ESCUELA COLOMBIANA DE INGENIERÍA
JULIO GARAVITO

PROYECTO DE GRADO

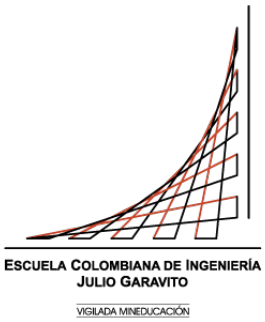
Advanced Natural Language Processing Techniques to Profile Cybercriminals

Autor:
Alejandro ANZOLA ÁVILA

Director:
Daniel Orlando DÍAZ LÓPEZ, *Ph.D.*

Programa de Ingeniería de Sistemas

BOGOTÁ, COLOMBIA



4 de agosto de 2019

Índice general

Índice de figuras

Índice de cuadros

Escuela Colombiana de Ingeniería
Julio Garavito

Resumen

Programa de Ingeniería de Sistemas

Estudiante de Ingeniería de Sistemas

Advanced Natural Language Processing Techniques to Profile Cybercriminals

por Alejandro ANZOLA ÁVILA

Se identifican diferentes metodologías de *Data Science* usadas comúnmente para el análisis de datos estructurados y no–estructurados. Luego se realiza una exploración del estado del arte en técnicas actuales de perfilamiento de cibercriminales y técnicas de clasificación de datos. Finalmente se realizan tres propuestas de modelos de *Natural Language Processing* que buscan soportar a Agencias de Seguridad del Estado en su lucha contra el crimen.

Escuela Colombiana de Ingeniería
Julio Garavito

Abstract

Programa de Ingeniería de Sistemas

Estudiante de Ingeniería de Sistemas

Advanced Natural Language Processing Techniques to Profile Cybercriminals

by Alejandro ANZOLA ÁVILA

Different commonly used *Data Science* methodologies are identified for structured and non-structured data analysis. After that a State of the Art exploration of current techniques for cybercriminal profiling and data classification is made. Lastly, three *Natural Language Processing* model proposals are made that seek to support State Security Agencies in their fight against crime.

Notación

Esta notación es una adaptación de la presentada en [1].

Números y Arreglos

a	Un escalar (entero o real)
$\boldsymbol{a}$	Un vector
$\boldsymbol{A}$	Una matriz
$\mathbf{A}$	Un tensor
a	Un token (n-grama, palabra, letra o byte)
$\mathbf{a}$	Una secuencia de tokens, $\mathbf{a} = \mathsf{a}_1 \ \ldots \ \mathsf{a}_N$
$\boldsymbol{I}_n$	Matriz identidad con n filas y n columnas
$\boldsymbol{I}$	Matriz identidad con dimensionalidad implícita por el contexto
$\boldsymbol{e}^{(i)}$	Vector estándar base $[0, \ldots, 0, 1, 0, \ldots, 0]$ con un 1 en la posición i
$\mathrm{diag}(\boldsymbol{a})$	Una matriz diagonal cuadrada con entradas diagonales dadas por $\boldsymbol{a}$
a	Una variable aleatoria real
$\mathbf{a}$	Un vector de variables aleatorias
$\mathbf{A}$	Una matriz de variables aleatorias

Conjuntos y Grafos

$\mathbb{A}$	Un conjunto
$\mathbb{R}$	El conjunto de números reales
$\mathbb{N}$	El conjunto de números naturales
$\{0, 1\}$	El conjunto que contiene al 0 y el 1
$\{0, 1, \dots, n\}$	El conjunto que contiene todos lo números entre 0 y n
$[a, b]$	El intervalo real que incluye a y b
$[a : b]$	El intervalo discreto que incluye a y b
$(a, b]$	El intervalo real que no incluye a pero si b
$\mathbb{A} \backslash \mathbb{B}$	Substracción de conjuntos, e.g., el conjunto que contiene los elementos de $\mathbb{A}$ que no están en $\mathbb{B}$
$\mathcal{G}$	Un grafo
$Pa_{\mathcal{G}}(\mathrm{x}_i)$	El padre de x_i en $\mathcal{G}$

Índices

a_i	Elemento i del vector $\boldsymbol{a}$, con índice empezando en 1
a_{-i}	Todos los elementos del vector $\boldsymbol{a}$ a excepción del elemento i
$A_{i,j}$	Elemento i, j de la matriz $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$	Fila i de la matriz $\boldsymbol{A}$
$\boldsymbol{A}_{:,i}$	Columna i de la matriz $\boldsymbol{A}$
$\boldsymbol{A}_{i,j,k}$	Elemento (i, j, k) de un tensor $\mathbf{A}$ 3-D
$\mathbf{A}_{:,:,i}$	Corte 2-D de un tensor 3-D
a_i	Elemento i del vector aleatorio $\mathbf{a}$

Operaciones de Álgebra Lineal

$\boldsymbol{A}^\top$	Traspuesta de la matriz $\boldsymbol{A}$
$\det(\boldsymbol{A})$	Determinante de la matriz $\boldsymbol{A}$

Calculo

$\frac{dy}{dx}$	Derivada de y con respecto a x
$\frac{\partial y}{\partial x}$	Derivada parcial de y con respecto a x
$\nabla_{\boldsymbol{x}} y$	Gradiente de y con respecto a $\boldsymbol{x}$
$\nabla_{\boldsymbol{X}} y$	Matriz de derivadas de y con respecto a $\boldsymbol{X}$
$\nabla_{\mathbf{X}} y$	Tensor que contiene derivadas de y con respecto a $\mathbf{X}$
$\int f(\boldsymbol{x}) d\boldsymbol{x}$	Integral definida sobre un dominio entero $\boldsymbol{x}$
$\int_{\mathbb{S}} f(\boldsymbol{x}) d\boldsymbol{x}$	Integral definida con respecto a $\boldsymbol{x}$ sobre el conjunto $\mathbb{S}$

Teoría de Probabilidad e Información

$\mathrm{a} \perp \mathrm{b}$	Las variables aleatorias a y b son independientes
$\mathrm{a} \perp \mathrm{b} \mid \mathrm{c}$	Son condicionalmente independientes dado c
$P(\mathrm{a})$	Una distribución de probabilidad de una variable aleatoria discreta
$p(\mathrm{a})$	Una distribución de probabilidad de una variable aleatoria continua
$\mathrm{a} \sim P$	La variable aleatoria a tiene distribución P
$\mathrm{Var}(f(x))$	Varianza de $f(x)$ sobre $P(\mathrm{x})$
$\mathrm{Cov}(f(x), g(x))$	Covarianza de $f(x)$ y $g(x)$ sobre $P(\mathrm{x})$
$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2)$	Distribución gaussiana sobre $\boldsymbol{x}$ con media $\boldsymbol{\mu}$ y varianza $\boldsymbol{\sigma}^2$

Funciones

$f : \mathbb{A} \to \mathbb{B}$	La función f con dominio $\mathbb{A}$ y rango $\mathbb{B}$
$f \circ g$	Composición de las funciones f y g
$f(\boldsymbol{x}; \boldsymbol{\theta})$	Una función de $\boldsymbol{x}$ parametrizado por $\boldsymbol{\theta}$.
$\log x$	Logaritmo natural de x
$\log_b x$	Logaritmo base b de x
$\sigma(x)$	Función sigmoid logística, $\frac{1}{1+\exp(-x)}$
$\zeta(x)$	Función Softplus, $\log(1+\exp(x))$
$\mathrm{softmax}(\boldsymbol{z})_i$	Función Softmax, $\exp(\boldsymbol{z}_i) / \sum_{j=1} \exp(\boldsymbol{z}_j)$
$\|\boldsymbol{x}\|_p$	Norma L^p de $\boldsymbol{x}$, $\|\boldsymbol{x}\|_p = \sqrt[p]{\sum_i x_i^p}$
$\|\boldsymbol{x}\|_\infty$	Norma L^∞ de $\boldsymbol{x}$, $\|\boldsymbol{x}\|_\infty = \max_i \lvert x_i \rvert$
$\|\boldsymbol{x}\| = \|\boldsymbol{x}\|_2$	Norma L^2 de $\boldsymbol{x}$
x^+	Parte positiva de x, e.g., $\max(0, x)$
$\mathbf{1}_{\text{condition}}$	Es 1 si $\text{condition} \in \mathbb{B}$ es verdadero, 0 de lo contrario
$[\text{condition}]$	Es 1 si $\text{condition} \in \mathbb{B}$ es verdadero, 0 de lo contrario

Datasets

p_{data}	La distribución generadora de datos
$\hat{p}_{\text{data}}$	La distribución empírica definida por el conjunto de entrenamiento
$\mathbb{X}$	Un conjunto de muestras de entrenamiento
$\boldsymbol{x}^{(i)}$	La i-ésima muestra (entrada) de un Dataset
$y^{(i)}$ or $\boldsymbol{y}^{(i)}$	El objetivo asociado con $\boldsymbol{x}^{(i)}$ para aprendizaje supervisado
$\boldsymbol{X}$	La matriz de $m \times n$ con muestra de entrada $\boldsymbol{x}^{(i)}$ en la fila $\boldsymbol{X}_{i,:}$

Capítulo 1

Introducción

1.1. Objetivo general

El objetivo de este proyecto es generar herramientas y estrategias para el perfilado de cibercriminales con ayuda de metodologías de *Natural Language Processing* (NLP) aplicado a datos recolectados de comunicaciones y redes sociales.

1.2. Objetivos específicos

- Identificar el estado del arte en sistemas que usan NLP para apoyar agencias de seguridad del Estado.
- Implementación de artefactos para la construcción de *Dataset*s con información recolectada de medios privados como de fuentes abiertas.
- Diseñar e implementar diferentes modelos de inteligencia artificial basados en procesamiento de lenguaje natural junto con heurísticas y meta–heurísticas para el perfilado de sospechosos.
- Validar la solución desarrollada frente a un escenario real.

Esto se planea hacer en el segundo periodo

Capítulo 2

Cronograma

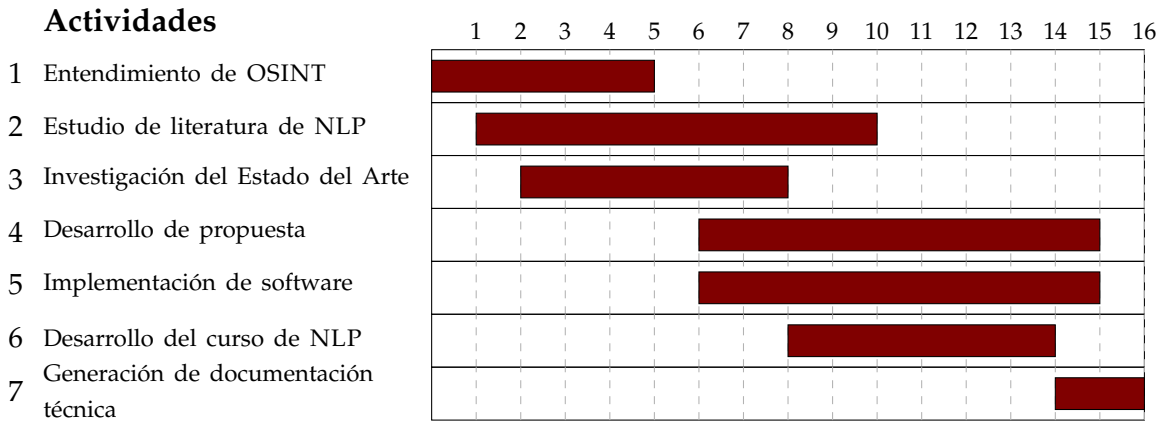


FIGURA 2.1: Diagrama Gantt de actividades de 1er periodo.

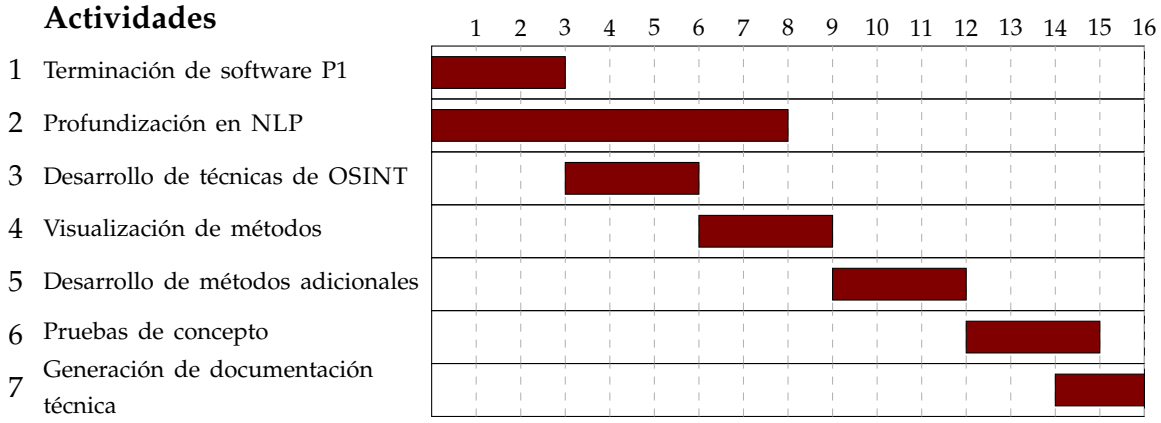


FIGURA 2.2: Diagrama Gantt de actividades de 2do periodo (Planeado).

#	**Descripción**
1.1	Entendimiento proyecto de *Inteligencia de fuentes abiertas para el contexto colombiano* [2].
1.2	Estudio de la literatura de NLP.
1.3	Investigación del Estado del Arte.
1.4	Desarrollo de la propuesta de solución.
1.5	Implementación de solución y pruebas.
1.6	Realización de curso de NLP de *National Research University Higher School of Economics* en *Coursera*.
1.7	Preparación final de documento de libro de proyectos del primer periodo.
2.1	Terminación y mejora de software desarrollado en el primer periodo.
2.2	Profundización de metodologías aplicadas de NLP.
2.3	Desarrollo de nuevas técnicas para la obtención de información de fuentes abiertas como también de identificar nuevas fuentes de información que tengan que ver con terrorismo.
2.4	Desarrollo de un *dashboard* detallado que ayuden un agente de ciberinteligencia realizar sus labores.
2.5	Desarrollo de métodos adicionales para el perfilamiento de cibercriminales.
2.6	Realizar pruebas demostrativas que permitan visualizar la obtención de información de las fuentes abiertas en tiempo real.
2.7	Generación de documentación técnica final.

CUADRO 2.1: Descripción del cronograma de actividades.

Capítulo 3

Logros y productos del proyecto

Entre los logros y productos obtenidos durante la ejecución de este proyecto de grado se encuentran los siguientes:

1. Entendimiento de las generalidades de *Data Science*:
   - Tipos de *Machine Learning*.
   - Sistemas de detección de anomalías (*Anomaly Detection Systems*).
   - Diferentes modalidades de *Clustering*.
2. Identificación de modelos de NLP aplicables para el perfilado de cibercriminales.
3. Entendimiento de los modelos de clasificación y *Clustering*:
   - Clasificador de Naïve Bayes.
   - Maquinas de soporte vectorial (*Support Vector Machine*).
   - Mapas autoorganizados (*Self-organizing Map*).
4. Entendimiento de los modelos utilizados en NLP:
   - Predicción de etiquetas con modelos de regresión lineal.
   - Reconocimiento de *Named Entities*.
   - Uso de *embeddings* generados con *StarSpace* para los k textos mas similares.
5. Propuesta de modelos de NLP para el perfilado de cibercriminales:
   - Modelo de predicción de hashtags de Twitter con modelos lineales junto con su implementación.
   - Modelo de reconocimiento de *Named Entities* con redes *Long Short Term Memory*.
   - Modelo de *Clustering* en redes SOM con *embeddings* de *StarSpace*.

Capítulo 4

Marco teórico

La Web contiene una gran cantidad de opiniones respecto a productos, políticos, y mucho mas, expresado en forma de noticias, sitios de opinión, reseñas en tiendas online, redes sociales. Como resultado, el problema de "Minería de opinión" ha obtenido una atención creciente en las ultimas dos décadas y es un factor decisivo para las nuevas organizaciones (como es mencionado en [3]). De esto mismo partimos que el análisis de textos para extraer el significado y demás componentes extraíbles del texto componen un factor que debe considerarse al momento de realizar decisiones, de manera que los avances hechos hasta ahora tienen como meta una aplicación practica de lo que se conoce como *Natural Language Processing*.

Luego de los ataques terroristas del 11 de Septiembre de 2001 en Estados Unidos, se realizaron fuertes criticas respecto a la inteligencia, donde el director del FBI llamado *Robert S. Mueller* indico que el principal problema que la agencia tuvo fue que se enfocaba demasiado en lidiar con el crimen luego de que fue cometido y ponía muy poco énfasis en prevenirlo (adaptado de [4]). Es por esto que el uso de NLP para temas de seguridad como también de metodologías de *Machine Learning* y *Deep Learning* han sido ampliamente utilizadas en ámbito de seguridad luego de estos eventos.

Para obtener una mejor inteligencia se necesito de mejores tecnologías a las que se tenían entonces (véase [4, pág 2]):

- Integración de datos (o *Data Integration* en ingles).
- Análisis de vínculos (o *Link Analysis* en ingles).
- Agentes de software (o *Software Agents* en ingles).
- Minería de texto (o *Text Mining* en ingles).
- Redes neuronales (o *Artificial Neural Network* (ANN) en ingles).
- Algoritmos de *Machine Learning* (o *Machine Learning Algorithms* en ingles).

4.1. Análisis de vínculos (*Link Analysis*)

Es la visualización de asociaciones entre entidades y eventos, por lo general involucran una visualización por medio de una gráfica o un mapa que muestre las relaciones entre sospechosos y ubicaciones, sea por medio físico o por comunicaciones en la red.

4.2. Agentes de software (*Software Agents*)

Es el software que realiza tareas asignadas por el usuario de manera autónoma, donde sus habilidades básicas son:

- **Realización de tareas:** Hacen obtención de información, filtrado, monitoreo y reporte.
- **Conocimiento:** Pueden usar reglas programadas, o pueden aprender reglas nuevas (véase 4.6.1).
- **Habilidades de comunicación:** Reportar a humanos e interactuar con otros agentes.

4.3. Aprendizaje de maquina (*Machine Learning*)

De acuerdo con [5], se define como un conjunto de métodos que pueden detectar patrones automáticamente en datos, y luego usar los patrones descubiertos para predecir los datos futuros, o realizar otra clase de toma de decisiones con un grado de incertidumbre, por tal motivo es necesario el uso de teoría de probabilidad, que puede ser aplicada a cualquier tipo de problema que involucra incertidumbre.

4.3.1. Tipos de *Machine Learning*

Machine Learning (ML) esta principalmente dividida en tres tipos. El método predictivo o bien **aprendizaje supervisado** (*Supervised Learning*), donde el objetivo es aprender un mapeo de las entradas $\boldsymbol{x}$ a las salidas y, dado un conjunto de pares de etiquetas de entrada–salida $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$. $\mathcal{D}$ se le llama el conjunto de entrenamiento y N es el numero de muestras de entrenamiento.

aprendizaje supervisado

En la forma mas sencilla, cada entrada de entrenamiento $\boldsymbol{x}_i$ es un vector $|\mathcal{D}|$–dimensional de números, a estos se le llaman *características* o *atributos*.

De manera similar la forma de la salida puede ser en principio cualquier cosa, pero la mayoría de métodos asumen que y_i es una variable *categórica* o *nominal* de algún conjunto finito, $y_i \in \{1, \ldots, C\}$, o que y_i es un escalar real, $y_i \in \mathbb{R}$. Cuando la variable y_i es categórica, al problema se le reconoce como **clasificación** o **reconocimiento de patrones**, y cuando es un valor real se le conoce como un problema de **regresión**.

clasificación
reconocimiento de patrones
regresión

El segundo tipo principal de ML es el descriptivo o **aprendizaje no–supervisado** (*Unsupervised Learning*), en este solo están disponibles los datos de entrada $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^{N}$, y la meta es encontrar "patrones interesantes" en los datos. Este es un problema mucho menos definido, debido a que no se conocen los tipos de patrones que se quieren encontrar, y no hay una métrica obvia de error (no como aprendizaje supervisado en la que se puede comparar nuestra predicción de y para un $\boldsymbol{x}$ con el valor observado).

aprendizaje no–supervisado

Un tercer tipo de aprendizaje de maquina es conocido como **aprendizaje de refuerzo** (o *Reinforcement Learning*en inglés), el cual es un tipo menos usado. Este es útil cuando se quiere aprender como actuar o comportarse cuando se recibe una recompensa ocasional o una señal de castigo.

aprendizaje de refuerzo

4.4. Minería de datos (*Data Mining*)

Según [6], la minería de datos se define como el proceso de descubrir información útil en repositorios grandes de datos. Las técnicas de minería de datos son desplegadas para limpiar grandes bases de datos para encontrar patrones nuevos y útiles que de lo contrario podrían permanecer desconocidos. También ofrecen capacidades para predecir la salida de observaciones futuras, tales como predecir si un cliente nuevo gastara mas de $100 en una tienda.

No todas las tareas de descubrimiento de información son considerados como *Data Mining*. Por ejemplo, realizar una consulta de campos individuales usando un sistema de base de datos o encontrar una pagina web por medio de una búsqueda en Internet son tareas relacionadas con *adquisición de información*.

4.4.1. Minería de texto (*Text Mining*)

Es un subcampo de Inteligencia Artificial conocida como *Natural Language Processing*, en donde las herramientas de minería de datos pueden capturar rasgos críticos del contenido de un documento basado en el análisis de sus características lingüísticas.

La mayoría de los crímenes son electrónicos por naturaleza, por lo que se dejan rastros textuales que investigadores pueden seguir y analizar. Estas se enfocan en el descubrimiento de relaciones en texto no–estructurado y pueden ser aplicados al problema de *búsqueda* y *localización de palabras clave*.

4.5. Redes Neuronales Artificiales (*Artificial Neural Network*)

El estudio de redes neuronales artificiales (ANN) fue inspirado por los intentos de simular los sistemas biológicos de neuronas. El cerebro humano se compone principalmente de células nerviosas llamadas *neuronas*, enlazadas con otras neuronas por medio de hebras de fibra conocidas como *axones*. Los axones son usados para transmitir impulsos nerviosos de una neurona a otra cada vez que las neuronas son estimuladas. Una neurona esta conectada a axones de otras neuronas por medio de *dendritas*, las cuales son extensiones desde el cuerpo de la neurona. El punto de contacto entre una dendrita y un axón se conoce como *sinapsis*. Los neurólogos han descubierto que el cerebro humano aprende por medio de cambiar la fuerza de la conexión sináptica entre las neuronas a través de estimulación repetitiva por el mismo impulso.

De manera análoga a la estructura del cerebro humano, una ANN se compone de una estructura interconectada de nodos y vínculos directos.

4.5.1. Redes neuronales profundas (*Deep Feedforward Networks*)

Una arquitectura muy popular de ANNs es el caso de las redes *Deep feedforward* o también conocidas como *Multilayer Perceptrons* (MLP) cuyo objetivo es aproximarse a una función f^*. A estos modelos se les llama **feedforward** debido a que la información fluye de una entrada $\boldsymbol{x}$ a través de unas computaciones intermedias usadas para definir f, y finalmente dar como salida a $\boldsymbol{y}$. feedforward

A estas se les llama redes debido a que son típicamente representadas con la composición de varias funciones y así mismo también son asociadas con grafos directos

acíclicos (o *Directed Acyclic Graph* (DAG) en inglés). La composición de funciones puede ser vista como $f(\boldsymbol{x}) = f^{(l)} \circ f^{(l-1)} \circ \cdots \circ f^{(1)}(\boldsymbol{x})$, siendo esto análogo a una red MLP de l capas, donde a $f^{(1)}$ se le llamaría la primera capa, a $f^{(2)}$ la segunda, y así sucesivamente. La longitud total de la cadena es la **profundidad** de la red, de ahí viene el nombre de profundas.

Considérese a d_i como la dimensión de la capa i-ésima, cada una de las capas intermedias $f^{(i)}$ son funciones toman como entrada un vector en $\mathbb{R}^{d_{i-1}}$ y da como resultado un vector $\mathbb{R}^{d_i}$ que es la entrada para la siguiente capa que pasara a ser $f^{(i)} : \mathbb{R}^{d_i} \to \mathbb{R}^{d_{i+1}}$.

Durante el entrenamiento de estas redes el objetivo final es que $\boldsymbol{y} \approx f^*(\boldsymbol{x})$, el algoritmo de aprendizaje debe decidir como acomodar las capas intermedias $f^{(i)}$ para dar el resultado deseado.

Una representación visual de las capas de una de estas redes se encuentra en la figura 4.1.

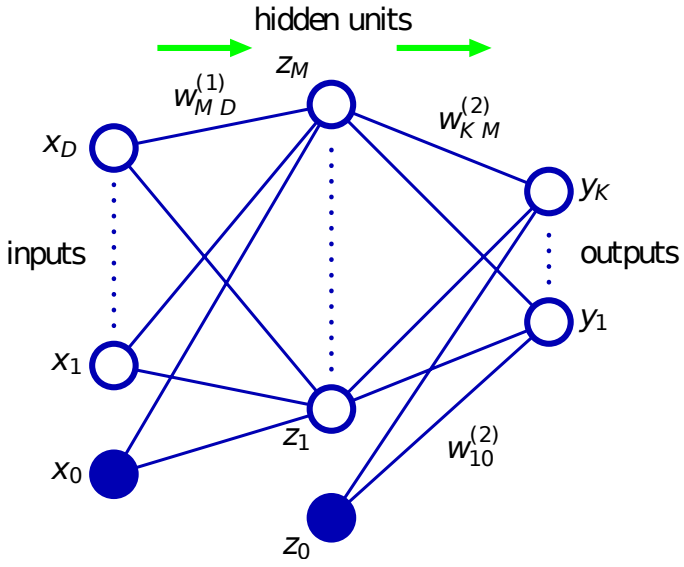


FIGURA 4.1: Representación de una *Deep Feedforward Network*. Tomado de [7].

Aunque esta es una representación simplificada de este tipo de redes, este libro no tratara de profundizar en los detalles técnicos al ser estos muy amplios (véase [1] y [7]).

4.5.2. Redes neuronales recurrentes (*Recurrent Neural Network*)

Las redes neuronales recurrentes (o *Recurrent Neural Network* (RNN) en inglés) son una familia de redes neuronales para el procesamiento de datos secuenciales. Una RNN es una red que se especializa para el procesamiento de una secuencia de valores $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}$. La mayoría de redes RNN pueden procesar secuencias de longitud variable.

Estas redes utilizan un concepto de ML llamado **compartido de parámetros** (o *parameter sharing* en inglés) a través de diferentes partes de un modelo. Esto nos permite extender y aplicar el modelo a muestras de diferentes longitudes y generalizar sobre

compartido de parámetros

ellas. Haciendo que el procesamiento de los parámetros sea independiente del orden de la secuencia con los que son dados a la red.

Podemos denotar entonces cada paso de una RNN (como se muestras en la figura 4.2) como la aplicación recursiva de la formula:

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}; \boldsymbol{\theta}) \tag{4.1}$$

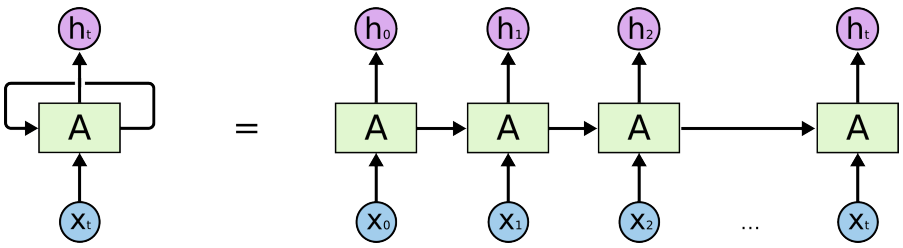


FIGURA 4.2: Red RNN simplificada. Tomado de [8].

Donde $\boldsymbol{h}^{(t)}$ es el estado del sistema en el instante t que además se le suele denotar con la letra 'h' por tratarse de estados ocultos (*hidden*) en la red. Los parámetros $\boldsymbol{\theta}$ no varían por el instante de tiempo en la que se este aplicando la función f (compartido de parámetros).

Aunque en la aplicación para secuencias sobre la red se agrega el parámetro de entrada $\boldsymbol{x}^{(t)}$ como:

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}) = f(f(\cdots f(\boldsymbol{h}^{(0)}, \boldsymbol{x}^{(1)}; \boldsymbol{\theta}) \cdots ; \boldsymbol{\theta}); \boldsymbol{\theta}) \tag{4.2}$$

En base a la figura 6.5, puede observarse el uso de una función $\tanh$, sin embargo las operaciones involucradas antes y después de esta función son importantes, y están dadas por medio de tres matrices particulares, donde la matriz $\boldsymbol{U}$ realiza conversión de los parámetros de entrada $\boldsymbol{x}^{(t)}$ a las capas ocultas de la red, la matriz $\boldsymbol{W}$ que realiza transformaciones entre una iteración de la red a la siguiente iteración, y finalmente la matriz $\boldsymbol{V}$ que realiza la transformación de las capas ocultas a la salida "sin procesar" $\boldsymbol{o}^{(t)}$, que luego pasa a ser el resultado $\hat{\boldsymbol{y}}^{(t)}$ con una operación de normalización. El proceso se da como:

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \tag{4.3}$$

$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{a}^{(t)}) \tag{4.4}$$

$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \tag{4.5}$$

$$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}(\boldsymbol{o}^{(t)}) \tag{4.6}$$

Los valores de $\boldsymbol{b}$ y $\boldsymbol{c}$ son valores de ajuste. La función $\mathrm{softmax}$ en la ecuación (4.6) normaliza los valores dados de $\boldsymbol{o}^{(t)}$ para estar en el intervalo $[0, 1]$, esto permite tratar las salidas de la red como probabilidades y también para evitar *Underfitting* por valores muy grandes.

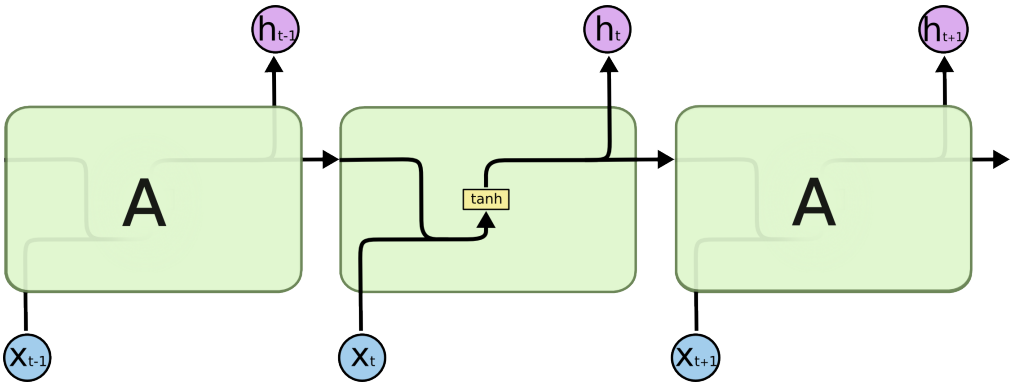


FIGURA 4.3: Red RNN clásica. Tomado de [8].

4.5.3. Redes neuronales recurrentes bidireccionales (*Bidirectional Recurrent Neural Network*)

Las redes RNN solo toman datos de forma *feedforward*, es decir, que la información de entrada y de la iteración anterior de la red solo van en una dirección, tomando solo valores del pasado en la secuencia de entrada.

En muchas aplicaciones quisiéramos tener en cuenta toda la secuencia. Es por esto que existen las redes *Bidirectional Recurrent Neural Network* (BI-RNN). Estas redes consisten en que igual a las redes RNN se toma las entradas previas de una secuencia, pero también se agrega otra red RNN en el sentido contrario de forma que la secuencia se procesa de final a inicio. Se puede entonces denotar el estado de ambas redes con $\boldsymbol{h}^{(t)}$ siendo el estado de la red RNN que va de inicio a fin, y $\boldsymbol{g}^{(t)}$ como el estado de la red RNN que va de fin a inicio, por lo que la salida $\boldsymbol{o}^{(t)}$ ahora depende de tanto entradas anteriores como posteriores en la secuencia.

4.6. Clasificación

Clasificación es la tarea de asignarle una de varias categorías predefinidas a objetos, y es una tarea que tiene una variedad extensa de aplicaciones. Ejemplos de esto se encuentran la detección de correos no deseados en mensajes de e–mails basándose del encabezado o el cuerpo del mensaje, categorización de células benignas de malignas basándose en los resultados de escaneados MRI o incluso la clasificación de galaxias basado en su forma.

Definido formalmente, clasificación es la tarea de aprender una función objetivo f que mapea cada conjunto de atributos $\boldsymbol{x}$ a una clase predefinida de etiquetas $\boldsymbol{y}$. La función objetivo también se define informalmente como un *modelo de clasificación*.

4.6.1. Sistemas Basados en Conocimiento (*Knowledge Based Systems*)

Explicar bien en que consiste la base de conocimiento y el motor de inferencia

Según [9], los *Knowledge Based Systems* (KBS) son uno de los mayores miembros de la familia de *Artificial Intelligence* (AI). El KBS consiste de una *Knowledge Base*

(KB) y un programa de búsqueda llamado *Inference Engine* (IE) representado en la figura 4.4. La KB puede ser usado como un repositorio de conocimiento de varias formas.

Existen 5 tipos de KBS, donde uno de ellos es conocido como *Expert System*, usados como *Rule–based Systems* (RBS), donde su KB esta dado como reglas y el IE esta dado por algo llamado *Working Memory* (WM), que representa los hechos que se conocen inicialmente del sistema junto con los nuevos hechos que se van dando como inferencia de las reglas.

Explicar brevemente cada uno de los 5 tipos

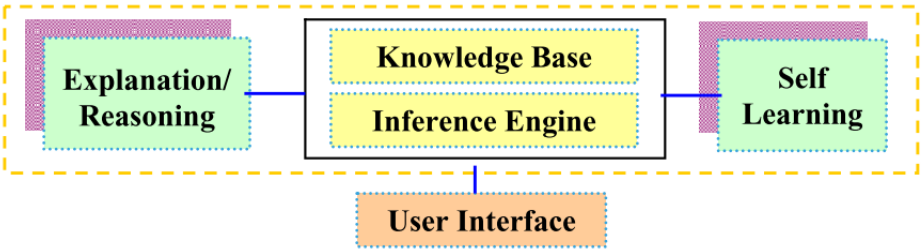


FIGURA 4.4: Arquitectura KBS. Tomado de [9]

Explicar esta figura: Que significa explanation/reasoning, self-learning y User Interface

Estas reglas pueden resumirse como una colección de condicionales de la forma **IF/ELSE** que se componen de un *antecedente* y un *consecuente*.

Existen dos tipos de RBS, definidos como *Deductive Systems* y *Reactive Systems*, donde el *Deductive Systems* tiene como objetivo realizar una conclusión en base a los hechos iniciales en la WM, por el otro lado se tienen los *Reactive Systems*, los cuales de igual manera a los *Deductive Systems*, toman los hechos de la WM y realizan sea una acción interactiva con su entorno o bien una modificación de los hechos que se encuentran en la WM tal como la adición o eliminación de hechos. Tómese el ejemplo de la ecuación (4.7) tomada de [10], donde x es la temperatura y AC es aire acondicionado.

$$\begin{cases} \text{IF x es moderado,} & \text{THEN y = ajustar AC a bajo} \\ \text{IF x es alto,} & \text{THEN y = ajustar AC a moderado a alto} \\ \text{IF x es muy alto,} & \text{THEN y = ajustar AC a alto} \end{cases} \tag{4.7}$$

Tipos de *Knowledge Based Systems*

Los cinco tipos de sistemas basados en conocimiento que existen son:

- Sistemas expertos (o *Expert Systems* en inglés)
- Sistemas de manipulación de hipertexto (o *Hypertext Manipulation Systems* en inglés)
- (o *CASE Based Systems* en inglés)
- (o *Database in conjunction with an Intelligent User Interface* en inglés)
- (o *Intelligent Tutoring Systems* en inglés)

Rule–based Systems

Por acomodar y expandir

4.6.2. Maquina de soporte vectorial (*Support Vector Machine*)

Support Vector Machine (SVM) es una técnica de clasificación que tiene sus raíces en la teoría de aprendizaje estadístico que ha mostrado resultados empíricos prometedores en muchas aplicaciones practicas, desde reconocimiento de dígitos escritos a mano a categorización de texto. SVM también funciona muy bien con datos de alta dimensionalidad. Otro aspecto destacable de esta aproximación es que representa la frontera de decisión usando un subconjunto de las muestras de entrenamiento, conocidos como los *support vectors*.

Maximum Margin Hyperplanes

Se puede entender a los *Maximum Margin Hyperplanes* como hiper-planos que ayudan a separar datos en un hiper-espacio y que poseen un margen de decisión entre los datos, como ejemplo tómese la figura 4.5, donde el hiper-plano B_1 tiene un margen de decisión mas grande que el hiper-plano B_2.

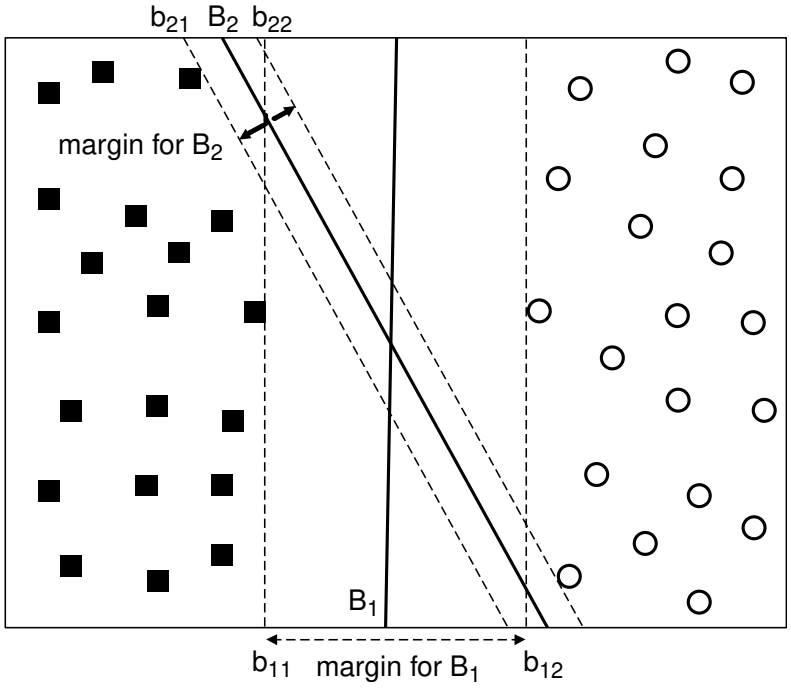


FIGURA 4.5: Maximum Margin Hyperplanes. Tomado de [6]

Finalmente, el objetivo final de los SVM es la búsqueda de un hiper-plano con el mayor margen de decisión. Existen dos tipos de SVM, el lineal y el no–lineal. El lineal realiza la separación de los datos con su hiper-plano a partir de los datos de entrada en

su espacio vectorial original, mientras que el no–lineal consta de realizar una transformación de los espacios de los datos de entrada a uno en que sea linealmente separable (véase como ejemplo la figura 4.6), sin embargo al realizar la transformación, el algoritmo de SVM se ve afectado por la dimensionalidad de la entrada, por lo que existe lo que se conoce como la función *kernel* para remediarlo.

Estas limitantes de dimensionalidad ocurren por un problema conocido como la **maldición de dimensionalidad** (o *curse of dimensionality* en inglés), y se manifiesta cuando el numero de dimensiones en los datos es alta. Particularmente se debe tener muy presente que el numero de configuraciones distintas posibles de un conjunto de variables crezcan de manera exponencial a medida que el numero de variables aumenta. Este caso ocurre con los SVMs cuando se vuelve necesaria la conversión de espacios vectoriales.

maldición de dimensionalidad

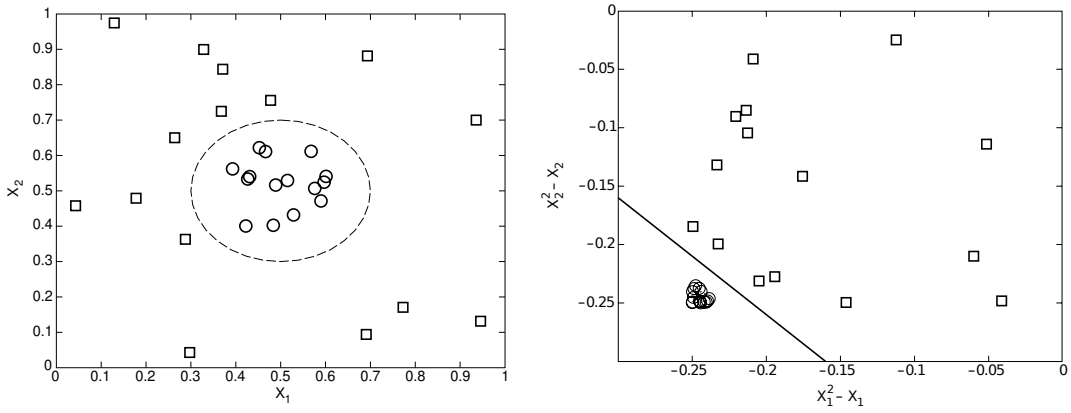


FIGURA 4.6: Transformación de espacios en *Support Vector Machine*. Tomado de [6]

Función Kernel

La función polinomial de similaridad, K, la cual es calculada en el espacio original de los datos de entrada, se le conoce como la **función Kernel**. En principio se asegura que la función kernel puede ser expresada siempre como el producto punto entre dos vectores de entrada en algún espacio de alta dimensionalidad, la función de kernel también tiene la particularidad de que el computo de los productos punto con la función toman considerablemente menos tiempo que realizar la transformación de espacios, dejando de lado la transformación, acelerando la tarea de clasificación.

función kernel

4.6.3. Clasificadores Bayesianos

En muchas aplicaciones practicas las relaciones entre un conjunto de atributos y una serie de clases son inciertas. Es decir, la clase a la que pertenece un atributo de prueba no es posible predecirlo con completa certeza incluso si el atributo es identico a alguno de entrenamiento. Esta situacion se puede presentar por ruido en el conjunto de datos o bien por factores ajenos que afectan la clasificacion pero que no fueron incluidos en el analisis del problema.

Es por esto que es necesario establecer un modelo que utilice relaciones probabilisticas entre sus atributos y sus clases. Debido a esto es util tener presente el **teorema de Bayes**, un principio estadistico que combina conocimiento previo con nueva evidencia recolectada a partir de los datos.

teorema de bayes

Teorema de Bayes

El teorema de Bayes dice que para un par de variables aleatorias x e y, donde $P(\mathrm{x} = x \mid \mathrm{y} = y)$ es la probabilidad de que la variable x tome el valor x dado que el valor de la variable y es y. Se tiene entonces la ecuación (4.8).

$$P(\mathrm{y} \mid \mathrm{x}) = \frac{P(\mathrm{x} \mid \mathrm{y})P(\mathrm{y})}{P(\mathrm{x})} \tag{4.8}$$

Usando el teorema de Bayes para clasificación Para denotar el problema de clasificación desde una perspectiva estadística, definimos a $\mathbb{X}$ y $\mathbb{C}$ como el conjunto de atributos y el conjunto de etiquetas de clase. Si las etiquetas de clase y los atributos tienen una relación incierta, entonces se puede tomar a $\mathrm{x} \in \mathbb{X}$ y $\mathrm{c} \in \mathbb{C}$ como variables aleatorias y establecemos a $P(\mathrm{c} \mid \mathrm{x})$ como su relación probabilística de que x pertenece a la clase c.

Durante la fase de entrenamiento, es necesario aprender las probabilidades de $P(\mathrm{c} \mid \mathrm{x})$ basándose únicamente en los datos recolectados del conjunto de entrenamiento.

Luego de entrenar el clasificador, la forma en que podemos determinar una clase concreta $c \in \mathbb{C}$ para un atributo concreto $\boldsymbol{x} \in \mathbb{X}$ es por medio de encontrar la combinación que nos de una mayor probabilidad de nuestro modelo, dándonos una estimación de la clase con $\hat{c}$, tal como se muestra en la ecuación (4.9).

$$\hat{c} = \arg\max_{c \in \mathbb{C}} P(\mathrm{c} = c \mid \mathrm{x} = x) \tag{4.9}$$

Clasificador Naïve Bayes

Un clasificador de Naïve Bayes estima la probabilidad condicional de las clases por medio de suponer que los atributos son condicionalmente independientes (e.g. $x_i \perp x_j \mid \mathrm{c}, \forall i, j$), dado la etiqueta de clasificación c. La suposición de independencia condicional se puede dar por la ecuación (4.10).

$$P(x_i \mid \mathrm{c}, x_j) = P(x_i \mid \mathrm{c}),\ i \neq j,\ \forall i, j \in \{1, \ldots, d\} \tag{4.10}$$

Donde cada conjunto de atributos $\mathbb{X} = \{x_1, \ldots, x_d\}$ consiste de d atributos.

Como funciona el clasificador Naïve Bayes Con la suposición de independencia condicional, en vez de computar la probabilidad condicional de clases para cada combinación de $\mathbb{X}$, solo se debe realizar para establecer la probabilidad condicional de cada x_i, dado c.

Para clasificar un dato de prueba, el clasificador computa la probabilidad posterior para cada clase y como se muestra en la ecuación (4.11).

$$P(\mathrm{c} \mid \mathbb{X}) = \frac{P(\mathrm{c}) \prod_{i=1}^{d} P(x_i \mid \mathrm{c})}{P(\mathbb{X})} \Rightarrow P(\mathrm{c}) \prod_{i=1}^{d} P(x_i \mid \mathrm{c}) \tag{4.11}$$

Puede ignorarse $P(\mathbb{X})$ debido a que es un termino constante. Para esto se realiza una normalización de forma que $\sum_{\forall c \in \mathbb{C}} P(\mathrm{c} \mid \mathbb{X}) = 1$.

4.6.4. Algoritmo de los k vecinos mas cercanos (k-*nearest neighbors*)

El algoritmo de los k vecinos mas cercanos (o k-*nearest neighbors* (KNN) en inglés) es un algoritmo no-parámetrico que tiene como objetivo encontrar los k elementos mas cercanos en un conjunto de datos de entrenamiento $\mathcal{D}$ a una entrada $\boldsymbol{x}$, cuenta cuantos miembros de cada clase están en el conjunto (cada elemento del conjunto ya tiene una clase definida), y retorna una fracción empírica como estimado. Formalmente como

$$p(y = c \mid \mathbf{x}, \mathcal{D}, k) = \frac{1}{k} \sum_{i \in N_k(\mathbf{x}, \mathcal{D})} [y_i = c] \tag{4.12}$$

donde $N_k(\boldsymbol{x}, \mathcal{D})$ son los índices de los k vecinos mas cercanos a $\mathbf{x}$ en $\mathcal{D}$ y $[\mathrm{condition}]$ es el **indicador**, que toma el valor de 1 cuando la condición es verdadera y 0 de lo contrario. indicador

Dada la probabilidad de pertenencia de una clase con la ecuación (4.12) se puede entonces estimar cual es la clase a la que pertenece un punto $\boldsymbol{x}$ por medio de estimar la clase que maximice la probabilidad que se estimo anteriormente como

$$\hat{y}(\boldsymbol{x}) = \arg\max_{c} p(y = c \mid \mathbf{x} = \boldsymbol{x}, \mathcal{D}, k) \tag{4.13}$$

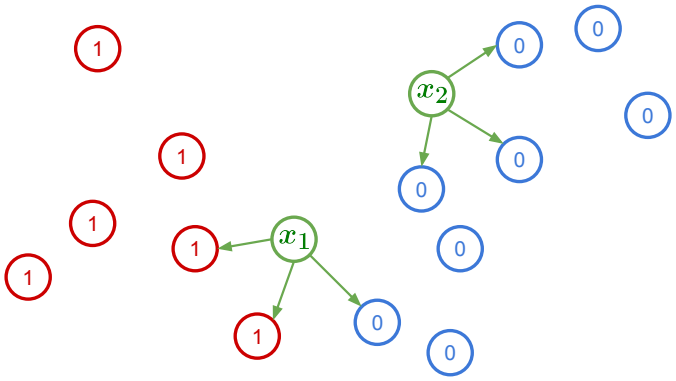


FIGURA 4.7: Ejemplo de algoritmo de k-*nearest neighbors* en 2-D para $k = 3$, los 3 vecinos mas cercanos a $\boldsymbol{x}_1$ tienen etiquetas de $1, 1, 0$, así que se predice la probabilidad $p(p = 1 \mid \mathbf{x} = \boldsymbol{x}_1, \mathcal{D}, k = 3) = 2/3$; Los 3 vecinos mas próximos a $\boldsymbol{x}_2$ tienen etiquetas de $0, 0, 0$, por lo que la probabilidad de predicción es $p(y = 1 \mid \mathbf{x} = \boldsymbol{x}_2, \mathcal{D}, k = 3) = 0/3$. Basado de [5].

Tómese como ejemplo la figura 4.8, en donde se muestra un conjunto de entrenamiento $\mathcal{D}$, las distribuciones de probabilidad dadas para las clases 1 y 2, como también una gráfica que muestra como se clasifica una muestra dependiendo de su ubicación en el espacio.

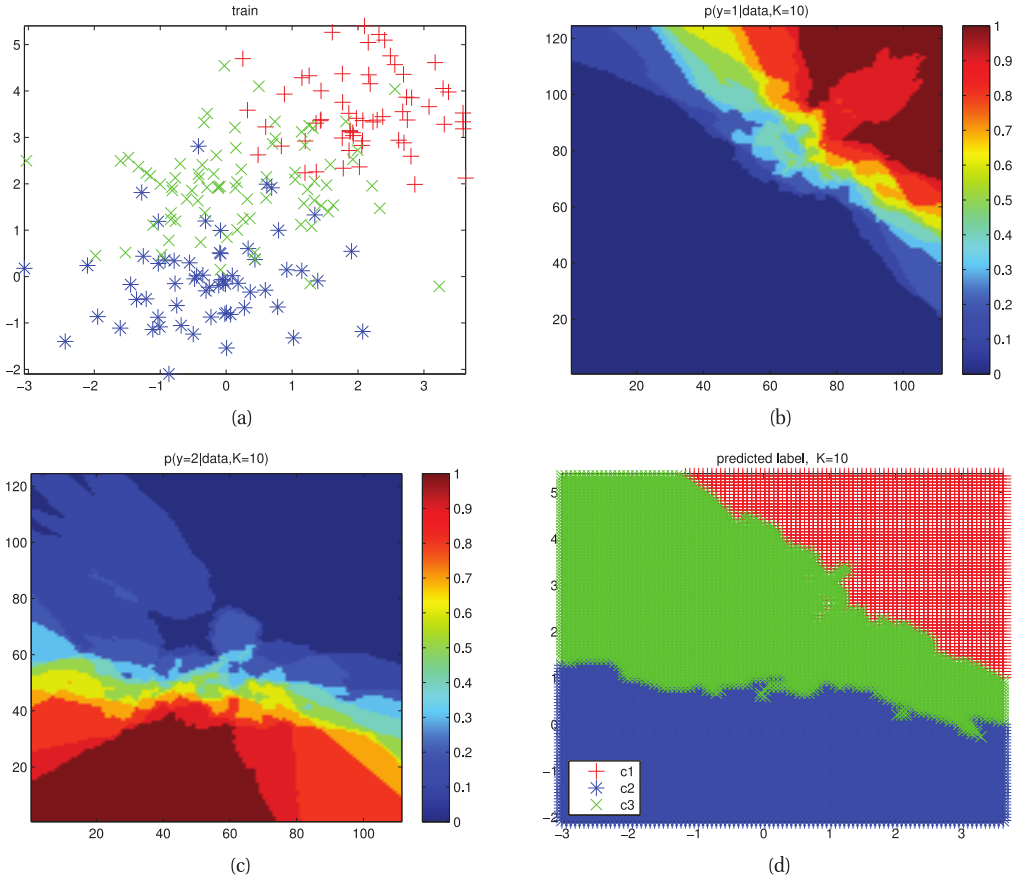


FIGURA 4.8: Comparaciones de *k-nearest neighbors*, $k = 10$. (a) Conjunto de entrenamiento $\mathcal{D}$, (b) Dibujo de $p(y = 1 \mid \mathbf{x}, \mathcal{D})$, (c) Dibujo de $p(y = 2 \mid \mathbf{x}, \mathcal{D})$, (d) Dibujo de $\hat{y}(\mathbf{x}) = \arg\max_c p(y = c \mid \mathbf{x}, \mathcal{D})$. Tomado de [5].

4.6.5. Sistemas de Detección de Anomalías (*Anomaly Detection Systems*)

Existen diferentes aproximaciones para los sistemas de anomalías, sin embargo una similitud entre todos estos sistemas es que se intenta realiza una *detección de desviaciones*, y su tarea es detectar los datos *atípicos* en un sistema [6].

Uno de los que se pueden encontrar en la literatura son los sistemas de detección de anomalías basados en realizar un estimado probabilístico con alguna distribución de probabilidad de donde para una serie de características m se trata de estimar una distribución gaussiana $\mathbf{X} \sim \mathcal{N}(\mu, \sigma^2)$ que tiene media μ y varianza σ^2 por cada característica, por lo que existirán m diferentes distribuciones .

Para estimar cada una las medias y variaciones de cada característica j, μ se estima con la ecuación (4.14) y σ^2 se estima con la ecuación (4.15).

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)} \tag{4.14}$$

$$\sigma_j^2 = \frac{1}{m} \sum_{i=1}^{m} (x_j^{(i)} - \mu_j)^2 \tag{4.15}$$

De donde para calcular la probabilidad de que una muestra se trata de una anomalía se calcula la probabilidad $p(x)$, luego de que fueron estimadas las distribuciones de cada característica del conjunto de entrenamiento, por lo que se define un ϵ de manera heurística, de forma que se determina que una muestra anómala si $p(x) < \epsilon$, la ecuación (4.16) representa cual es la probabilidad de una muestra x con una distribución gaussiana.

$$p(x) = \prod_{j=1}^{n} p(x_j, \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(- \frac{(x_j - \mu_j)^2}{2\sigma_j^2} \right) \tag{4.16}$$

Si el modelo esta dando como resultado muchos falsos positivos, lo que se debe hacer es reducir σ.

Este método de sistemas de detección de anomalía también es brevemente tratada en [2] con una representación de hiper-planos, que es equivalente con este descrito.

4.7. Clustering

El análisis de clusters agrupa objetos de datos basándose únicamente en la información encontrada en los datos que describen los objetos y sus relaciones. El objetivo es que objetos dentro de un grupo sean similares (o relacionados) el uno al otro, y que sean diferentes (o sin relación) a objetos en otros grupos. Entre mayor sea la similitud dentro de un grupo y entre mayor sea la diferencia entre grupos, sera mejor o mas distintivo el clustering.

Los métodos de clustering se hacen referencia comúnmente en ML como métodos no–supervisados. Unos de estos métodos son el algoritmo de k-means y los mapas autoorganizados, se describen en la apartado 4.7.1 y apartado 4.7.2.

4.7.1. Algoritmo de k-means

El algoritmo de k-means consiste en intentar encontrar k subconjuntos de datos disyuntos tales que la diferencia entre las características de los miembros sea lo mas disimilares. Esto lo logra a partir de definir k *centroides* inicializados aleatoriamente sobre el conjunto de datos. El valor k es puesto de antemano a la ejecución del algoritmo por el usuario. Cada elemento en el conjunto de datos es asignado al i-ésimo centroide mas cercano. Luego se recalculan los centroides a partir de ponderar todos los puntos que fueron asignados a cada centroide. Este proceso se repite hasta que el cambio de posición del centroide sea marginal. Tómese de referencia el Algoritmo 1 y la figura 4.9

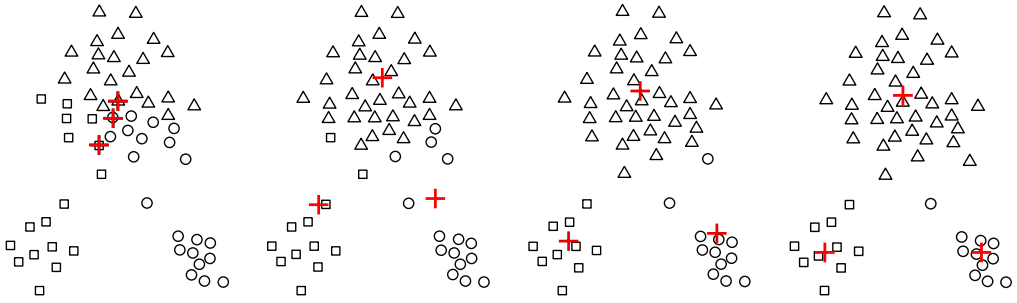
donde se pueden ver las iteraciones del algoritmo.

Algoritmo 1: k-means simplificado.

Entrada: Conjunto de datos $\boldsymbol{X} \in \mathbb{R}^{n \times m}$ y valor de k
Salida: Centroides $\{\boldsymbol{\mu}^{(1)}, \ldots, \boldsymbol{\mu}^{(k)}\}$

Inicialización aleatoria de los k puntos $\{\boldsymbol{\mu}^{(1)}, \ldots, \boldsymbol{\mu}^{(k)}\}$; $\forall j, \boldsymbol{\mu}^{(j)} \in \mathbb{R}^n$;
Define vector $\boldsymbol{h} \in \{1, \ldots, k\}^m$ representando la asignación de centroides a cada punto de $\boldsymbol{X}$;
repetir
  para todo $\boldsymbol{x}_i \in \boldsymbol{X}$ **hacer**
    $\boldsymbol{h}_i \leftarrow \arg\min_j \left\|\boldsymbol{x}_i - \boldsymbol{\mu}^{(j)}\right\|$;
  para todo $\boldsymbol{\mu}^{(j)} \in \{\boldsymbol{\mu}^{(1)}, \ldots, \boldsymbol{\mu}^{(k)}\}$ **hacer**
    $sz \leftarrow \sum_i [\boldsymbol{h}_i = j]$;
    $\boldsymbol{\mu}^{(j)} \leftarrow \frac{1}{sz} \sum_i \boldsymbol{x}_i \cdot [\boldsymbol{h}_i = j]$;
hasta que *Ningún centroide cambia (e.g.* $\left\|\boldsymbol{\mu}_t^{(i)} - \boldsymbol{\mu}_{t-1}^{(i)}\right\| \approx 0, \forall i$*)*;

Es útil mencionar que en este algoritmo puede valer la pena ejecutarlo mas de una vez. La inicialización aleatoria de los k centroides puede tener un inicio mas beneficioso para la clusterización de manera que este mejor distribuida una vez se alcanza un estado de convergencia.



(a) Iteration 1. (b) Iteration 2. (c) Iteration 3. (d) Iteration 4.

FIGURA 4.9: Iteraciones del algoritmo de k-means, $n = 2, k = 3$, centroides marcados en rojo. Tomado de [6].

4.7.2. Mapa autoorganizado (*Self-organizing Map*)

Las *Self-organizing Maps* (SOMs) son otra arquitectura de ANNs muy populares, sin embargo a diferencia de la arquitectura de las redes MLP, esta consta de una sola capa de neuronas, y no tiene un propósito de predicción, por lo que no es necesario aproximar ninguna función f a una salida $\boldsymbol{y}$ definida por un conjunto de entrenamiento.

El objetivo principal de los SOMs es de transformar una patrón de entrada m–dimensional en un mapa discreto uni– o bi–dimensional, donde sus principales características es que es un algoritmo que se basa en *Unsupervised Learning*, es *Feedforward*, tiene una sola capa de neuronas donde su propósito es realizar *Clustering* y una reducción de dimensionalidad sobre los datos de una forma topológicamente ordenada.

Los SOM tienen tres características distintivas:

- **Competencia:** por cada patrón de entrada, las neuronas en la red competirán entre ellas para determinar un ganador.
- **Cooperación:** la neurona ganadora determina la ubicación espacial (vecinos) alrededor de donde otras vecinas también se verán estimuladas.
- **Adaptación:** la neurona ganadora como también sus vecinas tendrán sus pesos asociados actualizados, y se tiene que los vecinos entre mas cerca estén del ganador, mayor es el grado de adaptación.

El algoritmo de aprendizaje de SOM parte de primero inicializar los pesos de las o neuronas con pesos aleatorios pequeños de una distribución de probabilidad aleatoria o uniforme, donde cada vector de entrada se define como $\boldsymbol{x} = [x_1, \ldots, x_m]^\top \in \mathbb{R}^m$ y la entrada general de N patrones como $\boldsymbol{X}^{m \times N}$, el vector de pesos de la neurona i es $\boldsymbol{w}_i = [w_{i1}, \ldots, w_{im}] \in \mathbb{R}^{1 \times m}$, con la matriz de pesos $\boldsymbol{W}^{o \times m}$.

Para alcanzar el objetivo de *competencia*, se realiza por cada patrón de entrada $\boldsymbol{x}_i$ una comparación con cada uno de los pesos de las o neuronas y se establece la de menor distancia $\|\boldsymbol{x}\|_p$ (típicamente la distancia Euclidiana o equivalentemente la norma L^2 e.g. $p = 2$), dejando un ganador winner, tal como en la ecuación (4.17).

$$\mathrm{winner} = \arg\min_j \|\boldsymbol{x}_i - \boldsymbol{w}_j\|_p \, ; j \in \{1, \ldots, o\} \tag{4.17}$$

Luego de establecer la neurona ganadora, se realiza el paso para alcanzar la *cooperación*, que consiste en que por medio de una función kernel h (típicamente una una distribución gaussiana), que permite establecer un área de afectación de las otras neuronas según su ubicación física en el mapa, definidos como $\boldsymbol{r}_{\mathrm{winner}}$ y $\boldsymbol{r}_j$ que son la ubicación de la neurona ganadora y la neurona vecina j, en el cual el grado de afectación de la neurona vecina depende de la distancia L^2 de la que esta de la neurona ganadora, definido en la ecuación (4.18).

$$h_{j,\mathrm{winner}}(t) = \exp\left(\frac{-\|\boldsymbol{r}_j - \boldsymbol{r}_{\mathrm{winner}}\|^2}{2\sigma(t)^2}\right) \tag{4.18}$$

Parte importante del proceso de convergencia del SOM es que a medida que avanzan las iteraciones t del algoritmo el área de afectación se va reduciendo como parte del proceso de adaptación, por lo que definimos $\sigma(t) = \sigma_0 \exp(-t/\tau_1)$, donde τ_1 es una constante heurística y σ_0 la dimensión del mapa SOM.

Finalmente para alcanzar la *adaptación* se realiza una actualización de los pesos de la matriz $\boldsymbol{W}$ en base a la influencia de área $\sigma(t)$ y de una tasa de aprendizaje $\alpha(t) = \alpha_0 \exp(-t/\tau_2)$, donde τ_2 es otra constante heurística y α_0 es una constante de aprendizaje inicial, que debe ser $0 \leq \alpha_0 \leq 1$, la actualización se describe por la ecuación (4.19) y el proceso puede ser visto gráficamente en la figura 4.10, tanto de forma uni– como bi–dimensional.

$$\boldsymbol{w}_j(t+1) = \boldsymbol{w}_j(t) + \alpha(t) h_{j,\mathrm{winner}}(t)\Big[\boldsymbol{x}_i - \boldsymbol{w}_j(t)\Big] \tag{4.19}$$

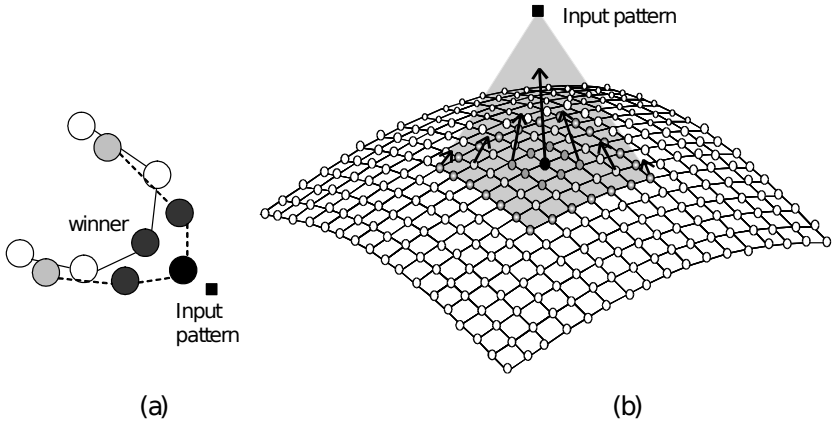


FIGURA 4.10: Proceso de adaptación de SOM, (a) uni–dimensional, (b) bi–dimensional. Tomado de [11]

Luego de que el algoritmo de aprendizaje termina de realizar las iteraciones, la salida de este es la matriz de pesos $\boldsymbol{W}$. En la figura 4.11 se puede apreciar una aproximación del algoritmo con un mapa uni–dimensional tratando de aproximar una función polar con ruido adicionado en un gráfico 2D. Adicionalmente pueden verse los efectos de *Underfitting* y *Overfitting* (véase el Apéndice B) con diferentes cantidades de neuronas en la figura 4.12.

En la figura 4.13 se puede ver una aplicación de los SOM, en donde se realiza una clusterizacion de casos de homicidios donde los parámetros son características de los homicidios, según [4] este resultado da una buena aproximación para sospechar de que estos son cometidos por personas distintas o si bien están siendo perpetrados por un mismo individuo o grupo.

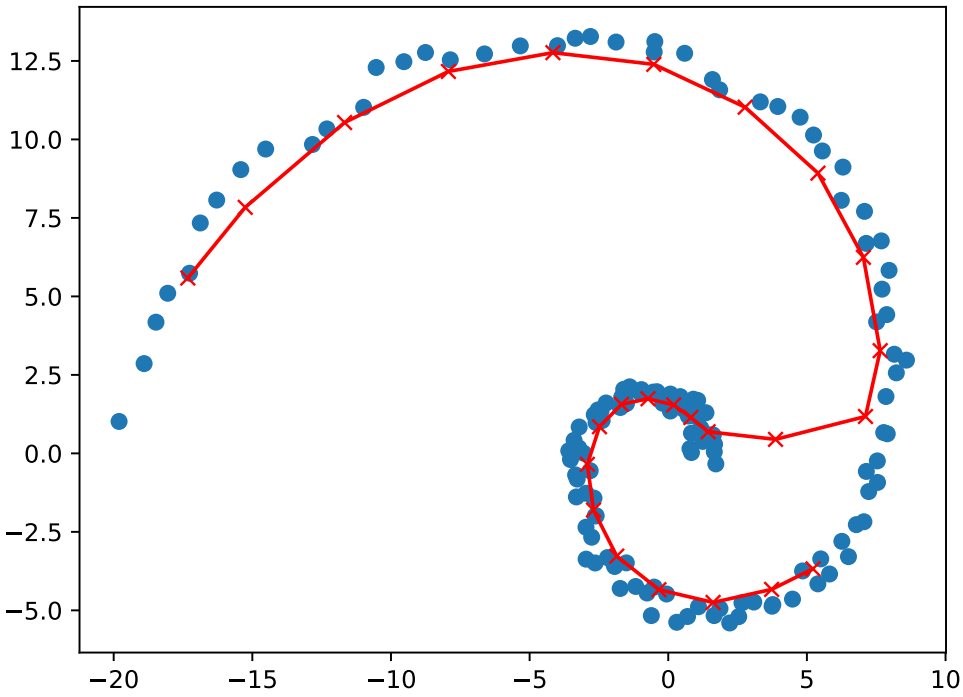
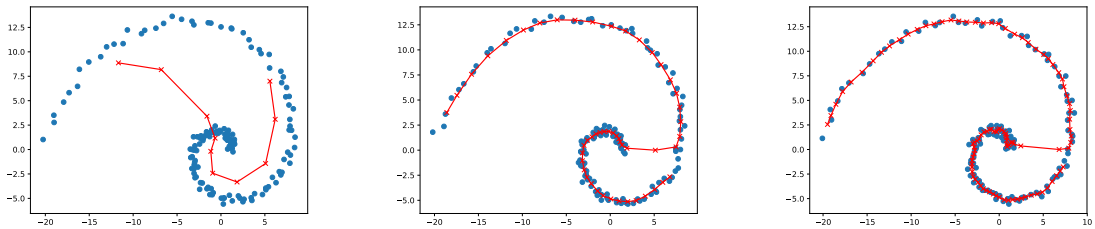


FIGURA 4.11: Ejemplo de salida de SOM uni-dimensional con 25 neuronas. Implementación propia.



(a) (b) (c)

FIGURA 4.12: Comparación de salidas de SOM uni-dimensional con (a) 10 neuronas (b) 50 neuronas y (c) 100 neuronas. Implementación propia.

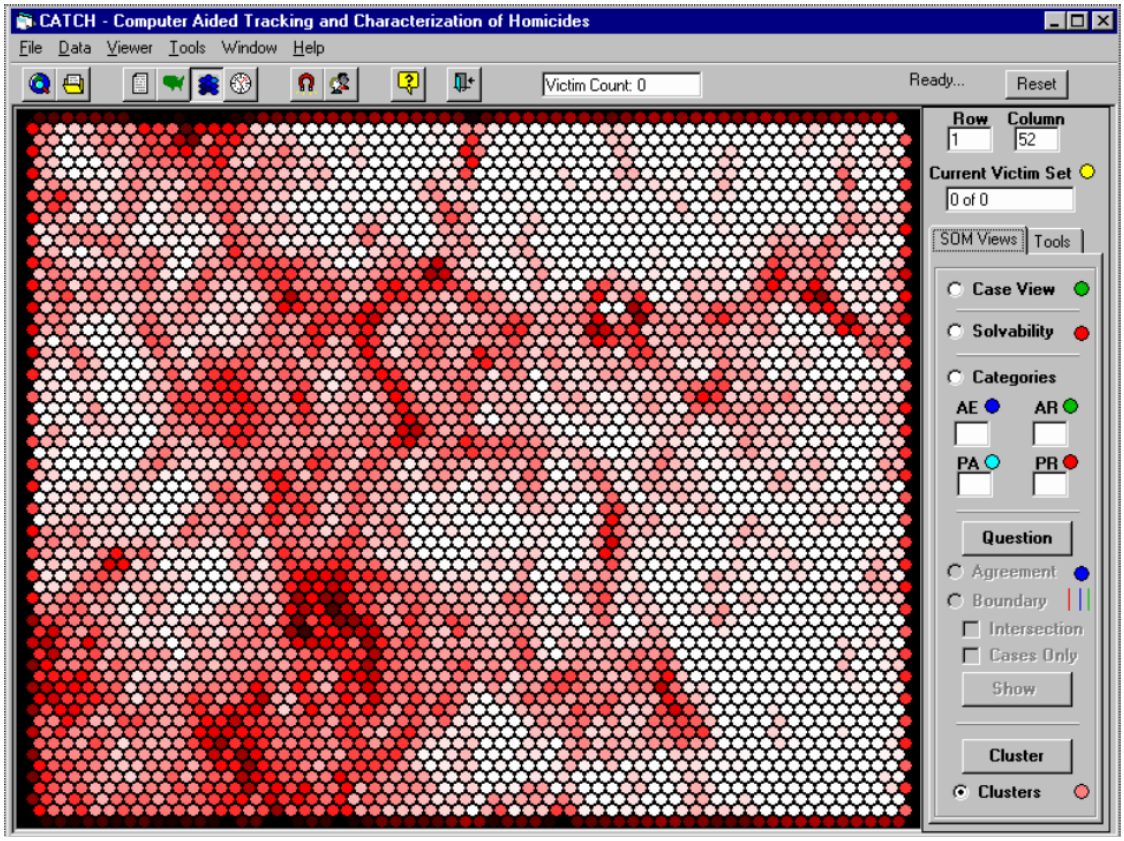


FIGURA 4.13: Ejemplo de uso de SOM en aplicaciones de perfilado. Tomado de [4]

4.8. Procesamiento de lenguaje natural (*Natural Language Processing*)

Por hacer, leer e incluir contenido de Speech and Language Processing [12]

Por hacer, adaptar la seccion de propuesta para mencionar esta seccion

El procesamiento de lenguaje natural (o *Natural Language Processing* en inglés), es el uso de lenguajes humanos, tales como el ingles, francés o español por un computador. Los programas de computador leen y emiten lenguajes especializados diseñados para un análisis eficiente y preciso por otros programas simples. Los lenguajes naturales son por lo general ambiguos y no poseen una definición formal. El procesamiento de lenguaje natural incluye aplicaciones tales como traducción de maquina, en el cual un aprendiz debe leer una sentencia en un lenguaje humano y emitir una sentencia equivalente en otro lenguaje humano. Muchas aplicaciones de NLP se basan en modelos de lenguaje que definen una distribución de probabilidad sobre secuencias de palabras, caracteres o bytes en un lenguaje natural.

Para construir un modelo eficiente de lenguaje natural, debemos usar técnicas que son especializadas para el procesamiento de datos secuenciales. Por lo general nos interesa que el lenguaje natural sea representado como secuencias de palabras mas que otras posibles representaciones. Debido a que el numero total de palabras que existen es muy grande, modelos de lenguaje basados en palabras deben operar en espacios de muy alta dimensionalidad y son discretamente dispersos.

4.8.1. n-gramas

Un **modelo de lenguaje** define una distribución de probabilidad sobre secuencias de tokens en un lenguaje natural. Dependiendo de como esta diseñado un modelo, un token puede ser una palabra, un carácter, o incluso un byte. Los tokens siempre son unidades discretas. Los primeros modelos de lenguaje exitosos se basaron en modelos de secuencias de tokens de tamaño fijo llamados n-gramas. Un n-grama es una secuencia de n tokens.

modelo de lenguaje

Modelos que se basan en n-gramas definen la probabilidad condicional del n-ésimo token dados los $n-1$ tokens anteriores. El modelo usa productos de estas distribuciones condicionales para definir la distribución de probabilidad sobre secuencias mas largas:

$$P(\mathsf{w}_1, \ldots, \mathsf{w}_\tau) = P(\mathsf{w}_1, \ldots, \mathsf{w}_{n-1}) \prod_{t=n}^{\tau} P(\mathsf{w}_t \mid \mathsf{w}_{t-n+1}, \ldots, \mathsf{w}_{t-1}) \tag{4.20}$$

Esta descomposición se justifica con la regla de la cadena de probabilidad. La distribución de probabilidad sobre la secuencia inicial $P(\mathsf{w}_1, \ldots, \mathsf{w}_{n-1})$ puede ser modelado por un modelo diferente con un valor mas pequeño de n.

Usualmente cuando se entrenan modelos de n-gramas, se están entrenando tanto un modelo de n-grama como también un modelo de $(n-1)$-grama simultáneamente. Esto vuelve sencillo el calculo de

$$P(\mathsf{w}_t \mid \mathsf{w}_{t-n+1}, \ldots, \mathsf{w}_{t-1}) = \frac{P_n(\mathsf{w}_{t-n+1}, \ldots, \mathsf{w}_t)}{P_{n-1}(\mathsf{w}_{t-n+1}, \ldots, \mathsf{w}_{t-1})} = \frac{P_n(\mathsf{w}_{t-n+1}^t)}{P_{n-1}(\mathsf{w}_{t-n+1}^{t-1})} \tag{4.21}$$

por medio de solo tener al menos dos probabilidades calculadas (dos casos base)[1].

Una limitante fundamental de estimación de máxima probabilidad sobre modelos de n-gramas es que P_n como es estimado a partir del conteo del conjunto de entrenamiento es muy probable que sea igual a cero en muchos casos, trayendo consigo dos posibles problemas.

El primero es que cuando P_{n-1} es diferente de cero pero P_n es cero, la probabilidad logarítmica es $-\infty$, una mitigación a este problema que usan muchos modelos de n-gramas es realizar un **suavizado** sobre la distribución de probabilidad, haciendo que la probabilidad de tuplas nunca vistas aumente con la probabilidad de las tuplas mas vistas.

suavizado

La segunda es que los modelos clásicos de n-gramas son particularmente vulnerables a la maldición de dimensionalidad (véase la apartado 4.6.2), debido a que existen $|\mathbb{V}|^n$ posibles n-gramas, donde $|\mathbb{V}|$ por lo general es grande. $\mathbb{V}$ es el vocabulario de palabras.

[1] Al tratarse de una recursión, existe la posibilidad de agilizar el tiempo de computo de este proceso con programación dinámica.

Entre las grandes utilidades de modelos de lenguajes con n-gramas es la de predicción de palabras [12]. Esto con la determinación de las distribuciones de probabilidad antes mencionada. Formalmente, dada una secuencia ordenada de k **unigramas** ($n = 1$), queremos determinar cual es son los unigramas con mayor probabilidad de ocurrir para formar una secuencia de $k+1$ unigramas que sea consistente con la distribución de probabilidad que determinamos del conjunto de datos. Gran parte los problemas presentes en el procesamiento de lenguaje natural implican alguna variación de este problema. Algunos de estos son reconocimiento de voz (o *speech recognition* en inglés), reconocimiento de escritura a mano (o *handwriting recognition* en inglés), siendo estos unos de los mas comunes para la predicción con probabilidades (que puede hacerse con ayuda de un estimador de máxima probabilidad, véase el Apéndice B.3.5).

unigramas

Aplicaciones que se también se benefician de estimar la máxima probabilidad de secuencias es la corrección de ortografía (o *spelling correction* en inglés), en donde queremos determinar que palabra es mas apropiada según su posición y las palabras que están presentes. Otra es la de la traducción de textos (o *machine translation* en inglés), en donde podemos tener un texto en chino, que si es traducido de palabra por palabra, probablemente no sera gramaticalmente valido, por lo que queremos una configuración (con mas o menos palabras) que sea muy próximo semánticamente a la frase original, pero correcto sintácticamente.

Suposición de Markov

Para calcular la probabilidad de una sentencia completa de n-gramas $P(\mathsf{w}_1, \ldots, \mathsf{w}_n)$ se logra típicamente con la **regla de la cadena en probabilidad**:

regla de la cadena en probabilidad

$$P(\mathsf{w}_1, \ldots, \mathsf{w}_n) = P(\mathsf{w}_1)\, P(\mathsf{w}_2 \mid \mathsf{w}_1)\, P(\mathsf{w}_3 \mid \mathsf{w}_1^2) \cdots P(\mathsf{w}_n \mid \mathsf{w}_1^{n-1}) = \prod_{k=1}^{n} P(\mathsf{w}_k \mid \mathsf{w}_1^{k-1}) \tag{4.22}$$

El modelo de **bigrama** ($n = 2$) aproxima la probabilidad de una palabra dadas todas la palabras anteriores con solo usar la probabilidad de una palabra dada la palabra inmediatamente anterior $P(\mathsf{w}_n \mid \mathsf{w}_{n-1})$, formalmente definido como

bigrama

$$P(\mathsf{w}_n \mid \mathsf{w}_1^{n-1}) \approx P(\mathsf{w}_n \mid \mathsf{w}_{n-1}) \tag{4.23}$$

Mientras que para un modelo general de n-gramas, se define como

$$P(\mathsf{w}_n \mid \mathsf{w}_1^{n-1}) \approx P(\mathsf{w}_n \mid \mathsf{w}_{n-N+1}^{n-1}) \tag{4.24}$$

La condición expuesta en la ecuación (4.24) y la ecuación (4.23) es conocida como la **suposición de Markov**. Con esta suposición podemos proceder a establecer la ecuación general de probabilidad condicional de la siguiente palabra (para *bigramas*) en una secuencia como

suposición de markov

$$P(\mathsf{w}_1^n) \approx \prod_{k=1}^{n} P(\mathsf{w}_k \mid \mathsf{w}_{k-1}) \tag{4.25}$$

Dado que el objetivos que se hablo anteriormente es la de maximizar la probabilidad de la siguiente palabra dada una secuencia de k unigramas, podemos entonces

establecerlo con la estimación de máxima probabilidad (*Maximum Likelihood Estimation* (MLE)). Sin embargo, para conseguir los estimados probabilidad de los n-gramas para el MLE se realiza un conteo de estos en el *Corpus*, normalizándola después para que las cuentas estén en $[0, 1]$ y sea considerada como una probabilidad.

Para computar la probabilidad de un bigrama para una palabra y dada una palabra anterior x, la calculamos con el conteo de los bigramas $C(\mathsf{x}\,\mathsf{y})$ y normalizamos por la suma de todos los bigramas que tienen esa misma palabra x como primera palabra:

$$P(\mathsf{w}_n \mid \mathsf{w}_{n-1}) = \frac{C(\mathsf{w}_{n-1}\,\mathsf{w}_n)}{\sum_{\mathsf{w}} C(\mathsf{w}_{n-1}\,\mathsf{w})} =^* \frac{C(\mathsf{w}_{n-1}\,\mathsf{w}_n)}{C(\mathsf{w}_{n-1})} \tag{4.26}$$

En la ecuación (4.26) la igualdad $=^*$ es la simplificación de la primera parte, debido a que el conteo de todas la palabras que contienen como inicio la primera palabra debe ser igual al conteo en general de la primera palabra.

Adicional a las palabras establecidas en una secuencia es necesario agregar tokens para indicar el inicio y el final de una secuencia, estos podemos denotarlos como `<s>` y `</s>` (según la notación expuesta en [12]) o bien como start y end en otras aplicaciones como también en este libro ahora en adelante.

4.8.2. Evaluación de modelos de lenguaje

La mejor manera para evaluar un modelo de lenguaje es por medio de incluirla en una aplicación y medir que tanto mejora su desempeño. Este tipo de evaluaciones se les conoce como **evaluación extrínseca**. Esta es la única manera de saber si una mejora en particular en un componente nos va a ayudar realmente en la tarea que estamos realizando [12]. Desafortunadamente, ejecutar sistemas NLP grandes de esta manera es muy costoso. Por otra parte, tener una métrica que nos ayude a evaluar rápidamente mejoras significativas en un modelo de lenguaje seria deseable. Una métrica de **evaluación intrínseca** es una que mide la calidad del modelo independiente de la aplicación que se le de.

evaluación extrínseca

evaluación intrínseca

Para realizar una evaluación intrínseca sobre un modelo de lenguaje es necesario un conjunto de prueba. El modelo se entrenara con un conjunto de entrenamiento (o *Corpus* en este contexto). Si usáramos una muestra de prueba que esta en el conjunto de entrenamiento le estaríamos asignando erróneamente una mayor probabilidad de la que deberíamos estar asignando, provocando una gran inexactitud en la perplejidad.

Perplejidad (*Perplexity*)

En la practica no se usa una probabilidad pura como métrica de evaluación para nuestro modelo, sino una variante llamada **perplejidad** (o *Perplexity* en inglés). La *Perplexity* $\mathcal{P}$ de un modelo de lenguaje sobre un conjunto de prueba es la probabilidad inversa del conjunto de prueba, normalizada por el numero de palabras. Para un conjunto de palabras de prueba $\mathbf{w} = \mathsf{w}_1\,\mathsf{w}_2 \ldots \mathsf{w}_N$, tenemos que la *Perplexity* se define como

perplejidad

$$\mathcal{P}(\mathbf{w}) = P(\mathsf{w}_1 \ldots \mathsf{w}_N)^{\frac{1}{N}} = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(\mathsf{w}_i \mid \mathsf{w}_{i-1})}} \tag{4.27}$$

El objetivo que queremos para nuestro modelo es que consigamos minimizar la *Perplexity*. En la ecuación (4.27) la segunda parte de la ecuación es una reorganización de la primera junto con la aplicación de la suposición de Markov.

4.8.3. Generalización y Ceros

El modelo de n-gramas, al igual que muchos otros modelos estadísticos, depende del tamaño del *Corpus*. Una implicación de esto es que las probabilidades por lo general codifican hechos específicos de un *Corpus* de entrenamiento. Otra implicación es que los n-gramas hacen un mejor trabajo a medida que el tamaño N del *Corpus* aumenta.

Existe un problema cuando se utilizan los modelos de n-gramas cuando se quiere realizar una tarea de predicción de la siguiente palabra dadas una palabras ya establecidas, y es que esta tarea es muy susceptible a que el *Corpus* de entrenamiento y de prueba sean conjuntos tomados de diferentes lugares. Esto nos ocasiona que las probabilidades que se estimen en el conjunto de pruebas cuando el modelo fue entrenado nos ocasione probabilidades iguales a 0, teniendo una *Perplexity* con valor ∞.

Existen diferentes recomendaciones para lidiar con estos problemas, una es que cuando se este entrenando nuestro modelo de n-gramas se asegure de tener conjunto de entrenamiento y de pruebas que tengan un **genero** similar a la tarea que intentamos solucionar.

genero

Otra recomendación igualmente importante es que los datos tengan un **dialecto** apropiado, especialmente cuando se manejan datos de redes sociales o transcritos verbales.

dialecto

Sin embargo asegurarnos de que se maneje un genero y dialecto apropiado en nuestros datos no es suficiente. Nuestros modelos pueden también sufrir del problema de **esparcidad** (o *sparcity* en inglés). Para cualquier n-grama que ocurre un numero suficiente de veces, podríamos tener un buen estimado de su probabilidad. Pero debido a que cualquier *Corpus* es limitado, algunas sentencias perfectamente aceptables pueden no encontrarse en el. Produciendo n-gramas con probabilidad de ocurrir igual a cero cuando no deberían.

esparcidad

Palabras desconocidas

En unos modelos de lenguaje podemos asumir que se están tratando con todas la posibles combinaciones de n-gramas que en un léxico se permiten, dejándonos con un **vocabulario cerrado** perfectamente valido. Sin embargo existen otros casos en donde tendríamos que lidiar con n-gramas que nunca hemos visto antes, que llamaríamos **palabras desconocidas** o bien palabras *Out of Vocabulary* (OOV). El porcentaje de palabras OOV que aparecen en el conjunto de prueba se le conoce como la tasa OOV (o OOV *rate* en inglés).

vocabulario cerrado

palabras desconocidas

Un sistema de **vocabulario abierto** es uno en el cual se modelan estas palabras desconocidas en el conjunto de prueba por medio de añadir una pseudo-palabra llamada `<UNK>`.

vocabulario abierto

Existen dos alternativas para tratar el problema de las palabras desconocidas:

- Escoger un vocabulario de tamaño fijo, luego convertir el problema a vocabulario cerrado reemplazando todas las palabras OOV al token `<UNK>` y finalmente estimar las probabilidades de manera habitual.

- Si no se tiene un vocabulario fijo de antemano, se crea uno de manera implícita, reemplazando palabras en los datos de entrenamiento por `<UNK>` basado en sus frecuencias (que ocurren menos de una cantidad especifica de veces, o bien establecer un tamaño fijo para el vocabulario, obtener las $|\mathbb{V}|$ palabras que mas ocurren y reemplazar todas las demás por `<UNK>`)

4.8.4. Suavizado

Para evitar que se asignen probabilidades con valor de cero a eventos no vistos, se toma un poco de probabilidad de los eventos mas vistos y se le asigna a los eventos menos o nunca antes vistos. Esta modificación a la metodología clásica se le conoce como **suavizado** o **descontado** (*discounting* en inglés). suavizado descontado

Suavizado Laplace o Suavizado Add-1

En este método todas los conteos de los tokens en nuestro modelo se aumentara por 1, haciendo que la probabilidad de estos sea mayor de cero. Sin embargo, este método no tiene un buen rendimiento en general para modelos de n-gramas, pero permite establecer una linea base de comparación.

Recuérdese la probabilidad para MLE que esta representada en la ecuación (4.26). Ahora en este método se establece la probabilidad como

$$P_{\text{Laplace}}(\mathsf{w}_n \mid \mathsf{w}_{n-1}) = \frac{C(\mathsf{w}_{n-1}\,\mathsf{w}_n) + 1}{C(\mathsf{w}_{n-1}) + |\mathbb{V}|} \tag{4.28}$$

Suavizado Add-k

Otra alternativa al método original del suavizado Laplace, es el suavizado Add-k, y consiste en desplazar un poco menos de las probabilidades de los eventos mas frecuentes a los menos frecuentes por medio de introducir una constante k. La nueva formula para calcular las probabilidades esta dada como

$$P_{\text{Add}-k}(\mathsf{w}_n \mid \mathsf{w}_{n-1}) = \frac{C(\mathsf{w}_{n-1}\,\mathsf{w}_n) + k}{C(\mathsf{w}_{n-1}) + k|\mathbb{V}|} \tag{4.29}$$

Retroceso (*Back-off*) e interpolación

Este método consiste en poder utilizar diferentes alternativas de diferentes valores de n en los n-gramas de manera simultanea. Supóngase que se estamos intentando de estimar la probabilidad $P(\mathsf{w}_n \mid \mathsf{w}_{n-2}\,\mathsf{w}_{n-1})$, pero existe al menos un **trigrama** ($n = 3$) que no existe en nuestros datos de entrenamiento, podemos entonces estimar la probabilidad de $P(\mathsf{w}_n \mid \mathsf{w}_{n-1})$. De manera similar sino tenemos un bigrama podemos retroceder a estimarlo con solo un unigrama como $P(\mathsf{w}_n)$. trigrama

Formalizándolo, se tiene entonces que en el **retroceso**, usamos un n-grama si existe evidencia suficiente, de lo contrario úsese un $(n-1)$-grama, de lo contrario sígase este procedimiento hasta que sea en el peor caso un unigrama. Es decir, solo se debe realizar un retroceso a un n-grama de menor orden si existe cero evidencia de un n-grama de mayor orden. Mientras que en **interpolación**, siempre se realiza una mezcla las estimaciones de probabilidad de todos los estimadores de los n-gramas. retroceso interpolación

En una interpolación lineal, se tiene la mezcla de todas la probabilidades de todos los n-gramas, $(n-1)$-gramas, ..., bigramas y unigramas por medio de darles un peso λ a cada estimación de probabilidad (tómese a k como el mayor numero para n que se quiere tomar, de forma que si queremos solo considerar trigramas, entonces $k = 3$):

$$\hat{P}(\mathsf{w}_n \mid \mathsf{w}_{n-k+1}^{n-1}) = \lambda_1 P(\mathsf{w}_n \mid \mathsf{w}_{n-k+1}^{n-1}) + \lambda_2 P(\mathsf{w}_n \mid \mathsf{w}_{n-k+2}^{n-1}) + \cdots + \lambda_k P(\mathsf{w}_n) \tag{4.30}$$

Simplificando la ecuación, nos resulta:

$$\hat{P}(\mathsf{w}_n \mid \mathsf{w}_{n-k+1}^{n-1}) = \sum_{i=1}^{k} \lambda_i P(\mathsf{w}_n \mid \mathsf{w}_{n-k+i}^{n-1}) \tag{4.31}$$

De forma que $\sum_i \lambda_i = 1$. Adicionalmente, para calcular los valores de λ, se debe considerar a esta como un **hiper-parámetro**, de forma que es necesario de un **conjunto de validación** para el calculo.

hiper-parámetro

conjunto de validación

Suavizado Kneser-Ney

Uno de los métodos de suavizado mas populares y de mejor rendimiento de modelos de n-gramas es el algoritmo de Kneser-Ney. Este método tiene sus raíces de un método llamado **descontado absoluto**.

descontado absoluto

Si consideramos realiza el conteo promedio de apariciones de cualquier bigrama dentro de un conjunto de entrenamiento y un conjunto de validación, donde ambos conjuntos sean del mismo tamaño y sean además de un tamaño considerable ($N \geq 20 \times 10^6$), tendremos un conteo como el expuesto en el cuadro 4.1.

Conteo de bigramas en conjunto de entrenamiento	**Conteo de bigramas en el conjunto de validación**
0	0,0000270
1	0,448
2	1,25
3	2,24
4	3,23
5	4,21
6	5,23
7	6,21
8	7,21
9	8,26

CUADRO 4.1: Comparación de conteo de bigramas en conjuntos de entrenamiento y validación. Tomado de [12].

Considérese que a excepción de los conteos del conjunto de entrenamiento con 0 y 1 repeticiones, se puede decir que todos estos puede tenerse un buen estimado de descontado de aproximadamente 0,75. El descontado absoluto formaliza esta intuición por medio de substraer un valor descontado fijo $d = 0{,}75$ para cada conteo.

Teniendo esto en cuenta, la ecuación para el descontado absoluto aplicado a bigramas es:

$$P_{\text{DescontadoAbsoluto}}(\mathsf{w}_i \mid \mathsf{w}_{i-1}) = \frac{C(\mathsf{w}_{i-1}\,\mathsf{w}_i) - d}{\sum_{\mathsf{v}} C(\mathsf{w}_{i-1}\,\mathsf{v})} + \lambda(\mathsf{w}_{i-1})P(\mathsf{w}_i) \tag{4.32}$$

El **descontado Kneser-Ney** mejora el descontado absoluto por medio de utilizar una manera mas sofisticada de lidiar con la distribución de unigramas de bajo orden. Esto lo logra por medio de introducir $P_{\text{continuation}}$ donde pretende responder la pregunta de "¿Que tan probable es que w ocurra como una continuación?", y se basa de el numero de contextos diferentes en los que aparece w. Así suponemos que palabras que han aparecido en mas contexto en el pasado son mas probables de ocurrir en algún contexto nuevo también. De esta manera se define a esta formula de continuación como:

descontado kneser-ney

$$P_{\text{continuation}} = \frac{|\{\mathsf{v} : C(\mathsf{v}\,\mathsf{w}) > 0\}|}{|\{(\mathsf{u}', \mathsf{w}') : C(\mathsf{u}'\,\mathsf{w}') > 0\}|} \tag{4.33}$$

La forma final de la ecuación de **interpolado de Kneser-Ney** para bigramas esta dado como:

interpolado de kneser-ney

$$P_{\text{KN}}(\mathsf{w}_i \mid \mathsf{w}_{i-1}) = \frac{\text{máx}(C(\mathsf{w}_{i-1}\,\mathsf{w}_i) - d, 0)}{C(\mathsf{w}_{i-1})} + \lambda(\mathsf{w}_{i-1})P_{\text{continuation}}(\mathsf{w}_i) \tag{4.34}$$

Donde λ es una constante que normaliza para distribuir la masa de probabilidad:

$$\lambda(\mathsf{w}_{i-1}) = \frac{d}{\sum_{\mathsf{v}} C(\mathsf{w}_{i-1}\,\mathsf{v})}|\{\mathsf{w} : C(\mathsf{w}_{i-1}\,\mathsf{w}) > 0\}| \tag{4.35}$$

Sin embargo, si queremos una forma general de la formula para cualquier n-grama, tenemos la siguiente formula recursiva:

$$P_{\text{KN}}(\mathsf{w}_i \mid \mathsf{w}_{i-n+1}^{i-1}) = \frac{\text{máx}(C_{\text{KN}}(\mathsf{w}_{i-n+1}^{i}) - d, 0)}{\sum_{\mathsf{v}} C_{\text{KN}}(\mathsf{w}_{i-n+1}^{i-1}\,\mathsf{v})} + \lambda(\mathsf{w}_{i-n+1}^{i-1})P_{\text{KN}}(\mathsf{w}_i \mid \mathsf{w}_{i-n+2}^{i-1}) \tag{4.36}$$

Donde su caso base es:

$$P_{\text{KN}}(\mathsf{w}) = \frac{\text{máx}(C_{\text{KN}}(\mathsf{w}) - d, 0)}{\sum_{\mathsf{w}'} C_{\text{KN}}(\mathsf{w}')} + \lambda(\epsilon)\frac{1}{|\mathbb{V}|} \tag{4.37}$$

Donde ϵ se considera el token (o en este caso palabra) vació. Y la definición de C_{KN} es:

$$C_{\text{KN}}(\cdot) = \begin{cases} C(\cdot) & \text{para los } n\text{-gramas de mayor orden} \\ C_{\text{continuation}}(\cdot) & \text{para los } n\text{-gramas de menor orden} \end{cases} \tag{4.38}$$

4.8.5. Representaciones de palabras y documentos como vectores

Por lo general cualquier modelo de AI, incluyendo los que se encuentran en ML y NLP deben utilizar internamente representaciones escalares, vectores, matrices y tensores. NLP por lo tanto debe utilizar algunas de estas para alimentar sus modelos, es por esto que es importante hablar de las diferentes representaciones que se les da a los diferentes elementos fundamentales, tales como los n-gramas (véase la apartado 4.8.1). Existen diferentes representaciones que se les puede dar a las palabras, todas con usos

diferentes, ventajas y desventajas, en particular interés están *Bag of Words*, *Term Frequency – Inverse Document Frequency* y embeddings de palabras.

Bag of Words

La representación de vectores *Bag of Words* (BOW) consiste en representar cada palabra dentro de un vocabulario $\mathbb{V}$ de forma que a cada palabra se le asigna un identificador inversamente proporcional a la cantidad de apariciones dentro de un *Corpus*, todos en el intervalo $1, \ldots, |\mathbb{V}|$. Cada vector se codifica de manera que para la i-ésima palabra (según la asignación dada por sus repeticiones) se le da el valor en vector de:

$$\boldsymbol{w} = \begin{bmatrix} 1 & & i & & |\mathbb{V}| \\ 0 & \ldots & 1 & \ldots & 0 \end{bmatrix}^\top$$

De modo que el vector solo tendrá un 1 en la posición i-ésima y ceros en el resto del vector. La notación que se usara para representar este tipo de vectores es con $\boldsymbol{e}^{(i)}$, a veces llamado como **vector base estándar**.

vector base estándar

Esta representación permite también codificar un conjunto de palabras presentes dentro de una secuencia $\mathbf{w} \in \mathbb{V}^k$ de k n-gramas por medio de realizar la suma vectorial $\boldsymbol{s}$ de cada representación, dado como:

$$\boldsymbol{s} = \sum_{\mathsf{w}_i \in \mathbf{w}} \boldsymbol{e}^{(i)} \tag{4.39}$$

Term Frequency – Inverse Document Frequency

Otra representación común en NLP es la de *Term Frequency – Inverse Document Frequency* (TF-IDF), que se divide en dos partes, expresadas en la ecuación (4.40), la ecuación (4.41) y la composición de ambas en la ecuación (4.42), donde $\mathcal{D}$ es el *Corpus* de palabras. Este es un peso asignado consiste en penalizar palabras que ocurren mucho en un documento pero no mucho en el *Corpus* o bien de penalizar la palabras que se repiten poco en un documento pero se repiten mucho en el *Corpus*, por lo que los puntos medios entre ambos tienen los mayores valores.

$$\mathrm{tf}(\mathsf{t}, \mathbf{d}) = \text{Frecuencia del } n\text{-grama } \mathsf{t} \text{ en el documento } \mathbf{d} \tag{4.40}$$

Existen diferentes variaciones para representar el conteo de términos **tf** de forma normalizada, como se representa en el cuadro 4.2.

Esquema	**Peso de tf**
Binario	$0, 1$
Conteo directo	$f_{\mathsf{t},\mathbf{d}}$
Frecuencia de términos	$f_{\mathsf{t},\mathbf{d}} / \sum_{\mathsf{t}' \in \mathbf{d}} f_{\mathsf{t}',\mathbf{d}}$
Normalización logarítmica	$1 + \log(f_{\mathsf{t},\mathbf{d}})$
Normalización k doble	$k + (1-k)\frac{f_{\mathsf{t},\mathbf{d}}}{\max_{\mathsf{t}' \in \mathbf{d}} f_{\mathsf{t}',\mathbf{d}}}$

CUADRO 4.2: Variaciones de tf

$$\text{idf}(\mathsf{t}, \mathcal{D}) = \log\left(\frac{N}{|\{\mathbf{d} \in \mathcal{D} : \mathsf{t} \in \mathbf{d}\}|}\right); N = |\mathcal{D}| \tag{4.41}$$

Otras variaciones de idf se encuentran en el cuadro 4.3, donde $n_{\mathsf{t}} = |\{\mathbf{d} \in \mathcal{D} : \mathsf{t} \in \mathbf{d}\}|$.

Esquema	**Peso de tf**
Unitario	1
idf suavizada	$\log(\frac{N}{1+n_{\mathsf{t}}})$
idf máxima	$\log(\frac{\max_{\mathsf{t}' \in \mathbf{d}} n_{\mathsf{t}'}}{1+n_{\mathsf{t}}})$
idf probabilística	$\log(\frac{N-n_{\mathsf{t}}}{n_{\mathsf{t}}})$

CUADRO 4.3: Variaciones de idf

$$\text{tfidf}(\mathsf{t}, \mathbf{d}, \mathcal{D}) = \text{tf}(\mathsf{t}, \mathbf{d}) \cdot \text{idf}(\mathsf{t}, \mathcal{D}) \tag{4.42}$$

Esta puntuación no se coloca directamente en vectores que irían a representar palabras individuales, sino que representan típicamente a documentos completos que tienen un vocabulario $\mathbb{V}$ en común. Esta representación es muy utilizada para encontrar similitud entre documentos. Formalmente definimos el vector $\boldsymbol{d}$ representativo de un documento $\mathbf{d}$ tomado de un *Corpus* $\mathcal{D}$ e $\text{idx}(\cdot)$ como el índice asignado de un termino $\mathsf{t} \in \mathbf{d}$ en la siguiente ecuación:

$$\boldsymbol{d} = \sum_{\mathsf{t} \in \mathbf{d}} \text{tfidf}(\mathsf{t}, \mathbf{d}, \mathcal{D}) \boldsymbol{e}^{(\text{idx}(\mathsf{t}))} \tag{4.43}$$

El vector $\boldsymbol{d}$ esta en $\mathbb{R}^{|\mathbb{V}|}$, por consecuencia $\boldsymbol{e}^{(\cdot)}$ también tiene la misma dimensión en este contexto. También se tiene que $\text{idx}(\cdot)$ concierne a índices dentro de $\mathcal{D}$, por lo que su rango es $\{1, \ldots, |\mathbb{V}|\}$.

Modelos de lenguaje neural (*Neural Language Model*)

Los modelos de lenguaje neural (o *Neural Language Model* (NLM) en inglés), son modelos de lenguaje diseñados para mitigar el problema de la *maldición de dimensionalidad* para el modelado de secuencias de lenguaje natural por medio de utilizar una representación distribuida de palabras. Estos modelos comparten fortalezas estadísticas entre una palabra (y su contexto) a otras palabras similares y contextos.

Algunas veces se les llama a estas representaciones de palabras como **embeddings de palabras** (o *word embeddings* en inglés). En esta interpretación, se representan en un espacio de características de dimensión menor al tamaño del vocabulario. En una representación tal como en BOW (a veces referidos como vectores "one-hot"), la distancia entre una representación de palabra con cualquier otra tiene una distancia de $\sqrt{2}$. En contraste, las representaciones de palabras en NLM pretenden que la distancia entre representaciones de palabras que ocurren en contextos similares sea pequeña. Esto por lo general produce que palabras con significados similares sean consideradas vecinos por su cercanía.

embeddings de palabras

Este tipo de representaciones son particularmente útiles para modelos de redes neuronales, debido a que estas trabajan de manera "nativa" con vectores en espacios dimensionales moderadamente grandes, mientras que un vector *one-hot* no permite crear una diferencia de semántica de las secuencias de texto. Particularmente es deseable para redes neuronales (especialmente con redes MLP) que los vectores de entrada no sean tan grandes debido a que pueden ocasionar *Overfitting*, dificultando optimizar nuestro modelo para que generalice.

Similaridad de embeddings de palabras o documentos

Para definir la similaridad entre dos términos u y v, es necesario establecer una métrica entre estos de forma que nos den una estimación de su similaridad. La métrica de similaridad mas usada es el coseno del ángulo entre ambos vectores representativos de ambas palabras, representada en la ecuación (4.44).

$$\text{similaridad}(\mathsf{u}, \mathsf{v}) = \cos(\theta) = \frac{\boldsymbol{u} \times \boldsymbol{v}}{\|\boldsymbol{u}\| \, \|\boldsymbol{v}\|} \tag{4.44}$$

Donde $\boldsymbol{u}$ y $\boldsymbol{v}$ son la representación en vector de los términos u y v respectivamente. No existe diferencia en el calculo cuando se realiza una estimación de similaridad entre dos documentos de un *Corpus* $\mathcal{D}$.

4.8.6. Etiquetado de *Part-of-Speech*

Por hacer, pagina 151 Speech and Language Processing [12]

4.8.7. *Hidden Markov Model*

Por hacer, pagina 312 y 603 Machine Learning [5]

Por hacer, pagina 464 Speech and Language Processing [12]

Capítulo 5

Estado del arte

Dentro de la literatura se encuentran diversos métodos específicamente para el perfilamiento de cibercriminales, como también métodos genéricos para el análisis de la información, aquí se exponen varios trabajos relacionados con estas tareas.

En [13] se propone un método de análisis automático de forums del bajo mundo. Ellos usan procesamiento de lenguaje natural y *Machine Learning* para generación de información de alto nivel sobre foros del bajo mundo, primero identificando publicaciones que involucran transacciones y luego extrayendo los productos y precios. También demuestran como un analista puede usar estas metodologías automatizadas para investigar otras categorías de productos y transacciones.

En [14] se introducen embeddings de palabras bilingües, es decir, representaciones de palabras asociadas entre dos lenguajes en el contexto de modelos de lenguaje neuronal. Ellos proponen un metodo para el aprendizaje de un gran corpus sin etiquetas. Estos nuevos embeddings mejoran significativamente en similaridad semántica.

En [15] desarrollan un método de minería de redes de cibercriminales poco supervisado para facilitar la forensia de cibercrimenes. El método propuesto es un modelo generativo probabilístico por un algoritmo de muestreo sensible al contexto y muestran un mejora significativa respecto a un método con *Latent Dirichlet Allocation* (LDA) y otro método basado en SVMs.

En [16] se realiza una visualización de categorías sacadas de Wikipedia de forma que se muestran sus relaciones con ayuda del meta-modelo SOM, ayudando así realizar una reducción de los espacios de búsqueda. Por medio de la selección de una neurona especifica era posible la obtención de categorías conceptualmente similares. La evaluación de las activaciones de neuronas individuales indicaban que formaban patrones de forma coherente que podrían ser útiles en la construcción de interfaces de usuario para la navegación sobre estructuras categóricas.

En [17] exploran diversas técnicas computacionales para realizar el *Clustering* de grupos de lenguajes con la categorización de participantes de foros de cibercriminales. Ellos hacen uso de un modelo de red neuronal de lenguaje para generar representaciones de vectores de tamaño fijo de mensajes publicados por los participantes de los foros. Ellos afirman que sus resultados muestran que Vectores de párrafos superan a las aproximaciones tradicionales de frecuencia de n–gramas para generar embeddings de documentos que son útiles en la clusterizacion de cibercriminales en grupos de lenguajes.

En [18] realizan clusterización de documentos HTML con lógica difusa, de donde

describen que en los resultados experimentales muestran una mejora significativa respecto a modelos de espacio vectorial con funciones tradicionales de pesado de parámetros en un dataset estándar de pruebas.

Capítulo 6

Propuesta de aplicación

La propuesta consta de a aplicación de modelos de NLP para el desarrollo de labores de inteligencia en escenarios de ciberterrorismo por medio de 3 modelos, pensados para el análisis de texto en redes sociales como Twitter en aras de realizar un perfilado de ciber-criminales potenciales. Esta propuesta estará fundamentada en varias tecnologías identificadas en el Estado-del-Arte.

Para esto se sigue el ciclo *Cross Industry Standard Process for Data Mining* (CRISP-DM), el cual contiene 6 fases, y 4 de las cuales fueron abordadas en este proyecto:

explicar por que no se siguieron las otras 2 fases

1. Entendimiento del negocio (*Business understanding*)
2. Adquisición de datos (*Data acquisition*)
3. Modelamiento (*Modelling*)
4. Despliegue (*Deployment*)

6.1. Entendimiento del negocio (Business understanding)

En el proceso de perfilado una tarea muy importante se basa en la búsqueda de pistas (como diversos ejemplos dados en [4]), por lo que la mayor parte de este proceso confiere de tratar de encontrar patrones que confieran algún tipo de relación entre lo que es un actor y una acción, si bien no necesariamente de forma directa. Es por esto que técnicas como las que se encuentran en la minería de texto (véase la apartado 4.4.1) confieren una posibilidad de realizar muchas de estas labores, tales como el *Clustering* (véase la apartado 4.7) de elementos, como el caso de la figura 4.13 en la apartado 4.7.2.

La necesidad cae en las agencias de seguridad que requieren de estar constantemente buscando posibles relaciones entre ataques pasados o futuros, y por tanto nuevas herramientas que permitan obtener nuevas habilidades en la esta labor son esenciales.

Finalmente, metodologías de análisis de textos a partir de fuentes abiertas, tales como en [2], permitirían no solo obtener la información, sino que tener una forma de analizarla e incluso de predecir relaciones a partir del contenido que no es explícito.

6.2. Adquisición de datos (Data acquisition)

Para esta labor de adquisición de datos, la intención es de recolectar datos de diversas fuentes, sin embargo para el alcance del proyecto se proponen dos metodologías de obtención de información de fuentes abiertas que serán explicadas en las subsecciones 6.2.1 y 6.2.2.

6.2.1. Obtención de datos con API de Twitter

Twitter permite el acceso de datos de su plataforma por medio de códigos de autorización provistos en ella, que luego son usados para obtener la información con criterios definidos por el usuario.

indicar cuales fueron los criterios de selección de tweets

Petición de autorización en Twitter

Para realizar una petición de datos de acceso a la plataforma es necesario crear una "App" por medio de visitar la pagina de Administración de aplicaciones (o *Application Management* en ingles). Se necesita una cuenta de Twitter previamente creada.

Luego se procede a crear la App, para esto sera necesario proveer información respecto a que se hará con la aplicación y que datos requerirá esta. En este proceso es importante tener en cuenta que debido al uso masivo de información dentro de Twitter, este ha puesto restricciones al acceso por temas de privacidad e influencia, es por esto que el proceso para pedir autorización de crear una nueva aplicación debe incluir la mayor cantidad de información que provea la intención del uso. Luego de llevar a cabo la solicitud luego de aproximadamente un día le sera notificado si fue autorizado para utilizar la plataforma.

de donde? de la cuenta de un usuario regular de twitter

En este punto podrá crear la aplicación y generar los tokens de autorización, de los cuales requerirá de:

- `CONSUMER_KEY`
- `CONSUMER_SECRET`
- `ACCESS_TOKEN`
- `ACCESS_SECRET`

Esta petición de datos se realiza por medio de una consulta `GET` donde la URI provee los datos como los codigos de autorización.

Dentro del directorio de Software (véase el Apéndice A) se incluye un programa que permite la obtención de tweets con todos los meta-datos ofrecidos por la plataforma, de manera que obtiene tweets con temas relacionados especificados en un archivo de texto externo.

6.2.2. Obtención de datos con datasets públicos en Archive

Otra posibilidad de obtener datos en una plataforma abierta es la de la pagina web de ARCHIVE[1].

En esta plataforma se puede descargar meses de contenido obtenido de cualquier origen en Twitter en el tiempo de un mes, aunque este información no fue obtenida con niveles altos de precisión, es decir, no se obtuvieron todos los tweets que ocurrieron durante ese mes ni tampoco fueron obtenidos en un ambiente de alta disponibilidad, por lo que no todos los tweets en ese instante fueron recolectados.

Una forma recomendada de descargar los datos de ARCHIVE es por medio de *torrents*, que permite una descarga fiable del contenido comprimido de ese mes[2] ya que

Indicar por favor las restricciones que impone twitter en cuanto a numero de twits descargados o ventana de descarga, en esta forma de descaga gratuita. Indicar tambien las opcio-

[1] https://archive.org/search.php?query=collection%3Atwitterstream&sort=-publicdate&page=2
[2] Cada mes pesa aproximadamente 50GB

además provee comprobación de integridad y reintento de descarga de secciones de archivos que se encuentran corruptas. Un motor de torrents Open Source es el de qBittorrent[3], y tiene implementaciones en Windows, MacOS y Linux.

Luego de haber descargado todo el archivo comprimido de tweets se puede proceder a descomprimirlo, de donde se encontrarán archivos comprimidos mas pequeños divididos por día del mes, sin embargo téngase en cuenta que descomprimir estos archivos puede llegar a pesar 500GB, por lo que una alternativa mas conveniente es de realizar un filtrado de los datos relevantes que se necesiten de manera individualpor cada uno de esos archivos, de esta forma solo se tendría la información requerida, en vez de ruido. Para esto existe una aplicación incluida en la carpeta de Software de este proyecto con un posible uso (véase el Apéndice A), donde también se hace uso de [19] para el procesamiento mas eficiente.

este filtrado se haría antes de descargar los archivos de ARCHIVE? especificar como se puede hacer

6.3. Modelamiento (Modelling)

Como parte de la propuesta se proponen 3 modelos para tratar diferentes necesidades los cuales estan representados en la figura 6.1.

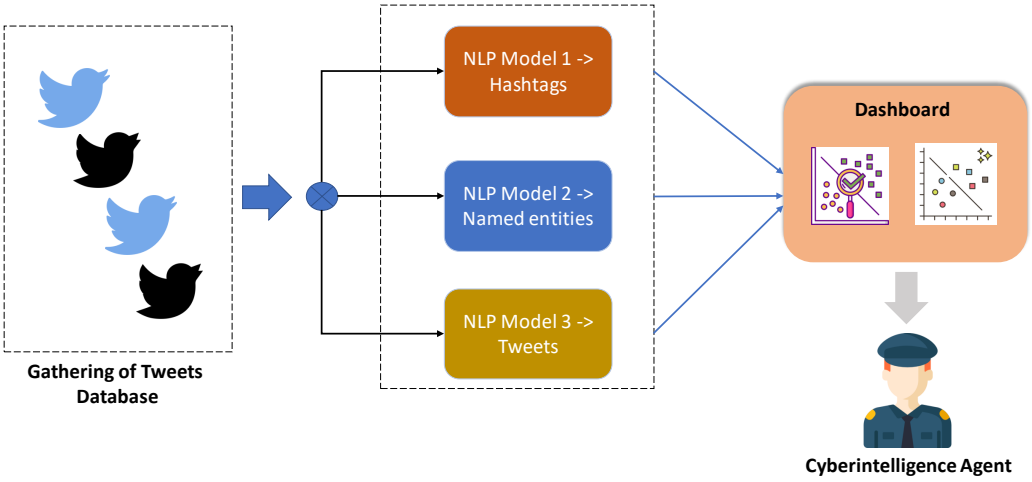


FIGURA 6.1: Arquitectura de propuesta.

6.3.1. Modelo 1: Predicción de etiquetas de Twitter con modelos lineales

En Twitter, las publicaciones que se realizan tienen la posibilidad de incluir menciones de temas de tendencia por el conocido *hashtag*, escrito como `#Tema`, y tiene la gran utilidad de realizar una mención explicita del tema que se quiere tratar y además facilita la tarea de encontrar textos directamente relacionados con un tema.

[3] https://www.qbittorrent.org/

Lo que se intenta solucionar con este modelo es de predecir que *hashtags* deberían estar presentes en un tweet basado en la combinación de palabras que se encuentran dentro. Para esto se propone un modelo lineal que usa regresión logística y realizar una multi-clasificación con las clases siendo los *hashtags* basándose del texto. Esto esta pensado con el propósito de ayudar al agente de inteligencia a identificar a que posibles tendencias pertenece un tweets de manera que pueda reconocer si es de un tema relevante para la seguridad.

Así mismo, en la literatura de NLP es muy común el uso de diferentes representaciones de palabras o conjuntos de palabras. Una representación de palabras típicas son los BOW, donde se establece un diccionario de palabras de tamaño N, y donde cada palabra tiene un vector que la representa. A cada palabra se le asigna un identificador único (vector) en ese diccionario, por lo que existiría una traducción de palabra a vector y cada palabra puede ser recuperada por medio del índice, como se muestra en la ecuación (6.1) y la ecuación (6.2).

$$\text{word2idx} = \Big\{(\text{word}_i, i) : \forall i \in \{1, \ldots, N\}\Big\} \tag{6.1}$$

$$\text{idx2word} = \Big[\text{word}_i\Big], \forall i \in \{1, \ldots, N\} \tag{6.2}$$

El lugar de donde son tomadas la palabras para representar al BOW están contenidas en un conjunto de palabras conocido como el *Corpus* de palabras, de esa misma manera también es posible tener *Corpus* de otros elementos como sentencias de texto (mas de una palabra compuesta), letras individuales, entre otras.

Actualizar seccion

Creación del *Corpus* de palabras

Debido a que los textos que se encuentran en Twitter no tienen forma estructurada es necesario realizar un preprocesamiento de cada post recolectado para luego contar la frecuencia de cada palabra dentro del *Corpus* y así establecer las primeras N palabras mas usadas que van a componer el *Corpus* de palabras.

Para realizar el preprocesamiento de cada texto se hacen los siguientes pasos:

1. Convertir todas las palabras a minúscula (e.g. “LaTeX” → “latex”)
2. Reemplazar todos los caracteres especiales de texto a espacios en blanco (e.g. “`@;,:\n\t\r`” → “␣␣␣␣␣␣␣”)
3. Remover todos los símbolos extraños, es decir todo lo que no sean numeros, ni letras ni los simbolos que se encuentran normalmente en tweets (e.g. “`%()&$!^`” → “”)
4. Remover todas las *stopwords*, que son palabras que no añaden ningún valor semántico al texto
   (e.g. “las palabras son una forma de expresarnos” → “palabras forma expresarnos”)

Luego se realiza un conteo de todas la palabras presentes dentro del *Corpus* de donde se sacan las N primeras palabras para incluirlas en el BOW. N es definido heurísticamente por el usuario.

Luego de esto se generan las etiquetas (o *tags* en ingles), equivalente a este contexto como los hashtags que se toman directamente de los textos de entrenamiento, de estos también se les realiza un conteo, que servirán para la predicción de las etiquetas según el contenido del texto.

Se realiza un conteo al texto de entrenamiento?

Conversión de textos a vectores

Para realizar la conversión de textos a vectores y así poder representar un texto como un vector se procede a primera realizar una generación de identificadores para cada palabra de manera como se describió en el inicio de la apartado 6.3.1.

La manera en que se representa un texto en forma de vector $\boldsymbol{s}$ es por medio de la sumatoria de los vectores que representan cada palabra como se representa en la ecuación (6.3), recuérdese que $\boldsymbol{e}^{(i)}$ es un vector con un 1 en la posición i y ceros en el resto de posiciones.

$$\boldsymbol{s} = \sum_{(\text{word},i)\in\text{word2idx}} \boldsymbol{e}^{(i)}, \text{word} \in d \tag{6.3}$$

Generación del conjunto de entrenamiento, validación y pruebas

Para la generación de los conjuntosse toman las sumatorias generadas de cada tweet en su forma de vector y se coloca en una matriz de $\boldsymbol{T}^{m\times N}$, donde m son el numero de muestras de Twitter y N el tamaño del *Corpus*.

cada conjunto debería tener un tamaño diferente, cierto? por favor explicar como se generan esos conjuntos

porque m y N?

Clasificador de regresión logística

La regresion logistica hace parte de uno de los algoritmos mas importantes en AI, esta consiste en procesar la entrada de un modelo, que para el caso actual es lineal, que se procesa con una serie de hiper-parámetros que se denominara $\boldsymbol{\theta}$ y la entrada del modelo como $\boldsymbol{x}$. Se tiene que al realizar una regresión con este modelo se calcula $\boldsymbol{\theta}^\top \boldsymbol{x}$, que da como resultado un valor en $\mathbb{R}$ con rango indefinido. La regresión logística simbolizada como $\sigma(x)$, conocida como la función *sigmoide* que es una función que para una entrada x de dominio $(-\infty, \infty)$ se tiene un rango de $(0, 1)$. La ecuación (6.4) define la función *sigmoide*.

$$\sigma(x) = \frac{1}{1+\exp(-x)} \tag{6.4}$$

Equivalentemente, la regresión logística con modelos lineales se calculan como $\sigma(\boldsymbol{\theta}^\top \boldsymbol{x})$ y permite realizar un suavizado de la regresión de forma que mitiga parte de los problemas de *Overfitting* y *Underfitting*, como se muestra en la figura 6.2.

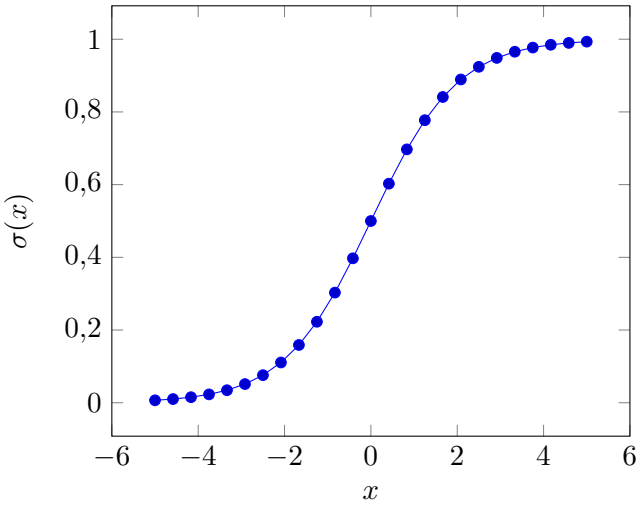


FIGURA 6.2: Gráfica de función sigmoide.

Multiclasificador de One vs Rest

La multiclasificacion como es brevemente introducido en la apartado 4.3, permite realizar una clasificación en C clases con una entrada $\boldsymbol{x}$. El multiclasificador de *One vs Rest* realiza la clasificación por medio de distinguir la separación de una muestra $\boldsymbol{x}_i$ en una clase $c \in \{1, \ldots, C\}$ respecto al resto de muestras, de forma que se calcula la pertenencia de la muestra i-ésima en esa clase con un estimador $\boldsymbol{\theta}_c$. De los resultados dados por cada estimadorse estima cual es el k-ésimo estimador que da el máximo valor, de donde se estima finalmente que la clase k es donde pertenece la muestra.

habria uno por cada clase? por favor agregar esto para mejorar la descripción si es el caso

El uso de la regresión logística se puede llevar a cabo para normalizar los resultados de los estimadores $\boldsymbol{\theta}$, de donde el proceso para calcular el estimador consta de realizar un proceso de gradiente descendiente, que utiliza como función de costo $J : \mathbb{R}^n \to \mathbb{R}$ la ecuación (6.5) para uso de regularización con L^2 y la ecuación (6.6) para el uso de L^1, según lo descrito en el manual de [20]. En ambas ecuaciones el resultado de optimizar la función de costo da como resultado el estimado final $\hat{\boldsymbol{\theta}}$ el cual es el estimado del modelo. La función ζ es la función *softplus* (véase la Notación).

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \frac{1}{2} \|\boldsymbol{\theta}\|_2 + C \sum_{i=1}^{n} \zeta(-y_i(\boldsymbol{X}_i^\top \boldsymbol{\theta})) \tag{6.5}$$

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \|\boldsymbol{\theta}\|_1 + C \sum_{i=1}^{n} \zeta(-y_i(\boldsymbol{X}_i^\top \boldsymbol{\theta})) \tag{6.6}$$

En el caso para realizar una multi-clasificación, la regresión logística solo puede realizar una clasificación binaria, esto es que para una entrada $\boldsymbol{x}$, esta solo puede realizar una clasificación con salidas $y \in [0, 1]$. Debido a esta restricción, existen una metodología de multiclasificacion llamada *One vs Rest*, que permite utilizar la regresión logística y realiza la multiclasificacion por medio de crear n estimadores que permiten estimar

cada una de las C clases respecto al resto de la entrada, por cada una de los estimadores se estima cual es la clase con mayor valor en la regresión logística individual de ese estimador, la figura 6.3 representa una versión simplificada del proceso.

esto creo que esta repetido con el parrafo anterior. Queda redundante volver a explicar en que consiste one vs rest. Por favor incorporar al parrafo anterior para mejorar la explicacion de lo que es one vs rest y dejar la imagen que esta muy diciente

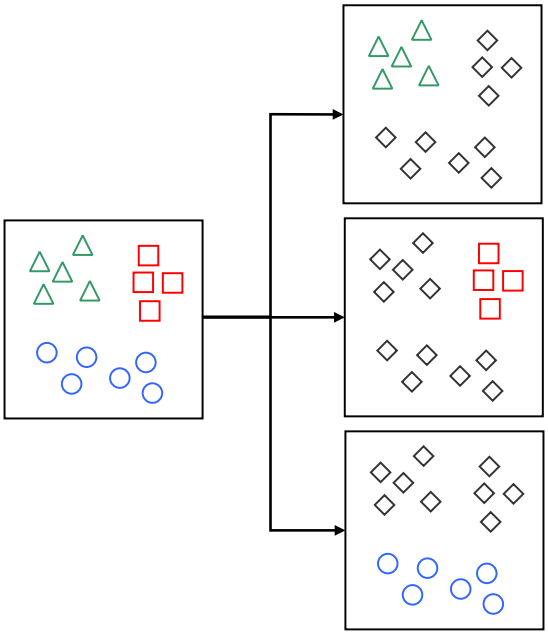


FIGURA 6.3: Algoritmo de One vs Rest.

La manera en que se entrena cada estimador $i \in \{1, \ldots, C\}$ es por medio de optimizar el valor de la función de predicción $\hat{y}(\boldsymbol{x}, \boldsymbol{\theta}_i)$, $\boldsymbol{x} \in \mathbb{X}$ (ecuación (6.7)). Como se muestra en la ecuación (6.8), y es el valor verdadero de la estimación y θ_0 se le conoce como el parámetro de ajuste del modelo.

$$\hat{y}(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \boldsymbol{x} = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n \tag{6.7}$$

$$\boldsymbol{\theta}_i = \arg\min_{\boldsymbol{\theta}_i} |\hat{y}(\boldsymbol{x}, \boldsymbol{\theta}_i) - y| \tag{6.8}$$

Luego de haberse entrenado cada estimador con sus valores óptimos, se puede realizar la predicción de las clases a las que pertenece una entrada $\boldsymbol{x}$ con la ecuación (6.9).

$$c = \arg\max_i \ \sigma(\hat{y}(\boldsymbol{x}, \boldsymbol{\theta}_i)) \tag{6.9}$$

Predicción de hashtags

El uso que se le puede dar a todo lo mencionado anteriormente en esta sección es de predecir etiquetas de Twitter típicas de terrorismo a partir del texto que se encuentra en el tweet con ayuda de alguna de las representaciones de BOW o TF-IDF, tal como se muestra en el ejemplo de la figura 6.4.

Cual de estos dos fue utilizado finalmente en la implementación?

"Really excited to add @plaidavenger to my deathlist along with Italy and @Plaid_Obama after receiving that information."	⇒	**#deathlist, #KillEveryone, #ISIS**

FIGURA 6.4: Predicción de hashtags con modelos lineales.

cada uno de estos hashtags tendría un valor diferente proveniente del respectivo estimador? cuantas clases (estimadores) se usaron en la implementación? por favor explicar e incorporar en el texto

De manera que se determinan los C hashtags de los datos de entrenamiento $\mathbb{X}$ para luego poder realizar la prediccion de los hashtags a partir un texto de entrada d que luego se convierte a representacion BOW o TF-IDF para pasarlo por los C estimadores del *One vs Rest* y estimar las etiquetas predecidas recuperandolas como la ecuación (6.10) de manera de diccionario indexado por el numero de la clase $i \in \{1, \ldots, C\}$ como llave y el hashtag como el valor.

$$\{(i, h)\}\ ;\ h \in \text{hashtags} \tag{6.10}$$

6.3.2. Modelo 2: Reconocimiento de *Named Entities* con redes *Long Short Term Memory*

Igual que en el modelo presentado en la apartado 6.3.1, en Twitter como en cualquier otra red social se presentan en sus textos muchas veces la mención de las llamadas *Named Entities*. Son objetos del mundo real, tales como personas, ubicaciones, organizaciones, productos, entre otros que pueden ser denotados con nombre propio, y pueden ser abstractos o tener una existencia física.

En consecuencia, este modelo tiene como propósito reconocer *Named Entities* que puedan dar una mejor aproximación a entender el contexto de conversaciones en masa de cibercriminales identificados o bien realizar una identificación de quienes son en base de cuales son los temas que tienden a mencionar mas habitualmente.

Despues de este parrafo por favor colocar un parrafo pequeño que describa la pregunta de data science que este modelo resuelve, por ejeemplo ¿Cual(es) es(son) lo(s) named entities identificado(s) en un twet?

Redes neuronales *Long Short Term Memory*

Expandir con contenido de pag 397 Deep Learning

En la literatura se puede encontrar usualmente el uso de las redes *Long Short Term Memory* (LSTM), las cuales proveen por medio de una arquitectura mas compleja la posibilidad de tener recordación de largo y corto plazo, como se muestra en la figura 6.6. Para que estas redes tengan la característica de tener recordación es necesario de una celda de memoria, donde esta es la entrada para la siguiente iteración de la red, junto con la entrada de datos a la red.

Colocar cual es la diferencia con respecto a las RNN, las unas son de corto y las otras de mediano/largo

Téngase en cuenta que la arquitectura presentada en esta sección hace referencia a la implementación de LSTM mas utilizada, sin embargo existen muchas mas variaciones de esta con diferentes propiedades que son deseables en diferentes situaciones.

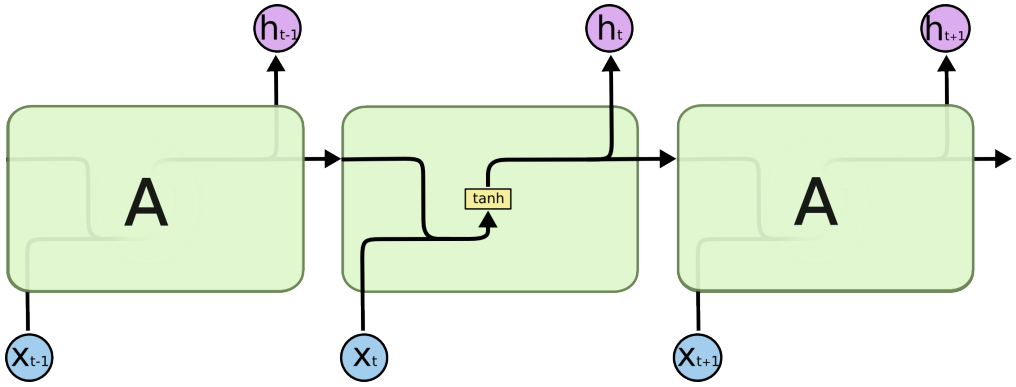


FIGURA 6.5: Red RNN clásica. Tomado de [8].

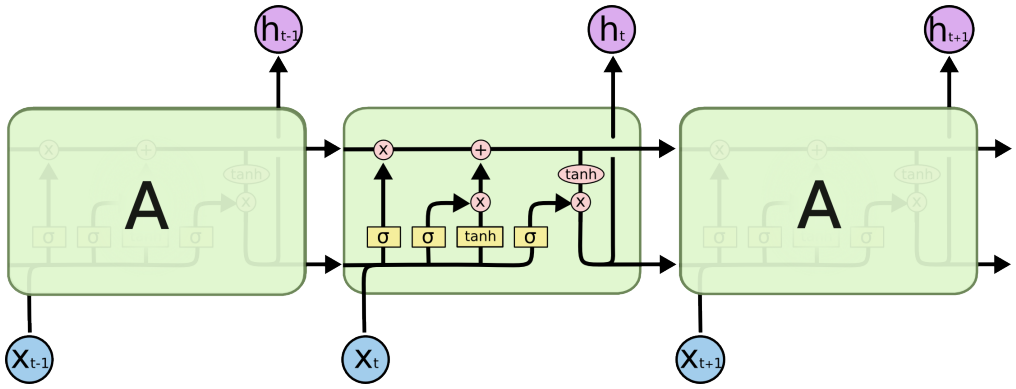


FIGURA 6.6: Red LSTM clásica. Tomado de [8].

Redes neuronales *Bidirectional Long Short Term Memory*

En la tarea de etiquetado de secuencias de texto las *Bidirectional Long Short Term Memory* (BI-LSTM) tienen acceso a muestras de entrada tanto de pasado, presente y futuro por una cierta cantidad de tiempo. El uso de esta característica se puede aprovechar para ir a estados en el futuro (por medio de estados hacia adelante) y en el pasado (por medio de estados hacia atrás) para un momento particular en el tiempo [21].

Con el fin de que la red sea bidireccional es necesario de una celda de memoria para el caso de desplazarse hacia adelante, y una para desplazarse hacia atrás.

Reconocimiento de *Named Entities*

Para el reconocimiento de *Named Entities* tómese como ejemplo el cuadro 6.1, donde las etiquetas asignadas corresponden al reconocimiento correspondiente de cada entidad. Las etiquetas (tags) de respuesta son *Otro* (O) o una de estas: *Persona* (PER), *Ubicación* (LOC), *Organización* (ORG) y *Misceláneo* (MISC), donde las partes de etiqueta B- y I- corresponden a una palabra que se encuentra en la primera posición (*Beginning*) o en una posición intermedia (*Intermediate*).

Texto	Donald	Trump	es	presidente	de	Estados	Unidos
Etiqueta	B-PER	I-PER	O	O	O	B-ORG	I-ORG

CUADRO 6.1: Ejemplo de reconocimiento de *Named Entities*.

La tarea de etiquetado para una LSTM y una BI-LSTM esta representada en la figura 6.7 y la figura 6.8.

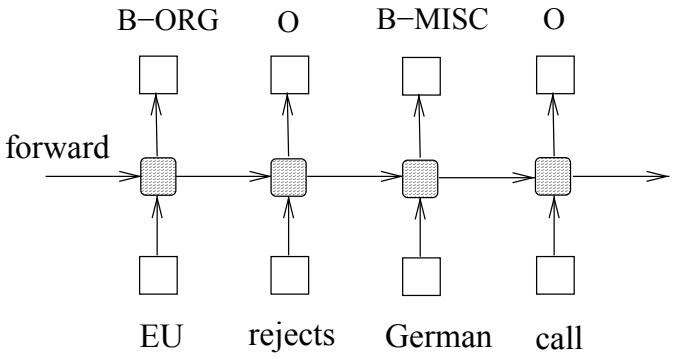


FIGURA 6.7: Etiquetado con una LSTM. Tomado de [21].

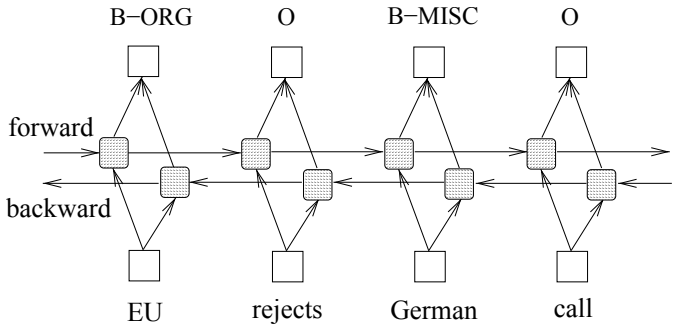


FIGURA 6.8: Etiquetado con una BI-LSTM. Tomado de [21].

Por favor explicar que ventaja tiene hacerlo con una BI-LSTM en lugar de una LSTM normal. En el caso de las figuras 6.8 y 6.9 los outputs son los mismos. ¿Este modelo (contribucion del pgr) se desarrollo finalmente como LSTM o BI-LSTM?

Entrenamiento de *Bidirectional Long Short Term Memory*

El entrenamiento de las redes BI-LSTM se empieza de manera similar a la descrita en la apartado 6.3.1 con el procedimiento de asignar un identificador a cada *token* y a cada *tag* como en las ecuaciones (6.11) y (6.12), donde N es el numero de tokens y T el numero de tags. De manera que el conjunto de entrenamiento $\mathbb{X}$ consta de tuplas $(\text{token}, \text{tag})$, de donde se agregan dos tipos especiales de tokens `<PAD>` y `<UNK>`, que representan relleno y algo desconocido respectivamente.

$$\text{token2id} = \Big\{(\text{token}_i, i) : \forall i \in \{1, \ldots, N\}\Big\} \tag{6.11}$$

$$\text{tag2id} = \Big\{(\text{tag}_i, i) : \forall i \in \{1, \ldots, T\}\Big\} \tag{6.12}$$

Generación de BATCHES para las redes BI-LSTM Las redes neuronales son comúnmente entrenadas por grupos (o *batches* en ingles) de entrada. Eso significa que las actualizaciones de los pesos dentro de la red se basan en secuencias en cada instante de tiempo. Pero estas secuencias dentro de un batch deben tener la misma longitud, así que estas se rellenaran con una etiqueta de relleno `<PAD>`. También es buena idea proveer la red con la longitud de las secuencias, para que así pueda saltarse computaciones innecesarias de las partes de relleno. A las partes de relleno se le asigna un tag de `O`.

Construcción de capas de la red neuronal Para esta construcción se procede a generar una matriz de *embeddings* aleatorios, que en el caso de *TensorFlow* son tensores aleatorios y se establecen las células de procesamiento *forward* y *backward* con valores de marginalización ξ (valor establecido heurísticamente) que es una forma de regularización importante en redes neuronales, de donde se especifica la probabilidad de retención de los valores de la iteración anterior.

La salida de las celdas de *forward* y *backward* serán dos tensores $\mathbf{F}$ y $\mathbf{B}$, que se concatenan en un tensor $\mathbf{O}$.

Computación de predicciones Se computa la función softmax sobre $\mathbf{O}$ para convertir las entradas en una forma de probabilidad, de forma que se estan calculando las probabilidades de cada una de las entradas para luego calcular cuales son las entrada de mayor probabilidad, estimando así la predicción P, como se muestra en la ecuación (6.13).

$$P = \arg\max_i(\text{softmax}(\sigma(\mathbf{O}_i))) \tag{6.13}$$

Sin embargo, durante el entrenamiento no necesitamos una predicción, sino una función de perdida (o *loss* en ingles), que se calcula como se muestra en la ecuación (6.14), y en la figura 6.9, donde $\hat{y}$ es la predicción y y es el valor verdadero.

$$\gamma(\hat{y}, y) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})) \tag{6.14}$$

Que es un token en este contexto? en la Seccion 6.3.1 no se mencionan tokens en ninguna parte

¿sería entonces un conjunto de tuplas (token, tag), que pueden ser: (<PAD>, tag1), (<UNK>, tag2)? No me queda claro que los tokens hagan parte del conjunto de datos de entrenamiento

Solo habrian estos 2 tipos de tokens?

por favor colocar que utilidad tienen estos valores de marginalizacion

es decir que la me-

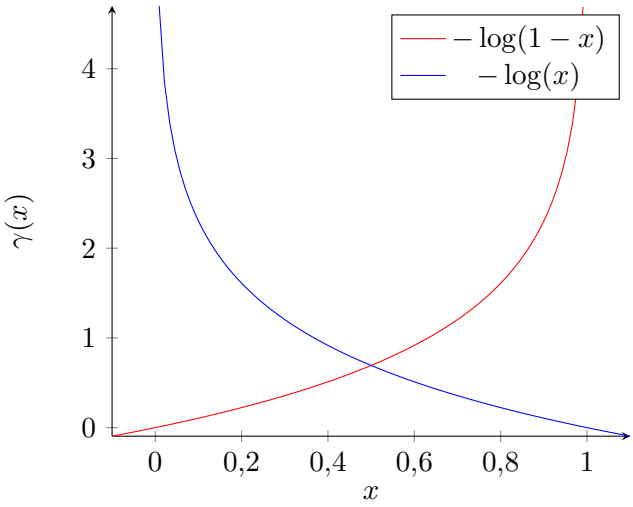


FIGURA 6.9: Gráficas de cross-entropy (azul) si $y = 1$ (rojo) si $y = 0$.

De manera que si se calcula una predicción de 1×10^{-8} cuando el valor verdadero es 1, dará como resultado un valor alto de perdida (e.g. $\gamma(1 \times 10^{-8}, 1) \approx 18.42$).

Se calcula un tensor de perdida $\mathbf{L}$ con $\mathrm{softmax}(\gamma(\sigma(\mathbf{O})))$, para luego calcular la media $\mu_{\mathbf{L}}$ del tensor $\mathbf{L}$. Y se pasa por una tensor mascara$\mathbf{M}$ (ecuación (6.15)) que multiplica por $\mathbf{L}$, donde $\mathcal{D}$ es el batch de entrenamiento. Entiéndase al tensor $\mathbf{M}$ como el tensor que representa las posiciones que no son <PAD> con valor de 1 y valor 0 si lo son dentro del batch.

por favr explicar cual es la utilidad de este tensor mascara?

$$\mathbf{M} = \sum_{i \in \{1,\ldots,|\mathcal{D}|\}} \boldsymbol{e}^{(i)}_{\mathrm{tag_i=PAD}} \tag{6.15}$$

Para realizar el entrenamiento con esta función de perdida se realiza una optimización (el optimizador estocástico ADAM [22] es una opción eficiente) con una tasa de aprendizaje α.

explicar brevemente porque es eficiente?

6.3.3. Modelo 3: Búsqueda de tweets relacionados con *embeddings* de StarSpace

En este modelo, la intención es que a partir de una colección de textos guardados en una base de datos se puedan obtener los k textos mas similares a un texto de consulta q, de manera que se encuentren los textos de mayor similaridad en cuanto a las palabras usadas por medio del uso de *embeddings*.

Por favor colocar aqui debajo un parrafo breve que represente la pregunta de data science que resuelve este modelo

¿Que son y como se usan los embeddings?

Los *embeddings* se pueden entender como vectores n–dimensionales que representan palabras dentro de un diccionario. Los modelos previamente vistos BOW y TF-IDF se pueden entender como *embeddings*, sin embargo para la aplicación en este modelo, es necesario otro *embedding* que permita encontrar la similaridad entre palabras de forma

que palabras que tengan un significado parecido tengan una distancia menor que si se comparara con una palabra que fuera antónima, de forma que $\text{dist}(\boldsymbol{p}_{\text{asombroso}}, \boldsymbol{p}_{\text{genial}}) \gg \text{dist}(\boldsymbol{p}_{\text{asombroso}}, \boldsymbol{p}_{\text{terrible}})$, donde dist típicamente seria la distancia de similaridad coseno (ecuación (6.16)).

$$\text{similaridad}(a, b) = \cos(\theta) = \frac{\boldsymbol{p}_a \times \boldsymbol{p}_b}{\|\boldsymbol{p}_a\| \, \|\boldsymbol{p}_b\|}, \; a, b \in d \tag{6.16}$$

Dentro de los ejemplos de uso popular de *embeddings* están *GloVe* [23], *word2vec* [24], *fastText* [25], entre muchos otros.

Calculo de vectores de varias palabras

Una adición a las ventajas dadas por estos *embeddings* es la capacidad de representar textos completos por medio de obtener la representación de cada termino (palabra) $t \in d$ y posteriormente calcular el promedio $\boldsymbol{s}$ (ecuación (6.17)).

$$\boldsymbol{s} = \frac{1}{|d|} \sum_{t \in d} \boldsymbol{p}_t \tag{6.17}$$

Evaluación de similaridad entre textos

Nos podemos imaginar que usando un buen *embedding* la similaridad coseno entre frases duplicadas[4] va a ser menor que para las frases aleatorias que generemos. Sobretodo que para cada par de frases duplicadas podemos generar R ejemplos de frases negativas y encontrar la posición del texto duplicado correcto.

Sin embargo, no es buena idea considerar que el primer candidato sera siempre el primer resultado que dio la mayor similaridad, por lo que se tomara como una lista ordenada y se formulará una métrica de comparación. Podemos tomar a K como un numero razonable de mejores elementos y N el numero de consultas (tamaño de la muestra).

Métrica Hits@K La primera métrica que se usaría seria el numero de "hits"correctos para algún K, vea la ecuación (6.18).

que significa un hit correcto?

$$\text{Hits@K} = \frac{1}{N} \sum_{i=1}^{N} [\text{dup}_i \in \text{topK}(q_i)] \tag{6.18}$$

Donde q_i es la i–ésima consulta, dup_i es su duplicado y $\text{topK}(q_i)$ son los K elementos mas similares provistos por nuestro modelo donde $[\text{dup}_i \in \text{topK}(q_i)]$ obtiene valor de 1 si la condición que el duplicado este dentro de los K elementos es verdadera y 0 de lo contrario[5].

[4] Entiéndase a una frase duplicada como textos que sean equivalentes "semánticamente" pero no necesariamente iguales sintácticamente

[5] A esta notación se le conoce como el paréntesis de Iverson (o *Iverson bracket* en inglés) wikipedia.org/wiki/Iverson_bracket

Métrica DCG@K La segunda métrica es conocida como *Discounted cumulative gain* (versión simplificada), dada por la ecuación (6.19).

$$\text{DCG@K} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\log_2(1+\text{rank}_{\text{dup}_i})}[\text{rank}_{\text{dup}_i} \leq K] \tag{6.19}$$

Donde $\text{rank}_{\text{dup}_i}$ es la posición del duplicado en la lista ordenada de frases mas cercanas al texto de la consulta q_i. De acuerdo a esta métrica, el modelo obtiene una mayor recompensa por una posición mas alta de la respuesta correcta. Si la respuesta no aparece en topK en absoluto, entonces la recompensa es 0.

StarSpace

StarSpace [26] es un modelo neuronal de propósito general para el aprendizaje eficiente de *embedding* de entidades para resolver una gran variedad de problemas, entre estas están:

- Aprendizaje de *embeddings* a nivel de palabras, frases y documentos.
- Obtención de información: "ranking" de conjuntos de entidades/documentos u objetos.
- Clasificación de textos.
- Aprendizaje de similaridad de textos.
- Entre otras.

¿Como funciona y cual es la principal diferencia con *word2vec*? *StarSpace* puede ser entrenado específicamente para algunas tareas. En contraste con *word2vec*, el cual intenta entrenar *embeddings* similares para palabras en contextos similares, *StarSpace* usa *embeddings* para las frases completas (tal como la suma de los *embeddings* para palabras y frases). A pesar de que se obtienen *embeddings* en ambos casos como resultado del entrenamiento, los *embeddings* de *StarSpace* son entrenados usando datos supervisados.

que ventaja hay en que el entrenamiento sea con datos supervisados?

Uso de *embeddings* creados con *StarSpace*

En este caso, *StarSpace* debería usar dos tipos de pares de secuencias para el entrenamiento: "positivas" y "negativas". Las muestras "positivas" son extraídas del conjunto de entrenamiento $\mathcal{D}$ (duplicados, alta similaridad) y las muestras "negativas" que son generadas aleatoriamente (se asume que tienen una baja similaridad).

De forma que se pueda usar *StarSpace* es necesario primero compilar el código fuente provisto del repositorio de código de GitHub[6] (Solo funciona MacOS o Linux). Se puede clonar el codigo fuente por medio de introducir los comandos en la Terminal de la figura 6.10.

Para luego ejecutarlo con diferentes parámetros de entrada y un conjunto de entrenamiento en un archivo `.tsv`, como se muestra en la figura 6.11.

Donde se ejecuta a *StarSpace* en modo de entrenamiento, tiene el archivo de entrenamiento (`prepared_train.tsv`), el modo de entrenamiento para exploración de si-

son datos etiquetados?

[6] https://github.com/facebookresearch/StarSpace

```
git clone https://github.com/facebookresearch/Starspace.git
cd Starspace
make
```

FIGURA 6.10: Ejemplo en compilación de *StarSpace* en la Terminal.

```
starspace train -trainFile prepared_train.tsv -model modelSaveFile
↪  -trainMode 3 -adagrad 1 -ngrams 1 -epoch 5 -dim 100 -similarity
↪  cosine -minCount 2 -verbose 1 -fileFormat labelDoc -negSearchLimit
↪  10 -lr 0.05
```

FIGURA 6.11: Ejemplo en ejecución de *StarSpace* en la Terminal.

milaridad de textos (`-trainMode 3`), uso de optimización ADAGRAD (`-adagrad 1`), con n-gramas de tamaño 1 (`-nrams 1`), 5 iteraciones (`-epoch 5`), vectores resultantes en $\mathbb{R}^{100}$ (`-dim 100`), uso de similaridad coseno (`-similarity cosine`), conteo mínimo de 2 términos (útil si no se quieren obtener *embeddings* de palabras extremadamente raras, `-minCount 2`), numero mínimo de ejemplos negativos como 10 usado durante el entrenamiento (`-negSearchLimit 10`) y una tasa de aprendizaje de 0,05 (`-lr 0.05`).

Luego de esto el resultado es un archivo de *embeddings*, donde cada linea esta compuesta de una palabra y los 100 números que componen al vector de esa palabra.

Estos *embeddings* resultantes permiten luego realizar la obtención de los k textos mas similares a un tweet de consulta q_i obtenido de *pool* de textos obtenidos de Twitter de fuentes relevantes de forma que se encuentren textos relacionados a individuos que tuvieran relación con grupos terroristas.

6.4. Despliegue (Deployment)

A la fecha, los productos disponibles para el despliegue son el primer modelo propuesto (apartado 6.3.1), una implementación del meta-modelo SOM (apartado 4.7.2) y la implementación de la recolección de información de tweets (apartado 6.2).

Capítulo 7

Conclusiones y trabajos futuros

7.1. Conclusiones

Revisar bien y comparar con los anteriores

- El procesamiento de lenguaje natural (NLP) aporta un conjunto de metodologías y técnicas que pueden apoyar labores de ciberinteligencia.
- El éxito en la aplicación de modelos de aprendizaje automático depende en gran medida de la calidad de los datos de entrenamiento.

7.2. Trabajos futuros

Revisar bien

- Utilizar el meta-modelo de SOM (apartado 4.7.2) como un cuarto modelo para el clustering de textos representados con embeddings pre-entrenados o generados con *StarSpace*.
- Utilizar metodologías de visualización de las redes BI-LSTM como la visualización expuesta en [27] con el fin de depurar su ejecución.
- Desarrollo de un *dashboard* para la ayuda de visualización de un agente de seguridad del estado a desempeñar su tarea de perfilado de cibercriminales.
- Desarrollo de técnicas avanzadas de obtención de información de redes sociales, tanto de redes sociales populares como impopulares, de manera similar al desarrollado con *Open Source Intelligence* (OSINT) en [2].
- Desarrollar un dashboard que integre los resultados obtenidos de los tres modelos.
- Extender la aplicación de los modelos desarrollados al lenguaje español (Colombia).

Apéndice A

Información general del proyecto

A.1. Repositorio del proyecto

El repositorio principal que contiene todos el Software, documentación y este documento se encuentran en GitHub.

Repositorio de GitHub

Hiper-vinculo al repositorio:
https://github.com/aanzolaavila/thesis-project-repository.git

Para clonar el repositorio es necesario ejecutar el comando en la Terminal (o bien *Git Bash* en Windows):

```
git clone --recurse-submodules
↪  https://github.com/aanzolaavila/thesis-project-repository
```

A.2. Software del proyecto

Cada software esta en repositorios independientes dentro de GitHub, donde tambien se puede encontrar una documentación de como ejecutarlos.

A.2.1. Mapas autoorganizados

Por hacer

A.2.2. Modelos de software

Por hacer

A.2.3. Extractor de tweets de ARCHIVE

Por hacer

A.2.4. Consumidor de Twitter

Por hacer

A.2.5. StarSpace

Por hacer

Apéndice B

Conceptos importantes

La mayor parte de este apéndice se basa en contenido expuesto en [1], y son conceptos que el autor considera pertinentes conocerlos de manera que se comprenda de manera adecuada el mundo del *Machine Learning*.

B.1. Capacidad, *Overfitting* y *Underfitting*

El reto central de ML es que nuestro algoritmo tenga un buen rendimiento sobre datos nuevos, entradas nunca antes vistas, no solo las entradas con las que entrenamos nuestro modelo. La habilidad de tener un buen rendimiento sobre nuevos datos se le llama **generalización**.

generalización

Típicamente cuando entrenamos un modelo de ML debemos acceder a un conjunto de entrenamiento; podemos computar una medida de error sobre el conjunto de entrenamiento, llamado el **error de entrenamiento**; y reducimos este error de entrenamiento. El **error de generalización** se define como el valor esperado de error sobre una nueva entrada de datos. Aquí el valor esperado se toma sobre diferentes entradas posibles, tomadas de una distribución de entradas que esperamos que el sistema se encontrara en la practica.

error de entrenamiento
error de generalización

Usualmente el error de generalización de un modelo de ML se estima por su rendimiento en un **conjunto de prueba** que se obtuvieron de manera separada al conjunto de entrenamiento.

conjunto de prueba

El conjunto de datos de entrenamiento y de prueba son generados por una distribucion de probabilidad sobre los conjuntos de datos llamado el **proceso de generación de datos** (o *data-generating process* en inglés).

proceso de generación de datos

Por lo general asumimos una serie de suposiciones (llamadas usualmente en inglés como *i.i.d. assumptions*), y es que una muestra de cada conjunto de datos es **independiente** una de otra y que los conjuntos de entrenamiento y prueba son **distribuidos idénticamente** tomados de la misma distribución de probabilidad. Esta suposición nos permite describir el proceso generativo de datos con una distribución de probabilidad sobre una sola muestra. Esta misma distribución es luego usada para generar cualquier muestra de entrenamiento y prueba. Llamamos a esta distribución compartida la **distribución generadora de datos** (o *data-generating distribution* en inglés), denotada como p_{data}.

independiente
distribuidos idénticamente
distribución generadora de datos

Cuando usamos una algoritmo de ML, no establecemos los parámetros de antemano para luego tomar muestras de ambos conjuntos de datos. Tomamos muestras del conjunto de entrenamiento, luego ajustamos los parámetros para reducir el error en el

conjunto de entrenamiento, y luego tomamos muestras del conjunto de prueba. Bajo este proceso, el valor esperado de error en el conjunto de prueba es igual o mayor al del conjunto de entrenamiento. Los factores que determinan que tan bien un algoritmo de ML se desempeñara son su habilidad para

- Hacer que el error en el conjunto de entrenamiento sea pequeño.
- Hacer que la diferencia entre el error de prueba y de entrenamiento sea pequeño.

Estos dos factores corresponden a los dos grandes retos en ML: ***Underfitting*** y ***Overfitting***. *Underfitting* es cuando el modelo no es capaz de obtener un valor de error suficientemente bajo sobre el conjunto de entrenamiento. *Overfitting* ocurre cuando la diferencia entre el valor de error del conjunto de entrenamiento y el conjunto de prueba es muy grande. *Underfitting* *Overfitting*

Podemos controlar si un modelo es mas propenso a hacer *Underfitting* u *Overfitting* cambiando su **capacidad**. Informalmente, la capacidad de un modelo es su habilidad de acomodarse a una gran variedad de funciones. Modelos con baja capacidad pueden tener problemas para adaptarse al conjunto de entrenamiento. Modelos con una alta capacidad pueden adaptarse demasiado por memorizar propiedades del conjunto de entrenamiento que no le son útiles con el conjunto de pruebas. capacidad

Una forma de controlar la capacidad de nuestro algoritmo de ML es escogiendo su **espacio de hipótesis**, el conjunto de funciones que el algoritmo de aprendizaje se le es permitido tomar como función solución. espacio de hipótesis

Un ejemplo de esto es regresión lineal, y es que podemos permitir que nuestro algoritmo de aprendizaje en vez de tomar funciones lineales (de grado $p = 1$, ecuación (B.1)), también pueda tomar funciones polinomiales (grado $p > 1$, ecuación (B.2)).

$$\hat{y} = b + wx \tag{B.1}$$

$$\hat{y} = b + \sum_{i=1}^{p} w_i x^i \tag{B.2}$$

En este ejemplo, es útil mencionar que existe un teorema que dictamina que para cualquier entero $n \geq 0$ y una lista de $n + 1$ puntos en $\mathbb{R}^2$ $(x_0, y_0), \ldots, (x_n, y_n)$, donde $x_0 < x_1 < \cdots < x_n$, existe un polinomio f de grado n tal que $f(x_i) = y_i$ para todo i [28, pág 15]. Análogamente a los algoritmos con una gran capacidad, siempre se tiene una función que se adapta perfectamente a los datos de entrenamiento, sin embargo esta función probablemente no sera útil para los datos de prueba. En la practica no siempre es posible obtener una función que se adapte perfectamente a los datos.

Los algoritmos de ML por lo general se desempeñan mejor cuando su capacidad es apropiada para la verdadera complejidad de la tarea que se necesita desarrollar y la cantidad apropiada de datos de entrenamiento que le son provistos.

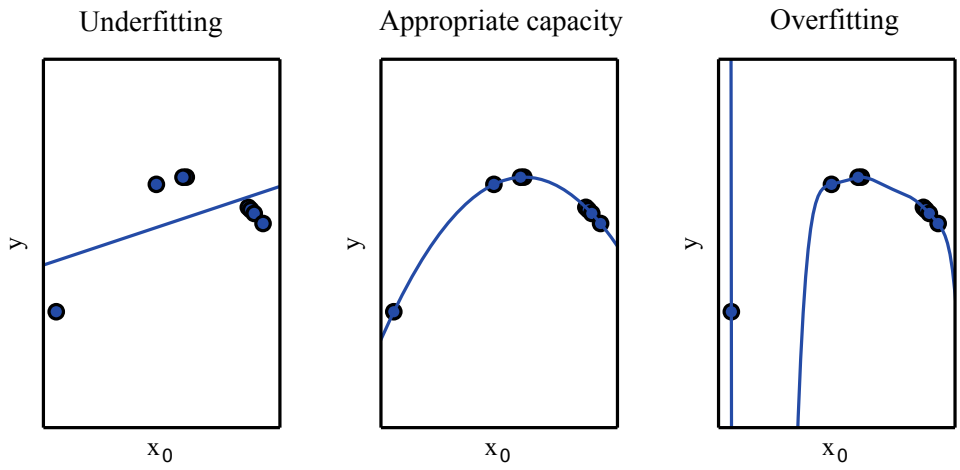


FIGURA B.1: Comparación de capacidades. Tomado de [1].

Existen muchas maneras de cambiar la capacidad de un modelo, la capacidad no esta solo determinada por la elección del modelo. El modelo especifica cual familia de funciones se pueden escoger cuando se varían los parámetros para alcanzar un objetivo de entrenamiento. A esto se le llama la **capacidad representativa** (o *representational capacity* en inglés) del modelo. En muchos casos encontrar la mejor función dentro de esta familia es un problema difícil de optimización. En la practica, el algoritmo de aprendizaje no encuentra la mejor función, sino una que meramente reduce el error de entrenamiento. Estas limitaciones adicionales, tales como la imperfección del algoritmo de optimización, significa que la **capacidad efectiva** del algoritmo puede ser menor al de la capacidad representativa.

capacidad representativa

capacidad efectiva

B.1.1. Teorema "No free lunch"

La teoría de aprendizaje dice que un algoritmo de ML puede generalizar de manera aceptable a partir de un numero finito de muestras de conjuntos de entrenamiento. Esto parece contradecir algunos principios básicos de la lógica. Inferir reglas generales de un conjunto de muestras limitadas no es lógicamente valido. Para inferir lógicamente una regla para describir cada miembro de un conjunto es necesario tener información de todos los miembros de ese conjunto.

ML evita es problema por medio de ofrecer reglas probabilísticas, en vez de ofrecer reglas completamente ciertas como las que ofrece el razonamiento lógico. ML promete encontrar reglas que son *probablemente* ciertas para la *mayor* parte de los miembros de un conjunto que les concierne.

Desafortunadamente, esto no resuelve el problema completo. Existe un teorema llamado "no free lunch" el cual afirma que, promediando sobre todas las posibles distribuciones generadoras de datos, cualquier algoritmo de clasificación tiene la misma tasa de error cuando clasifica puntos antes no vistos. Esto quiere decir que no existe un algoritmo de ML que es mejor universalmente a los demás.

Sin embargo, esto solo es aplicable cuando se habla del promedio de *todas* las posibles distribuciones generadoras de datos. Si hacemos suposiciones sobre los tipos de

distribución de probabilidad que nos encontramos en la vida real, entonces podemos diseñar algoritmos de aprendizaje que se desempeñan mejor sobre esas distribuciones.

Esto significa que el objetivo de la investigación en ML no busca encontrar un algoritmo de aprendizaje universal, sino que el objetivo es encontrar que tipos de distribuciones son relevantes al "mundo real" que un agente de AI experimenta, y que tipos de algoritmos de ML se desempeñan bien con datos tomados del tipo de distribuciones generadoras de datos que nos interesan.

B.1.2. Regularización

El teorema de "no free lunch" nos dice que debemos diseñar nuestro algoritmo de ML para tener un buen rendimiento en una tarea especifica, esto lo logramos por medio de tener una serie de preferencias dentro del algoritmo de aprendizaje, cuando estas preferencias se alinean hacia el objetivo que queremos lograr este tiene un mejor desempeño.

Aunque una forma de lograr ese objetivo se puede conseguir en parte por medio de modificar la capacidad, no es la única forma de lograrlo. El comportamiento de nuestro algoritmo esta fuertemente influenciado no solo por que tan grande hacemos el conjunto de funciones permitido en su espacio hipótesis, sino también la identidad especifica de esas funciones.

Un método popular para realizar un ajuste de forma que se controle el *Underfitting* y el *Overfitting* es por medio de modificar nuestra función de costo J agregándole un hiper-parámetro λ que se establece de antemano sobre el algoritmo de manera *heurística*. Modificamos J de manera que cuando $\lambda = 0$ no se modifique el valor original de J, permitiéndonos controlar el nivel de *Underfitting* y *Overfitting* con el valor de λ.

De manera mas general, podemos regularizar un modelo que aprende una función $f(\boldsymbol{x}, \boldsymbol{\theta})$ por medio de añadir una penalidad llamada **regulador** (o *regularizer* en inglés) a la función de costo J. Como ejemplo, téngase una función hipótesis $\hat{f}(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \boldsymbol{x}$ con parámetros $\boldsymbol{\theta}$ y una función objetivo $f(\boldsymbol{x})$ de manera que podemos definir una función de costo que queremos minimizar como:

regulador

$$J(\boldsymbol{x}, \boldsymbol{\theta}) = |\hat{f}(\boldsymbol{x}, \boldsymbol{\theta}) - f(\boldsymbol{x})| \tag{B.3}$$

Agregando el regulador $\Omega(\boldsymbol{\theta}) = \lambda \boldsymbol{\theta}^\top \boldsymbol{\theta}$ a la función de costo, la ecuación resulta ser

$$J(\boldsymbol{x}, \boldsymbol{\theta}) = |\hat{f}(\boldsymbol{x}, \boldsymbol{\theta}) - f(\boldsymbol{x})| + \Omega(\boldsymbol{\theta}) \tag{B.4}$$

De esta manera penalizamos los valores grandes de $\boldsymbol{\theta}$ (cuando $\lambda > 0$) que pueden ocasionar *Overfitting* de forma que regularizan el avance del algoritmo de optimización que se usa para minimizar la función de costo. Sin embargo, un valor muy grande en λ ocasionara *Underfitting*.

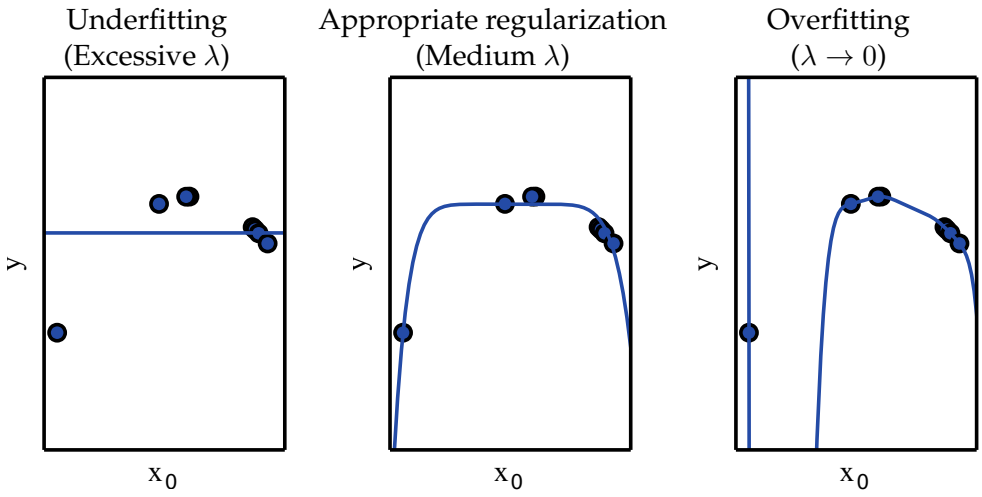


FIGURA B.2: Comparación de valores de regularización. Tomado de [1].

B.2. Hiper-parámetros y conjuntos de Validación

La mayor parte de los algoritmos de ML tienen hiper-parámetros, que son configuraciones que se pueden usar para controlar el comportamiento del algoritmo. Los valores de los hiper-parámetros no esta controlados por el algoritmo en si mismo.

Algunas veces una configuración se decide poner como un hiper-parámetro que el algoritmo no aprende debido a que la configuración es difícil de optimizar. Es mas común encontrar que la configuración debe ser un hiper-parámetro debido a que no es apropiado ajustarlo con un conjunto de entrenamiento. Esto aplica para todos los hiper-parámetros que controlar la capacidad de un modelo. Si se aprendiera sobre un conjunto de entrenamiento, veríamos que siempre se escogerían hiper-parámetros que llegarían a la capacidad mas alta que sea posible para el modelo, dando como resultado *Overfitting*.

Para resolver este problema, necesitamos de un **conjunto de validación** que el algoritmo de aprendizaje no pueda observar.

conjunto de validación

Es importante que el conjunto de pruebas no sea usado para ajustar de ninguna manera los parámetros e hiper-parámetros del modelo, de esta manera la forma en que construimos el conjunto de validación es a partir del conjunto de entrenamiento. Específicamente, separamos los datos de entrenamiento en dos conjuntos disyuntos, uno de estos es para aprender los parámetros del modelo; el otro es nuestro conjunto de validación, usado para estimar nuestro error de generalización durante o después del entrenamiento, permitiendo que lo hiper-parámetros sean ajustados adecuadamente. Después de que la optimización de nuestros hiper-parámetros este completa, podemos estimar el error de generalización de nuestro modelo con el conjunto de pruebas.

B.2.1. Validación cruzada

Dividir el conjunto de datos en un conjunto de entrenamiento y pruebas fijo puede ser problemático si resulta que el conjunto de pruebas es pequeño. Un conjunto de pruebas pequeño implica incertidumbre estadística sobre el valor estimado del error de

prueba promedio, haciendo difícil decir que un algoritmo A es mejor que el algoritmo B en una tarea especifica.

Cuando el conjunto de datos es muy pequeño, procedimientos alternativos nos permiten utilizar todas las muestras en la estimación del error de prueba promedio, al precio de mayor costo computacional. Estos procedimientos se basan en la idea de repetir las computaciones de entrenamiento y de prueba sobre diferentes subconjuntos escogidos aleatoriamente, o bien divisiones del conjunto de datos original. El procedimiento mas común el de validación cruzada de k partes (conocido como k-*fold cross-validation* en inglés), en donde una partición del conjunto de datos se construye por medio de dividirlo en k subconjuntos disyuntos. El error de prueba se estima por medio de tomar el promedio del error de prueba de todas las k pruebas.

B.3. Estimadores, Parcialidad (*Bias*) y Varianza

El campo de las estadísticas nos da muchas herramientas para conseguir nuestro objetivo de ML de resolver una tarea no solo con el conjunto de entrenamiento sino que también para generalizar. Conceptos fundamentales como la estimación de puntos, la parcialidad y la varianza son útiles para caracterizar formalmente las nociones de generalidad, *Underfitting* y *Overfitting*.

B.3.1. Estimación de puntos

La estimación de puntos es el intento de proveer la única "mejor" predicción de alguna cantidad de interés. En general esta cantidad de interés puede ser un único parámetro o un vector de parámetros en algún modelo paramétrico.

Para distinguir estimados de parámetros de su valor real se utiliza la distinción de un parámetro $\boldsymbol{\theta}$ con $\hat{\boldsymbol{\theta}}$.

Sea $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ un conjunto de m puntos de datos independientes e idénticamente distribuidos. Un **estimador** o **estadístico** es cualquier función en base a los datos: estimador estadístico

$$\hat{\boldsymbol{\theta}}_m = g(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}) \tag{B.5}$$

Sin embargo, la definición no requiere que g sea una buena estimación del verdadero $\boldsymbol{\theta}$ ni tampoco que la respuesta este en el mismo conjunto de valores validos de $\boldsymbol{\theta}$. Esta definición supone una gran flexibilidad para definir un estimador, sin embargo un buen estimador es una función cuya salida sea cercana al verdadero $\boldsymbol{\theta}$ que genero los datos de entrenamiento.

Podemos asumir (desde la perspectiva de estadístico frecuentista) que el valor verdadero del parámetro $\boldsymbol{\theta}$ es fijo pero desconocido, y que $\hat{\boldsymbol{\theta}}$ es una estimación en función a los datos.

Estimación de funciones

La estimación de puntos también puede hacer referencia a la estimación de la relación entre variables entrada y objetivo, nos referimos a estos tipos de estimación de puntos como estimación de funciones.

En este escenario tratamos de predecir una variable $\boldsymbol{y}$ dado un vector de entrada $\boldsymbol{x}$. Asumimos que existe una función $f(\boldsymbol{x})$ que describe una relación aproximada entre $\boldsymbol{y}$

con $\boldsymbol{x}$. Podemos asumir que $\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\epsilon}$, donde $\boldsymbol{\epsilon}$ es la parte de $\boldsymbol{y}$ que no es predecible a partir de $\boldsymbol{x}$. En la estimación de funciones nos interesa aproximar f con un modelo o estimado $\hat{f}$.

B.3.2. Parcialidad (*Bias*)

La parcialidad (o *Bias* en inglés) es un estimador definido como:

$$\text{bias}(\hat{\boldsymbol{\theta}}_m) = \mathbb{E}(\hat{\boldsymbol{\theta}}_m) - \boldsymbol{\theta} \tag{B.6}$$

Donde el valor esperado es sobre los datos (vistos como muestras de una variable aleatoria), donde $\boldsymbol{\theta}$ es el valor verdadero de $\boldsymbol{\theta}$ usado para definir la distribución que genera los datos. Un estimador se dice que es imparcial (o *unbiased* en inglés) si $\text{bias}(\hat{\boldsymbol{\theta}}_m) = \mathbf{0}$, lo cual implica que $\mathbb{E}(\hat{\boldsymbol{\theta}}_m) = \boldsymbol{\theta}$. Un estimador se dice que es **asintótico imparcial** si $\lim_{m\to\infty} \text{bias}(\hat{\boldsymbol{\theta}}_m) = \mathbf{0}$.

asintótico imparcial

Los estimadores imparciales son ciertamente deseables, sin embargo no son siempre los "mejores" estimadores.

B.3.3. Varianza y Error Estándar

Otra propiedad de un estimador que puede que debamos tener en cuenta, es que tanto se espera que van a variar nuestros resultados como función de la muestra de datos.

La **varianza** de un estimador se define como:

varianza

$$\text{Var}(\hat{\theta}) \tag{B.7}$$

donde la variable aleatoria $\hat{\theta}$ es el conjunto de entrenamiento. De manera alternativa la raíz cuadrada de la varianza es el **error estándar**, denotado como $\text{SE}(\hat{\theta})$.

error estándar

La varianza (o el error estándar) de un estimador nos provee de una medida de como esperaríamos que el estimado que computamos de los datos variara cuando retomamos una muestra independiente del conjunto de datos. De la misma manera en que nosotros querríamos una parcialidad baja, querríamos también una varianza baja.

El error estándar de la media esta dado por:

$$\text{SE}(\hat{\mu}_m) = \sqrt{\text{Var}\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right]} = \frac{\sigma}{\sqrt{m}} \tag{B.8}$$

La varianza de un estimador disminuye en función de m, el numero de muestras de nuestro conjunto de datos.

B.3.4. Consistencia

En la labor de escoger un mejor estimador nos interesa especialmente como este se comportara a medida que la cantidad de datos de entrenamiento aumenta. En particular

nos interesa que a medida que la cantidad de datos aumenta nuestros estimados se aproximan cada vez mas al verdadero valor de los parámetros. Formalmente deseamos:

$$\operatorname*{plim}_{m\to\infty} \hat{\theta} = \theta \tag{B.9}$$

El símbolo plim indica convergencia en probabilidad, indicando que para cualquier $\epsilon > 0$ se tiene que $P(|\hat{\theta} - \theta| > \epsilon) \to 0$ cuando $m \to \infty$. La condición descrita en la ecuación (B.9) se le conoce como **consistencia**. Algunas veces se le llama consistencia débil, cuando se refiere a consistencia fuerte se hace referencia a la convergencia **casi segura** de $\hat{\theta}$ a θ. La convergencia casi segura de una secuencia de variables aleatorias $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots$ a un valor $\boldsymbol{x}$ ocurre cuando $p(\lim_{m\to\infty} \mathbf{x}^{(m)} = \boldsymbol{x}) = 1$. consistencia

La consistencia nos asegura que la parcialidad inducida por el estimador se elimina a medida que la cantidad de muestras de datos aumenta. Sin embargo lo contrario no es verdad — imparcialidad asintótica no implica consistencia.

A medida que la capacidad del modelo aumenta, la parcialidad tiende a disminuir y la varianza tiende a aumentar, dándonos así una curva en forma de U para el error de generalización, tal como se muestra en la figura B.3.

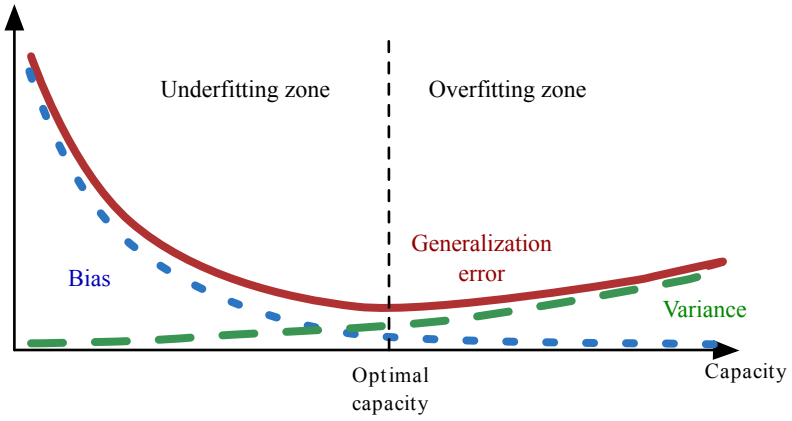


FIGURA B.3: Consistencia. Tomado de [1].

B.3.5. Estimación de máxima probabilidad

En vez de adivinar si una función se comportara como un buen estimador y luego analizar su parcialidad y su varianza, querríamos tener un principio del cual pudiéramos derivar funciones especificas que son buenos estimadores para modelos diferentes.

Uno de estos principios es el de la estimación de máxima probabilidad (o *Maximum Likelihood Estimation* en inglés).

Considérese un conjunto de m muestras $\mathbb{X} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ tomadas de manera independiente de una distribución generadora de datos $p_{\text{data}}(\mathbf{x})$ verdadera pero desconocida.

Sea además $p_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})$ una familia paramétrica de distribuciones de probabilidad sobre el mismo espacio que $\boldsymbol{\theta}$. O en otras palabras, $p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})$ mapea cualquier configuración $\boldsymbol{x}$ a un numero real que estima la probabilidad verdadera $p_{\text{data}}(\boldsymbol{x})$.

El estimador de máxima probabilidad para $\boldsymbol{\theta}$ es definido como

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} p_{\mathrm{model}}(\mathbb{X}; \boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\mathrm{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \tag{B.10}$$

Debido a que se trata de probabilidades, se generan varios problemas a la hora de utilizarlo. Las probabilidades al estar en un intervalo de $[0, 1]$, al multiplicarse muchas probabilidades eso puede generar un problema llamada **underflow**, en la que el hecho de que un numero se acerque demasiado al cero provoca que en la representación interna del computador se trunque a 0 perdiendo una precisión considerable. Como se trata de una maximizaron de valores, hacer una modificación a la función de costo no cambia el valor objetivo, por lo que podemos aplicarle la función logaritmo a la función de costo, de esta manera reducimos considerablemente el problema de underflow, dejando ahora el único problema al que nos enfrentamos conocido como **overflow**. La ecuación queda como

underflow

overflow

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\mathrm{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \tag{B.11}$$

Pasando el nombre a ser estimación de máxima probabilidad logarítmica (o *maximum log-likelihood* en inglés).

Propiedades de estimación de máxima probabilidad

Bajo las condiciones apropiadas, el estimador de máxima probabilidad tiene la propiedad de consistencia (véase el apéndice B.3.4), por lo que a medida que el numero de muestras se aproxima a infinito, el estimado de un parámetro converge al verdadero valor, dándonos las siguientes condiciones:

- La distribución verdadera p_{data} debe estar en la familia de modelos $p_{\mathrm{model}}(\cdot; \boldsymbol{\theta})$.
- La verdadera distribución de p_{data} debe corresponder a exactamente un valor para $\boldsymbol{\theta}$. De lo contrario, el estimador de máxima probabilidad puede recuperar una distribución p_{data} correcta, pero no sera capaz de determinar cual valor de $\boldsymbol{\theta}$ es el correcto por el proceso generador de datos.

B.4. Métodos de optimización

Por hacer, mirar si incluirlo de verdad.

B.4.1. Gradiente descendiente

Por hacer, pagina 79 Deep Learning

B.4.2. Gradiente descendiente estocástica

Por hacer, pagina 147 y 286 Deep Learning

B.4.3. AdaGrad

Por hacer, pagina 299 Deep Learning

B.4.4. Adam

Por hacer, pagina 301 Deep Learning

B.5. Evaluación: Precisión, Exactitud y medida F

Por hacer, pagina 73 Speech and Recognition [12]

B.6. Curva Receiver-Operating Characteristic (ROC) y valor Area Under the Curve (AUC)

Por hacer, usar [29], esto se necesitara para los resultados de los experimentos

Receiver-Operating Characteristic (ROC) y *Area Under the Curve* (AUC)

Glosario

Símbolos

k-nearest neighbors (KNN)

Pendiente

. 15

A

Area Under the Curve (AUC)

Pendiente

. 62

Artificial Intelligence (AI)

Inteligencia Artificial, área de las ciencia de la computación que se encarga de la creación de maquinas inteligentes que tienen un comportamiento que se asemeja a tener inteligencia. 10, 29, 39, 56

Artificial Neural Network (ANN)

Para predecir la probabilidad de crímenes y nuevos ataques terroristas. 5, 7, 18

B

Bag of Words (BoW)

Es una representación simplificada usada en procesamiento de lenguaje natural. En este modelo, un texto es representado como una bolsa (multiconjunto) de sus palabras. 30, 31, 38, 41, 42, 46

Bias

Es la parcialidad de un modelo, donde entre mayor sea peor sera la predicción del modelo respecto a cualquier dato que se le de como entrada. 59

Bidirectional Long Short Term Memory (Bi-LSTM)

Red neuronal LSTM bidireccional, que permite el uso de secuencias de palabras futuras y pasadas para una mejor clasificacion de la actual. 43–45, 50

Bidirectional Recurrent Neural Network (Bi-RNN)

Pendiente

. 10

C

Clustering

Es la tarea de agrupar conjuntos de objetos de manera que este en grupos de mismo tipo. 4, 19, 33, 35

Corpus

Es un conjunto grande y estructurado de textos, compuesto principalmente por documentos que contienen terminos (palabras) ordenados. 25, 26, 30–32, 38, 39

Coursera

Sitio web de cursos de aprendizaje en https://www.coursera.org. 3

Cross Industry Standard Process for Data Mining (CRISP-DM)

Proporciona una descripción normalizada del ciclo de vida de un proyecto estándar de análisis de datos. El modelo CRISP-DM cubre las fases de un proyecto, sus tareas respectivas, y las relaciones entre estas tareas. 35

D

Data Integration

Para acceder a múltiples y diversas fuentes de información. 5

Data Mining

Proceso de descubrir automáticamente información útil en repositorios grandes de datos. 7

Data Science

Es una campo multidisciplinario que usa metodos, procesos, algoritmos y sistemas cientificos para extraer conocimiento y revelaciones de datos estructurados y no estructurados. 4

Dataset

Conjunto de datos, en el campo de Inteligencia Artificial generalmente se refiere a conjuntos públicos o de acceso limitado de donde es posible obtener información útil con sistemas de computo. 1

Deductive Systems

Sistemas deductivos, son sistemas basados en conocimiento que en base al estado de la memoria de trabajo y a las reglas presentes en la base de conocimiento, agregan nuevas inferencias a la memoria de trabajo. 11

Directed Acyclic Graph (DAG)

Pendiente

. 8

E

Expert System

Sistemas expertos, son sistemas de computo que emulan la abeldad de toma de decisiones de un humano experto. 11

F

Feedforward

Es un termino que describe típicamente un elemento o un camino dentro de un sistema de control que pasa una señal de control de una fuente externa. En la inteligencia artificial este termino no difiere en significado. 18

I

Inference Engine (IE)

Motor de inferencia, es un componente del sistema que aplica reglas lógicas a la base de conocimiento para deducir información nueva. 11

K

Knowledge Base (KB)

Base de conocimiento, es una tecnología usada para almacenar información estructurada y no-estructurada usada por un sistema de computo. 11

Knowledge Based Systems (KBS)

Sistemas Basados en Conocimiento, su objetivo es abstraer conocimiento de un experto de un área en una representación dentro de un computador. 10, 11

L

Link Analysis

Para visualizar asociaciones y relaciones criminales y terroristas. 5

Long Short Term Memory (LSTM)

Red neuronal recurrente que tiene memoria Largo y Corto Plazo, son un tipo de redes neuronales recurrentes que son capaces de aprender dependencias de largo plazo. 42–44

M

Machine Learning Algorithms

Para extraer perfiles de perpetradores y mapas gráficos de crímenes. 5

Machine Learning (ML)

Informalmente ha sido definido como "El campo de estudio que le da a computadores la habilidad de aprender sin ser explícitamente programados", este tiene tres tipos de algoritmos de aprendizaje: aprendizaje supervisado, aprendizaje no–supervisado, y aprendizaje por refuerzo. 6, 8, 17, 29, 53–58

Maximum Likelihood Estimation (MLE)

Pendiente

. 25, 27

Maximum Margin Hyperplanes

Hiperplanos que permiten separar datos en espacios de alta dimensionalidad con un margen asociado para separarlos. 12

Multilayer Perceptrons (MLP)

Pendiente

. 7, 8, 18, 32

N

Named Entities

Son típicamente palabras que denotan personas particulares u organizaciones, pero no son sustantivos. 4, 42, 43

Natural Language Processing (NLP)

Rama de la inteligencia artificial que lidia con la interacción entre computadores y humanos usando el lenguaje natural. 1–5, 22, 25, 29, 30, 38

Neural Language Model (NLM)

Pendiente

. 31

O

Open Source Intelligence (OSINT)

Disciplina responsable de la adquisición, procesamiento y posterior transformación en inteligencia de información obtenida de fuentes públicas como prensa, radio, televisión, internet, informes de diferentes sectores y, en general, cualquier recurso de acceso público (Tomado de [2]). 50

Out of Vocabulary (OOV)

Pendiente

. 26

Overfitting

El modelo esta demasiado acoplado a la entrada por lo que para un modelo predictivo dará resultados solo condicionados para la entrada de entrenamiento, pero tendrá mal desempeño en cualquier otro conjunto de prueba. 20, 32, 39, 54, 56–58

P

Perplexity

Pendiente

. 25, 26

R

Reactive Systems

Sistemas reactivos, son sistemas basados en conocimiento que en base al estado de la memoria de trabajo y a las reglas presentes en la base de conocimiento, agregan o eliminan inferencias de la memoria de trabajo. 11

Receiver-Operating Characteristic (ROC)

Pendiente

. 62

Recurrent Neural Network (RNN)

Redes que recuerdan salidas y entradas pasadas de datos, de forma que decisiones futuras respecto a una clasificación pueden tener un mejor resultado. 8–10

Reinforcement Learning

Aprendizaje de refuerzo, es el aprendizaje de que acción se debe tomar de forma que se logre una señal de recompensa lo mas alto posible. 6

Rule–based Systems (RBS)

Sistemas basados en reglas, son sistemas expertos que en su base de conocimiento contienen reglas para crear nuevas inferencias. 11

S

Self-organizing Map (SOM)

Mapas auto-organizados, son redes neuronales especializadas para el clustering de datos semejantes según una función de distancia, da como resultado un mapa discreto uni- o bi-dimensional. 18–20, 33, 49, 50

Software Agents

Para el monitoreo, obtención, análisis y actuación sobre la información. 5

StarSpace

Es un modelo neuronal de propósito general para el aprendizaje eficiente de embeddings. 4, 48, 50

Supervised Learning

Aprendizaje supervisado, su objetivo es aprender un mapeo de unas entradas a unas salidas definidas. 6

Support Vector Machine (SVM)

Maquinas de Soporte Vectorial, son modelos de aprendizaje supervisado usados para el problema de clasificación y análisis de regresión. 12, 13, 33

T

TensorFlow

TensorFlow es una plataforma de codigo abierto para machine learning. Tiene un ecosistema comprensible y flexible de herramientas, librerias y recursos de la comunidad que le permite a investigadores impulsar el estado del arte en machine learning y desarrolladores puede construir y desplegar facilmente aplicaciones con Machine Learning incorporado. 45

Term Frequency – Inverse Document Frequency (TF-IDF)

Frecuencia de términos – Frecuencia inversa de documento, es una estadística numérica que esta diseñada para reflejar que tan importante es una palabra a un documento dentro de un corpus. 30, 41, 42, 46

Text Mining

Búsqueda sobre terabytes de información en documentos, paginas web y correos electrónicos. 5

U

Underfitting

El modelo predice pobremente la salida correcta en base a la entrada, de forma que no da buenos resultados para cualquier conjunto de entrenamiento o de pruebas. 9, 20, 39, 54, 56, 58

Unsupervised Learning

Aprendizaje no-supervisado, su objetivo es obtener patrones estructurados de una serie de datos no estructurados. 6, 18

W

Working Memory (WM)

Memoria de trabajo, es donde se encuentra el estado actual del sistema antes y después de haber ejecutado reglas de un sistema basado en reglas, y es donde se encuentran las inferencias hechas por el motor de inferencia. 11

Bibliografía

[1] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[2] Martín José Hernández Medina, Ricardo Andrés Pinto Rico y Cristian Camilo Pinzón Hernández. *Inteligencia de fuentes abiertas para el contexto colombiano*. Inf. téc. Escuela Colombiana de Ingeniería Julio Garavito, 2018.

[3] Ana Maria Popescu y Orena Etzioni. «Extracting product features and opinions from reviews». En: *Natural Language Processing and Text Mining* (2007), págs. 9-28. ISSN: 08247935. DOI: 10.1007/978-1-84628-754-1_2. arXiv: 0309034 [cs].

[4] Jesús Mena. *Investigative Data Mining for Security and Criminal Detection*. Elsevier Science, 2003. ISBN: 9780080509389. URL: https://books.google.com.co/books?id=3mDlrtJuZv4C.

[5] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. Cambridge, Mass. [u.a.]: MIT Press, 2013. ISBN: 9780262018029 0262018020. URL: https://www.amazon.com/Machine-Learning-Probabilistic-Perspective-Computation/dp/0262018020/ref=sr_1_2?ie=UTF8&qid=1336857747&sr=8-2.

[6] Pang-Ning Tan, Michael Steinbach y Vipin Kumar. *Introduction to Data Mining*. US ed. Addison Wesley, mayo de 2005. ISBN: 0321321367. URL: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/0321321367.

[7] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[8] *Understanding LSTM Networks*. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[9] Priti Srinivas Sajja y Rajendra Akerkar. «Knowledge-based systems for development». En: *Advanced Knowledge Based Systems: Model, Applications & Research* 1 (2010), págs. 1-11.

[10] Jerry M Mendel. *Uncertain Rule-Based Fuzzy Systems*. ISBN: 9783319513690. DOI: 10.1007/978-3-319-51370-6.

[11] Leandro Nunes De Castro. *Fundamentals of natural computing: basic concepts, algorithms, and applications*. Chapman y Hall/CRC, 2006.

[12] Martin Jurafsky. *Speech and Language Processing*. Prentice Hall, 2009.

[13] Rebecca S. Portnoff y col. «Tools for Automated Analysis of Cybercriminal Markets». En: (2017), págs. 657-666. DOI: 10.1145/3038912.3052600.

[14] Will Y Zou y col. «Bilingual Word Embeddings for Phrase-Based Machine Translation». En: *Proceedings ofthe 2013 Conference on Empirical Methods in Natural Language Processing* (2013), págs. 1393-1398. URL: http://www.aclweb.org/anthology/D13-1141.

[15] Raymond Y.K. Lau, Yunqing Xia y Yunming Ye. «A probabilistic generative model for mining cybercriminal networks from online social media». En: *IEEE Computational Intelligence Magazine* 9.1 (2014), págs. 31-43. ISSN: 1556603X. DOI: 10.1109/MCI.2013.2291689.

[16] Julian Szymański y Włodzisław Duch. «Self Organizing Maps for Visualization of Categories». En: *Neural Information Processing*. Ed. por Tingwen Huang y col. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, págs. 160-167. ISBN: 978-3-642-34475-6.

[17] Victor Benjamin y Hsinchun Chen. «Identifying language groups within multilingual cybercriminal forums». En: *IEEE International Conference on Intelligence and Security Informatics: Cybersecurity and Big Data, ISI 2016* (2016), págs. 205-207. DOI: 10.1109/ISI.2016.7745471.

[18] Alberto P. García-Plaza, Víctor Fresno y Raquel Martínez. «Web page clustering using a fuzzy logic based representation and Self-Organizing Maps». En: *Proceedings - 2008 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2008* (2008), págs. 851-854. DOI: 10.1109/WIIAT.2008.249.

[19] O. Tange. «GNU Parallel - The Command-Line Power Tool». En: *;login: The USENIX Magazine* 36.1 (2011), págs. 42-47. DOI: http://dx.doi.org/10.5281/zenodo.16303. URL: http://www.gnu.org/s/parallel.

[20] Lars Buitinck y col. «API design for machine learning software: experiences from the scikit-learn project». En: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, págs. 108-122.

[21] Zhiheng Huang, Wei Xu y Kai Yu. «Bidirectional LSTM-CRF Models for Sequence Tagging». En: (2015). arXiv: 1508.01991. URL: http://arxiv.org/abs/1508.01991.

[22] Diederik P. Kingma y Jimmy Ba. «Adam: A Method for Stochastic Optimization». En: (2014), págs. 1-15. arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

[23] Jeffrey Pennington, Richard Socher y Christopher Manning. «Glove: Global vectors for word representation». En: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, págs. 1532-1543.

[24] Tomas Mikolov y col. «Efficient Estimation of Word Representations in Vector Space». En: *Intelligent Systems Reference Library* 58 (2014), págs. 1-12. ISSN: 18684408. DOI: 10.1007/978-3-319-01967-3_1. arXiv: arXiv:1301.3781v3.

[25] Armand Joulin y col. «FastText.zip: Compressing text classification models». En: *arXiv preprint arXiv:1612.03651* (2016).

[26] L. Wu y col. «StarSpace: Embed All The Things!» En: *arXiv preprint arXiv:1709.03856* (2017).

[27] Andreas Madsen. «Visualizing memorization in RNNs». En: *Distill* (2019). DOI: 10.23915/distill.00016. URL: https://distill.pub/2019/memorization-in-rnns.

[28] Jeremy Kun. *A programmers introduction to mathematics*. CreateSpace Independent Publishing Platform, 2018.

[29] Kelly H. Zou, A. James O'Malley y Laura Mauri. «Receiver-Operating Characteristic Analysis for Evaluating Diagnostic Tests and Predictive Models». En: *Circulation* 115.5 (2007), págs. 654-657. ISSN: 0009-7322. DOI: 10.1161/CIRCULATIONAHA.105.594929. URL: https://www.ahajournals.org/doi/10.1161/CIRCULATIONAHA.105.594929.

[30] Patrick Henry Winston. *Artificial Intelligence (3rd Ed.)* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1992. ISBN: 0-201-53377-4.

[31] Steven Bird, Ewan Klein y Edward Loper. *Natural Language Processing with Python*. 1st. O'Reilly Media, Inc., 2009. ISBN: 0596516495, 9780596516499.