



TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

Diseño e implementación de un analizador de dependencias para procesamiento de lenguaje natural en Español

Mediante Máquinas de Soporte Vectoriales

Autor

Alejandro Alcalde Barros

Directores

Salvador García López



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
8 de diciembre de 2016

Alejandro Alcalde Barros: *Diseño e implementación de un analizador de dependencias para procesamiento de lenguaje natural en Español*, Mediante Máquinas de Soporte Vectoriales, Grado en Ingeniería Informática, © 8 de diciembre de 2016

DIRECTOR:
Salvador García López

LOCALIZACIÓN:
Granada

ÚLTIMA MODIFICACIÓN:
8 de diciembre de 2016

Ohana means family.
Family means nobody gets left behind, or forgotten.
— Lilo & Stitch

Dedicated to the loving memory of Rudolf Miede.
1939–2005

RESUMEN

En este trabajo se implementa un método para analizar dependencias palabra a palabra con una estrategia de abajo a arriba (*Bottom-Up*) usando *Support Vector Machine* (SVM). En concreto, este trabajo se ha centrado en analizar las dependencias entre palabras en Castellano. El algoritmo no usa información sobre la estructura de las frases, realiza un análisis determinístico de las dependencias del idioma. El funcionamiento consiste en ir generando un árbol de dependencias para la frase mediante la ejecución de tres acciones — DESPLAZAR, IZQUIERDA, DERECHA — Inicialmente se comienza con tantos nodos como palabras tiene la frase, a medida que se ejecuta el algoritmo, se aplican las acciones mencionadas para ir construyendo el árbol. Por ejemplo, al aplicar la acción DERECHA a dos nodos contiguos, el nodo posicionado a la derecha pasa a ser padre del nodo izquierdo. En cuanto a los resultados obtenidos, han sido coherentes a la descripción proporcionada por los autores, obteniéndose resultados muy similares, incluso ligeramente mejores. La implementación original – Yamada y Matsumoto [3] – no se basa en conjuntos de datos en castellano, de modo que las comparaciones se han realizado frente al trabajo de Jain [2].

ETIQUETAS: PNL, SVM, Parseo de dependencias.

ABSTRACT

In this project, a method for analyzing word to word dependencies is implemented using a bottom-up strategy with the help of Support Vector Machine (SVM). In particular, this work has focused in analyzing dependencies between words for spanish language. The algorithm does not use any information about sentence's structure, it performs a deterministic analysis of the language's dependencies. Dependencies are build by generating a dependency tree for the sentence being analized. This tree is build with the help of three parsing actions — SHIFT, LEFT, RIGHT — At the beginning the algorithm has as many nodes as words has the sentence, in each iteration an action is applied to the nodes in order to build the dependency tree. For instance, when RIGHT action is applied to a pair of nodes, the node on the right becomes the parent of the left node. The accuracy obtained by this project its coherent with the one obtained for other authors. Our accuracy its very similar, even a little bit better. The original implementation by Yamada and Matsumoto [3] did not use any Spanish

datasets for its experiments, so comparison has been made against the work of Jain [2].

TAGS: NLP, SVM, Dependency parsing.

*An approximate answer to the right problem
is worth a good deal more than an exact answer
to an approximate problem.*

— John Tukey

AGRADECIMIENTOS

Muchas gracias a mi familia por apoyarme durante todos estos años de carrera, especialmente en los últimos meses de desarrollo de este proyecto. A mi tutor, por apoyar la idea.

Y por último, a Cristina, por estar siempre ahí.


ÍNDICE GENERAL

I	PUESTA EN ESCENA	1
II	OBJETIVOS	3
III	RESOLUCIÓN DEL TRABAJO	5
1	ALGORITMO SELECCIONADO: STATISTICAL DEPENDENCY ANALYSIS WITH SUPPORT VECTOR MACHINES	7
1.1	Una introducción a las SVMs	8
1.2	Análisis de dependencias determinístico	9
1.2.1	Acciones para el parseo	9
1.2.2	Descripción del algoritmo	10
1.2.3	Extracción de características	12
1.2.4	Agrupando SVMs para reducir costes	12
1.3	Ejemplo práctico	13
IV	CONCLUSIONES Y VÍAS FUTURAS	15
V	APÉNDICE	17
	BIBLIOGRAFÍA	19

ÍNDICE DE FIGURAS

Figura 1	Estructura en árbol de la frase “Rolls-Royce Motor Cars Inc. said it expects its U.S. sales to remain steady at about 1,200 cars.”	7
Figura 2	Árbol de parseo de dependencias	8
Figura 3	Ejemplo SVM	9
Figura 4	Ejemplo de la acción DESPLAZAR. (a) muestra el estado antes de aplicar la acción. (b) el resultado tras aplicarla	10
Figura 5	Ejemplo de la acción DERECHA. (a) Estado antes de la acción. (b) Estado tras aplicar la acción	10
Figura 6	Ejemplo de la acción LEFT. (a) Estado antes de la acción. (b) Estado tras aplicar la acción	10

TODO LIST

Figure: SVM example	9
 Con las figuras de arriba quizá no haya que poner esta sección.	13

ÍNDICE DE TABLAS

Tabla 1	Descripción del tipo de características y sus valores	12
---------	---	----

ÍNDICE DE CÓDIGO FUENTE

LISTA DE ALGORITMOS

Algoritmo 1 Algoritmo de parseo 11

ACRÓNIMOS

POS Part-Of-Speech

AA Aprendizaje Automático

PTB Penn Tree Bank

SVM Support Vector Machine

Parte I

PUESTA EN ESCENA

La memoria se organiza en cuatro partes.

La primera, en el ??, es una introducción al Procesamiento del Lenguaje Natural, en ella se definen los tipos de aplicaciones que permite, así como una descripción del *pipeline* usual que suele seguirse en este tipo de sistemas y el estado del arte.

En la segunda parte, ?? se listan los objetivos del proyecto.

La tercera parte se compone de los Capítulos (?? 1 ?? y ??). En ellos se introduce al lenguaje SCALA, se explican los detalles técnicos del algoritmo de parseo de dependencias, se discute la implementación y se muestran los casos de prueba realizados, respectivamente.

Por último, la cuarta parte consta del ?? y ??, en donde se exponen los resultados obtenidos, se comparan los resultados con el algoritmo original y se proponen trabajos futuros.

Parte II

OBJETIVOS

Parte III

RESOLUCIÓN DEL TRABAJO

Esta parte del trabajo se centra en los pasos seguidos para el desarrollo del proyecto. Se organiza como sigue:

En el ?? se discuten las características del lenguaje SCALA, y por qué ha sido escogido para el desarrollo.

El [Capítulo 1](#) detalla los aspectos técnicos del algoritmo de parseo de dependencias seleccionado.

En el ?? se narra las distintas etapas seguidas para el desarrollo, desde el análisis hasta la implementación.

Por último, el ?? explica los distintos casos de prueba llevados a cabo.

ALGORITMO SELECCIONADO: STATISTICAL DEPENDENCY ANALYSIS WITH SUPPORT VECTOR MACHINES

Yamada y Matsumoto [3] proponen un método para analizar las dependencias palabra-a-palabra mediante una estrategia *bottom-up* – de abajo a arriba. – Para ello se hace uso de la técnica de Aprendizaje Automático (AA) *Support Vector Machine* (SVM). Sus experimentos se basan en árboles de dependencias creados a partir del corpus Penn Tree Bank (PTB), logrando una precisión superior al 90 % para dependencias palabra-a-palabra. Aún siendo esta precisión inferior al ??, hay que tener en cuenta que este método no utiliza información sobre la estructura de las frases.

El tipo de anotaciones usadas en este método pueden verse en la Figura 2. Esta forma de ilustrar las dependencias palabra-a-palabra es más sencilla de entender para los anotadores que el usual estilo PTB — Figura 1 — El problema del estilo PTB es que requiere que los anotadores tengan un amplio conocimiento de la teoría lingüística del idioma, así como de la estructura de las frases, además del dominio específico que trata el problema. Como ventaja adicional, la representación del árbol de dependencias de la Figura 2 hace que la construcción de los datos de entrenamiento sea menos ruidosa, al ser el proceso de anotación más simple. En la estructura propuesta por

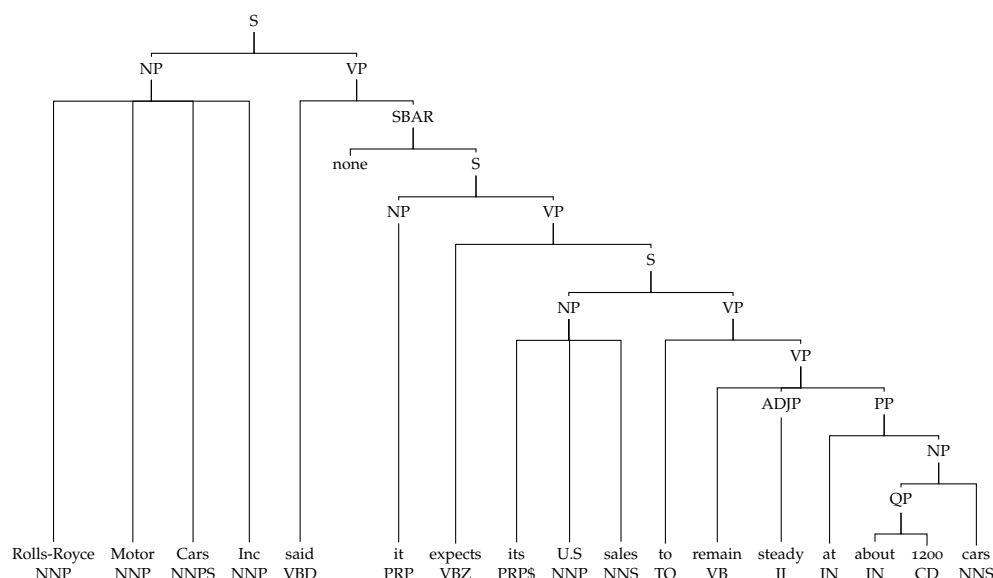


Figura 1: Estructura en árbol de la frase “Rolls-Royce Motor Cars Inc. said it expects its U.S. sales to remain steady at about 1,200 cars.”

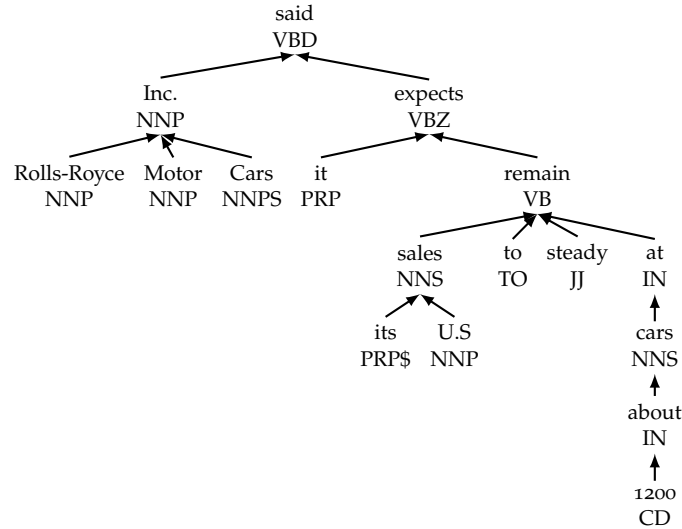


Figura 2: Árbol de parseo de dependencias

Yamada y Matsumoto se realiza un análisis estadístico de las dependencias de un idioma. Para lograr este análisis, se usa la técnica de aprendizaje conocida como [SVM](#), ya que es capaz de tratar con espacios de características a gran escala. En los siguientes párrafos se hará una breve introducción a esta técnica.

1.1 UNA INTRODUCCIÓN A LAS SVMs

Los modelos lineales son muy poderosos, mediante transformaciones lineales es posible incrementar en gran medida su capacidad expresiva. Sin embargo, incrementar dicha capacidad tiene un precio, el sobre ajuste y más tiempo de cómputo. [SVM](#) usa un cojín de seguridad cuando separa los datos. Este cojín logra que la [SVM](#) sea más robusta al ruido, reduciendo así el sobre ajuste. Además, [SVM](#) es capaz de trabajar con la herramienta conocida como *kernel* — la cual permite operar de forma eficiente con transformaciones no lineales de gran dimensión —. Estas dos características – el cojín de seguridad y el *kernel* – hacen de la [SVM](#) un modelo no lineal muy robusto y potente con regularización automática. Las [SVM](#) son muy populares por su facilidad de uso y su buen rendimiento.

[SVM](#) usa la estrategia del máximo margen ideada por *Vapnik*. Supongamos l datos de entrenamiento $(x_i, y_i), (1 \leq i \leq l)$, donde x_i es un vector de características en un espacio de dimensionalidad n , y_i es la etiqueta de la clase $\{-1, +1\}$ de x . Las [SVMs](#) encuentran un hiperplano $w \cdot x + b = 0$ que separe correctamente los datos de entrenamiento de forma que tengan margen máximo, es decir, con la máxima distancia entre dos hiperplanos $w \cdot x + b \geq 1$ y $w \cdot x + b \leq -1$, como se aprecia en la [Figura 3](#) El uso de [SVM](#) para el análisis estadístico de dependencias presenta principalmente dos ventajas. La primera de ellas es un

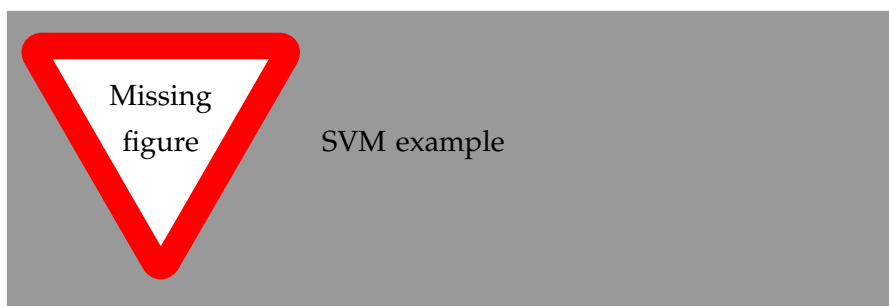


Figura 3: Ejemplo SVM

gran poder de generalización en espacios de características de grandes dimensiones. Segunda, gracias al *kernel trick* es posible entrenar al modelo para que aprenda a partir de la combinación de múltiples características.

Para poder trabajar con clasificaciones no lineales uno de los *kernels* posibles es el polinomial $-(\mathbf{x}' \cdot \mathbf{x}'' + 1)^d$ — Con este *kernel* se habilita la posibilidad de tener en cuenta combinaciones de d características sin incrementar demasiado el tiempo de cálculo. En el problema que nos ocupa, esto se traduce en la capacidad de entrenar reglas de dependencias usando varias características, como Part-Of-Speech (POS) *tags*, las palabras en sí y sus combinaciones.

1.2 ANÁLISIS DE DEPENDENCIAS DETERMINÍSTICO

1.2.1 Acciones para el parseo

El trabajo de Yamada y Matsumoto propone tres acciones para construir el árbol de dependencias. Es aquí cuando entra en juego la SVM, ya que aprenderá las acciones de los datos de entrenamiento y predecirá qué acciones realizar para datos nuevos. El árbol de dependencias se construye de izquierda a derecha, siguiendo la dirección de lectura habitual. Las tres acciones posibles son *Shift*, *Right* y *Left* — DESPLAZAR, DERECHA e IZQUIERDA. — Estas acciones se aplican a dos palabras vecinas, nombradas a partir de ahora nodos objetivo. Se pasa ahora a describir cada una de las acciones.

DESPLAZAR indica que no se ha realizado ninguna construcción en el árbol de dependencias entre los nodos objetivo. La acción para esta situación simplemente desplaza una posición a la derecha la ventana que apunta a los nodos objetivo. En la figura Figura 4 muestra un ejemplo concreto — Los ejemplos han sido extraídos de Yamada y Matsumoto [3] —

DERECHA construye una relación de dependencia entre los nodos objetivo. De los dos nodos objetivo, el de la izquierda pasa a ser hijo del nodo de la derecha. En la Figura 5 puede observarse el efecto de esta acción. IZQUIERDA construye una relación de dependencia entre

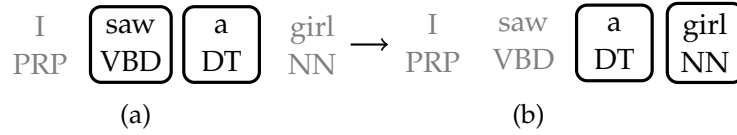


Figura 4: Ejemplo de la acción DESPLAZAR. (a) muestra el estado antes de aplicar la acción. (b) el resultado tras aplicarla

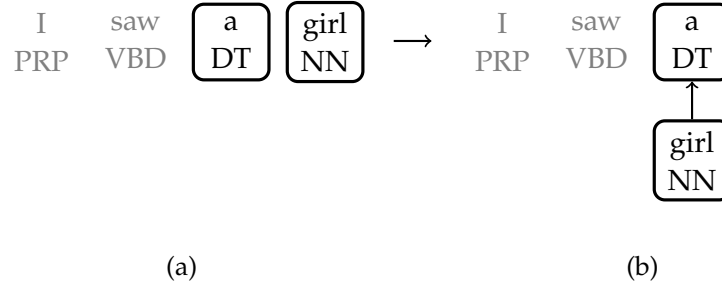


Figura 5: Ejemplo de la acción DERECHA. (a) Estado antes de la acción. (b) Estado tras aplicar la acción

los nodos objetivo. De los dos nodos objetivo, el de la derecha pasa a ser hijo del nodo de la izquierda. En la [Figura 6](#) se ilustra esta situación.

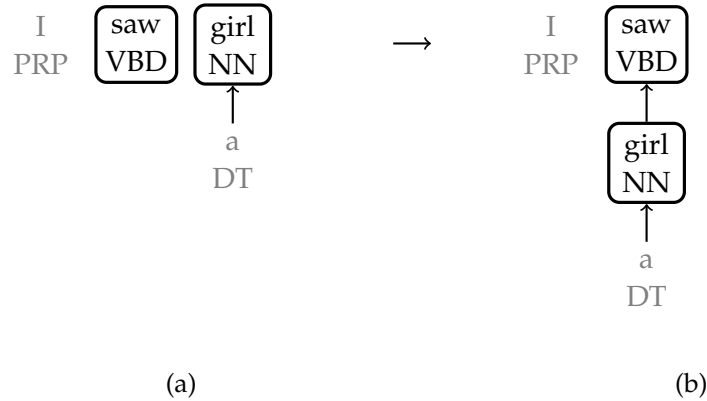


Figura 6: Ejemplo de la acción LEFT. (a) Estado antes de la acción. (b) Estado tras aplicar la acción

1.2.2 Descripción del algoritmo

El Algoritmo 1 consiste de dos partes. Primero se estima la acción apropiada usando la información contextual rodeando los nodos objetivo. Segundo, se construye el árbol de dependencias ejecutando las acciones estimadas en el primer paso.

Algoritmo 1 Algoritmo de parseo

```

1: Input Sentence:  $(w_1, p_1), (w_2, p_2), \dots, (w_n, p_n)$ 
2: Initialize:
3:    $i \leftarrow 1$ 
4:    $\mathcal{T} \leftarrow \{(w_1, p_1), (w_2, p_2), \dots, (w_n, p_n)\}$ 
5:    $\text{no\_construction} \leftarrow \text{true}$ 
6: Start
7:   while  $|\mathcal{T}| \geq 1$  do
8:     if  $i == |\mathcal{T}|$  then
9:       if  $\text{no\_construction} == \text{true}$  then break
10:      end if
11:       $\text{no\_construction} \leftarrow \text{true}$ 
12:       $i \leftarrow 1$ 
13:    else
14:       $x \leftarrow \text{getContextualFeatures}(\mathcal{T}, i)$ 
15:       $y \leftarrow \text{estimateAction}(\text{model}, x)$ 
16:       $\text{construction}(\mathcal{T}, i, y)$ 
17:      if  $y == \text{Left or Right}$  then  $\text{no\_construction} \leftarrow \text{false}$ 
18:      end if
19:    end if
20:  end while

```

Cuando el algoritmo se ejecuta, la variable i representa el índice del nodo de la izquierda del par de nodos objetivos, $i + 1$ el de la derecha, dichos nodos están en \mathcal{T} . \mathcal{T} contiene la secuencia de nodos a los que se debe estimar una acción, estos nodos se corresponden con los nodos raíz de los árboles que se van construyendo a lo largo del proceso de parseado. Como es lógico, inicialmente todos los nodos $t_i \in \mathcal{T}$ están compuestos únicamente por la raíz, sin tener hijos. La información que guarda cada nodo es el par (w_i, p_i) , donde w_i es una palabra y p_i su [POS tag](#).

La estimación sobre qué acción aplicar se lleva a cabo mediante las funciones *getContextualFeatures* y *estimateAction*. La primera extrae características de x en función del contexto que la rodea por i , en la sección [Extracción de características](#) se profundiza sobre este tema. La segunda función estima la acción más apropiada.

La variable *no_construction* se encarga de comprobar cuando se han producido acciones que hayan resultado en la contrucción de dependencias al terminar de leer la frase. Cuando esta variable tiene valor verdadero significa que se ha producido la acción DESPLAZAR para todos los nodos objetivo, y por tanto no se han creado nuevas dependencias. En ese caso se detiene el proceso y se devuelven los árboles en \mathcal{T} , ya que no es posible aplicar más acciones. Cuando la variable es falsa, los nodos objetivo se colocan al principio de la frase y se vuelve a repetir el proceso hasta que $|\mathcal{T}| = 1$.

1.2.3 Extracción de características

En el proceso de entrenamiento de las SVM, cada estimación de una acción y en el contexto x se corresponde con una observación (x, y) de la SVM. Al tener tres tipos de acciones, el problema que nos ocupa es de clasificación multi clase. Para resolverlo se crean tres clasificadores binarios correspondientes a cada acción, lo cual se conoce como método por parejas. Los tres clasificadores binarios son IZQUIERDA vs. DERECHA, IZQUIERDA vs. DESPLAZAR y DERECHA vs. DESPLAZAR.

Como se mencionó en la Subsección 1.2.2, el POS tag y la propia palabra se usan como características de los nodos para los contextos de la izquierda y derecha. Se pasa ahora a describir en qué consisten dichos contextos.

Como ya se sabe, i e $i + 1$ son los índices que apuntan los nodos objetivo en \mathcal{T} . El contexto a la izquierda se puede definir como los nodos posicionados a la izquierda de los nodos objetivo, es decir t_l , ($l < i$). El contexto a la derecha, de forma análoga, son los nodos a la derecha de los nodos objetivo t_r , ($i + 1 < r$)

La representación de una característica viene dada por la tupla (p, k, v) . p es la posición partiendo de los nodos objetivo, k codifica el tipo de característica y v almacena el valor para dicha característica. Cuando $p < 0$ se está representando el nodo del contexto izquierdo, $p = \{0-, 0+\}$ representa el nodo izquierdo ($0-$) o derecho ($0+$) de los nodos objetivo — recordemos que los nodos objetivo estaban formados por dos nodos. — Análogamente, $p > 0$ indica los nodos en el contexto derecho. La Tabla 1 muestra los valores del tipo de característica k y sus valores v . Las características $ch\{-L,R\}\{-pos,lex\}$ se denominan características hijas y se calculan de forma dinámica durante el análisis. Estas características ayudan a determinar qué acción tomar.

Tabla 1: Descripción del tipo de características y sus valores

Tipo	Valor
pos	POS tag
lex	La palabra
ch-L-pos	POS tag del nodo hijo modificando al padre desde la izquierda
ch-L-lex	Palabra del correspondiente ch-L-pos
ch-R-pos	POS tag del nodo hijo que modifica al padre desde la derecha
ch-R-lex	Palabra del correspondiente ch-R-pos

1.2.4 Agrupando SVMs para reducir costes

Debido a la gran cantidad de datos de entrenamiento disponibles, Yamada y Matsumoto proponen dividirlos en varios grupos. La di-

visión se realiza en base al *POS tag* del nodo de la izquierda de los nodos objetivo. Por ejemplo, si el *POS tag* del nodo izquierdo es “VB”, se estima la acción usando SVM_s^{VB}

1.3 EJEMPLO PRÁCTICO

Con las figuras de arriba quizá no haya que poner esta sección.

Parte IV

CONCLUSIONES Y VÍAS FUTURAS

Esta última parte de la memoria se dedica a presentar los resultados obtenidos con el parseador introducido en [Capítulo 1](#), así como comparar los resultados con la implementación original. Por último, se comentarán algunas ideas para trabajos futuros.

Parte V

APÉNDICE

BIBLIOGRAFÍA

- [1] Yaser S. Abu-Mostafa, Malik Magdon-Ismail y Hsuan-Tien Lin. *Learning From Data, a short course*. AML, 2012.
- [2] Rohit Jain. *Re-Implementing Statistical Dependency Analysis With Support Vector Machines*. Inf. téc. Cornell Tech, 2016.
- [3] Hiroyasu Yamada y Yuji Matsumoto. «Statistical dependency analysis with support vector machines». En: *Proceedings of IWPT*. Vol. 3. 2003, págs. 195-206.

DECLARACIÓN

Yo, ALEJANDRO ALCALDE BARROS, alumno del GRADO EN INGENIERÍA INFORMÁTICA de la ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE TELECOMUNICACIONES de la UNIVERSIDAD DE GRANADA, declaro explícitamente que el trabajo presentado es original, entendiéndose en el sentido de que no se ha utilizado ninguna fuente sin citarla debidamente.

Granada, 8 de diciembre de 2016

Alejandro Alcalde Barros

COLOPHON

This document was typeset using the typographical look-and-feel *classicthesis* developed by André Miede. The style was inspired by Robert Bringhurst’s seminal book on typography “*The Elements of Typographic Style*”. *classicthesis* is available for both \LaTeX and \LyX :

<https://bitbucket.org/amiede/classicthesis/>

Happy users of *classicthesis* usually send a real postcard to the author, a collection of postcards received so far is featured here:

<http://postcards.miede.de/>

Hermann Zapf’s *Palatino* and *Euler* type faces (Type 1 PostScript fonts *URW Palladio L* and *FPL*) are used. The “typewriter” text is typeset in *Bera Mono*, originally developed by Bitstream, Inc. as “Bitstream Vera”. (Type 1 PostScript fonts were made available by Malte Rosenau and Ulrich Dirr.)