un cambio entre dominios al momento de entrenar y ser probado. Este método es nombrado como *Baseline*¹, y representa el caso que se busca resolver mediante la adaptación entre dominios.

Cabe recordar que menores valores de *transfer loss* significan mejores resultados, y además que los puntos a tener en cuenta son: el valor que muestra el método *Baseline* y si un método alcanza valores menores o iguales a 0. Lo primero es debido a que el valor obtenido por el método *Baseline* representa la pérdida de exactitud que se obtiene entrenando normalmente un clasificador, y lo segundo es debido a que un valor igual a 0 significa que se alcanza la exactitud al clasificar que se obtendría en un caso ideal, mientras que un valor menor a 0 significa que se obtienen mejores resultados que los

¹Para evitar confusiones entre el método *Baseline* y el término de *baseline in-domain error*, se refiere al primero de forma capitalizada, mientras que el segundo siempre se encuentra en minúsculas.

que obtendría un clasificador en un caso ideal.

Los gráficos presentados muestran en el eje x nombres de la forma " $f \rightarrow o$ ", en donde " f " y " o " corresponden a las iniciales de los dominios fuente y objetivo respectivamente. Por ejemplo, el valor " $b \rightarrow d$ " significa que la adaptación tiene a "*books*" y "*dvd*" como dominios fuente y objetivo respectivamente.

5.1.1. Amazon

In-domain Error

En la Tabla 5.1 se obtienen los valores de *baseline in-domain error* para ambas dimensionalidades. Cabe recordar que este error corresponde al error de clasificación que se obtiene entrenando y probando un clasificador con los datos de un mismo dominio, y los resultados obtenidos muestran que todos los casos, al pasar de 1000 a 3000 dimensiones, disminuyen su error.

En la figura 5.1 se observa gráficamente la diferencia entre los valores del *baseline in-domain error*, notando que en algunos casos los valores disminuyen casi a la mitad cuando se pasa de 1000 a 3000 dimensiones.

Uno de los factores atribuibles al aumento de los errores al disminuir las dimensiones es el hecho de que a una mayor cantidad de dimensiones, los clasificadores tienen una mayor cantidad de información disponible que les permite poder distinguir entre una y otra etiqueta. Considerando que la cantidad de instancias disponibles para el entrenamiento de los modelos es lo suficientemente grande.

Similitud entre dominios

Los valores de la Tabla 5.2 muestran los valores de la métrica *Proxy A-distance (PAD)* entre los distintos pares de dominios utilizando 3000 dimensiones. Por cada

Tabla 5.1: Baseline In-domain error de los dominios del *dataset Amazon*.

Dominio	1000 Dimensiones	3000 Dimensiones
dvd	16.900	15.762
kitchen	16.215	8.587
books	23.710	15.157
electronics	16.397	11.127

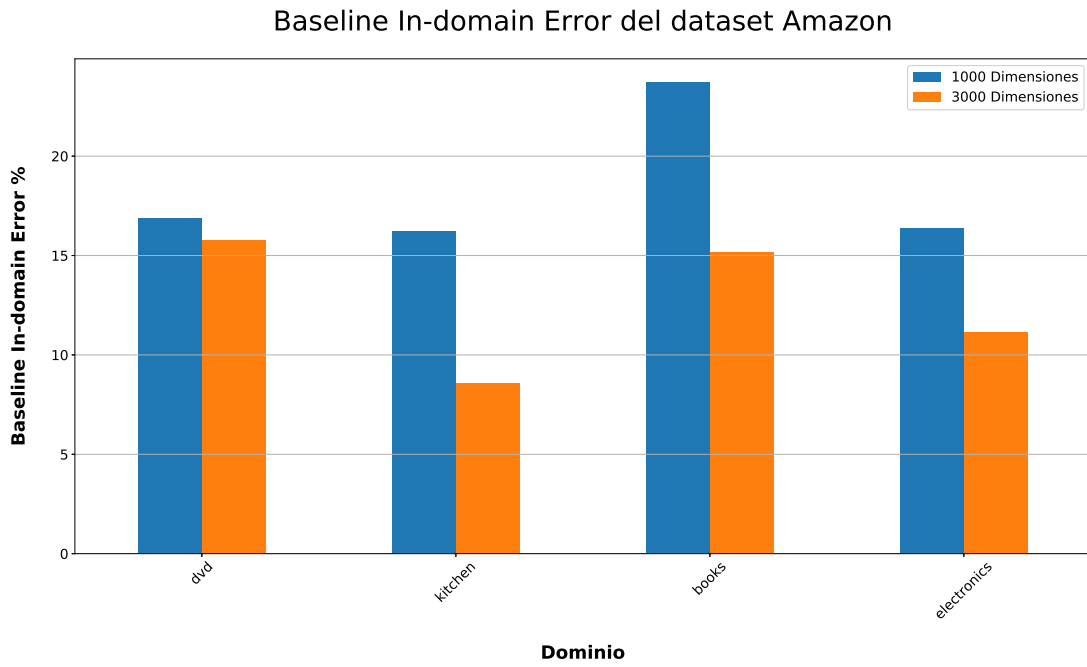


Figura 5.1: Baseline In-domain error para el dataset Amazon en ambas dimensiones.

dominio existente, además se incluye la combinación de los dominios restantes, permitiendo también ver la distancia que hay entre un dominio y la distribución de los dominios restantes combinados. Cabe recordar que menores valores implican una mayor similitud entre los dominios, lo que podría llevar a mejores adaptaciones al utilizar los dominios con mayor similitud.

En la Tabla 5.2 se observa que algunos dominios tienen mayor similitud con un dominio específico, mientras que otros tienen mayor similitud con la union de todos los dominios. Por ejemplo, los dominios *electronics* y *kitchen* tienen mayor similitud entre sí. Por otra parte, *books* y *dvd* tienen mayor similitud con la union de todos dominios, y en segundo lugar tienen mayor similitud entre ellos.

En la Tabla 5.3 se observa que con 1000 dimensiones los valores obtenidos, en general, son menores que los obtenidos con 3000 dimensiones. Al utilizar una menor cantidad de dimensiones, solamente los términos más frecuentes quedan disponibles para su uso, y al disminuir la diversidad de estos términos, es razonable que la similitud existente entre los distintos dominios sea mayor, disminuyendo así los valores obtenidos. Por otra parte, los dominios *dvd*, *electronics* y *kitchen* siguen teniendo mayor similitud con los mismos dominios obtenidos al utilizar 1000 dimensiones. Sin embargo, el dominio *books* pasa de tener mayor similitud con el dominio *dvd* a la union de todos los demás dominios.

La Tabla 5.4 muestra la comparación con ambas dimensionalidades, solamente para los dominios con menores valores, en donde no solamente se observa la casi total repetición de los dominios con mayor similitud (como fue mencionado anteriormente), sino que también se puede observar mejor la diferencia entre los valores que se obtienen con ambas dimensionalidades.

Los valores obtenidos sirven para tener idea de lo que se puede esperar al momento de realizar las adaptaciones. Por ejemplo, se puede esperar que los dominios *electronics* y *kitchen* den mejores resultados al utilizarse como dominios fuente y objetivo en las adaptaciones. Así mismo, el hecho de que aparezca un solo dominio o el conjunto de estos, da una muestra de cuál de los dos marcos experimentales podría

Tabla 5.2: *Proxy A-distance* de los dominios del *dataset Amazon* con 3000 dimensiones

Dominio 1	Dominio 2	PAD
books	(kitchen, dvd, electronics)	1.947
	dvd	1.966
	electronics	1.983
	kitchen	1.990
dvd	(kitchen, electronics, books)	1.934
	books	1.966
	kitchen	1.972
	electronics	1.973
electronics	kitchen	1.644
	(kitchen, dvd, books)	1.812
	dvd	1.973
	books	1.983
kitchen	electronics	1.644
	(dvd, electronics, books)	1.808
	dvd	1.972
	books	1.990

Tabla 5.3: *Proxy A-distance* de los dominios del *dataset Amazon* con 1000 dimensiones

Dominio 1	Dominio 2	PAD
books	dvd	1.830
	(kitchen, dvd, electronics)	1.847
	kitchen	1.960
	electronics	1.967
dvd	(kitchen, electronics, books)	1.755
	books	1.830
	electronics	1.868
	kitchen	1.887
electronics	kitchen	1.420
	(kitchen, dvd, books)	1.707
	dvd	1.868
	books	1.967
kitchen	electronics	1.420
	(dvd, electronics, books)	1.639
	dvd	1.887
	books	1.960

Tabla 5.4: Mejores valores de *Proxy A-distance* para cada dominio del *dataset Amazon* en ambas dimensiones.

Dominio 1	Dimensiones	Dominio 2	PAD
books	1000	dvd	1.830
	3000	(kitchen, dvd, electronics)	1.947
dvd	1000	(kitchen, electronics, books)	1.755
	3000	(kitchen, electronics, books)	1.934
electronics	1000	kitchen	1.420
	3000	kitchen	1.644
kitchen	1000	electronics	1.420
	3000	electronics	1.644

tener mejor desempeño, en donde el marco experimental 1 usa todos los dominios disponibles para la adaptación, mientras que el marco experimental 2 usa solamente un par por adaptación.

Por último, cabe recordar que no necesariamente los resultados a obtener en las adaptaciones van a coincidir con lo que se espera, sin embargo, el hecho de que existan coincidencias en los resultados servirá para establecer la medida *PAD* como una métrica confiable para futuras pruebas.

Marco Experimental 1

Al realizar las pruebas con 3000 dimensiones lo primero a comprobar en los resultados de la figura 5.2 es que efectivamente existen casos en donde los algoritmos mejoran la clasificación. Es decir, existen casos en donde el valor que obtiene cada algoritmo es menor a lo que muestra el método *Baseline*, lo cual se comprueba que sucede. Sin embargo, no ocurren mejoras en todos los casos, ya que en las pruebas " $b \rightarrow d$ " y " $b \rightarrow e$ " ningún algoritmo pudo obtener mejor desempeño que el método *Baseline*, siendo *GFK* y *mSDA* los algoritmos que más se aproximan en cada caso, respectivamente. Por otra parte, se puede observar que en los casos " $e \rightarrow k$ " y " $k \rightarrow e$ " se cumple con el método *mSDA* una de las predicciones realizadas en la Tabla 5.2.

En la Figura 5.3 se observan los efectos al utilizar 1000 dimensiones, en donde uno de los efectos más notorios es la existencia de resultados menores a 0, lo que implica que las adaptaciones realizadas han superado incluso a las de los casos ideales. Lo segundo a notar es la disminución total de casos en donde la adaptación no mejora los resultados del método *Baseline*. También, en este caso se puede notar mejor la diferencia de los valores de *transfer loss* que obtienen algunas tareas al utilizar un dominio en específico. Por ejemplo: se tiene que los dominios *books* y *dvd* realizan mejores adaptaciones entre sí, y lo mismo ocurre con los dominios *electronics* y *kitchen*, lo cual viene a apoyar los valores que se presentan en la Tabla 5.3 y más importante, a la medida *PAD*.

Lo siguiente a analizar es la cantidad de veces que mejora los resultados cada

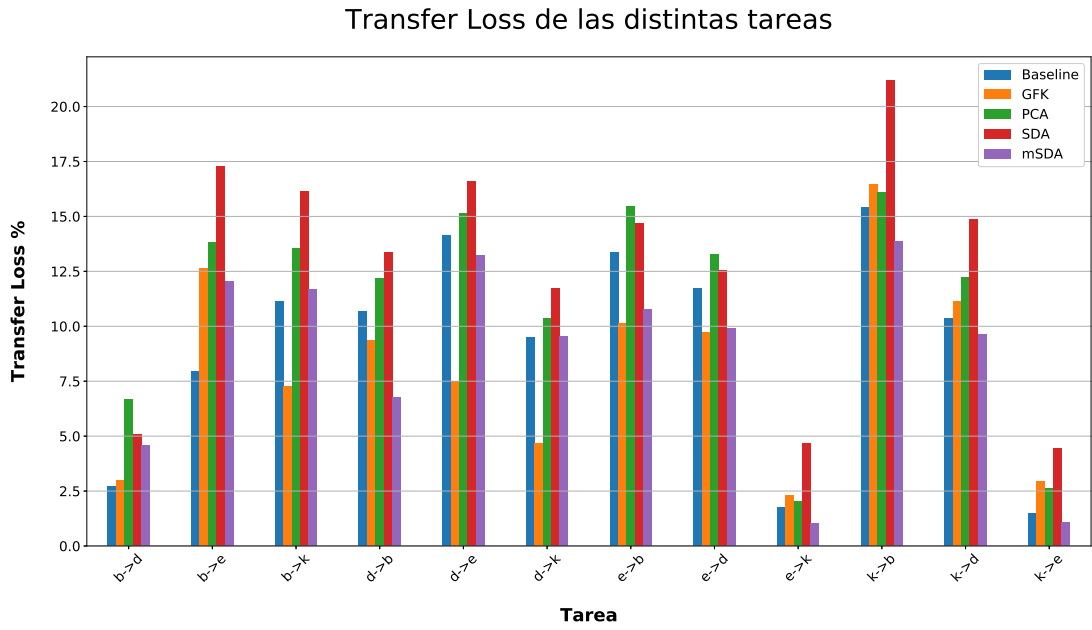


Figura 5.2: *Transfer loss* para el dataset Amazon utilizando 3000 dimensiones en el marco experimental 1.

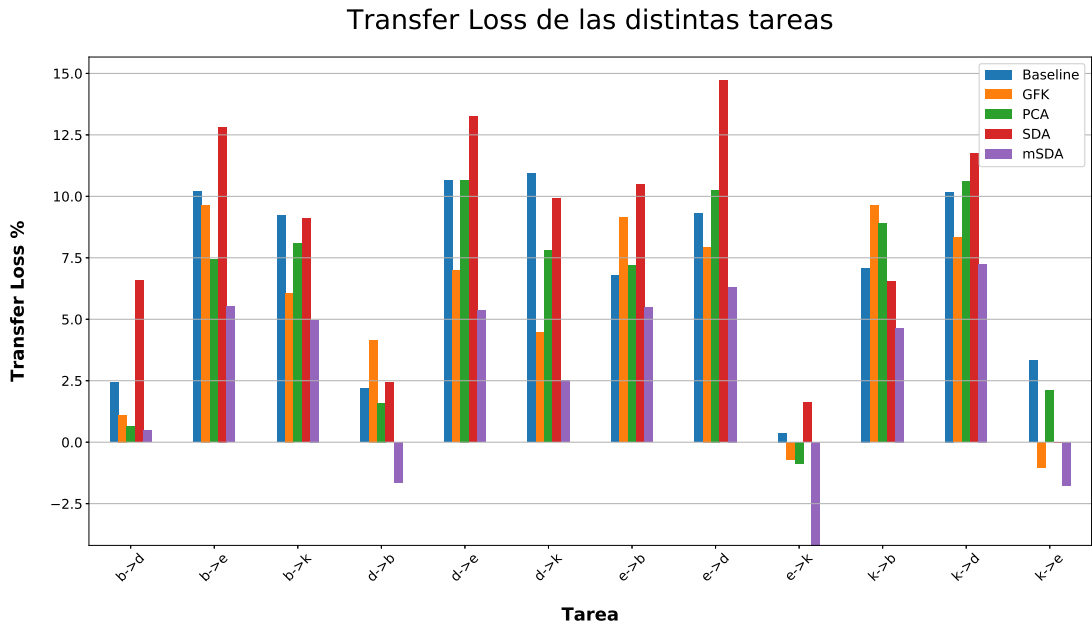


Figura 5.3: *Transfer loss* para el dataset Amazon utilizando 1000 dimensiones en el marco experimental 1.

algoritmo. En la Tabla 5.5 se observa un recuento de las comparaciones realizadas de acuerdo a la cantidad de tareas en que cada algoritmo mejora o empeora los resultados del método *Baseline*.

Se observa que con 3000 dimensiones el algoritmo *mSDA* obtiene la mayor cantidad de tareas con resultados mejorados, seguido del algoritmo *GFK*. Mientras que los algoritmos *SDA* y *PCA* obtienen peores desempeños, sin lograr mejorar las clasificaciones en ningún caso.

Por otra parte, al utilizar 1000 dimensiones se observa una mejora significativa en los resultados obtenidos. Se tiene en este caso que todos los métodos implementados aumentan la cantidad de tareas de adaptación mejoradas. Se tiene que *mSDA* mejora la totalidad de las adaptaciones, luego *GFK* mejora 3 tareas más, *PCA* aumenta considerablemente la cantidad, en donde pasó de 0 a 7 tareas mejoradas, y finalmente *SDA* mejora 4 tareas de adaptación. Así mismo, los dos algoritmos que realizan la mayor cantidad de mejoras son *mSDA* y *GFK*.

Tabla 5.5: Comparación de los resultados obtenidos por cada algoritmo en comparación al método *Baseline* en el marco experimental 1.

Algoritmo	3000 Dimensiones		1000 Dimensiones	
	Clasificaciones mejoradas	Clasificaciones empeoradas	Clasificaciones mejoradas	Clasificaciones empeoradas
<i>GFK</i>	6	6	9	3
<i>PCA</i>	0	12	7	5
<i>SDA</i>	0	12	4	8
<i>mSDA</i>	8	4	12	0

Luego, al contar la cantidad de veces que cada algoritmo obtiene el mejor resultado (solamente cuando se supera al método *Baseline*) se obtiene en la Tabla 5.6 que utilizando 3000 dimensiones los algoritmos *GFK* y *mSDA* obtienen la mayor cantidad de veces los mejores resultados. Mientras que al utilizar 1000 dimensiones, el algoritmo *mSDA* obtiene los mejores resultados en cada una de las tareas.

Tabla 5.6: Conteo de la cantidad de veces que cada método obtiene los mejores resultados con el marco experimental 1.

	3000 Dimensiones	1000 Dimensiones
Algoritmo	Mejores Resultados	Mejores Resultados
<i>GFK</i>	5	0
<i>PCA</i>	0	0
<i>SDA</i>	0	0
<i>mSDA</i>	5	12

En general, los resultados coinciden parcialmente con los esperados. A pesar de no usar la misma cantidad de dimensiones que en las publicaciones², se obtiene al menos un mismo comportamiento por parte de los algoritmos. Al utilizar 1000 dimensiones se obtienen comportamientos positivos por parte de todos los algoritmos. Sin embargo, al aumentar la cantidad de dimensiones a 3000, el comportamiento de algunos de los algoritmos empieza a decaer un poco, en los casos de *mSDA* y *GFK*, y a decaer de manera más abrupta en los casos de *PCA* y *SDA*.

Los datos anteriores muestran que disminuir la dimensionalidad mejora las adaptaciones si se miden con respecto a un clasificador entrenado con los datos del dominio objetivo, pero con la misma dimensionalidad. También permiten eliminar la suposición de una incorrecta implementación de los algoritmos que muestran resultados poco satisfactorios con 3000 dimensiones, ya que al ser comparados con los resultados de sus publicaciones originales, estos no muestran una completa concordancia.

A pesar de lo anterior, no es posible asegurar que la disminución en los errores se debe completamente a la disminución de las dimensionalidades, ya que, como fue mostrado inicialmente, los valores de *baseline in-domain error* son diferentes al utilizar distintas dimensiones. El caso es el siguiente: al utilizar una mayor cantidad de dimensiones, los valores de *baseline in-domain error* son menores, sin embargo, las

²En las publicaciones respectivas de los métodos *mSDA* y *SDA* se utilizan 5000 dimensiones, sin embargo, en los experimentos se utiliza una menor cantidad debido a limitantes computacionales.

adaptaciones dan peores resultados. Entonces, una suposición a comprobar es la siguiente: el desempeño de los algoritmos no varía en mayor magnitud al utilizar menos dimensiones, sino que, los algoritmos tienden a mejorar mucho más las clasificaciones cuando estas no tienen buenos rendimientos, y de la misma forma, cuando las clasificaciones ya tienen buenos resultados, entonces los algoritmos van a adaptar en menor medida.

Una forma de comprobar la suposición anterior es utilizando la métrica *transfer ratio*, la cual al ser una división de dos valores, es más útil para los casos en donde la dimensión cambia³. En la Figura 5.4 se observa el promedio de *transfer ratio* de todas las tareas, y se tiene que los promedios, cuando se usan 1000 dimensiones, son menores a cuando se usan 3000, siendo esta una tendencia que se repite en todos los algoritmos. Además, cuando se utilizan 1000 dimensiones, los valores llegan a ser menos de la mitad de los valores cuando se utilizan 3000 dimensiones.

Habiendo establecido lo anterior, se tiene como opción lo siguiente: si dos algoritmos obtienen el mismo valor de *transfer ratio*, esto significa que su desempeño ha sido el mismo. Luego, si un algoritmo obtiene un menor valor de *transfer ratio* que otro algoritmo, esto significa que el primero ha obtenido un mejor desempeño, lo cual se cumple en los casos en donde se utilizan 1000 dimensiones y permite atribuir la mejora en las adaptaciones a la disminución de dimensiones.

Por otra parte, cuando se comparan los valores finales de las clasificaciones, en vez de los rendimientos de las adaptaciones, se observa en la Tabla A.1 una comparación entre las distintas dimensionalidades, por cada algoritmo y prueba de adaptación, de los valores de error de las clasificaciones (que corresponden a los valores de *transfer error*) y se observa que en mayor medida, cuando se utilizan 3000 dimensiones se obtienen 44 casos con menores errores, mientras que cuando se utilizan 1000 dimensiones, solo se obtienen 16 casos⁴. Luego, de la misma manera, al considerar solo el

³Cabe recordar que *transfer ratio* se define como la división entre *transfer error* e *in-domain baseline error*, por lo que resulta más fácil hacer comparaciones utilizando esta métrica cuando el *in-domain baseline error* cambia, producto de la variación de dimensiones en este caso. Así mismo, también es necesario recordar que menores valores implican mejores resultados.

⁴En total se consideran 60 casos, ya que se tienen 12 pares de tareas por cada uno de los 5 métodos probados, incluyendo al método *Baseline*, en donde no se realiza ninguna adaptación.

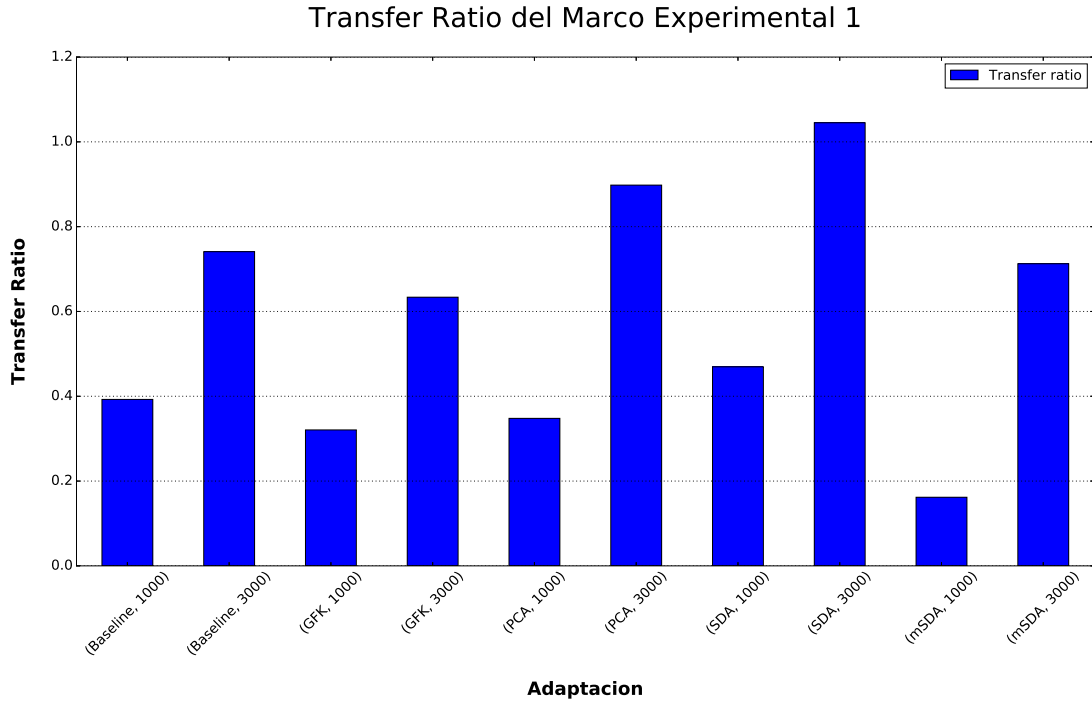


Figura 5.4: Promedio de los valores de *transfer ratio* para el *dataset Amazon* en el marco experimental 1.

menor error por cada tarea de adaptación, se observa en la Tabla 5.7 que los menores errores se obtienen cuando se utilizan 3000 dimensiones. Por lo que, a pesar de que se considera que con 1000 dimensiones las adaptaciones realizan un mejor desempeño, uno de los criterios más importantes es el error que se obtiene por cada clasificación, y en este caso se obtiene que con 1000 dimensiones no se generan en mayor medida mejores resultados.

Marco Experimental 2

En el marco experimental 2 se espera ver una diferencia en los resultados con respecto al marco experimental 1, al ser las adaptaciones solamente dependientes de los datos de los dominios fuente y objetivo.

Al realizar adaptaciones con 3000 dimensiones se observa en la figura 5.5 que hay casos en los que algunos algoritmos mejoran sus resultados, mientras que otros

Tabla 5.7: Menores valores de *transfer error* por cada tarea de adaptación con el *dataset Amazon* en el marco experimental 1.

Tarea	Algoritmo	Dimensiones	Transfer Error
b->d	mSDA	1000	17.375
b->e	Baseline	3000	19.085
b->k	GFK	3000	15.825
d->b	mSDA	3000	21.901
d->e	GFK	3000	18.610
d->k	GFK	3000	13.233
e->b	GFK	3000	25.273
e->d	mSDA	1000	23.216
e->k	mSDA	3000	9.610
k->b	mSDA	1000	28.361
k->d	mSDA	1000	24.141
k->e	mSDA	3000	12.200

empeoran. De la misma manera que en el marco experimental 1, existen casos en donde no es posible obtener mejores resultados que los del método *Baseline*, esto ocurre en 4 casos: " $b \rightarrow e$ ", " $e \rightarrow k$ ", " $k \rightarrow d$ " y " $k \rightarrow e$ ", siendo el primero también un caso que ocurre en el marco experimental anterior. También se encuentra que la correspondencia entre dominios para obtener menores valores de *transfer loss*, vista en el marco experimental 1, se vuelve a repetir. Siendo los dominios *books* y *dvd* un par, y *electronics* y *kitchen* el otro, que vienen a reforzar los valores entregados por la medida *PAD*.

En la Figura 5.6 se observa que utilizar el marco experimental 2 con 1000 dimensiones también genera una mejora general en las adaptaciones realizadas, en comparación a las obtenidas utilizando 3000 dimensiones bajo el mismo marco experimental. Para comenzar, hay solo 2 tareas cuyas clasificaciones no hayan sido mejoradas por algún algoritmo, además, ahora se tienen 2 tareas en donde hay valores menores a 0.

Luego, analizando la cantidad de veces que cada algoritmo mejora o empeora las

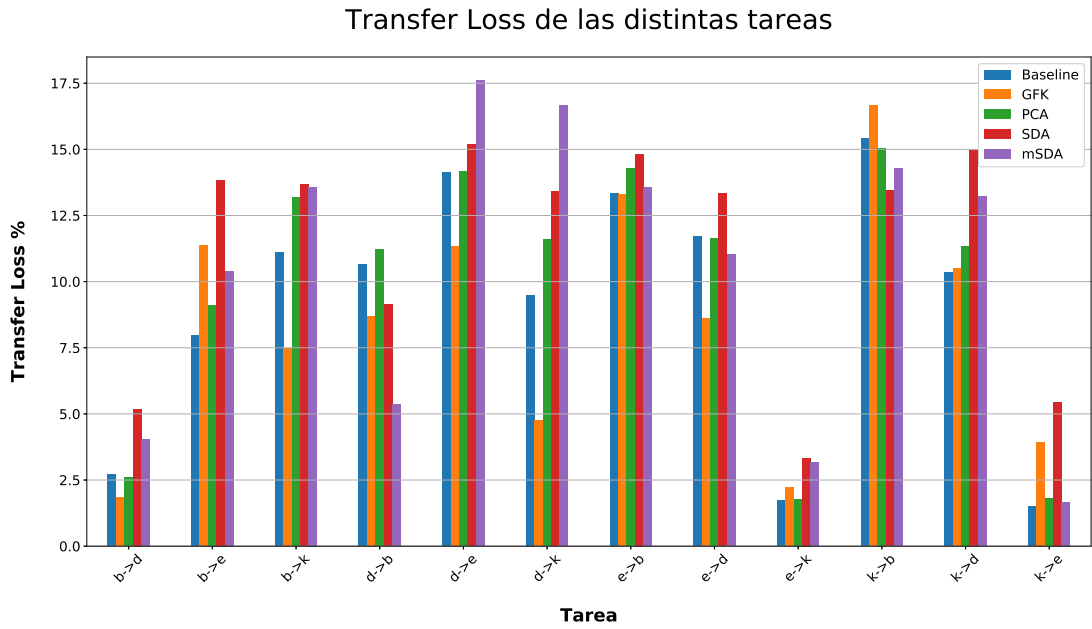


Figura 5.5: *Transfer loss* para el dataset Amazon utilizando 3000 dimensiones en el marco experimental 2.

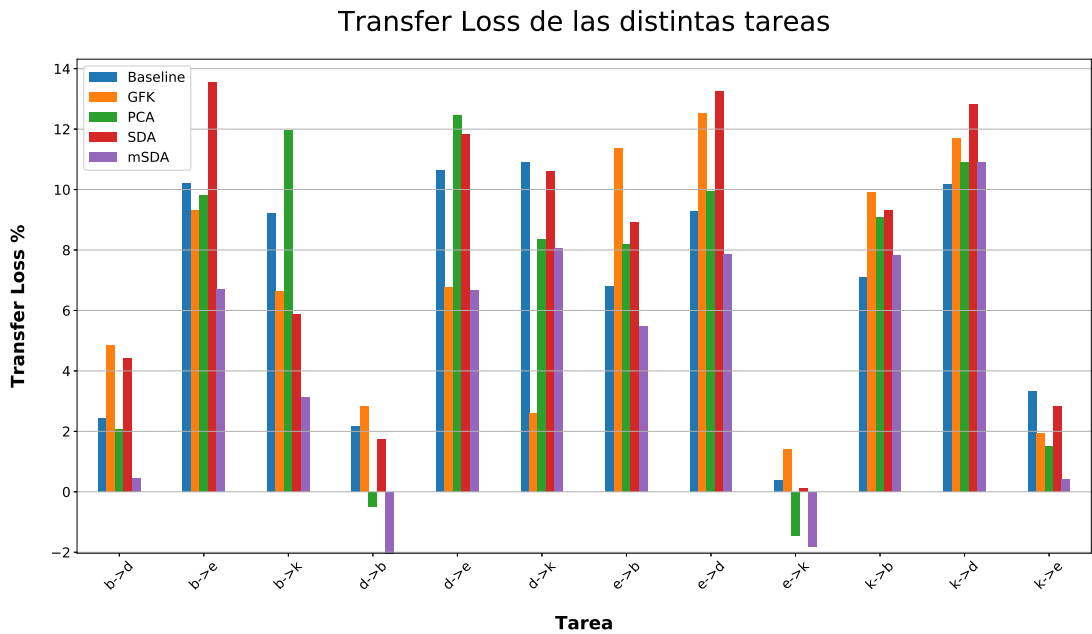


Figura 5.6: *Transfer loss* para el dataset Amazon utilizando 1000 dimensiones en el marco experimental 2.

clasificaciones se obtienen los resultados de la Tabla 5.8, en donde los resultados al pasar de 3000 a 1000 dimensiones mejoran considerablemente. Para el algoritmos *mSDA*, *PCA* y *SDA*, la cantidad de veces que mejoran las clasificaciones aumenta considerablemente al utilizar 1000 dimensiones. Mientras que por otra parte, el algoritmo *GFK* disminuye la cantidad de veces que mejora las clasificaciones.

Tabla 5.8: Comparación de los resultados obtenidos por cada algoritmo en comparación al método *Baseline* en el marco experimental 2.

Algoritmo	3000 Dimensiones		1000 Dimensiones	
	Clasificaciones mejoradas	Clasificaciones empeoradas	Clasificaciones mejoradas	Clasificaciones empeoradas
<i>GFK</i>	7	5	5	7
<i>PCA</i>	3	9	6	6
<i>SDA</i>	2	10	5	7
<i>mSDA</i>	3	9	10	2

En la Tabla 5.9 se observa que la disminución a 1000 dimensiones también beneficia en mayor medida al algoritmo *mSDA*, pasando a tener la mayor cantidad de mejores resultados. También se debe considerar el hecho de que al disminuir las dimensiones existe una cantidad de tareas que utilizando 3000 dimensiones no habían podido ser mejoradas mediante la adaptación.

Tabla 5.9: Conteo de la cantidad de veces que cada método obtiene los mejores resultados con el marco experimental 2.

Algoritmo	3000 Dimensiones	1000 Dimensiones
	Mejores Resultados	Mejores Resultados
<i>GFK</i>	6	1
<i>PCA</i>	0	0
<i>SDA</i>	1	0
<i>mSDA</i>	1	9

En la Figura 5.7 se observan los promedios de *transfer ratio*, en donde se observa

un comportamiento similar al del marco experimental 1. Los promedios son menores cuando se utilizan menos dimensiones. Sin embargo, al igual que en el marco experimental 1, la cantidad de pruebas que tienen menor *transfer error* son las realizadas con 3000 dimensiones, que son 49 casos de los 60 en total, mientras que con 1000 dimensiones se obtienen los 11 casos restantes.

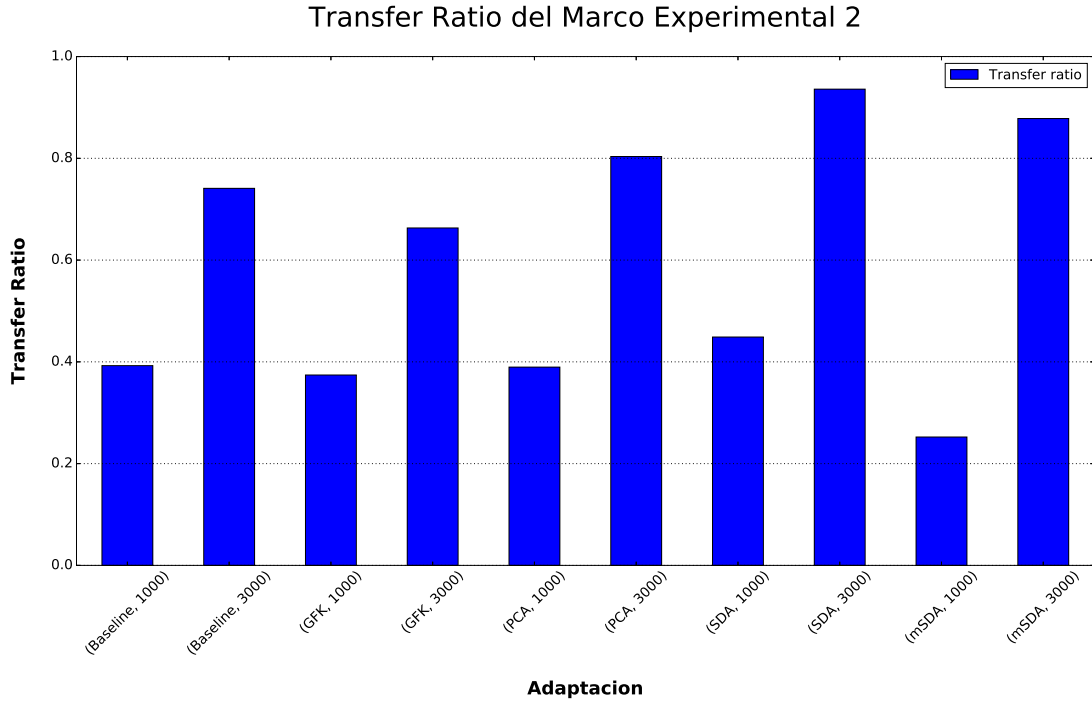


Figura 5.7: Promedio de los valores de *transfer ratio* para el *dataset Amazon* en el marco experimental 2.

Comparación de Algoritmos Entre Marcos Experimentales

Comparando los resultados del algoritmo *GFK* en ambos marcos experimentales, se obtiene en la Figura 5.8 que utilizando 3000 dimensiones no existe una diferencia significativa entre ambos marcos, siendo un par de casos en donde se observa una mayor diferencia de valores, como por ejemplo las pruebas " $d \rightarrow e$ " y " $e \rightarrow b$ ".

Por otra parte, con 1000 dimensiones, en la Figura 5.9 las diferencias se hacen más notorias, favoreciendo al marco experimental 1, que muestra una mayor cantidad

Tabla 5.10: Menores valores de *transfer error* por cada tarea de adaptación con el *dataset Amazon* en el marco experimental 2.

Tarea	Algoritmo	Dimensiones	Transfer Error
b->d	mSDA	1000	17.338
b->e	Baseline	3000	19.085
b->k	GFK	3000	16.083
d->b	mSDA	3000	20.513
d->e	GFK	3000	22.468
d->k	GFK	3000	13.340
e->b	GFK	3000	28.463
e->d	GFK	3000	24.378
e->k	Baseline	3000	10.323
k->b	SDA	3000	28.598
k->d	Baseline	3000	26.113
k->e	Baseline	3000	12.625

Comparación de marcos para el algoritmo GFK

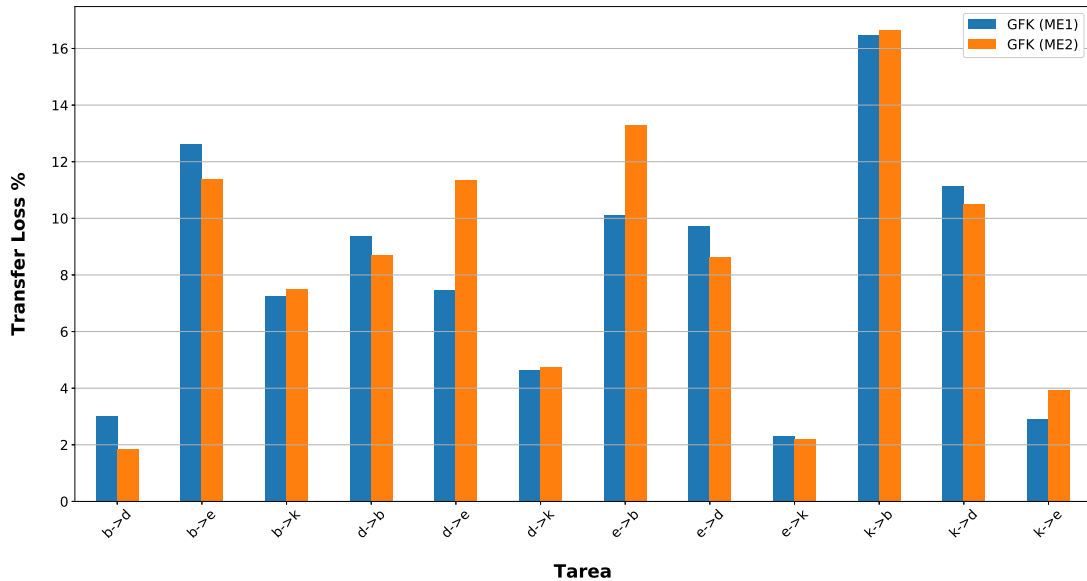


Figura 5.8: *Transfer loss* para el algoritmo *GFK* utilizando 3000 dimensiones en ambos marcos experimentales.

de casos en donde obtiene menores valores, e incluso presenta valores negativos.

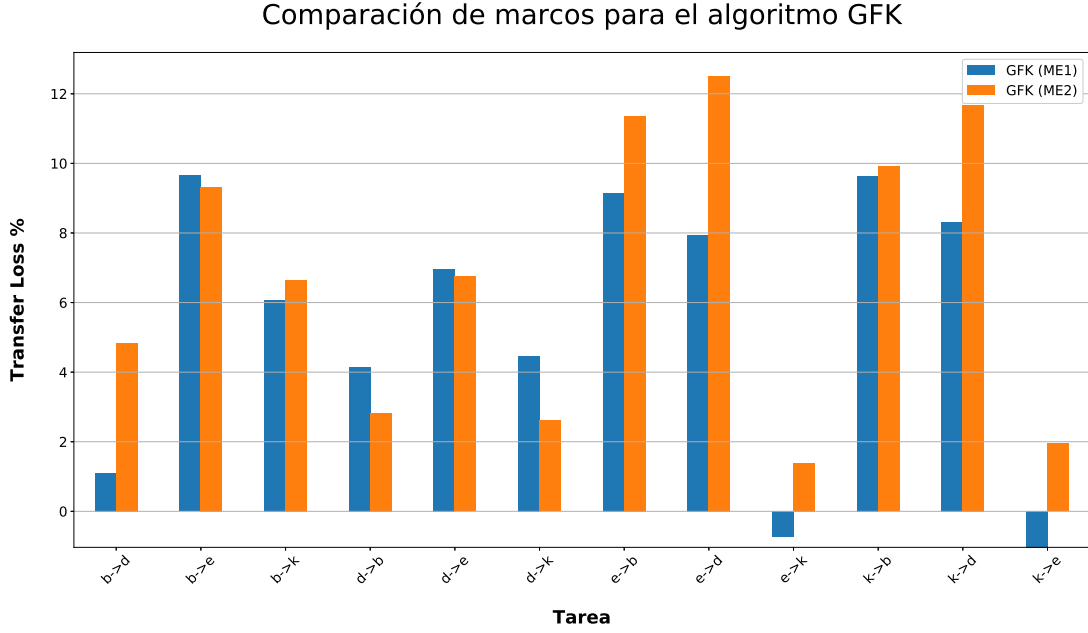


Figura 5.9: *Transfer loss* para el algoritmo *GFK* utilizando 1000 dimensiones en ambos marcos experimentales.

En el caso de *PCA*, al utilizar 3000 dimensiones, se observa en la Figura 5.10 una mayor predominancia del marco experimental 2 para obtener menores valores, aunque la diferencia existente entre los valores de ambos marcos no se considera tan significativa. Luego, en la Figura 5.11 se observa que con 1000 dimensiones se intercambian los roles, obteniendo mejores resultados el marco experimental 1.

Para el algoritmo *SDA* con 3000 dimensiones, se observa en la Figura 5.12 que existen diferencias grandes en los valores de *transfer loss* en los casos " $b \rightarrow e$ ", " $d \rightarrow b$ " y " $k \rightarrow b$ ", siendo estas diferencias más perjudiciales para el marco experimental 1. Por otra parte, el resto de los resultados se ve balanceado entre ambos marcos experimentales. En el caso de las pruebas con 1000 dimensiones, visto en la Figura 5.13, se tiene una variación en los resultados de varias pruebas, sin embargo, en general los marcos experimentales se ven balanceados en términos de la cantidad de veces que cada uno obtiene los menores resultados.

Los resultados del algoritmo *mSDA* con 3000 dimensiones, mostrados en la Figura

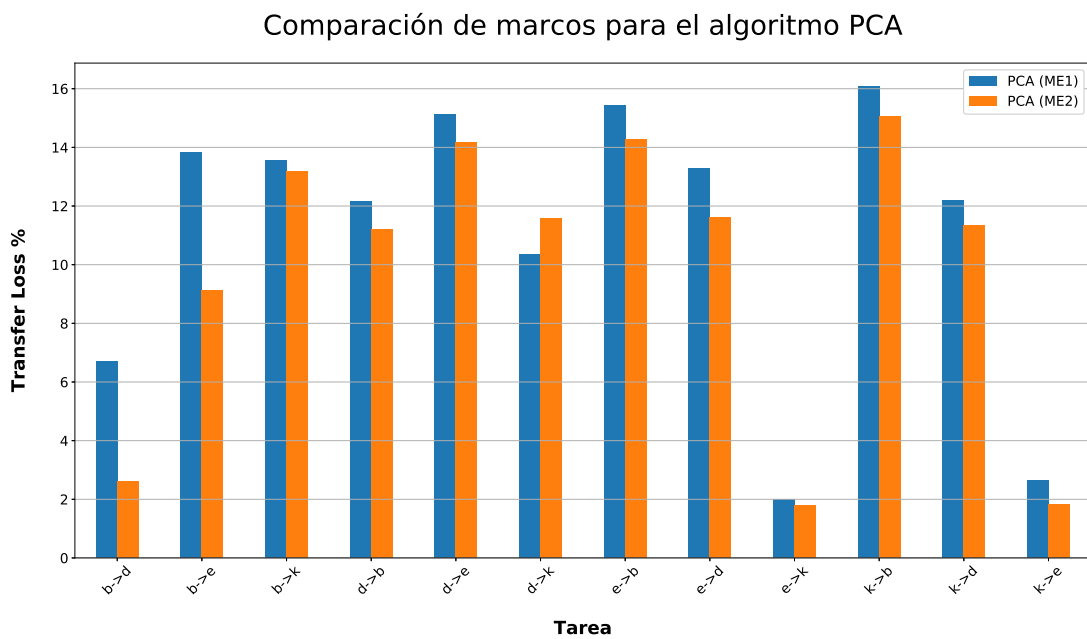


Figura 5.10: *Transfer loss* para el algoritmo *PCA* utilizando 3000 dimensiones en ambos marcos experimentales.

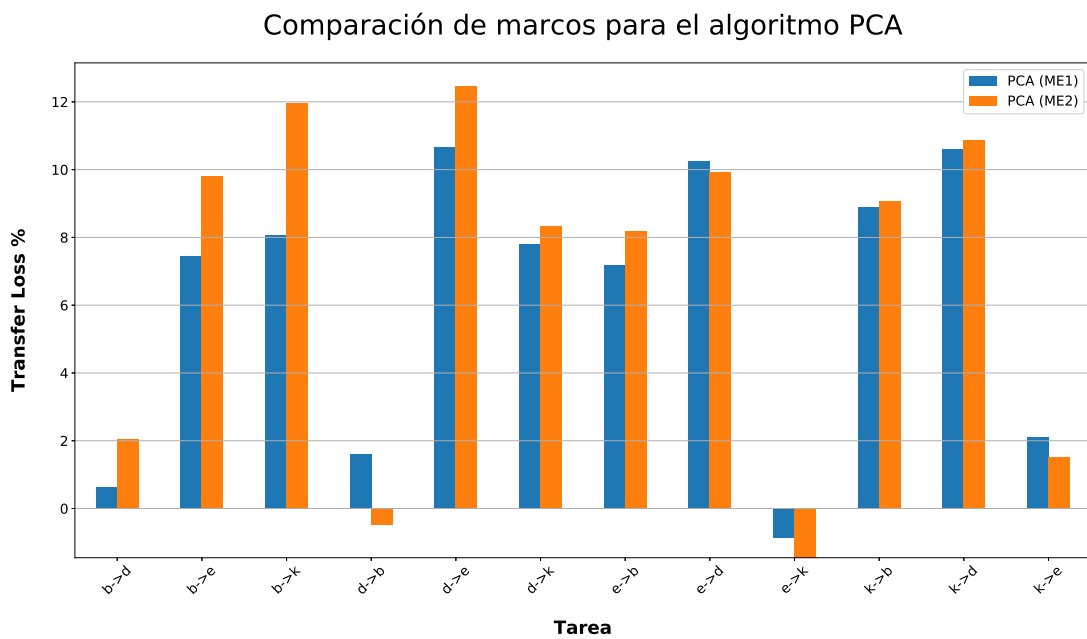


Figura 5.11: *Transfer loss* para el algoritmo *PCA* utilizando 1000 dimensiones en ambos marcos experimentales.

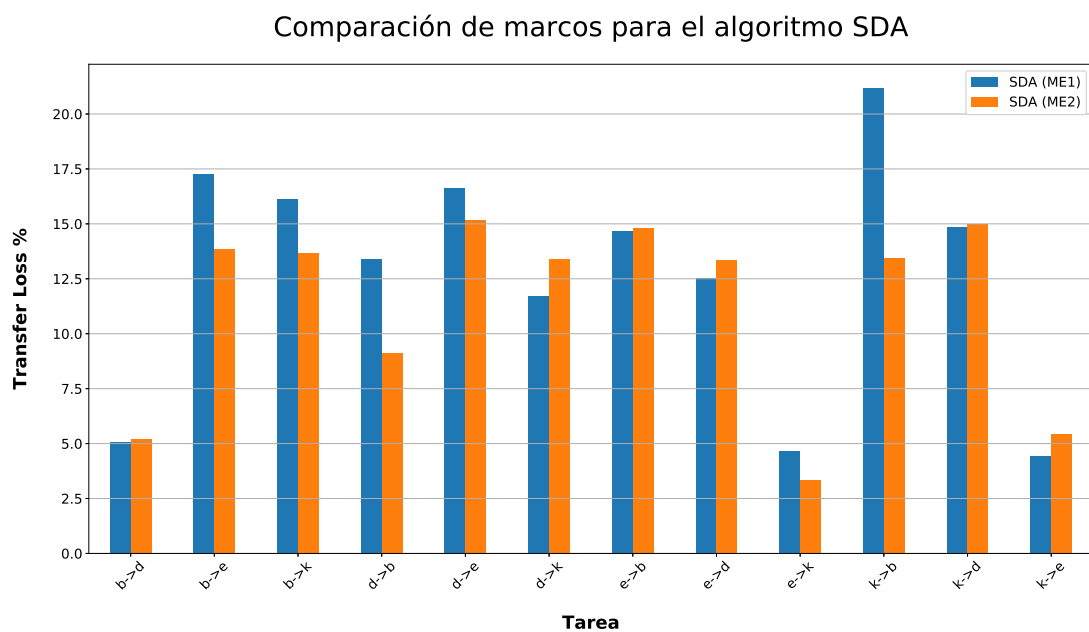


Figura 5.12: *Transfer loss* para el algoritmo SDA utilizando 3000 dimensiones en ambos marcos experimentales.

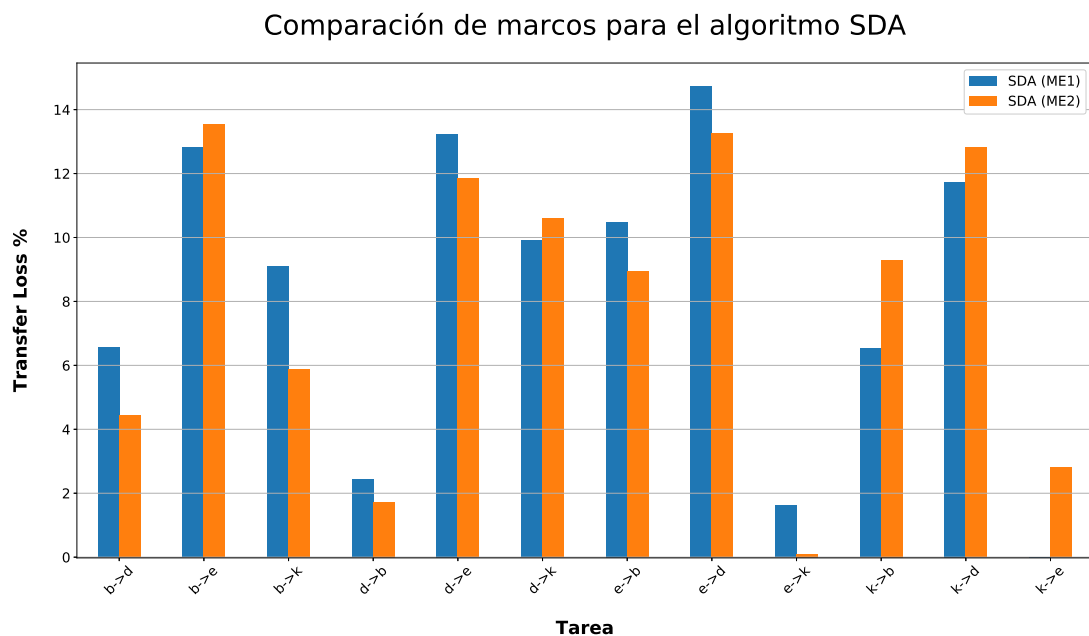


Figura 5.13: *Transfer loss* para el algoritmo SDA utilizando 1000 dimensiones en ambos marcos experimentales.

5.14, muestran que se tienen menores valores con el marco experimental 1, logrando diferencias notorias en tareas como " $d \rightarrow e$ " y " $d \rightarrow k$ ". Luego, en la Figura 5.15 se observa un comportamiento similar al utilizar 1000 dimensiones, en donde el marco experimental 1 tiene mejores resultados y con mayor diferencia.

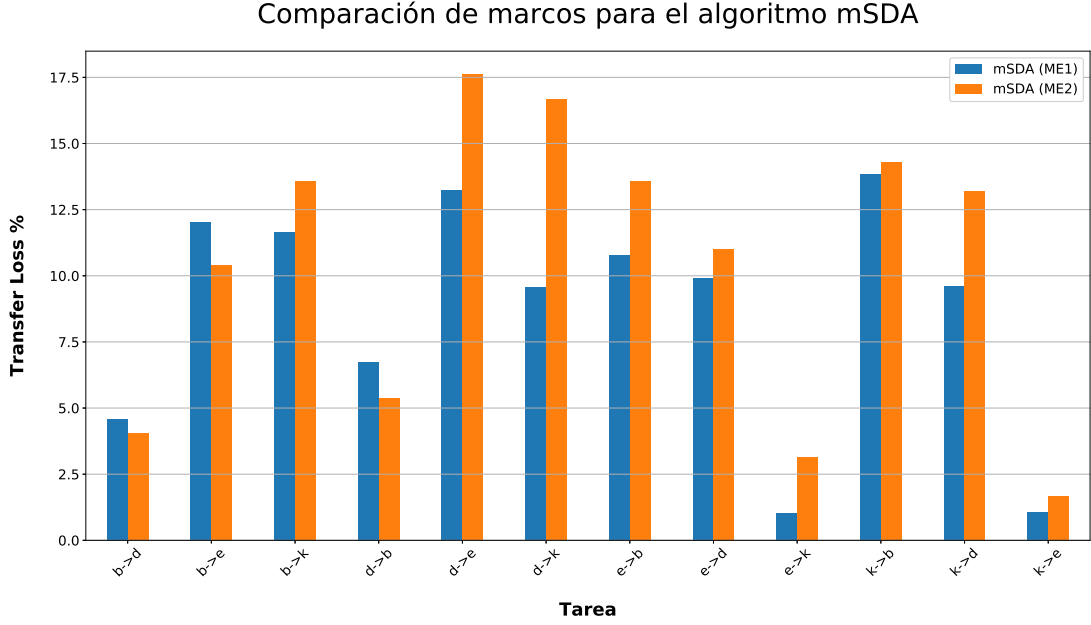


Figura 5.14: *Transfer loss* para el algoritmo mSDA utilizando 3000 dimensiones en ambos marcos experimentales.

En la Tabla 5.11 se observa un conteo de las veces que cada marco experimental obtiene menores resultados que el otro, en donde se observa que, utilizando 3000 dimensiones: los algoritmos *GFK* y *SDA* tienen un rendimiento similar en ambos marcos, *PCA* muestra un mejor rendimiento en el marco experimental 2, y *mSDA* en general obtiene mejores resultados con el marco experimental 1. Luego, al utilizar 1000 dimensiones: los algoritmos *GFK* y *PCA* pasan a tener mejores resultados con el marco experimental 1, mientras que los algoritmos *SDA* y *mSDA* tienen pocas variaciones con respecto a lo obtenido con 3000 dimensiones.

En la Tabla 5.12 se observa, por cada algoritmo y dominio objetivo, cuáles dominios fuentes fueron los que lograron la mejor adaptación utilizando 3000 dimensiones, separando si es que estos fueron utilizados en el marco experimental 1 o 2. Lo primero a notar es la predominancia que tienen los resultados del marco experimental 2 para

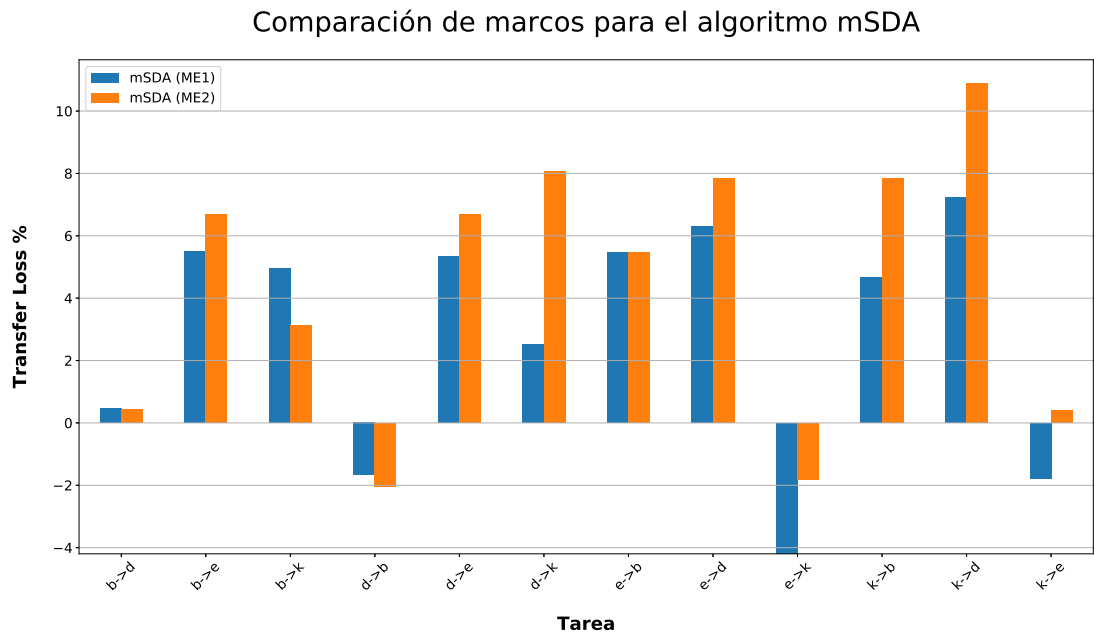


Figura 5.15: *Transfer loss* para el algoritmo mSDA utilizando 1000 dimensiones en ambos marcos experimentales.

Tabla 5.11: Comparación de los resultados obtenidos por ambos marcos experimentales en el *dataset Amazon*.

	3000 Dimensiones		1000 Dimensiones	
Algoritmo	M.E. 1	M.E. 2	M.E. 1	M.E. 2
GFK	6	6	8	4
PCA	1	11	8	4
SDA	6	6	5	7
mSDA	9	3	8	3

obtener los menores valores⁵. Luego, se puede observar también una simetría entre los dominios que generan las mejores adaptaciones, existiendo así los pares (*kitchen*, *electronics*) y (*books*, *dvd*). De la misma forma se observa que para todos los algoritmos, los dominios fuente que realizan la mejor adaptación por cada dominio objetivo se mantienen constantes, independiente del marco experimental al que pertenecen⁶.

Al comparar los resultados con los valores de la Tabla 5.4, se obtienen en mayor parte coincidencias. En la Tabla 5.4 se obtienen menores distancias entre los dominios *electronics* y *kitchen* de manera simétrica, lo cual se cumple con la mayoría de los algoritmos. Para los casos en donde el marco experimental 2 obtiene los mejores resultados esto se cumple, dado que se utilizan solamente estos dos dominios. Mientras que con los resultados que provienen del marco experimental 1, a pesar de que se utilizan todos los dominios para la adaptación, el uso de estos dominios para la selección de modelos es de gran importancia, ya que se utilizan para simular la existencia de datos etiquetados, y sus distribuciones de probabilidad, en el dominio objetivo. Por otra parte, a pesar de que los dominios *dvd* y *books* tienen menor distancia con la unión de todos los dominios, se tiene que los siguientes dominios con menor distancia son ellos mismos, formando un par (*dvd*, *books*), lo cual se cumple con la mayoría de los algoritmos, en donde estos dominios dan los mejores resultados con el marco experimental 2.

En la Tabla 5.13 se observa que con 1000 dimensiones, se mantiene la correspondencia entre pares de dominios. También se observa una mayor cantidad de adaptaciones en donde el marco experimental 2 obtiene los menores valores, aunque la diferencia en la cantidad que obtiene cada marco es mínima. Observando cada dominio por separado, se puede ver que el dominio *books* tiene mejores valores con el marco experimental 2. Los dominios *dvd* y *kitchen* no tienen una predominancia de alguno de los dos marcos, y finalmente, el dominio *electronics* tiene mejores resultados con el marco experimental 1. Lo cual se cumple en cierta medida con lo esperado según la

⁵Es necesario recordar que los resultados del método *Baseline* son ambos iguales en ambos marcos, dado que no se realiza adaptación en este método.

⁶A pesar de que en el marco experimental 1 se utiliza la unión de todos los dominios para realizar el entrenamiento de modelos de adaptación, al momento de seleccionar modelos para realizar la adaptación de los datos se utiliza solamente un dominio.

Tabla 5.12: Mejores resultados obtenidos por cada algoritmo para adaptar cada dominio objetivo en el *dataset Amazon* utilizando 3000 dimensiones.

Algoritmo	Dominio Objetivo	Dominio Fuente	Transfer loss
Baseline	books	dvd	10.650
	dvd	books	2.708
	electronics	kitchen	1.498
	kitchen	electronics	1.735
GFK	books	dvd (M.E. 2)	8.700
	dvd	books (M.E. 2)	1.840
	electronics	kitchen (M.E. 1)	2.918
	kitchen	electronics (M.E. 2)	2.205
PCA	books	dvd (M.E. 2)	11.218
	dvd	books (M.E. 2)	2.593
	electronics	kitchen (M.E. 2)	1.810
	kitchen	electronics (M.E. 2)	1.783
SDA	books	dvd (M.E. 2)	9.125
	dvd	books (M.E. 1)	5.085
	electronics	kitchen (M.E. 1)	4.438
	kitchen	electronics (M.E. 2)	3.338
mSDA	books	dvd (M.E. 2)	5.355
	dvd	books (M.E. 2)	4.033
	electronics	kitchen (M.E. 1)	1.073
	kitchen	electronics (M.E. 1)	1.023

Tabla 5.4, ya que los dominios con los cuales se obtienen mejores resultados coinciden en mayor medida, pero no ocurre así con el marco experimental esperado, es decir, con algún dominio por si solo o con la union de estos.

Tabla 5.13: Mejores resultados obtenidos por cada algoritmo para adaptar cada dominio objetivo en el *dataset Amazon* utilizando 1000 dimensiones.

Algoritmo	Dominio Objetivo	Dominio Fuente	Transfer loss
Baseline	books	dvd	2.183
	dvd	books	2.428
	electronics	kitchen	3.323
	kitchen	electronics	0.368
GFK	books	dvd (M.E. 2)	2.830
	dvd	books (M.E. 1)	1.110
	electronics	kitchen (M.E. 1)	-1.035
	kitchen	electronics (M.E. 1)	-0.728
PCA	books	dvd (M.E. 2)	-0.480
	dvd	books (M.E. 1)	0.628
	electronics	kitchen (M.E. 2)	1.513
	kitchen	electronics (M.E. 2)	-1.453
SDA	books	dvd (M.E. 2)	1.723
	dvd	books (M.E. 2)	4.430
	electronics	kitchen (M.E. 1)	-0.023
	kitchen	electronics (M.E. 2)	0.100
mSDA	books	dvd (M.E. 2)	-2.033
	dvd	books (M.E. 2)	0.438
	electronics	kitchen (M.E. 1)	-1.785
	kitchen	electronics (M.E. 1)	-4.198

En la Tabla 5.14 se observa una recopilación de los mejores resultados en ambos marcos experimentales utilizando 3000 dimensiones. En donde se observa que el marco experimental 1 obtiene 7 veces los mejores resultados, mientras que el marco experimental 2 obtiene 4 veces los mejores resultados. Por otra parte en la Tabla 5.15

se tiene que, utilizando 1000 dimensiones, el marco experimental 1 obtiene 9 veces las mejores adaptaciones, mientras que el marco experimental 2 obtiene las mejores adaptaciones 3 veces.

Finalmente, en la Tabla 5.16 se observa que con 3000 dimensiones, la mayoría de los mejores resultados son obtenidos al utilizar el algoritmo *GFK*, seguido por *mSDA* y *SDA*. Mientras que con 1000 dimensiones, la totalidad de los resultados se obtienen utilizando el algoritmo *mSDA*.

Tabla 5.14: Mejores adaptaciones obtenidas en ambos marcos experimentales, utilizando 3000 dimensiones con el *dataset Amazon*.

Tarea	Algoritmo	Marco	Transfer loss
$b \rightarrow d$	GFK	M.E. 2	1.840
$b \rightarrow e$	Baseline	–	7.958
$b \rightarrow k$	GFK	M.E. 1	7.238
$d \rightarrow b$	mSDA	M.E. 2	5.355
$d \rightarrow e$	GFK	M.E. 1	7.483
$d \rightarrow k$	GFK	M.E. 1	4.645
$e \rightarrow b$	GFK	M.E. 1	10.115
$e \rightarrow d$	GFK	M.E. 2	8.615
$e \rightarrow k$	mSDA	M.E. 1	1.023
$k \rightarrow b$	SDA	M.E. 2	13.440
$k \rightarrow d$	mSDA	M.E. 1	9.615
$k \rightarrow e$	mSDA	M.E. 1	1.073

5.1.2. Twitter

En este *dataset* se espera obtener una tendencia similar a la obtenida con el *dataset Amazon*, con el fin de asegurar una correcta implementación de los algoritmos y el de probar su comportamiento con datos reales, además de probar el caso en donde se tiene un dominio con una cantidad mucho mayor de datos en comparación al resto.

Tabla 5.15: Mejores adaptaciones obtenidas en ambos marcos experimentales, utilizando 1000 dimensiones con el *dataset Amazon*.

Tarea	Algoritmo	Marco	Transfer loss
$b \rightarrow d$	mSDA	M.E. 2	0.438
$b \rightarrow e$	mSDA	M.E. 1	5.515
$b \rightarrow k$	mSDA	M.E. 2	3.118
$d \rightarrow b$	mSDA	M.E. 2	-2.033
$d \rightarrow e$	mSDA	M.E. 1	5.345
$d \rightarrow k$	mSDA	M.E. 1	2.508
$e \rightarrow b$	mSDA	M.E. 1	5.468
$e \rightarrow d$	mSDA	M.E. 1	6.315
$e \rightarrow k$	mSDA	M.E. 1	-4.198
$k \rightarrow b$	mSDA	M.E. 1	4.650
$k \rightarrow d$	mSDA	M.E. 1	7.240
$k \rightarrow e$	mSDA	M.E. 1	-1.785

Tabla 5.16: Conteo de la cantidad de veces que cada método obtiene los mejores resultados en ambos marcos experimentales en el *dataset Amazon*.

	3000 Dimensiones	1000 Dimensiones
Algoritmo	Mejores Resultados	Mejores Resultados
<i>GFK</i>	6	0
<i>PCA</i>	0	0
<i>SDA</i>	1	0
<i>mSDA</i>	4	12

In-domain Error

En la Tabla 5.17 se obtienen los valores de *baseline in-domain error* para ambas dimensionalidades. Se observa que en al aumentar las dimensiones disponibles, los errores disminuyen, con excepción del dominio *rio2016*. Sin embargo, la variación que se observa es menor a la que muestran los otros dos dominios. Además, la justificación dada anteriormente, en donde se atribuye la disminución del *baseline in-domain error* al aumento de las dimensiones, sigue vigente, ya que es solamente un caso en donde esto no se cumple.

Tabla 5.17: *Baseline In-domain error* de los dominios del *dataset Twitter*.

Objetivo	1000 Dimensiones	2000 Dimensiones
thevoice	19.045	7.362
general	30.407	26.998
rio2016	7.372	10.208

En la figura 5.16 se observa gráficamente la diferencia entre los valores del *baseline in-domain error*, notando que en el caso del dominio *thevoice* el error disminuye a más de la mitad.

Similitud entre dominios

Los valores de la Tabla 5.18 muestran los valores de la métrica *PAD* entre los distintos pares de dominios utilizando 2000 dimensiones, en donde se observa una mayor cercanía de entre los pares de dominios de forma independiente, en comparación a la unión de ellos. Por otra parte, también se observa que los valores, en general, son menores a los que se obtuvieron con el *dataset Amazon*.

En la Tabla 5.19 se observa que con 1000 dimensiones los valores obtenidos son menores que los obtenidos con 2000 dimensiones. Atribuyendo nuevamente esta disminución en los valores a la disminución de la diversidad de los términos que ocurre al disminuir las dimensiones.

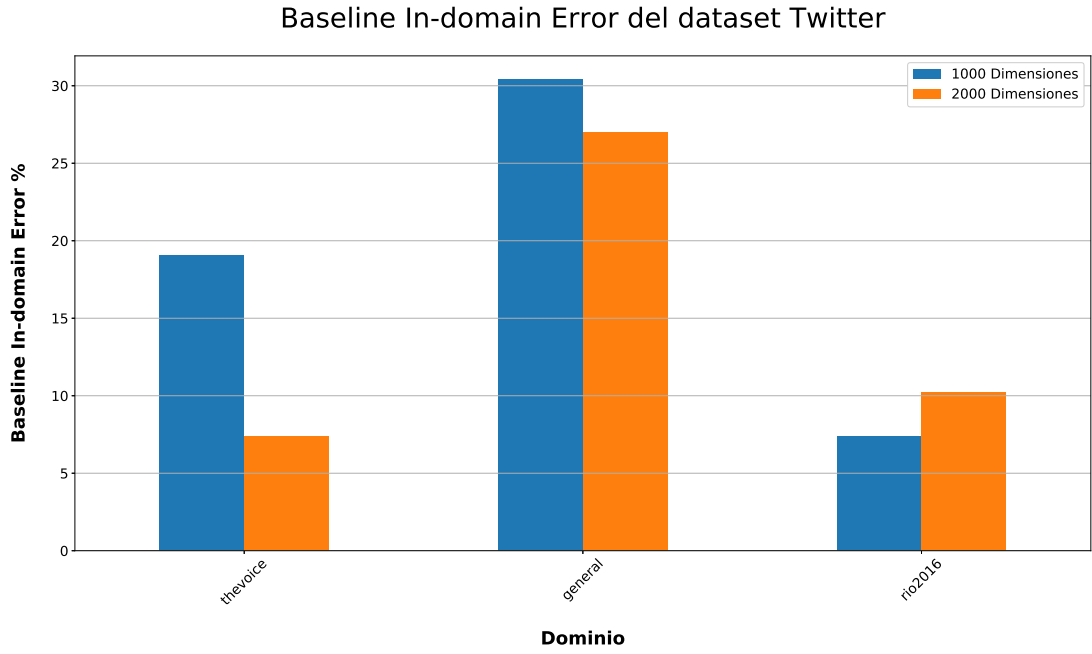


Figura 5.16: *Baseline In-domain error* para el *dataset Twitter* en ambas dimensiones.

Tabla 5.18: *Proxy A-distance* de los dominios del *dataset Twitter* con 2000 dimensiones

Dominio 1	Dominio 2	PAD
general	thevoice	1.603
	rio2016	1.712
	(thevoice, rio2016)	1.749
rio2016	general	1.712
	thevoice	1.718
	(thevoice, general)	1.725
thevoice	general	1.603
	(general, rio2016)	1.698
	rio2016	1.718

Tabla 5.19: *Proxy A-distance* de los dominios del *dataset Twitter* con 1000 dimensiones

Dominio 1	Dominio 2	PAD
general	thevoice	1.377
	rio2016	1.563
	('thevoice', 'rio2016')	1.585
rio2016	general	1.563
	thevoice	1.676
	('thevoice', 'general')	1.748
thevoice	general	1.377
	('general', 'rio2016')	1.467
	rio2016	1.676

La Tabla 5.20 muestra la comparación con ambas dimensionalidades, en donde se observa que para ambas dimensionalidades los dominios con los cuales se tendría una mejor adaptación son los mismos. Cabe notar también la presencia del dominio *general*, ya que no solamente por la distancia obtenida se espera que este permita hacer buenas adaptaciones, sino que además este cuenta con una mayor cantidad de datos de entrenamiento.

Tabla 5.20: Mejores valores de *Proxy A-distance* para cada dominio del *dataset Twitter* en ambas dimensiones.

Dominio 1	Dimensiones	Dominio 2	PAD
general	1000	thevoice	1.377
	2000	thevoice	1.603
rio2016	1000	general	1.563
	2000	general	1.712
thevoice	1000	general	1.377
	2000	general	1.603

Marco Experimental 1

En la Figura 5.17 se observa principalmente que no todos los algoritmos son capaces de mejorar las clasificaciones. Se tienen casos en donde los algoritmos dan buenos resultados en algunas tareas, mientras que en otras no alcanzan a dar valores menores a los del método *Baseline*. Sin embargo, se observa que en todas las tareas existe al menos un algoritmo que haya sido capaz de mejorar las clasificaciones.

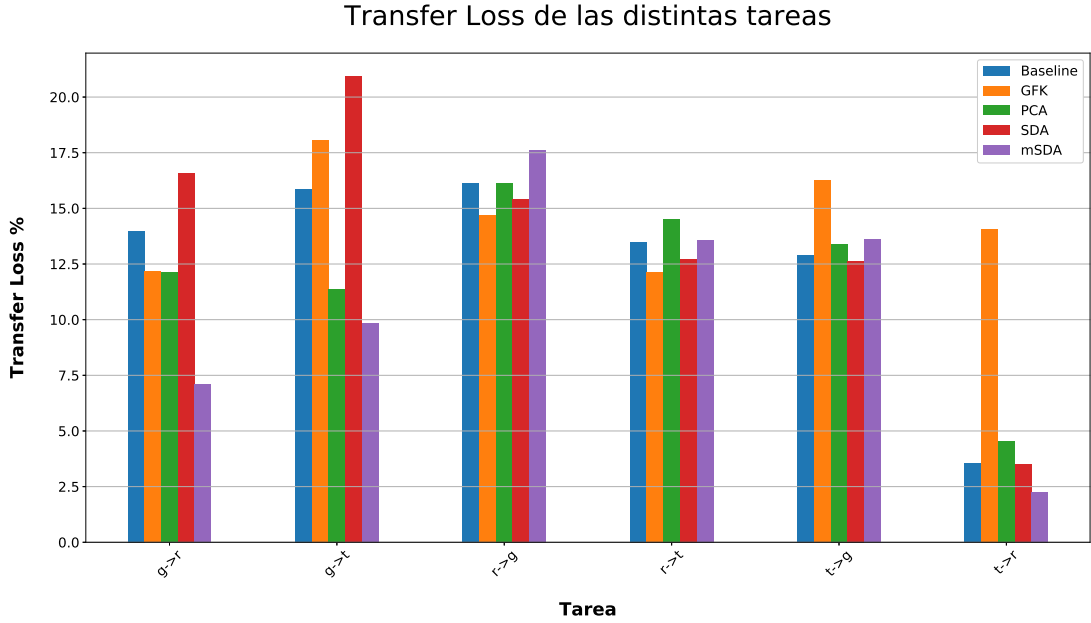


Figura 5.17: *Transfer loss* para el dataset *Twitter* utilizando 2000 dimensiones en el marco experimental 1.

Al disminuir la cantidad de dimensiones a 1000, se observa en la Figura 5.18 que no todos los resultados mejoran en comparación a los que se obtienen con 2000 dimensiones. En este caso se tienen 2 tareas en las cuales no es posible mejorar las clasificaciones. Por otra parte, con ambas dimensionalidades no se obtienen resultados negativos de *transfer loss*, al contrario de lo ocurrido en el dataset *Amazon*.

En la Tabla 5.21 se observa que no todos los algoritmos presentan mejoras al disminuir la cantidad de dimensiones. Por ejemplo, los algoritmos *GFK* y *SDA* pasan a tener 0 clasificaciones mejoradas al utilizar 1000 dimensiones, mientras que los algoritmos *PCA* y *mSDA* solamente aumentan la cantidad de clasificaciones mejoradas en

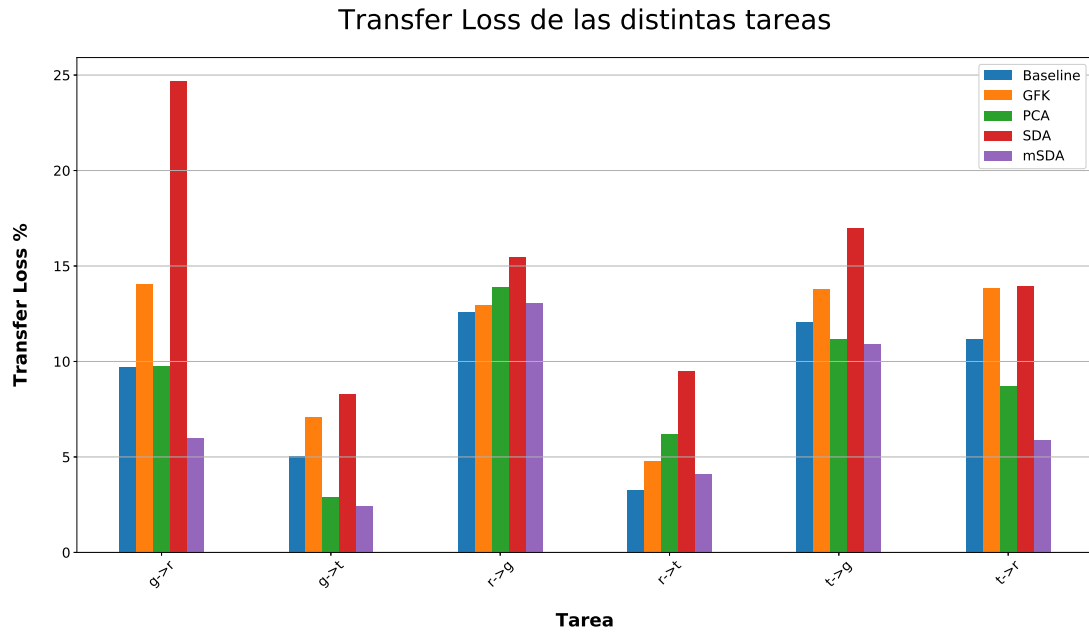


Figura 5.18: *Transfer loss* para el *dataset Twitter* utilizando 1000 dimensiones en el marco experimental 1.

1 tarea.

Tabla 5.21: Comparación de los resultados obtenidos por cada algoritmo en comparación al método *Baseline* en el marco experimental 1.

Algoritmo	2000 Dimensiones		1000 Dimensiones	
	Clasificaciones mejoradas	Clasificaciones empeoradas	Clasificaciones mejoradas	Clasificaciones empeoradas
<i>GFK</i>	3	3	0	6
<i>PCA</i>	2	4	3	3
<i>SDA</i>	4	2	0	6
<i>mSDA</i>	3	3	4	2

En la Tabla 5.22 se observan también los cambios al disminuir las dimensiones, en donde se observa que el algoritmo *mSDA* realiza siempre las mejores adaptaciones.

En la Figura 5.19 se observa un comportamiento similar al del *dataset Amazon*,

Tabla 5.22: Conteo de la cantidad veces que cada método obtiene los mejores resultados con el marco experimental 1.

	2000 Dimensiones	1000 Dimensiones
Algoritmo	Mejores Resultados	Mejores Resultados
<i>GFK</i>	2	0
<i>PCA</i>	0	0
<i>SDA</i>	1	0
<i>mSDA</i>	3	4

en donde los promedios de *transfer ratio* son menores cuando se utilizan menos dimensiones, pero, en este *dataset* se cuenta con la diferencia de que las adaptaciones con menores dimensiones no siempre mejoran. Por otra parte, cuando se comparan los valores de *transfer error* entre las distintas dimensiones, en la Tabla A.3 se observa que la mayoría de las veces los menores valores se obtienen usando 2000 dimensiones. Se tiene que, con 2000 dimensiones, son 22 tareas las que obtienen menores valores, mientras que con 1000 dimensiones solamente se obtienen 8. Luego, de la misma forma, en la Tabla 5.23 se observa que los mejores resultados para la mayoría de las pruebas se obtienen con 2000 dimensiones.

Tabla 5.23: Menores valores de *transfer error* por cada tarea de adaptación con el *dataset Twitter* en el marco experimental 1.

Tarea	Algoritmo	Dimensiones	Transfer Error
g->r	mSDA	1000	13.362
g->t	mSDA	2000	17.211
r->g	GFK	2000	41.676
r->t	GFK	2000	19.508
t->g	SDA	2000	39.635
t->r	mSDA	2000	12.477

En las pruebas realizadas bajo el marco experimental 1 no se observa, en general, el mismo comportamiento obtenido con el *dataset Amazon*, ya que no todas las

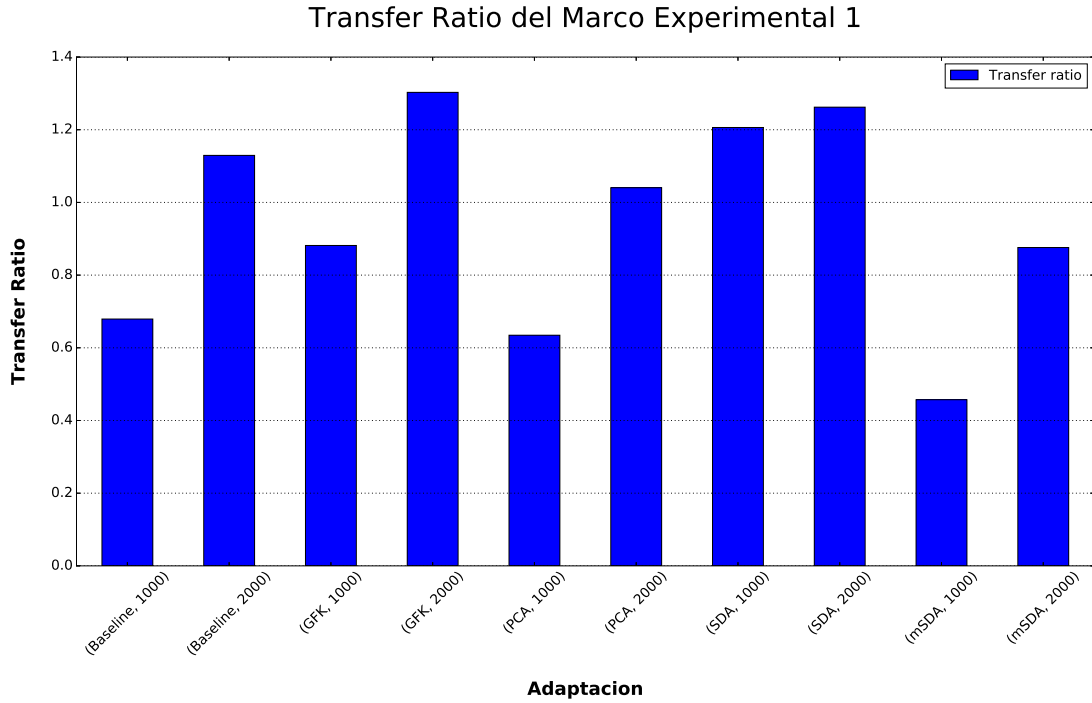


Figura 5.19: Promedio de los valores de *transfer ratio* para el *dataset Twitter* en el marco experimental 1.

adaptaciones dan buenos resultados, además, tampoco se obtienen valores menores a 0. Uno de los casos a notar es el del algoritmo *GFK*, el cual muestra un buen comportamiento con ambas dimensionalidades con el *dataset Amazon*, mientras que con el *dataset Twitter* los resultados empeoran drásticamente al disminuir la dimensiones. Sin embargo, a pesar de que los resultados de las adaptaciones parecen empeorar a medida que disminuyen las dimensiones, y a pesar de que la medida de *transfer ratio* muestra que el desempeño de los algoritmos parece ser mejor cuando las dimensiones disminuyen, en realidad los resultados son más favorables al utilizar una mayor cantidad de dimensiones, que es posible ver comparando los valores de *transfer error*.

Marco Experimental 2

Para las pruebas con 2000 dimensiones, en la Figura 5.20 se observa que no todos los algoritmos dan buenos resultados, siendo *mSDA* el que a simple vista muestra los

mejores resultados. Además, se tienen 3 tareas en donde ningún algoritmo da resultados menores a los del método *Baseline*. Por otra parte, se observa que cuando *general* es el dominio objetivo, las adaptaciones dan peores resultados.

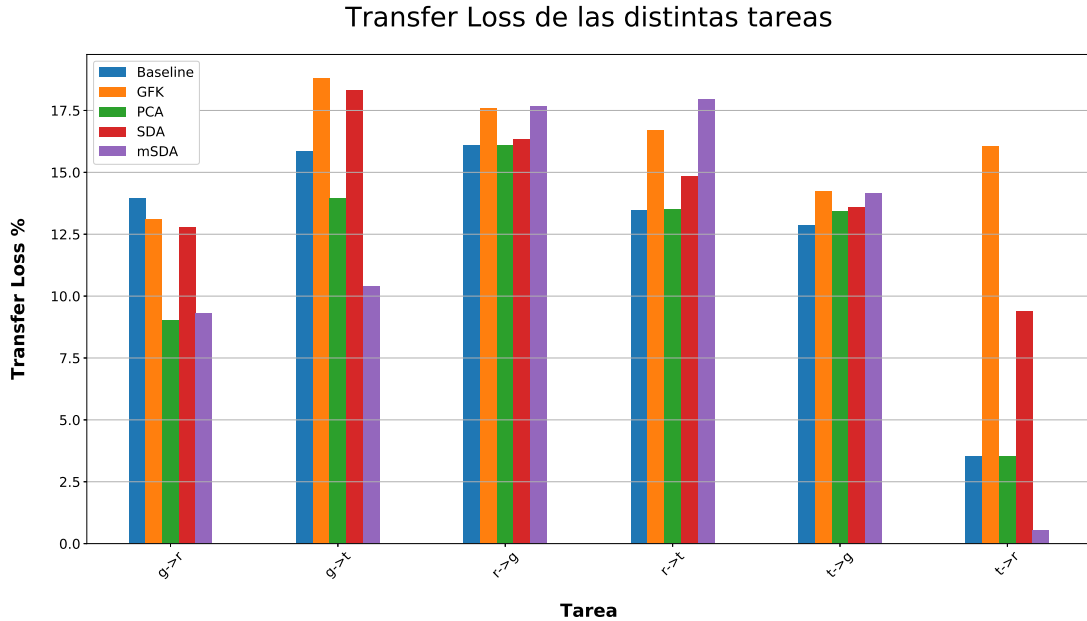


Figura 5.20: Transfer loss para el dataset Twitter utilizando 2000 dimensiones en el marco experimental 2.

Al disminuir las dimensiones a 1000 se observa en la Figura 5.21 que, en general, las adaptaciones dan mejores resultados, con excepción de los algoritmos *GFK* y *PCA*. Los valores que se obtienen son menores en la mayoría de los casos, y todas las pruebas tienen algún algoritmo que mejora las clasificaciones. Por otra parte, nuevamente se observa que cuando *general* es el dominio objetivo, las adaptaciones suelen dar peores resultados.

Al comparar el desempeño de los algoritmos, en la Tabla 5.24, se observa que los algoritmos *GFK* y *PCA* se ven afectados por la disminución de dimensiones, de manera más drástica en el primer caso, mientras que el algoritmo *SDA* mantiene la misma cantidad de adaptaciones mejoradas, y que *mSDA* obtiene mejores resultados al disminuir sus dimensiones.

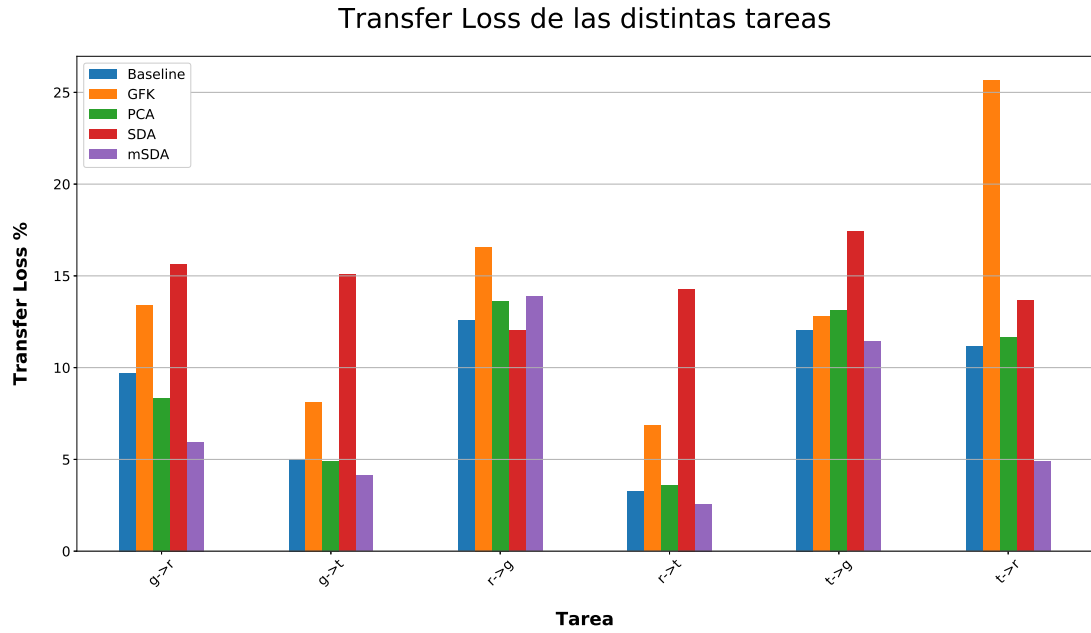


Figura 5.21: Transfer loss para el dataset Twitter utilizando 1000 dimensiones en el marco experimental 2.

Tabla 5.24: Comparación de los resultados obtenidos por cada algoritmo en comparación al método *Baseline* en el marco experimental 2.

Algoritmo	2000 Dimensiones		1000 Dimensiones	
	Clasificaciones mejoradas	Clasificaciones empeoradas	Clasificaciones mejoradas	Clasificaciones empeoradas
<i>GFK</i>	1	5	0	6
<i>PCA</i>	3	2	2	4
<i>SDA</i>	1	5	1	5
<i>mSDA</i>	3	3	5	1

Luego, viendo la cantidad de veces que cada algoritmo obtiene los mejores resultados, en la Tabla 5.25 se observa la predominancia que obtiene el algoritmo *mSDA* al disminuir la dimensiones. Además, nuevamente se observa que el algoritmo *GFK* no tiene resultados similares a los obtenidos con el *dataset Amazon*.

Tabla 5.25: Conteo de la cantidad veces que cada método obtiene los mejores resultados con el marco experimental 2.

	2000 Dimensiones	1000 Dimensiones
Algoritmo	Mejores Resultados	Mejores Resultados
<i>GFK</i>	0	0
<i>PCA</i>	2	0
<i>SDA</i>	0	1
<i>mSDA</i>	2	5

En la Figura 5.22 se observa que los promedios de *transfer ratio* mantienen la tendencia a disminuir cuando se disminuyen las dimensiones. Luego, al igual que lo ocurrido en el marco experimental 1, no todas las adaptaciones mejoran, sin embargo los valores de los algoritmos que no tienen un buen desempeño (*SDA* y *GFK* principalmente) no muestran tanta variación como los algoritmos que sí tienen un buen desempeño (*mSDA*). A pesar de esto, de la misma forma que ha ocurrido anteriormente, en la Tabla A.4 se observa que los mejores resultados para la mayoría de las pruebas se obtienen con 2000 dimensiones, en donde se obtiene que en 22 de las tareas los mejores resultados son utilizando 2000 dimensiones, mientras que solamente se obtienen 8 cuando se utilizan 1000 dimensiones. Además, en la Tabla 5.26 nuevamente se puede observar que las adaptaciones con 2000 dimensiones son las que obtienen menores valores de *transfer error*.

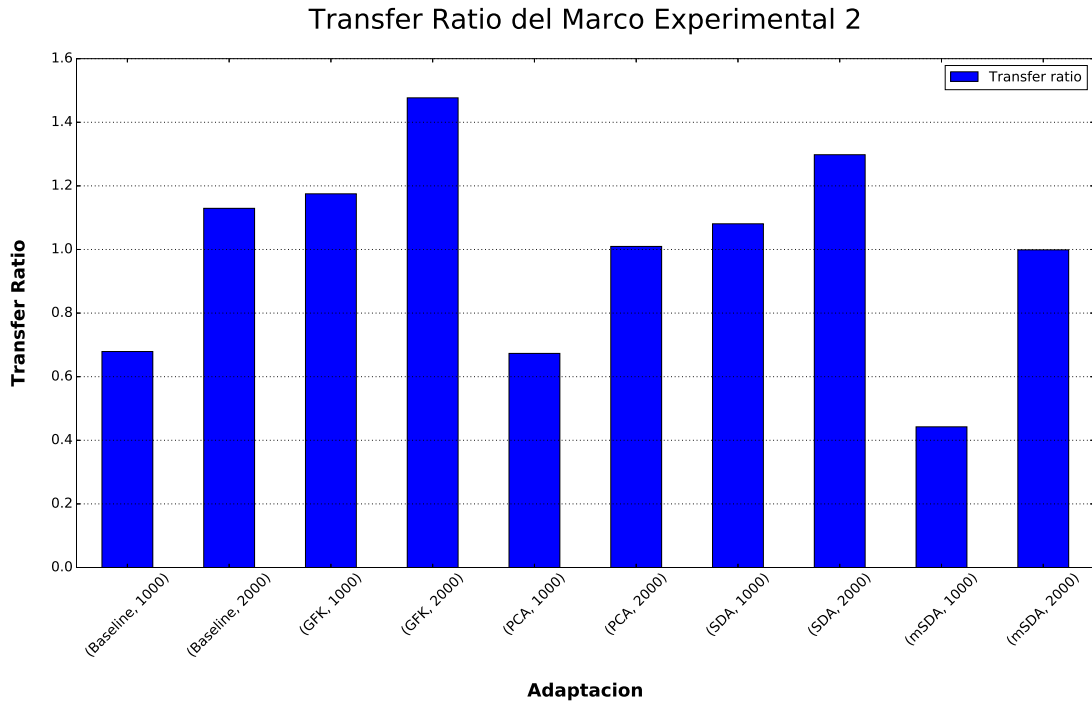


Figura 5.22: Promedio de los valores de *transfer ratio* para el *dataset Twitter* en el marco experimental 2.

Tabla 5.26: Menores valores de *transfer error* por cada tarea de adaptación con el *dataset Twitter* en el marco experimental 2.

Tarea	Algoritmo	Dimensiones	Transfer Error
g->r	mSDA	1000	13.317
g->t	mSDA	2000	17.756
r->g	SDA	1000	42.426
r->t	Baseline	2000	20.845
t->g	Baseline	2000	39.875
t->r	mSDA	2000	10.753

Comparación de Algoritmos Entre Marcos Experimentales

La comparación de los valores de *transfer loss* del algoritmo *GFK* en ambos marcos experimentales, presentados en la Figura 5.23, muestran que utilizando 2000 dimensiones la mayoría de los resultados son menores en el marco experimental 1. Luego, con 1000 dimensiones, en la Figura 5.24 se observa el mismo comportamiento, favoreciendo en mayor medida los resultados del marco experimental 1.

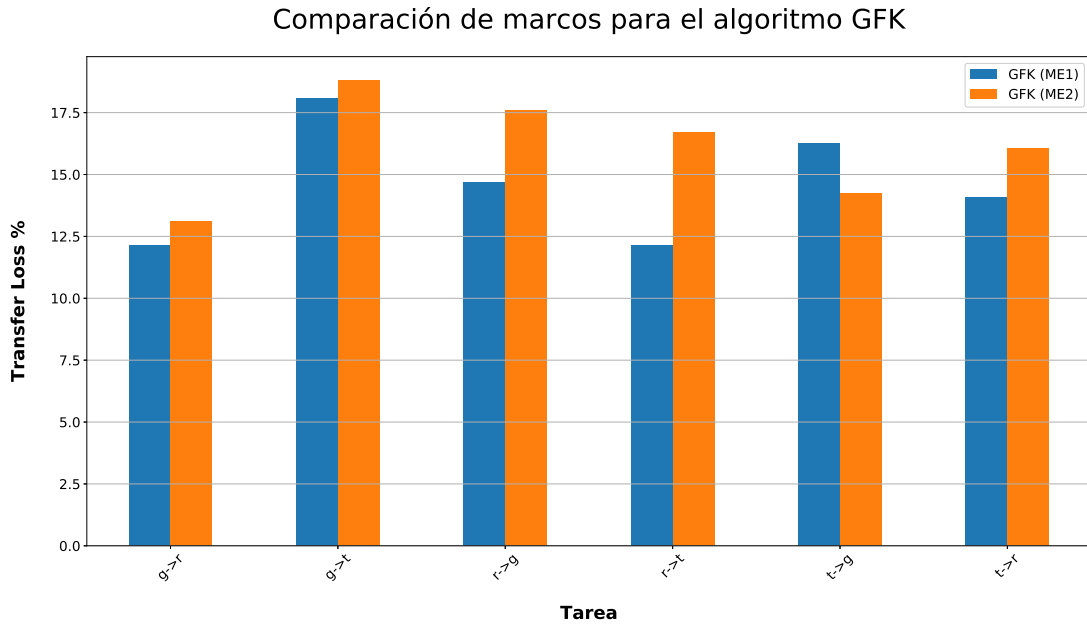


Figura 5.23: *Transfer loss* para el algoritmo *GFK* utilizando 2000 dimensiones en ambos marcos experimentales.

Las pruebas con el algoritmo *PCA*, utilizando 2000 dimensiones, muestran en la Figura 5.25 menores valores para los resultados del marco experimental 2, sin embargo, las diferencias son pocas en algunos casos, como por ejemplo las tareas " $r \rightarrow g$ " y " $t \rightarrow g$ ". Por otra parte, en la Figura 5.26 se observa que con 1000 dimensiones los valores se encuentran un poco más separados entre ambos marcos experimentales, en general, sin haber una predominancia de un marco por sobre otro.

En la Figura 5.27 se observa que el algoritmo *SDA*, al utilizar 2000 dimensiones los valores del marco experimental 1 son, en general, menores. Luego, en la Figura 5.28, utilizando 1000 dimensiones, los resultados no muestran una tendencia clara

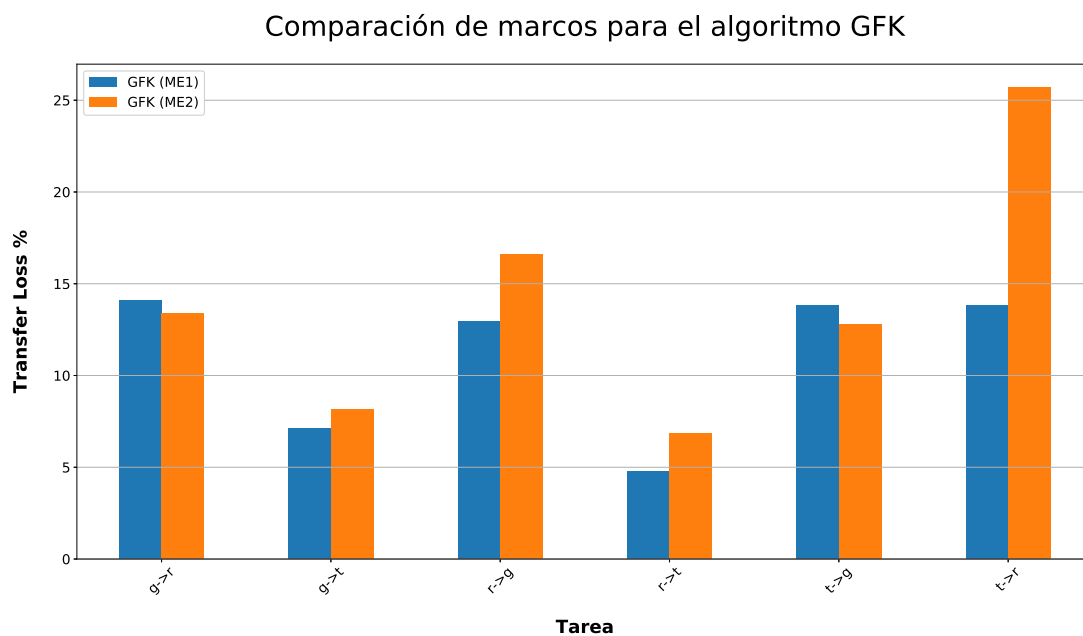


Figura 5.24: *Transfer loss* para el algoritmo GFK utilizando 1000 dimensiones en ambos marcos experimentales.

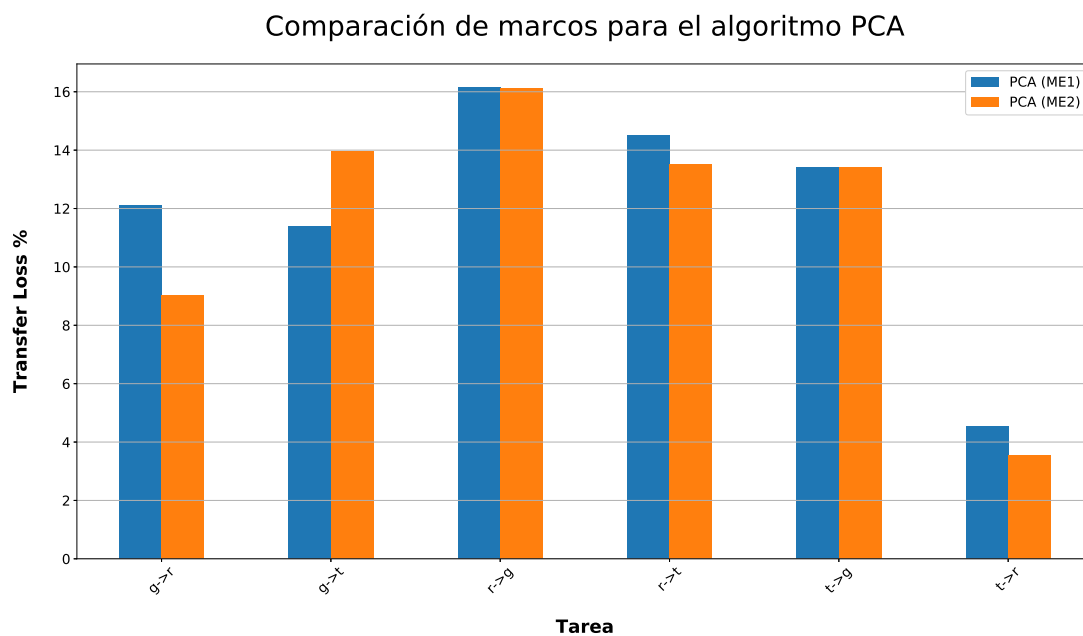


Figura 5.25: *Transfer loss* para el algoritmo PCA utilizando 2000 dimensiones en ambos marcos experimentales.

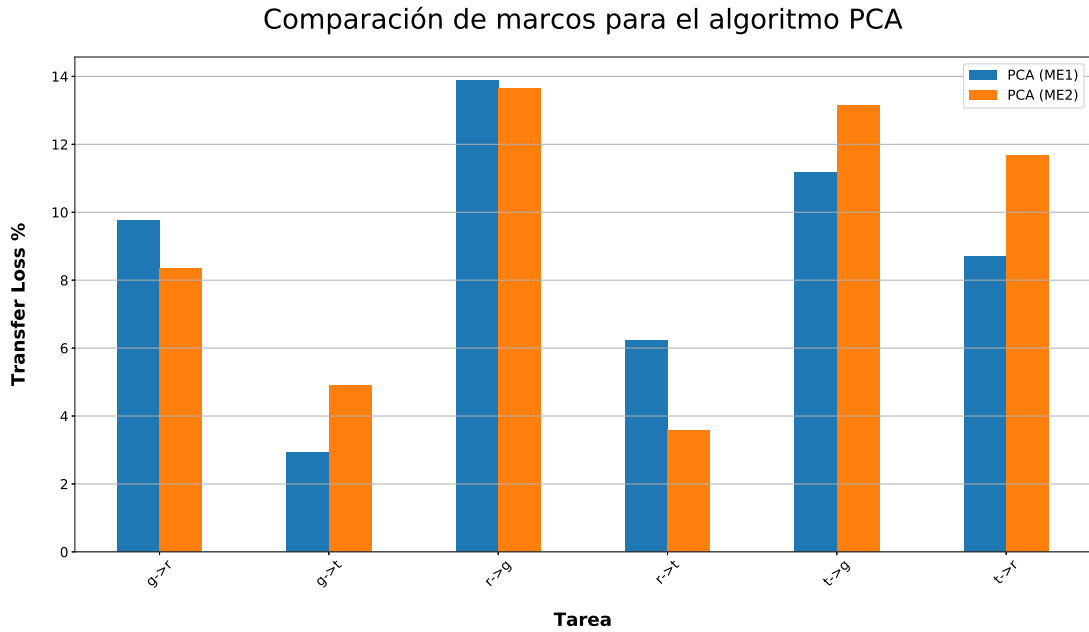


Figura 5.26: *Transfer loss* para el algoritmo PCA utilizando 1000 dimensiones en ambos marcos experimentales.

hacia alguno de los dos marcos experimentales.

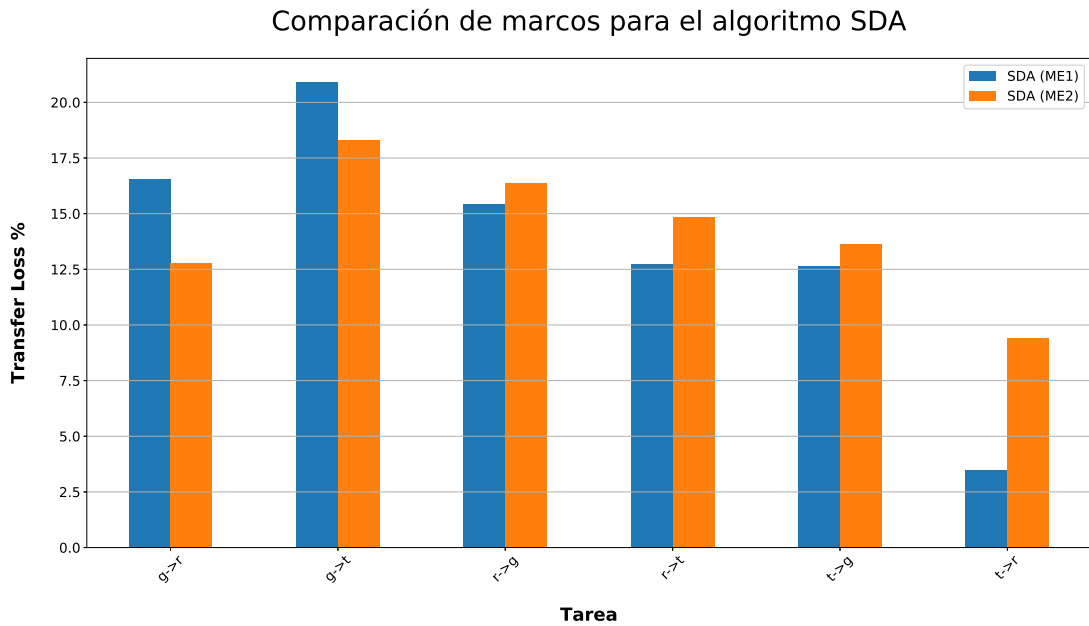


Figura 5.27: *Transfer loss* para el algoritmo SDA utilizando 2000 dimensiones en ambos marcos experimentales.

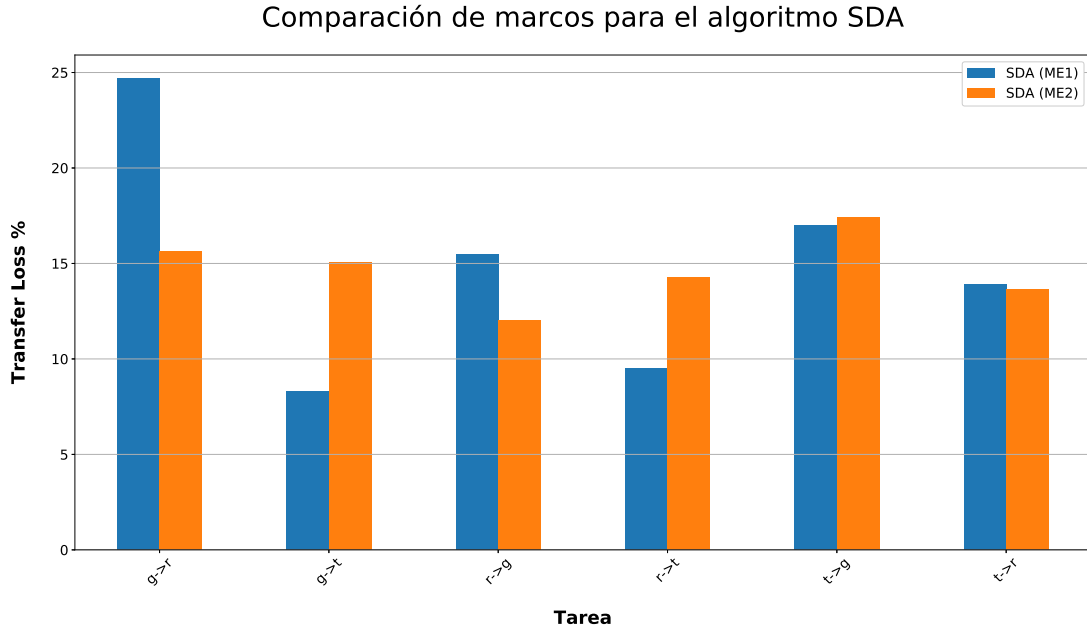


Figura 5.28: *Transfer loss* para el algoritmo SDA utilizando 1000 dimensiones en ambos marcos experimentales.

Los resultados del algoritmo *mSDA* con 2000 dimensiones, mostrados en la Figura 5.29, muestran que el marco experimental 1 obtiene la mayoría de las veces menores valores, sin haber una mayor diferencia entre estos y los del marco experimental 2. Por otra parte, en la Figura 5.30 se muestra que al utilizar 1000 dimensiones, los valores no tienen una tendencia clara, teniendo ambos marcos experimentales un desempeño similar.

En la Tabla 5.27 se observa la comparación de ambos marcos en ambas dimensiones utilizadas. Se observa que al utilizar 2000 dimensiones, los algoritmos en general tienden a tener mejores resultados con el marco experimental 1, a excepción del algoritmo *PCA*. Mientras que con 1000 dimensiones, los algoritmos tienden a tener desempeños similares en ambos marcos experimentales, con excepción del algoritmo *GFK*, que nuevamente obtiene mejores resultados con el marco experimental 1.

En la Tabla 5.28 se observan, por cada tarea y algoritmo, los mejores resultados obtenidos al utilizar 2000 dimensiones, en donde se observa una predominancia del

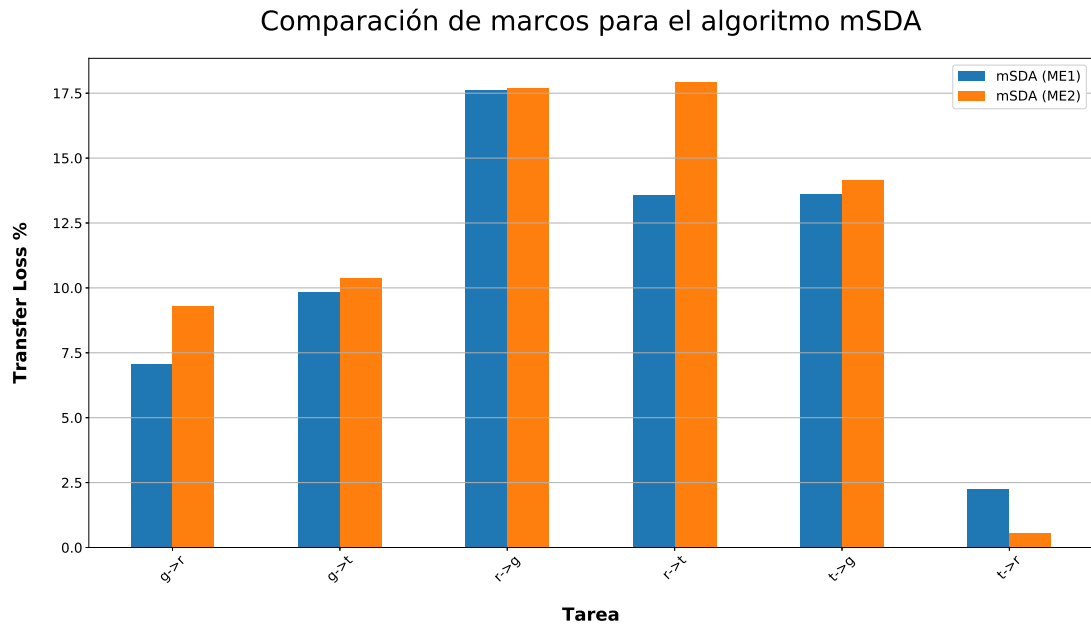


Figura 5.29: *Transfer loss* para el algoritmo mSDA utilizando 2000 dimensiones en ambos marcos experimentales.

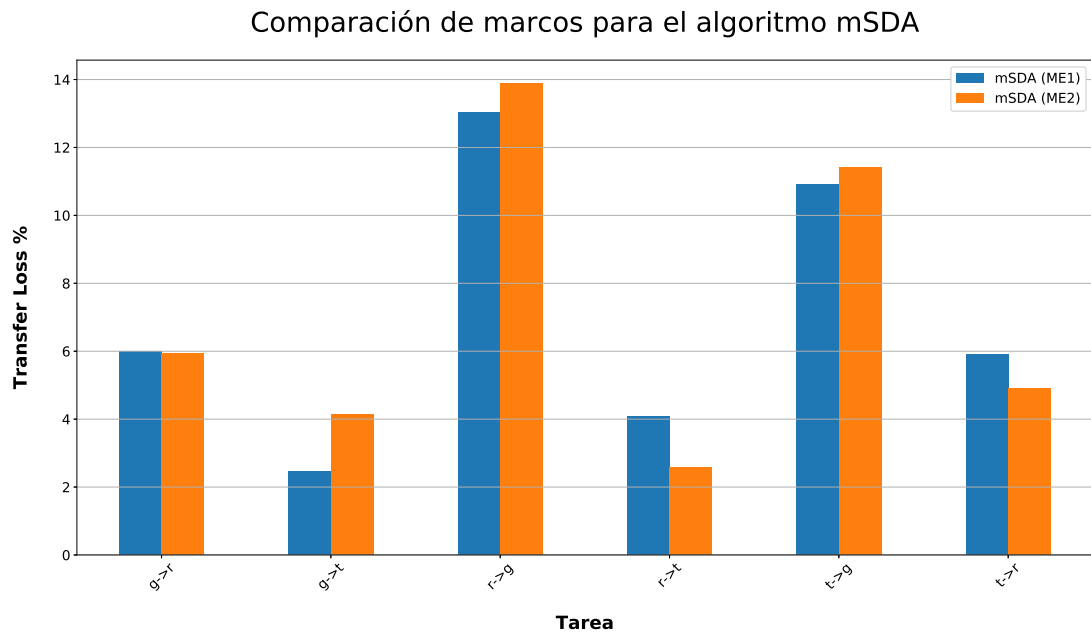


Figura 5.30: *Transfer loss* para el algoritmo mSDA utilizando 1000 dimensiones en ambos marcos experimentales.

Tabla 5.27: Comparación de los resultados obtenidos por ambos marcos experimentales en el *dataset Twitter*.

	2000 Dimensiones		1000 Dimensiones	
Algoritmo	M.E. 1	M.E. 2	M.E. 1	M.E. 2
GFK	5	1	4	2
PCA	2	4	3	3
SDA	4	2	3	3
mSDA	5	1	3	3

marco experimental 1 al obtener los menores resultados. Además se observa una tendencia marcada por algunos dominios a obtener los menores resultados. Por ejemplo: el dominio *general* tiene mejores adaptaciones cuando el dominio fuente es *thevoice*, con mayor frecuencia en el marco experimental 1, coincidiendo en el dominio esperado por la Tabla 5.20, mas no en el marco experimental. Luego, para los dominios *thevoice* y *rio2016* se espera que el dominio *general* por sí solo sea el que genere las mejores adaptaciones. Lo cual se cumple parcialmente. El dominio *rio2016* muestra mejores resultados, en general, con el dominio *thevoice*, sin un marco experimental que sobresalga en comparación al otro. Por otra parte, el dominio *thevoice* no muestra una predominancia por algún dominio individual, sin embargo, todos sus resultados son realizados bajo el marco experimental 1, por lo que la inclusión del dominio *general* se considera dentro de los datos que permiten realizar la adaptación entre dominios, cumpliendo así en cierta forma lo esperado según la Tabla 5.20.

En la Tabla 5.29, utilizando 1000 dimensiones, no se observa una predominancia por algún marco experimental. Para el dominio *general* se vuelve a observar que una mayor parte de los dominios que generan los mejores resultados provienen del uso del dominio *thevoice*, sin un marco experimental que predomine por sobre el otro. En el caso del dominio *rio2016*, este tiene resultados tanto del dominio *general* como *thevoice*, siendo todos los resultados provenientes del marco experimental 2. Del mismo modo, el dominio *thevoice* tiene resultados provenientes de los dominios *general* y *rio2016*, con la diferencia de que en este caso se tienen valores pertenecientes al marco

Tabla 5.28: Mejores resultados obtenidos por cada algoritmo para adaptar cada dominio objetivo en el *dataset Twitter* utilizando 2000 dimensiones.

Algoritmo	Objetivo	Dominio Fuente	Transfer loss
Baseline	general	thevoice	12.877
	rio2016	thevoice	3.539
	thevoice	rio2016	13.482
GFK	general	thevoice (M.E. 2)	14.259
	rio2016	general (M.E. 1)	12.160
	thevoice	rio2016 (M.E. 1)	12.145
PCA	general	thevoice (M.E. 1)	13.399
	rio2016	thevoice (M.E. 2)	3.539
	thevoice	general (M.E. 1)	11.387
SDA	general	thevoice (M.E. 1)	12.637
	rio2016	thevoice (M.E. 1)	3.494
	thevoice	rio2016 (M.E. 1)	12.713
mSDA	general	thevoice (M.E. 1)	13.610
	rio2016	thevoice (M.E. 2)	0.544
	thevoice	general (M.E. 1)	9.848

experimental 1.

Nuevamente, se tienen similitudes parciales con los valores esperados de la Tabla 5.20. Las comparaciones realizables entre los valores obtenidos y los esperados son más fáciles de realizar cuando se trata del marco experimental 2, ya que en este caso los datos del dominio fuente son utilizados tanto para la adaptación como para la selección de modelos, teniendo el dominio fuente una gran influencia en los resultados. Mientras que, por otra parte, en el marco experimental 1 realizar comparaciones respecto a los dominios esperados de la Tabla 5.20 se torna un poco más difícil, ya que en este los datos del dominio fuente son utilizados en conjunto con todos los otros dominios disponibles al momento de entrenar los modelos de adaptación, y pasan a ser utilizados por si solos al momento de la selección de modelos.

Tabla 5.29: Mejores resultados obtenidos por cada algoritmo para adaptar cada dominio objetivo en el *dataset Twitter* utilizando 1000 dimensiones.

Algoritmo	Dominio Objetivo	Dominio Fuente	Transfer loss
Baseline	general	thevoice	12.044
	rio2016	general	9.710
	thevoice	rio2016	3.243
GFK	general	thevoice (M.E. 2)	12.771
	rio2016	general (M.E. 2)	13.385
	thevoice	rio2016 (M.E. 1)	4.794
PCA	general	thevoice (M.E. 1)	11.163
	rio2016	general (M.E. 2)	8.348
	thevoice	general (M.E. 1)	2.924
SDA	general	rio2016 (M.E. 2)	12.018
	rio2016	thevoice (M.E. 2)	13.657
	thevoice	general (M.E. 1)	8.298
mSDA	general	thevoice (M.E. 1)	10.912
	rio2016	thevoice (M.E. 2)	4.900
	thevoice	general (M.E. 1)	2.450

En la Tabla 5.30 se observan los mejores resultados en ambos marcos experimentales utilizando 2000 dimensiones. En donde se observa que el marco experimental 1 obtiene 5 veces los mejores resultados, mientras que el marco experimental 2 obtiene solamente 1 vez el mejor resultado. Por otra parte en la Tabla 5.31 se observa que con 1000 dimensiones, el marco experimental 4 obtiene 4 veces los mejores resultados, mientras que el marco experimental 1 obtiene 2 veces los mejores resultados.

Finalmente, en la Tabla 5.32 se observa que con 2000 dimensiones, la mayoría de los mejores resultados son obtenidos al utilizar el algoritmo *mSDA*, seguido por *GFK* y *SDA*. Mientras que con 1000 dimensiones, nuevamente el algoritmo *mSDA* obtiene la mayoría de los resultados, seguido por *SDA*.

Tabla 5.30: Mejores adaptaciones obtenidas en ambos marcos experimentales, utilizando 2000 dimensiones con el *dataset Twitter*.

Tarea	Algoritmo	Marco	Transfer loss
$g \rightarrow r$	mSDA	M.E. 1	7.078
$g \rightarrow t$	mSDA	M.E. 1	9.848
$r \rightarrow g$	GFK	M.E. 1	14.678
$r \rightarrow t$	GFK	M.E. 1	12.145
$t \rightarrow g$	SDA	M.E. 1	12.637
$t \rightarrow r$	mSDA	M.E. 2	0.544

Tabla 5.31: Mejores adaptaciones obtenidas en ambos marcos experimentales, utilizando 1000 dimensiones con el *dataset Twitter*.

Tarea	Algoritmo	Marco	Transfer loss
$g \rightarrow r$	mSDA	M.E. 2	5.944
$g \rightarrow t$	mSDA	M.E. 1	2.450
$r \rightarrow g$	SDA	M.E. 2	12.018
$r \rightarrow t$	mSDA	M.E. 2	2.569
$t \rightarrow g$	mSDA	M.E. 1	10.912
$t \rightarrow r$	mSDA	M.E. 2	4.900

Tabla 5.32: Conteo de la cantidad veces que cada método obtiene los mejores resultados en ambos marcos experimentales en el *dataset Twitter*.

	2000 Dimensiones	1000 Dimensiones
Algoritmo	Mejores Resultados	Mejores Resultados
<i>GFK</i>	2	0
<i>PCA</i>	0	0
<i>SDA</i>	1	1
<i>mSDA</i>	3	5

5.2. Pruebas de Tiempo

Estas pruebas muestran, en minutos, el tiempo que demoran en ejecutar los algoritmos. A través del eje de las abscisas se distribuyen los distintos algoritmos, ordenados de izquierda a derecha según el tiempo tomado, y diferenciados por color según el algoritmo. Esto es realizado sin ningún fin en particular más que el de entregar la información de una forma más comprensible.

Marco Experimental 1

Los resultados observados en la Figura 5.31 muestran una clara división entre los algoritmos. Primero, se observa que *SDA* es el algoritmo que en general demora más tiempo, superando en varios casos los 15 minutos. Luego, con una cantidad menor de tiempo, se encuentra el algoritmo *mSDA*, que en general demora menos de 15 minutos de entrenamiento. Además, como es de esperar, en ambos algoritmos se tiene que el tiempo demorado aumenta a medida que aumenta el número de capas utilizadas. En el caso de *SDA*, la cantidad de *epochs* realizados también es un factor importante a considerar, en donde por ejemplo se ve que un modelo entrenado con 50 epochs puede demorar más tiempo que otro modelo entrenado con 25 epochs, aun teniendo una cantidad mayor de capas.

Por otra parte, se tiene que los algoritmos que toman menos tiempo son *PCA* y

GFK, existiendo poca diferencia entre estos dos algoritmos, y además mostrando pocos cambios al variar sus parámetros, en comparación a los obtenidos por *SDA* y *mSDA*.

Tiempos de entrenamiento en el Marco Experimental 1

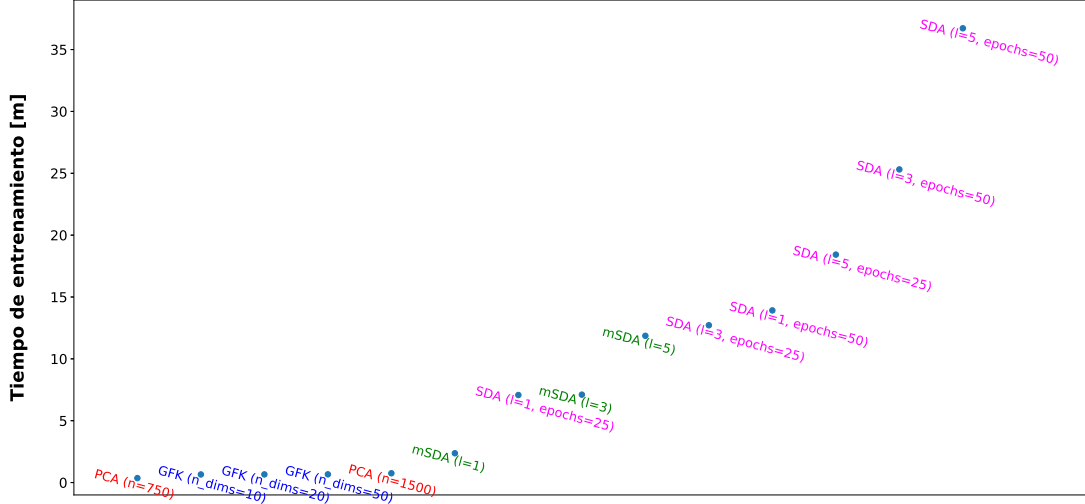


Figura 5.31: Tiempos de ejecución para el dataset Amazon utilizando 3000 dimensiones en el marco experimental 1.

Marco Experimental 2

Los tiempos obtenidos al utilizar el marco experimental 2, mostrados en la Figura 5.32, no muestran mayor variación en comparación a los resultados anteriores, con excepción del algoritmo *SDA*, en donde los tiempos obtenidos por este algoritmo son menores que los que obtiene *mSDA*, y además la totalidad de sus tiempos demoran menos de 5 minutos, en comparación a los más de 15 que obtuvo en las pruebas anteriores. Es necesario recordar que la gran diferencia con las pruebas anteriores es la utilización de los datos correspondientes a un par de dominios, en comparación a las pruebas anteriores, donde se utiliza la totalidad de los datos del *dataset*. Por otra parte, tal como fue mencionado anteriormente, el resto de los algoritmos no ven una variación significativa al utilizar menos datos.

Tiempos de entrenamiento en el Marco Experimental 2

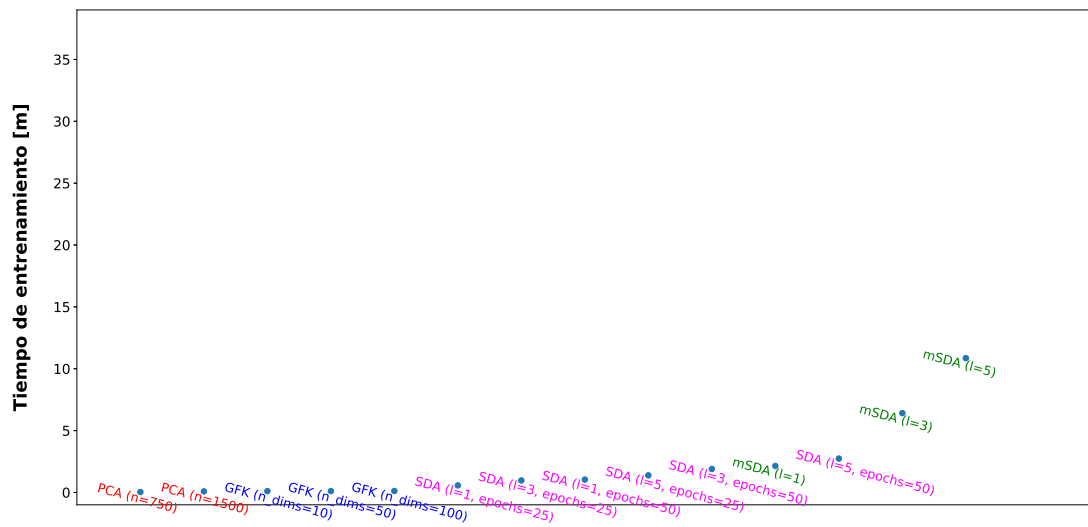


Figura 5.32: Tiempos de ejecución para el dataset Amazon utilizando 3000 dimensiones en el marco experimental 2.

Conclusiones

En este capítulo se muestran las conclusiones, tanto generales como específicas, obtenidas durante el desarrollo de este trabajo, consistente de la investigación de algoritmos de adaptación entre dominios, la implementación de algoritmos, la instalación y configuración de un entorno de trabajo, y la prueba de un par de *datasets* con los algoritmos implementados. Finalmente, se presentan opciones de trabajo futuro a realizar.

Conclusiones Generales

Los objetivos principales de este trabajo son la adaptación entre dominios aplicada a problemas de clasificación de texto, y los procesos que involucra la realización de esta tarea. O mejor dicho, las etapas de investigación de distintos métodos de adaptación entre dominios, la implementación de algoritmos, la configuración de un marco de trabajo y la prueba de una selección de estos algoritmos.

Para cumplir los objetivos propuestos, se realizó un estudio de cinco publicaciones relacionadas a *transfer learning*, o adaptación entre dominios, de las cuales se seleccionaron 3, debido a su relevancia, la calidad de sus resultados, y la facilidad que tienen sus propuestas para ser implementadas. Además, se realizó la implementación de un cuarto algoritmo, que si bien no es propuesta en una publicación, se utiliza para realizar comparaciones en varias de las publicaciones estudiadas. Los algoritmos implementados son conocidos bajo los nombres *SDA*, *mSDA*, *GFK* y *PCA*, y cómo algoritmo base de comparación también se agrega un método definido como *Baseline*.

Las pruebas a ejecutar se realizan utilizando el lenguaje *Python*, utilizando librerías que permiten el uso directo de algunos de los algoritmos o la implementación de estos en el caso contrario. Además, se configura un entorno de trabajo que permite la utilización de *datasets* por parte de los algoritmos, y un entorno virtual en la nube, utilizando *Amazon Web Services*, que permite la ejecución de las pruebas de forma replicable.

Para la realización de las pruebas se establecieron 2 tipos: de adaptación y de tiempo. Las pruebas de adaptación miden la calidad de los algoritmos de adaptación, utilizando las métricas de *transfer loss*, *transfer ratio* y *transfer error*, bajo distintas configuraciones de los datos disponibles, que se diferencian en la cantidad de datos que se utilizan para el entrenamiento de modelos de adaptación. Las pruebas de adaptación se realizan dentro de los llamados marcos experimentales 1 y 2, en donde el primero utiliza todos los datos de entrenamiento de los dominios disponibles para realizar las adaptaciones, mientras que el segundo utiliza solamente los datos correspondientes al par de dominios que participan en la adaptación. Luego, las pruebas de tiempo consideran solamente el tiempo que requiere entrenar un algoritmo, sin contar con las fases de selección de modelos, adaptación de los datos y entrenamiento de nuevos clasificadores.

Las pruebas se ejecutaron en 2 *datasets* distintos, el primero corresponde al *dataset Amazon*, que es utilizado como *benchmark* por otros algoritmos de adaptación entre dominios, mientras que el segundo *dataset* corresponde al *dataset Twitter*, que fue creado manualmente para su uso en este trabajo y consiste de una recopilación de *tweets* etiquetados sobre diversos temas. Luego, por cada *dataset* se realizan las pruebas de adaptación en los dos marcos experimentales descritos, utilizando dos dimensionalidades distintas por *dataset* y variando los parámetros de cada algoritmo. Además, las pruebas de tiempo se realizan solamente con el *dataset Amazon* utilizando 3000 dimensiones.

Además de lo anterior, se analizaron 2 medidas, que sirven para tener un comportamiento a esperar de los algoritmos. Estas son el *baseline in-domain error* y la métrica *proxy a-distance*. El primero es el error obtenido al entrenar y probar un clasificador

utilizando solamente los datos del dominio objetivo, el cuál varía al utilizar distintas dimensiones. La segunda medida es una estimación de las distancias, o similitudes, existentes entre los dominios, que sirven para tener una idea de qué adaptaciones van a dar mejores resultados.

Las pruebas ejecutadas mostraron que los algoritmos tienen comportamientos similares a los mostrados en sus respectivas publicaciones (para los casos en donde es más posible realizar una comparación). La tendencia general observada por los algoritmos, es a realizar mejores adaptaciones cuando se utiliza una mayor cantidad de dimensiones y se utilizan todos los datos de entrenamiento disponibles de un *dataset* para realizar las adaptaciones.

Con lo anterior, se cumplen los objetivos generales planteados al inicio de este trabajo. Se ha realizado un estudio de las propuestas más relevantes de técnicas relacionadas a la adaptación entre dominios, se ha probado experimentalmente una selección de estas propuestas, asegurando su correcta implementación y teniendo conocimiento de su desempeño. Y finalmente, en conjunto con la implementación, se ha creado un entorno capaz de expandirse a nuevas investigaciones y propuestas, contando con las herramientas necesarias y 2 conjuntos de datos con los cuales trabajar.

Conclusiones Específicas

Definición de adaptación entre dominios

La definición de adaptación entre dominios partió bajo la suposición de que esta forma parte de un conjunto de técnicas agrupadas bajo el nombre de *transfer learning*. De *transfer learning* se estudiaron las clasificaciones de las técnicas y formas de resolver los problemas que engloba esta disciplina. Las clasificaciones encontradas se dan de acuerdo a la disponibilidad de datos en los dominios fuente y objetivo, dentro de las cuales se encuentra *transductive transfer learning*, en donde se considera la existencia de datos etiquetados en el dominio fuente, pero no se considera la existencia de

etiquetas en los datos del dominio objetivo.

Dentro de las técnicas posibles que caen en la categoría de *transductive transfer learning* se encuentra la adaptación entre dominios. Sin embargo, las restricciones que se encontraron parecen limitar mucho la cantidad de soluciones aplicables y también reduce los problemas de interés a resolver, por lo que, para evitar esto, se adopta la definición de adaptación entre dominios como el mismo concepto de *transfer learning*.

Investigación de técnicas de adaptación entre dominios

La investigación de técnicas relacionadas a la adaptación entre dominios consistió en el estudio de 5 publicaciones, en donde en algunas de estas se referían al problema como adaptación entre dominios o como *cross-domain sentiment classification*. Las publicaciones estudiadas proponían los métodos *SFA*, *SDA*, *SGF*, *GFK* y *mSDA*, además de utilizar como comparación el algoritmo *PCA*, en donde se aprendió que algunos de estos algoritmos fueron específicamente diseñados para resolver el problema de adaptación entre dominios, tales como *SFA*, *SGF*, *GFK* y *mSDA*, mientras que *SDA* y *PCA* eran algoritmos ya existentes y que, debido a la utilidad que tienen en el ámbito de *machine learning*, fueron utilizados para resolver el problema de interés.

De las técnicas estudiadas se escogió un conjunto de estas, debido a los resultados mostrados y a la disponibilidad de código disponible de estas, en algunos casos, o bien, la facilidad con la cual se podrían implementar. Las técnicas escogidas fueron *SDA*, *PCA* y *mSDA*, *GFK*. En donde las tres primeras utilizaban el *dataset Amazon* como *benchmark*.

Configuración del entorno de trabajo e implementación

Para implementar los algoritmos, fue necesario establecer *Python* como el lenguaje en donde se ejecutarían los algoritmos y pruebas. Esto conllevó la creación de estructuras de datos y clases con las cuales los *datasets* pudieran ser almacenados y

utilizados por los distintos programas. Además, se realizó la instalación del entorno *Anaconda*, para facilitar la instalación de paquetes concernientes a la manipulación de datos y aprendizaje de máquinas, en conjunto con *Jupyter Notebooks*, para permitir una codificación más interactiva y que permitiera un mejor desarrollo y transmisión de conceptos utilizando textos altamente descriptivos.

Luego, dado que la ejecución de las pruebas ocupa muchos recursos computacionales, y que la replicación de los resultados también dependen en cierta medida del computador en donde se ejecutan, se utilizó el servicio de virtualización en la nube de *Amazon*, conocido como *Elastic Compute Cloud*, del cual se escogió un tipo de instancia que tuviera la capacidad de cómputo y almacenamiento suficiente para ejecutar las pruebas sin aumentar demasiado el costo monetario.

La instalación del entorno de trabajo tuvo poca dificultad debido a las herramientas que provee *Anaconda* para la instalación de paquetes, mientras que el uso de entornos virtuales (también provisto por *Anaconda*) facilitó la replicación del entorno en la máquina virtual en la nube.

La implementación de los algoritmos tuvo una dificultad media de realizar. Para los algoritmos *GFK* y *PCA* ya existían implementaciones realizadas, de parte de las librerías *Bob* y *scikit-learn* respectivamente. Mientras que los algoritmos *SDA* y *mSDA* tuvieron que ser implementados utilizando las librerías *Keras* y *Theano* respectivamente. Para el uso de estas librerías es necesario un entendimiento un poco más profundo de su funcionamiento, ya que las instrucciones escritas no son ejecutadas directamente, como suele suceder normalmente en *Python*, sino que deben ser compiladas para luego ser ejecutadas, además de entender que la asignación de los datos a las distintas variables se realiza de una forma diferente.

Resultados de las adaptaciones

Al realizar los dos tipos de pruebas sobre ambos *datasets*, utilizando las dos dimensiones posibles por cada *dataset*. Se observa un comportamiento mayormente

constante en general, y en algunos casos también por parte de algunos algoritmos. Una vez que las pruebas fueron ejecutadas, se obtuvieron las siguientes conclusiones:

- La implementación de los algoritmos se ha realizado correctamente, al obtener resultados positivos, en varios casos, de parte de todos los algoritmos. Además, varios de estos se comportan de acuerdo a lo presentado en sus respectivas publicaciones, considerando que se utilizaron las mismas configuraciones de parámetros recomendadas en distintas condiciones experimentales.
- El desempeño de los algoritmos se ha mantenido sin muchas variaciones al ser probados estos en el *dataset Twitter*, permitiendo asegurar el uso de algunos de estos algoritmos en otros *datasets* similares.
- La disminución de dimensiones en los *datasets* da la impresión de mejorar el desempeño general de los algoritmos, lo cual se pudo observar utilizando la métrica de *transfer ratio*. Sin embargo, al observar el error final de las clasificaciones luego de haber realizado las adaptaciones (*transfer error*), se obtiene que las clasificaciones son mejores en la mayoría de los casos cuando se utilizan más dimensiones.
- El punto anterior también permite dejar establecido que hay que tener precaución al utilizar la métrica de *transfer ratio*.
- La métrica *proxy α -distance* ha demostrado ser confiable para saber de antemano qué dominios serían más aptos para realizar una adaptación entre dominios o para la selección de modelos.
- En general, los algoritmos implementados que mejor desempeños obtienen en todas las pruebas son *mSDA* y *GFK*. Teniendo en menor medida también buenos desempeños el método *PCA*.
- En las pruebas realizadas con ambos marcos experimentales, se obtiene que el marco experimental 1 da en general mejores resultados. Lo cual se comprueba

comparando la cantidad de veces que cada marco experimental obtiene los mejores resultados. Sin embargo, al comparar las diferencias entre estos resultados, no se encuentran diferencias abismales entre ambos.

Al medir los tiempos de los algoritmos al ser ejecutados en el *dataset Amazon*, se obtuvo que la mayoría de ellos presentan variaciones al cambiar la cantidad de datos con los cuales son entrenados. La cantidad de datos que se utilizan en el marco experimental 1 es al menos el doble de la que se utiliza en el marco experimental 2, por lo que es de esperar los cambios en los tiempos obtenidos, tal como ocurrió en el algoritmo *SDA*, y en menor medida en los algoritmos *PCA*, *GFK* y *mSDA*. El algoritmo *mSDA* no presenta una gran variación respecto a la cantidad de datos utilizados, solamente presentado cambios a medida que se aumenta la cantidad de capas utilizadas. De una forma similar, el algoritmo *SDA* presenta grandes aumentos de tiempo a medida que se aumenta la cantidad de capas, *epochs* y datos. Por otra parte, el algoritmo *PCA* presenta cambios menores al aumentar la cantidad de dimensiones a las cuales se reducen los datos, mientras que el algoritmo *GFK* presenta cambios menores de tiempo al variar la cantidad de sub-espacios utilizados.

Lo que se puede concluir de los desempeños individuales de cada algoritmo es:

- ***GFK***: da mejores resultados cuando se utiliza en el marco experimental 1, es decir, cuando se utiliza con datos de varios dominios como dominio fuente, sin presentar una disminución significativa en sus resultados cuando se utilizan solamente los datos como se realiza en el marco experimental 2. Obtiene mejores resultados cuando se tiene una mayor cantidad de dimensiones. Los tiempos obtenidos por este algoritmo se encuentran entre los menores, y la variación de estos tiempo es poca al variar la cantidad de dimensiones.
- ***mSDA***: tiene un comportamiento similar al de *GFK*, teniendo buenos resultados en el marco experimental 1, sin decaer mucho en el marco experimental 2. Obtiene mejores resultados cuando se tienen pocas dimensiones, obteniendo mejor desempeño que *GFK*. Así mismo, sus tiempos de entrenamiento son bajos, aunque pueden aumentar a medida que se agregan más capas para entrenar.

Por otra parte, los resultados obtenidos coinciden con los esperados al realizar la investigación de este algoritmo.

- **SDA**: sus mejores resultados no tienen un marco en específico que los genere, por otra parte, los resultados que muestra en general no son los mejores entre los que se obtienen de todos los algoritmos. Una causa atribuible a esto es la implementación, que a pesar de funcionar correctamente, puede diferir de la realizada por sus autores. Además, los tiempos de entrenamiento obtenidos experimentalmente difieren demasiado de los expuestos en su publicación, por lo que esto también puede considerarse una causa. Sin embargo, dados los resultados obtenidos por *GFK* y *mSDA*, que superan a los de *SDA*, tanto en la calidad de las adaptaciones como en el tiempo, no se encuentra necesario dedicar más tiempo a entrenar este algoritmo por varias horas.
- **PCA**: su desempeño es irregular en ambos marcos experimentales, sin presentar comportamientos que se puedan considerar constantes al utilizarse en distintos *datasets* o con distintas dimensionalidades, aunque en general su desempeño es levemente mejor que el que obtiene *SDA*. En términos de tiempo, es el algoritmo que menos tiempo demora en ser entrenado. Los resultados obtenidos en general por este algoritmo no son sorprendidos, dado que es un algoritmo cuyo uso original es en tareas de reducción de dimensionalidad y que fue utilizado para la adaptación entre dominios por la similitud que presenta con *SDA*.

En general, se encuentra conveniente utilizar los algoritmos *GFK* y *mSDA* para realizar adaptaciones, debido a los desempeños que obtienen tanto al adaptar, como en los bajos tiempos que toman para entrenar sus respectivos modelos. Además de la baja cantidad de parámetros que se deben ajustar para entrenar cada algoritmo. También, fue posible observar cómo los algoritmos que fueron específicamente diseñados para realizar adaptación entre dominios (*GFK* y *mSDA*), tienen mejores desempeños que los que ya existían y se les dio un uso en esta disciplina (*SDA* y *PCA*). Luego, también se recomienda utilizar una configuración similar a la del experimental 1, es decir, utilizando la mayor cantidad de datos posibles existentes en un *dataset*, en conjunto con el uso de la métrica *PAD* para saber qué dominio es conveniente utilizar para la

selección de modelos. En caso de tener opciones disponibles sobre las dimensiones a utilizar, se prefiere no utilizar dimensiones muy bajas, sino que una cantidad moderada, tal que asegure bajos errores finales de clasificación y tiempos no muy largos de entrenamiento de los algoritmos a utilizar.

Trabajo a Futuro

Los algoritmos implementados han dado buenos resultados. Sin embargo, una de las limitantes que se encontraba en algunos casos, era el tiempo que toman en entrenar, limitando con esto la cantidad de dimensiones a utilizar o los parámetros a probar en algunos algoritmos, como *mSDA* por ejemplo. Es entonces donde una opción a realizar a futuro es el uso de procesamiento por la tarjeta gráfica (*gpu*) para entrenar algunos algoritmos, lo cual disminuiría considerablemente el tiempo de ejecución de los algoritmos.

Por otra parte, se podría expandir el uso de los algoritmos a otras tareas, tales como la adaptación entre dominios para la clasificación de imágenes, ya que, es necesario recordar que el problema inicial a resolver de *GFK* era este, y su uso para clasificación de texto dio buenos resultados, por lo que el uso de otros algoritmos que solo han sido utilizados en clasificación de texto, como *mSDA* por ejemplo, sería una buena alternativa a explorar.

Las pruebas en general dieron buenos resultados, sin embargo, estas estuvieron limitadas a la disponibilidad de datos etiquetados en el dominio fuente para realizar la selección de dominios, por lo que una mejora en los resultados podría darse teniendo una cantidad de datos etiquetados en el dominio objetivo, con los cuales poder realizar una selección de modelos de adaptación basada en datos más cercanos a los que se utilizan realmente en las pruebas de los modelos.

Finalmente, dada la investigación realizada y el conocimiento acumulado sobre la adaptación entre dominios, una opción de trabajo a futuro es la propuesta de un algoritmo. Dentro de las técnicas estudiadas, se encuentra *SDA*, la cual utiliza un algoritmo de

deep learning ya existente para resolver el problema de adaptación entre dominios, por lo que podrían existir otros algoritmos de *deep learning* que permitan realizar buenas adaptaciones.

Bibliografía

- [1] Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. Analysis of representations for domain adaptation. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS'06, pages 137–144, Cambridge, MA, USA, 2006. MIT Press.
- [2] John Blitzer, Mark Dredze, Fernando Pereira, et al. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *ACL*, volume 7, pages 440–447, 2007.
- [3] John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 conference on empirical methods in natural language processing*, pages 120–128. Association for Computational Linguistics, 2006.
- [4] Minmin Chen, Kilian Q Weinberger, and John Blitzer. Co-training for domain adaptation. In *Advances in neural information processing systems*, pages 2456–2464, 2011.
- [5] Minmin Chen, Zhixiang Xu, Kilian Weinberger, and Fei Sha. Marginalized denoising autoencoders for domain adaptation. *arXiv preprint arXiv:1206.4683*, 2012.
- [6] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 513–520, 2011.
- [7] Boqing Gong, Yuan Shi, Fei Sha, and Kristen Grauman. Geodesic flow kernel for unsupervised domain adaptation. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2066–2073. IEEE, 2012.
- [8] Raghuraman Gopalan, Ruonan Li, and Rama Chellappa. Domain adaptation for object recognition: An unsupervised approach. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 999–1006. IEEE, 2011.

- [9] Gregory Griffin, Alex Holub, and Pietro Perona. Caltech-256 object category dataset. 2007.
- [10] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting change in data streams. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 180–191. VLDB Endowment, 2004.
- [11] Tom M Mitchell. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 1:2, 1997.
- [12] Sinno Jialin Pan, Xiaochuan Ni, Jian-Tao Sun, Qiang Yang, and Zheng Chen. Cross-domain sentiment classification via spectral feature alignment. In *Proceedings of the 19th international conference on World wide web*, pages 751–760. ACM, 2010.
- [13] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *Knowledge and Data Engineering, IEEE Transactions on*, 22(10):1345–1359, 2010.
- [14] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In *European conference on computer vision*, pages 213–226. Springer, 2010.

Apéndice A

Tablas

Tabla A.1: Comparación de los valores de *transfer error* entre las distintas dimensiones utilizadas en el *dataset Amazon* en el marco experimental 1.

		Transfer error											
Adaptacion	Dimensiones	b->d	b->e	b->k	d->b	d->e	d->k	e->b	e->d	e->k	k->b	k->d	k->e
Baseline	1000	19.328	26.591	25.441	25.893	27.041	27.133	30.493	26.193	16.583	30.798	27.081	19.720
	3000	18.470	19.085	19.700	25.808	25.263	18.083	28.493	27.473	10.323	30.571	26.113	12.625
GFK	1000	20.236	25.428	17.875	24.571	22.753	16.265	29.571	27.058	11.083	30.053	27.448	14.745
	3000	18.765	23.748	15.825	24.518	18.610	13.233	25.273	25.478	10.890	31.618	26.888	14.045
PCA	1000	17.528	23.841	24.291	25.308	27.056	24.011	30.886	27.166	15.350	32.606	27.498	18.510
	3000	22.448	24.941	22.138	27.326	26.251	18.938	30.591	29.048	10.595	31.228	27.961	13.765
SDA	1000	23.466	29.218	25.318	26.138	29.633	26.131	34.198	31.621	17.823	30.246	28.636	16.375
	3000	20.848	28.381	24.703	28.536	27.738	20.323	29.818	28.316	13.265	36.361	30.638	15.565
mSDA	1000	17.375	21.913	21.186	22.043	21.743	18.723	29.178	23.216	12.018	28.361	24.141	14.613
	3000	20.343	23.143	20.248	21.901	24.348	18.143	25.921	25.671	9.610	29.001	25.378	12.200

Tabla A.2: Comparación de los valores de *transfer error* entre las distintas dimensiones utilizadas en el *dataset Amazon* en el marco experimental 2.

		Transfer error											
Adaptacion	Dimensiones	b->d	b->e	b->k	d->b	d->e	d->k	e->b	e->d	e->k	k->b	k->d	k->e
Baseline	1000	19.328	26.591	25.441	25.893	27.041	27.133	30.493	26.193	16.583	30.798	27.081	19.720
	3000	18.470	19.085	19.700	25.808	25.263	18.083	28.493	27.473	10.323	30.571	26.113	12.625
GFK	1000	21.738	25.716	22.866	26.541	23.151	18.828	35.066	29.411	17.605	33.626	28.581	18.348
	3000	17.603	22.501	16.083	23.858	22.468	13.340	28.463	24.378	10.793	31.806	26.263	15.070
PCA	1000	18.958	26.216	28.188	23.231	28.853	24.558	31.911	26.841	14.763	32.783	27.788	17.910
	3000	18.355	20.246	21.768	26.376	25.291	20.176	29.431	27.386	10.370	30.208	27.106	12.938
SDA	1000	21.331	29.933	22.086	25.433	28.241	26.803	32.643	30.151	16.315	33.008	29.708	19.210
	3000	20.953	24.971	22.263	24.283	26.318	21.986	29.983	29.096	11.925	28.598	30.738	16.573
mSDA	1000	17.338	23.091	19.333	21.678	23.073	24.276	29.178	24.751	14.403	31.536	27.791	16.813
	3000	19.795	21.523	22.166	20.513	28.738	25.248	28.721	26.778	11.743	29.443	28.976	12.793

Tabla A.3: Comparación de los valores de *transfer error* entre las distintas dimensiones utilizadas en el *dataset Twitter* en el marco experimental 1.

		Transfer error					
Adaptacion	Dimensiones	g->r	g->t	r->g	r->t	t->g	t->r
Baseline	1000	17.083	24.077	42.989	22.289	42.452	18.534
	2000	24.183	23.236	43.116	20.845	39.875	13.748
GFK	1000	21.438	26.136	43.375	23.840	44.225	21.211
	2000	22.368	25.426	41.676	19.508	43.243	24.274
PCA	1000	17.128	21.970	44.284	25.260	41.571	16.084
	2000	22.323	18.750	43.144	21.875	40.397	14.746
SDA	1000	32.055	27.344	45.876	28.527	47.377	21.302
	2000	26.770	28.291	42.418	20.076	39.635	13.702
mSDA	1000	13.362	21.496	43.451	23.130	41.320	13.271
	2000	17.287	17.211	44.610	20.952	40.609	12.477

Tabla A.4: Comparación de los valores de *transfer error* entre las distintas dimensiones utilizadas en el *dataset Twitter* en el marco experimental 2.

		Transfer error					
		g->r	g->t	r->g	r->t	t->g	t->r
Adaptacion	Dimensiones						
Baseline	1000	17.083	24.077	42.989	22.289	42.452	18.534
	2000	24.183	23.236	43.116	20.845	39.875	13.748
GFK	1000	20.758	27.178	46.986	25.900	43.179	33.054
	2000	23.321	26.184	44.609	24.053	41.257	26.270
PCA	1000	15.721	23.958	44.042	22.609	43.553	19.034
	2000	19.238	21.330	43.102	20.881	40.418	13.748
SDA	1000	23.026	34.115	42.426	33.333	47.831	21.030
	2000	23.004	25.687	43.352	22.206	40.611	19.601
mSDA	1000	13.317	23.177	44.286	21.615	41.823	12.273
	2000	19.510	17.756	44.689	25.308	41.157	10.753

Apéndice B

Principal Components Analysis

El Análisis de Componentes Principales (*Principal Components Analysis* o *PCA*) es una técnica que permite identificar patrones en los datos, expresándolos de forma en que se destaquen las similitudes y diferencias entre estos datos, sirviendo así como una herramienta de análisis de datos. Otra ventaja de esta técnica es que, habiendo encontrado los patrones en los datos, permite reducir la cantidad de dimensiones sin generar mucha pérdida de información.

Utilizando las propiedades de los autovectores y autovalores, se pueden obtener las direcciones que generan la mayor varianza dentro de un conjunto de datos, permitiendo así reordenar estos datos de forma que se pueda diferenciar más entre ellos. De la misma forma, se pueden descartar dimensiones no aportan información útil para diferenciar de mejor manera los datos.

Considerando un conjunto de datos $\mathbf{X}$, los pasos para realizar *PCA* son:

- Sustraer la media de los datos.
- Calcular la matriz de covarianza de los datos.
- Calcular los autovectores y autovalores de la matriz de covarianza.
- Ordenar de mayor a menor los autovalores y generar un vector de características con los autovectores (o bien una matriz), posicionándolos de acuerdo al orden de sus autovalores correspondientes.
- Multiplicar la matriz de autovectores por el conjunto de datos para reordenar estos de acuerdo a los componentes principales.

Luego, en caso de querer reducir la cantidad de dimensiones, solo es necesario modificar un poco el último paso, multiplicando el conjunto de datos por una submatriz de autovectores con la cantidad de columnas correspondientes a las dimensiones deseadas.

Apéndice C

Partial Least Squares

Partial Least Squares, o *PLS*, es un método similar a *PCA* que puede ser utilizado para reducción de dimensionalidad y para obtener características importantes en un conjunto de datos. La diferencia principal con *PCA* es que *PLS* utiliza información sobre las etiquetas de un conjunto de datos. En contraste a *PCA*, que busca una alta covarianza entre las dimensiones de un conjunto de datos, *PLS* busca además una alta correlación entre las dimensiones y las etiquetas utilizadas.

Al igual que en *PCA*, se buscan las direcciones que generen la mayor varianza en un conjunto de datos. Sin embargo, estas direcciones son obtenidas utilizando solamente la información de los datos, sin considerar las etiquetas. Mientras que, por otra parte, *PLS* busca obtener las direcciones utilizando representaciones obtenidas a partir del producto interno entre los datos y las etiquetas, haciendo que estas nuevas representaciones se encuentren "pesadas" por la variación que los datos originales causan en las etiquetas.

Apéndice D

Marginalized Denoising Autoencoders

Tomando una instancia x , esta se corrompe estableciendo sus características a 0 con una probabilidad $p \geq 0$. Se denota a la versión corrompida de esta como $\tilde{x}$.

El *denoiser* de una capa (*Single-layer Denoiser*) trata de reconstruir la instancia x a partir de una instancia $\tilde{x}$ y una matriz de mapeo $\mathbf{W} : R^d \rightarrow R^d$, tal que $\mathbf{W}$ minimice el error de reconstrucción de la función D.1.

$$\frac{1}{2n} \sum_{i=1}^n \|\mathbf{x}_i - \mathbf{W}\tilde{\mathbf{x}}_i\|^2. \quad (\text{D.1})$$

Dado que la solución de la función D.1 depende de cuáles características sean corrompidas, se realizan múltiples pasadas en el conjunto de datos cada vez con distintas corrupciones. De este modo se busca encontrar un $\mathbf{W}$ que minimice el error general de pérdida de la ecuación D.2

$$\mathcal{L}_{sq}(\mathbf{W}) = \frac{1}{2mn} \sum_{j=1}^m \sum_{i=1}^n \|\mathbf{x}_i - \mathbf{W}\tilde{\mathbf{x}}_{i,j}\|^2. \quad (\text{D.2})$$

En donde $\tilde{x}_{i,j}$ representa la j -ésima versión corrompida del valor x_i .

Dado lo anterior, se define la matriz $\mathbf{X} = [x_1, \dots, x_n] \in R^{d \times n}$, y la matriz m veces repetida $\bar{\mathbf{X}} = [\mathbf{X}, \dots, \mathbf{X}]$, la cual en su versión corrompida queda denotada $\tilde{\bar{\mathbf{X}}}$. Luego,

con esto la ecuación D.1 se reduce a la ecuación D.3.

$$\mathcal{L}_{sq}(\mathbf{W}) = \frac{1}{2nm} \text{tr}[(\bar{\mathbf{X}} - \mathbf{W}\tilde{\mathbf{X}})^\top (\bar{\mathbf{X}} - \mathbf{W}\tilde{\mathbf{X}})]. \quad (\text{D.3})$$

La solución de la ecuación D.3 puede ser expresada en forma cerrada como en D.4.

$$\mathbf{W} = \mathbf{P}\mathbf{Q}^{-1}, \text{ con } \mathbf{Q} = \tilde{\mathbf{X}}\tilde{\mathbf{X}}^\top, \text{ y } \mathbf{P} = \bar{\mathbf{X}}\tilde{\mathbf{X}}^\top. \quad (\text{D.4})$$

Por la ley débil de los grandes números¹, las matrices $\mathbf{P}$ y $\mathbf{Q}$, definidas en la ecuación D.4, convergen a sus valores esperados cuando m se convierte en un valor muy grande. En el caso límite, en donde $m \rightarrow \infty$, se pueden derivar los valores esperados de $\mathbf{P}$ y $\mathbf{Q}$ y es posible expresar $\mathbf{W}$ como en la ecuación D.5.

$$\mathbf{W} = E[\mathbf{P}]E[\mathbf{Q}]^{-1}. \quad (\text{D.5})$$

Para encontrar los valores esperados de las matrices $E[\mathbf{P}]$ y $E[\mathbf{Q}]$ es conveniente enfocarse primero en $E[\mathbf{Q}]$. Teniendo en cuenta la ecuación D.6, un valor no diagonal de la matriz $\tilde{\mathbf{x}}_i\tilde{\mathbf{x}}_i^\top$ no está corrompida si dos características α y β han “sobrevivido” al proceso de corrupción, lo cual ocurre con probabilidad $(1-p)^2$, mientras que para los valores diagonales esto ocurre con probabilidad $(1-p)$. Luego, se define el vector $\mathbf{q} = [1-p, \dots, 1-p, 1]^\top \in \mathcal{R}^{d+1}$, en donde q_α representa la probabilidad de la característica α de “sobrevivir” a la corrupción, y el valor $q_{d+1} = 1$ se mantiene constante dado que nunca es corrompido.

$$E[\mathbf{Q}] = \sum_{i=1}^n E[\tilde{\mathbf{x}}_i\tilde{\mathbf{x}}_i^\top]. \quad (\text{D.6})$$

Definiendo la matriz dispersa de los valores de entrada sin corromper como $\mathbf{S} = \mathbf{X}\mathbf{X}^\top$, se puede expresar la esperanza de la matriz $\mathbf{Q}$ como se indica en la ecuación D.7. Y de forma similar la esperanza de la matriz $\mathbf{P}$ se puede expresar en forma cerrada según la ecuación D.8.

$$E[\mathbf{Q}]_{\alpha\beta} = \begin{cases} \mathbf{S}_{\alpha\beta}\mathbf{q}_\alpha\mathbf{q}_\beta & \text{si } \alpha \neq \beta \\ \mathbf{S}_{\alpha\beta}\mathbf{q}_\alpha & \text{si } \alpha = \beta \end{cases} \quad (\text{D.7})$$

¹La ley débil de los grandes números establece que el promedio de una sucesión infinita de variables aleatorias independientes converge a su valor esperado.

$$E[\mathbf{P}]_{\alpha\beta} = \mathbf{S}_{\alpha\beta} \mathbf{q}_{\beta} \quad (\text{D.8})$$

Finalmente, se puede obtener la matriz $\mathbf{W}$ en forma directa según la ecuación [D.4](#) sin crear explícitamente un valor de entrada corrompido $\tilde{x}_i$.

Apéndice E

Transfer Loss

Transfer loss es una métrica propuesta en [6] para medir el desempeño de algoritmos de adaptación entre dominios. Para calcular esta métrica es necesario conocer primero las siguientes medidas:

- *Transfer error* ($e(S, T)$): Se define como el error obtenido por un modelo al ser entrenado en el dominio S y ser probado en el dominio T .
- *In-domain error* ($e(T, T)$): Se define como el error obtenido por un modelo al ser entrenado y probado en un dominio T .
- *Baseline in-domain error* ($e_b(T, T)$): Se define como el *in-domain error* obtenido utilizando un método base (Ej: Una SVM sin realizar algún tipo de adaptación).

Finalmente, *transfer loss* se define como:

$$t(S, T) = e(S, T) - e_b(T, T)$$

De una manera más intuitiva, se puede entender el *baseline in-domain error* como el error que se obtiene en condiciones ideales, en donde se tienen los datos etiquetados necesarios para entrenar y probar un clasificador, siendo este error una especie de error al que se quiere llegar en otras circunstancias. Por otra parte, *transfer error* se puede entender como el error que se obtiene en las condiciones que busca resolver la adaptación entre dominios, es decir cuando se tienen pocos datos de un dominio nuevo y se requiere utilizar datos de un dominio similar para entrenar un clasificador. De esto se puede inferir que *transfer loss* representa la cantidad de error que se debe disminuir

para alcanzar lo que se obtendría en condiciones ideales, con un valor de 0 representando que se ha alcanzado lo esperado, y además un valor negativo representando que el clasificador adaptado ha superado los valores esperados.