



C-YANGUAS

CLASIFICADOR DOCUMENTAL - 2021

Procesamiento del lenguaje natural

PILA TECNOLÓGICA



LIBRERÍAS UTILIZADAS

- NLTK
- PYPANDOC
- NUMPY
- SKLEARN
- MATPLOTLIB
- KERAS

Contenidos

Contenidos	I
Lista de Figuras	II
1. Introducción	1
1.1. Organización	1
2. Conjunto de datos	3
2.1. Metodología, fuentes de información y organización	3
2.2. Preprocesado	4
2.2.1. Librerías	4
2.2.2. Operaciones realizadas	7
3. Obtención de glosario	8
4. Clasificadores	10
4.1. Máquinas de vector soporte (SVM)	10
4.1.1. Clasificador SVM con glosario supervisado	10
4.1.2. Clasificador SVM con SVD	11
4.2. Naïve bayes con glosario	12
4.3. Random forest TFIDF-SVD	12
4.4. Redes neuronales con SVD	13
5. Evaluación de los clasificadores	17
5.1. Evaluación del clasificador basado en SVM con glosario	18
5.2. Evaluación del clasificador basado en SVM con SVD	18
5.3. Evaluación del clasificador basado en redes neuronales	19
5.4. Evaluación del clasificador naïve Bayes	19
5.5. Evaluación del clasificador Random Forest	20
5.6. Comparativa	21
6. Conclusiones	23
Bibliografía	25
Apéndice	26
.1. Figuras	26
.2. Notebooks	35
.2.1. Preprocesamiento - transformacion.ipynb	35
.2.2. Modelo SVM con glosario - Modelo_SVM.ipynb	43
.2.3. Modelo SVM con SVD - Modelo_SVM_SVD.ipynb	56
.2.4. Modelo basado en redes neuronales con SVD Modelo_NN.ipynb	65
.2.5. Modelo Naïve Bayes - Modelo_NB.ipynb	83
.2.6. Modelo Random Forest - Modelo_NB.ipynb	92

Listado de Figuras

1.1. Estructuración de los directorios y archivos profundidad 1-2-3.	2
1.2. Estructuración de los directorios y archivos profundidad 3-4-5.	2
2.1. Estructuración de los directorios y archivos.	4
3.1. Diferencia simétrica.	9
4.1. Margen de un hiperplano de separación.	11
4.2. Aproximación One vs One.	11
4.3. Arquitectura de una red neuronal profunda.	13
4.4. Ejemplo de traducción automática de DeepL.	14
4.5. Ejemplo de como se pueden distribuir las funciones de activación.	15
4.6. Funciones de activación utilizadas en el modelo.	15
5.1. Informe de métricas del modelo SVM con glosario.	18
5.2. Matriz de confusión del modelo SVM con glosario.	18
5.3. Informe de métricas del modelo SVM con SVD.	19
5.4. Matriz de confusión del modelo SVM con SVD.	19
5.5. Informe de métricas del modelo NN.	19
5.6. Matriz de confusión del modelo NN.	20
5.7. Informe de métricas del modelo naïve Bayes.	20
5.8. Matriz de confusión del modelo naïve Bayes.	20
5.9. Informe de métricas del modelo Random Forest.	21
5.10. Matriz de confusión del modelo Random Forest.	21
5.11. Comparativa de las puntuaciones medias de los modelos para las métricas seleccionadas.	21
5.12. Comparativa de las puntuaciones de la precisión y desviación típica de las validaciones cruzadas aplicadas a los clasificadores.	22
6.1. Resumen de los procedimientos llevados a cabo para la realización de la práctica.	23
2. Glosario inicial con 100 términos por temática.	27
3. Glosario por temática supervisado.	28
4. Arquitectura del modelo basado en redes neuronales.	29
5. Predicciones probabilísticas del modelo SVM con glosario.	30
6. Predicciones probabilísticas del modelo VM con SVD.	31
7. Predicciones probabilísticas del modelo basado en redes neuronales.	32
8. Predicciones probabilísticas del modelo Naive Bayes.	33
9. Predicciones probabilísticas del modelo Random Forest.	34

Capítulo 1

Introducción

El objetivo de este documento es explicar de la mejor manera posible el trabajo llevado a cabo, este consiste en responder a la necesidad de clasificación de documentos en una serie de temáticas dadas a priori, en nuestro caso serán deportes, salud y política. Veamos los diferentes enfoques que pueden ser llevados a cabo para responder al problema:

1. **Clasificación manual:** Consiste en encargar a una o varias personas que se lea cada uno de los documentos que deben ser clasificados y después, trate de clasificarlos en sus respectivas temáticas según su criterio.
2. **Clasificación automática:** Consiste en automatizar la clasificación de documentos, es decir, dando un determinado texto, conseguir que se introduzca en la carpeta adecuadamente de manera automática. Para llevar este objetivo a cabo existen dos posibilidades:
 - Clasificador automático con glosario: Consiste en utilizar un glosario¹ de manera que el clasificador pueda ponderar la aparición de dichas palabras en un documento que es necesario clasificar para así clasificarlo en aquel que resulte en una mayor puntuación. Por tanto, el glosario en este enfoque es de vital importancia que sea adecuado.
 - Clasificador automático sin glosario: Esta aproximación consiste en tratar de clasificar los documentos sin una serie de temáticas previamente conocidas. Para ello, un enfoque habitual es la aplicación de un algoritmo de clustering para conseguir agrupar por temáticas los documentos, aunque no se conozca exactamente a qué se refiere cada temática, y después crear un glosario para cada una de estas temáticas.

Dado que en nuestro trabajo tenemos una serie de temáticas establecidas a priori utilizaremos la aproximación de clasificador automático con glosario. Este trabajo es útil en el sentido de que, se evita la necesidad de que esta tarea sea llevada a cabo por una persona de forma manual, lo que ahorra tiempo y dinero en cualquier contexto al que se aplique. Además, al conocer las temáticas a priori y los documentos estar correctamente etiquetados, se podrá entrenar los diferentes clasificadores de una manera más sencilla y eficaz, lo que permitirá que su precisión sea elevada.

1.1. Organización

A continuación se exponen las diferentes secciones en las que esta dividida el trabajo, procurando cubrir cada una de ellas la explicación que cubre una parte fundamental del trabajo realizado.

- **Capítulo 2: Conjunto de datos:** En este capítulo describiremos la metodología de extracción de los documentos con los que trabajaremos, ofreciendo además la fuente a partir de la cual se obtuvo. Además se ofrecerán las diferentes transformaciones que se realizaron sobre los documentos extraídos inicialmente para que sean más eficientes para los modelos que trataremos, así como su consecuente explicación del porqué.
- **Capítulo 3: Obtención de glosario:** En este capítulo describiremos cuales han sido las aproximaciones realizadas para obtener los glosarios que utilizaremos para nuestros modelos.

¹Glosario: Conjunto de temáticas en los que se puede clasificar el documento así como un conjunto de palabras que se consideren importantes o relevantes para cada una de las temáticas.

■ **Capítulo 4: Creación y desarrollo de clasificadores:** En este capítulo describiremos los diferentes enfoques y arquitecturas utilizadas para establecer cada uno de los clasificadores.

■ **Capítulo 5: Evaluación de los clasificadores:** En este capítulo se ofrece una breve descripción de cada una de las métricas de los diferentes clasificadores, así como una comparativa cuantitativa entre estos.

Capítulo 6: Conclusiones: En este capítulo determinaremos las conclusiones más relevantes que hemos extraído del conjunto de resultados obtenidos.

Apéndice: En este área nos encontraremos todo lo relativo a esquemas, grandes tablas o los cuadernos sobre los que se ha desarrollado el código.

Además en Figuras [1.1, 1.2] se ofrece un esquema de la organización de los archivos entregados.

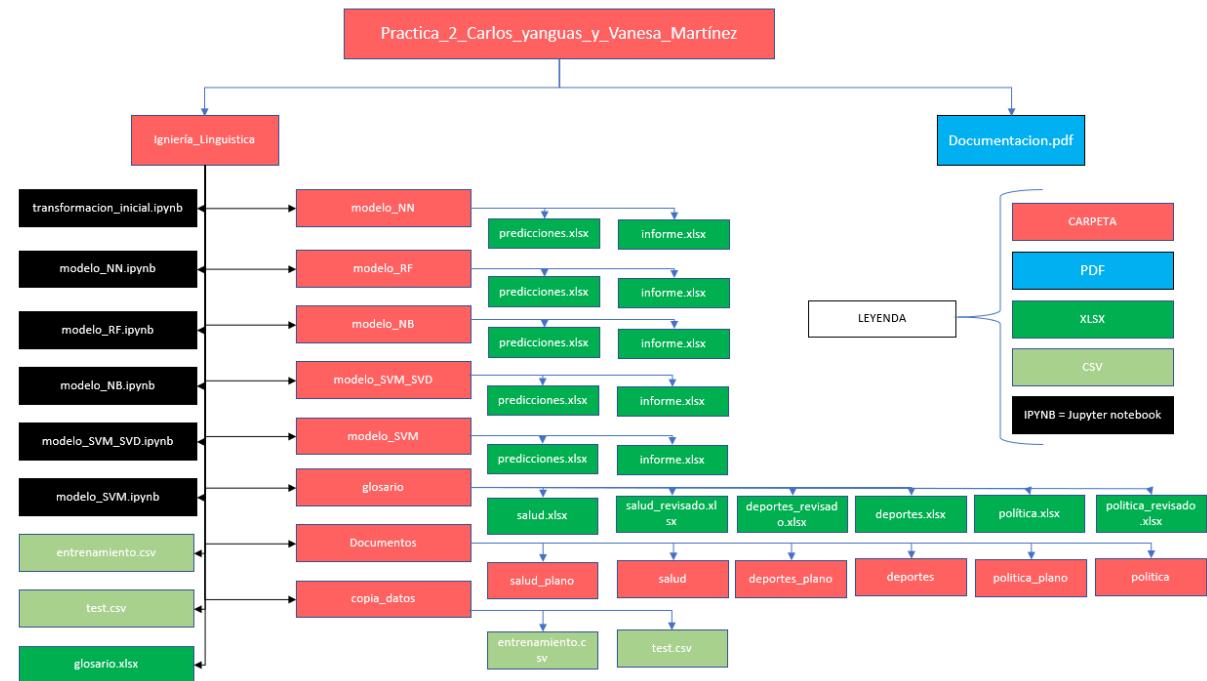


Figura 1.1: Estructuración de los directorios y archivos profundidad 1-2-3.

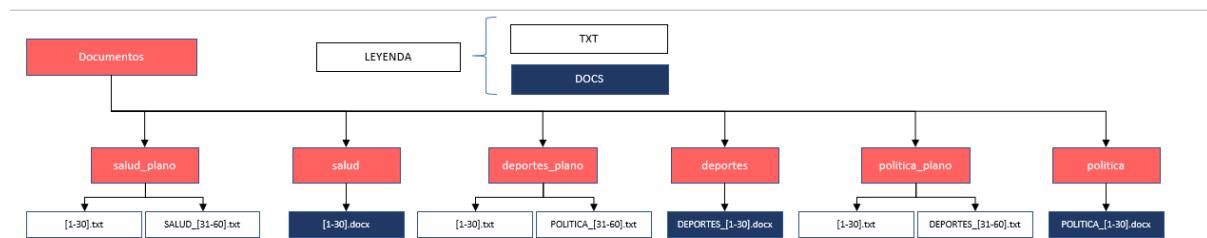


Figura 1.2: Estructuración de los directorios y archivos profundidad 3-4-5.

Se ha querido añadir la carpeta **copia_datos** para que en caso de querer ejecutar el notebook **transformacion_inicial**, se siga contando con la misma distribución de los conjuntos de datos de entrenamiento y test en dicha carpeta. Esto es debido a que estos datos se sobrescribirían en la carpeta inicial, ya que este cuaderno realiza una mezcla aleatoria y se obtendrían distintos resultados.

Capítulo 2

Conjunto de datos

En este capítulo se expone la metodología para la obtención de los documentos, la organización para los mismos y el preprocesado llevado a cabo para poder extraer información relevante de los mismos.

2.1. Metodología, fuentes de información y organización

La metodología para la obtención de documentos ha sido manual. Es la que hemos considerado más adecuada, ya que, pese a poder haber adoptado otras metodologías automáticas, el hecho de haberlas aprendido y aplicado hubiese supuesto más tiempo. Cabe aclarar que esto es debido a que se va a tratar con un conjunto de datos muy reducidos, y que no sería conveniente para un caso de aplicación real. Las fuentes utilizadas se exponen a continuación:

- Deportes
 - 1. **Marca**: Es un diario español de información deportiva de carácter y alcance nacional. También abarca noticias y evoluciones relativas a la mayoría de deportes practicados en España y/o con repercusión internacional.
 - 2. **El Mundo**: Es el segundo periódico español más importante por difusión y audiencia. Es hoy uno de los diarios más influyentes en la sociedad española. Su página web, es líder entre los diarios generalistas.
 - 3. **ABC**: Ha sido definido como un periódico conservador, monárquico y católico. El editorial del 1 de junio de 1905 afirmaba que «en política, (ABC) no seguirá bandera alguna para no mermar su independencia, dentro de la cual se propone vivir sin abdicar uno solo de sus fueros».
 - 4. **El Confidencial**: es un diario digital español de información general, especializado en noticias económicas, financieras y de actualidad política fundado en 2001. Está orientado hacia un público profesional de mediana edad. Se financia sobre todo mediante publicidad, eventos y contenido de marca (branded content).
 - 5. **El Español**: Es un diario digital editado en España y en idioma español, disponible en Internet desde octubre de 2015, con sede principal en Madrid.
 - 6. **Mundo Deportivo**: Es un diario deportivo español editado en Barcelona por el Grupo Godó. se centra principalmente en informar acerca de la actualidad futbolística, en especial la del Fútbol Club Barcelona, y polideportiva.
- Política
 - 1. **El Periódico**: Es un diario español de información política de carácter y alcance nacional.
 - 2. **El Confidencial**
 - 3. **ABC**
 - 4. **El Español**
 - 5. **El Mundo**
- Salud

1. **Noticias en salud:** Noticiario independiente donde se publican las noticias más relevantes del mundo sanitario desde un punto de vista diferente.
2. **Yo salud:** Se trata de un blog de salud y bienestar.

Con el objetivo de dividir el trabajo, cada uno de los integrantes del grupo ha buscado 30 documentos para cada una de las temáticas establecidas. Además, fueron subidas a una carpeta compartida de onedrive denominada práctica-2 en la cual existen otras tres carpetas, una para cada temática, y en cada carpeta existen 30 documentos word representando cada uno de ellos un texto de la temática a la que pertenece según su carpeta. Podemos visualizar esta organización en Figura 2.1.

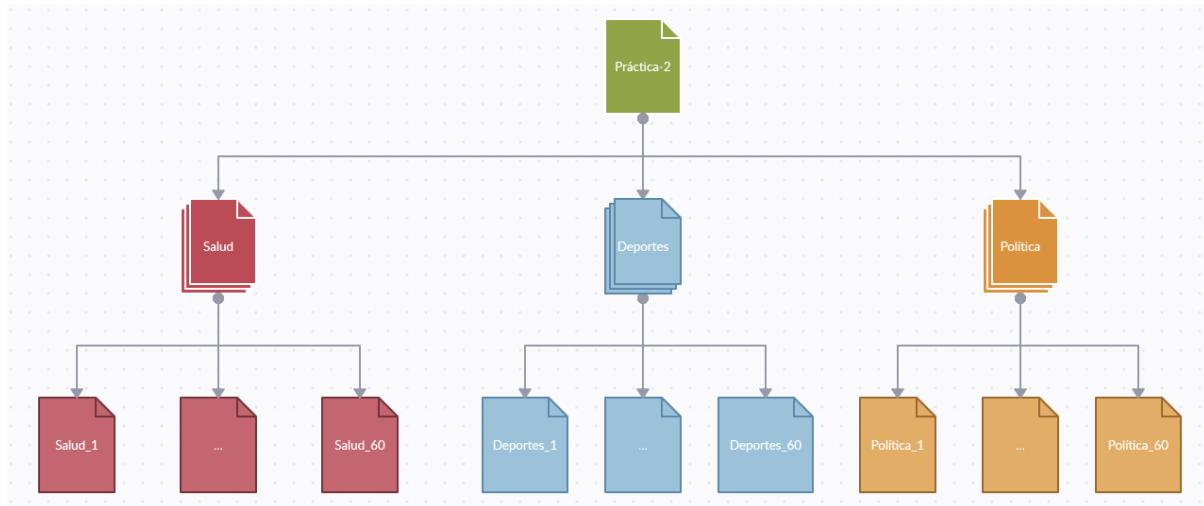


Figura 2.1: Estructuración de los directorios y archivos.

2.2. Preprocesado

El preprocesado de los datos es una parte fundamental del problema, ya que es el conjunto de transformaciones sobre los documentos extraídos directamente que permitirá obtener las palabras de manera clara para poder incorporar las más relevantes a nuestro glosario.

Con el objetivo de llevar estas transformaciones a cabo y a la vez incorporar comentarios y rapidez, hemos decidido utilizar **Google Colab** que es una herramienta que permite ejecutar y programar en Python desde un navegador. Los cuadernos Colab permiten combinar código ejecutable y texto enriquecido en un mismo documento, lo que facilita la legibilidad y estructuración. Estos cuadernos se almacenan en las cuentas de Google Drive, pudiendo compartir los mismos fácilmente. Las principales ventajas de Google Colab es que, no requiere de ninguna configuración y que el código de los mismos se ejecutan en los servidores en la nube de Google, lo que permite aprovechar la potencia de los recursos hardware que Google ofrece, entre otros, GPUs.

Por su parte, el lenguaje de programación **Python** surge bajo la filosofía de legibilidad y menor complejidad, es por ello que muchos desarrolladores eligen este lenguaje para llevar a cabo sus proyectos. Esto tiene como consecuencia la aparición de una gran variedad de repositorios de código abierto desarrollados por diferentes usuarios de la comunidad, lo que permite que algoritmos complejos y difíciles de desarrollar estén al alcance de cualquiera.

Además de esto, Python ofrece ventajas en términos de velocidad de ejecución. Esto es debido a que puede hacer llamadas a bibliotecas externas convirtiéndose en una envoltura para el código C/C++. El lenguaje de programación C/C++ es conocido entre los desarrolladores por ser el lenguaje de programación más potente. Todo esto junto con las diversas herramientas que ofrece para las diferentes áreas del aprendizaje automático es lo que lo convierte líder en este área.

Por tanto, elegiremos Python para desarrollar nuestro proyecto, ya que ofrece un código legible y poco complejo con velocidades de ejecución cercanas a los lenguajes más eficientes en este sentido.

2.2.1. Librerías

A continuación se presentan las librerías más relevantes utilizadas para desarrollar nuestro proyecto.

1. **Numpy para reducción de tiempos:** Numpy es uno de los principales paquetes orientados a la computación científica en Python. Esta librería es muy útil debido a que permite realizar las siguientes operaciones entre otras:

- a) Operaciones matemáticas.
- b) Operaciones lógicas.
- c) Operaciones de redimensionamiento.
- d) Operaciones de ordenación.
- e) Operaciones de selección.
- f) Operaciones de entrada y salida.
- g) Operaciones de álgebra lineal básica.
- h) Operaciones estadísticas.
- i) Operaciones de simulación aleatoria.
- j) Transformadas discretas de Fourier.

El núcleo del paquete NumPy es el objeto ndarray. Este encapsula matrices n-dimensionales de tipos de datos homogéneos. Hay varias diferencias importantes entre las matrices de NumPy y las secuencias estándar de Python:

- a) Las matrices NumPy tienen un tamaño fijo en el momento de su creación, a diferencia de las listas de Python (que pueden crecer dinámicamente). Si se cambia el tamaño de una matriz NumPy, se creará una nueva matriz y se borrará la original.
 - b) Los elementos de una matriz NumPy deben ser todos del mismo tipo de datos, y por tanto tendrán el mismo tamaño en memoria. En caso de tener la necesidad de incorporar diferentes objetos en una matriz, es posible la creación de matrices de objetos, permitiendo así matrices de elementos de diferentes tamaños.
 - c) Las matrices NumPy facilitan las operaciones matemáticas avanzadas y otros tipos de operaciones sobre grandes cantidades de datos. Normalmente, estas operaciones se ejecutan de forma más eficiente y con menos código de lo que es posible utilizando las secuencias incorporadas de Python. La velocidad con la que se realizan estas operaciones se debe a la vectorización, la cual permite la ausencia de bucles, indexaciones y este tipo de código que puede derivar a la ralentización cuando tratamos con una gran cantidad de datos. Estas operaciones se realizan a través de código C precompilado y optimizado.
 - d) En NumPy es importante tener en cuenta el concepto de broadcasting, es decir, que todas las operaciones se realizan de manera implícita a todos los elementos de la matriz a la cual aplicamos dicha operación.
2. **Pandas para lectura, estructuración, transformación y limpieza de datos:** Pandas se utiliza muy frecuentemente en tareas de ciencia de datos, análisis de datos y aprendizaje automático. Esta librería surge debido a la necesidad de una herramienta de análisis de datos de un alto nivel. Está construida sobre la librería Numpy, por tanto, esta librería nos permitirá realizar todo tipo de operaciones ofrecidas por NumPy pero de una manera más sencilla. Las características principales de Pandas son:

- a) DataFrame: Se trata de un objeto rápido y eficiente con indexación predeterminada y personalizada.
- b) Lectura de datos: Para la lectura de datos también es muy utilizada esta librería, esto es debido a que podemos leer los mismos desde fuentes como CSV, texto, Excel, bases de datos SQL o formato HDF5 utilizado en bases de datos distribuidas y almacenarlos directamente en un objeto DataFrame.
- c) Limpieza de datos: Pandas ofrece diferentes métodos para manipular los datos nulos que se encuentren en nuestro DataFrame.
- d) Visualización de datos: Pandas permite visualizar los datos de una manera muy sencilla a través de su objeto DataFrame, ya que hace uso de la librería Matplotlib que veremos en [5](#).

3. Scikit-learn para preprocessado de datos y selección y validación de modelos: Scikit-learn es una de las librerías más útiles para el aprendizaje automático en Python. Esta librería contiene múltiples herramientas que entre otras, permite llevar a cabo:

- a) Algoritmos de aprendizaje supervisados¹: Entre otros, permite aplicar, regresiones lineales, árboles de decisión o métodos bayesianos.
- b) Validación cruzada²: Dispone de varios métodos para comprobar la precisión de los modelos.
- c) Algoritmos de aprendizaje no supervisados³: Esta librería ofrece diferentes algoritmos como agrupación, o redes neuronales no supervisadas.
- d) Obtención de varios conjuntos de datos de prueba: Ofrece diferentes conjuntos de datos para comprender mejor el funcionamiento de sus algoritmos.
- e) Extracción de características: Nos permite extraer las características más importantes de un conjunto de datos de manera simple. Por ejemplo, las palabras más utilizadas en las opiniones de los clientes de una empresa.
- f) Normalización de datos: Permite aplicar diferentes técnicas de normalización de datos según la naturaleza del problema.
- g) Métricas: Esta librería contiene diferentes métricas que permiten evaluar modelos de manera sencilla y comprensiva.

4. NLTK Natural Language Toolkit para preprocessado de datos: NLTK es una plataforma líder para crear programas en Python que trabajen con datos del lenguaje humano. Ofrece interfaces fáciles de usar para más de 50 corpus y recursos léxicos como WordNet, junto con un conjunto de bibliotecas de procesamiento de texto para la clasificación, la tokenización, el stemming, el etiquetado, el análisis sintáctico y el razonamiento semántico, envoltorios para bibliotecas de PNL industriales y un foro de discusión activo.

Gracias a una guía práctica que introduce los fundamentos de la programación junto con temas de lingüística computacional, además de una completa documentación de la API, NLTK es adecuado para lingüistas, ingenieros, estudiantes, educadores, investigadores y usuarios de la industria por igual. NLTK está disponible para Windows, Mac OS X y Linux. Lo mejor de todo es que NLTK es un proyecto gratuito, de código abierto e impulsado por la comunidad.

NLTK ha sido calificado como "una herramienta maravillosa para enseñar y trabajar en lingüística computacional usando Python; una biblioteca increíble para jugar con el lenguaje natural".

Natural Language Processing with Python ofrece una introducción práctica a la programación para el procesamiento del lenguaje. Escrito por los creadores de NLTK, guía al lector por los fundamentos de la escritura de programas en Python, el trabajo con corpus, la categorización de textos, el análisis de la estructura lingüística, etc.

5. Matplotlib para visualización de resultados: Matplotlib es una biblioteca multiplataforma de visualización de datos y trazado de gráficos para Python y su extensión numérica NumPy. Como tal, ofrece una alternativa viable de código abierto a MATLAB. Uno de los mayores beneficios de la visualización que ofrece, es que permite el acceso visual a enormes cantidades de datos en gráficos fácilmente interpretables. Matplotlib consta de varios tipos de gráficos como línea, barra, dispersión o histograma.

6. Keras para construcción y entrenamiento de modelos predictivos basados en redes neuronales: Keras es una envoltura para el framework Tensorflow, ya que, mientras que Keras es una biblioteca centrada en redes neuronales, tensorflow se encarga de diversas tareas del aprendizaje automático. Keras es una API⁴ diseñada para crear modelos de una manera sencilla, ya que minimiza el número de acciones necesarias para crear los mismos y ofrece una respuesta clara en caso de error. Todo ello hace que Keras sea una librería muy atractiva, ya que aumenta la productividad de sus desarrolladores, permitiéndoles a los mismos realizar diferentes pruebas. Además de

¹ Aprendizaje supervisado: Técnica para deducir una función a partir de datos de entrenamiento

² Validación cruzada: técnica utilizada para evaluar los resultados de un análisis estadístico y garantizar que son independientes de la partición entre datos de entrenamiento y de validación. Se utiliza en entornos donde el objetivo principal es la predicción y se requiere estimar la precisión de un modelo.

³ Aprendizaje no supervisado: algoritmo de aprendizaje automático donde un modelo se ajusta a las observaciones.

⁴ API (interfaz de programación de aplicaciones): es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones.[?]

todo esto, hay que destacar que Keras, al estar integrada con tensorflow, trabaja a bajo nivel, lo que permite tiempos de ejecución bajos, así como el uso eficiente de los diferentes recursos hardware.

2.2.2. Operaciones realizadas

Una vez elegidos el lenguaje de programación, sus librerías y el entorno de ejecución. Veamos las distintas transformaciones llevadas a cabo:

1. **Paso de documentos word a csv:** Dado que inicialmente se volcaron los textos extraídos de la web directamente sobre documentos word, es importante pasar los mismos a texto plano, es decir, que el texto no tenga colores o tamaños de fuente diferentes entre otras cosas. Cabe decir que esta operación se anida sobre un bucle para transformar todos los words a txt sin mucha complejidad. Otro detalle a aclarar es que 30 documentos fueron subidos en formato word (del 1 al 30) y los otros 30 en formato txt, por lo que solo es necesario transformar 30 de los 60 totales. Una vez hecho esto, creamos tres listas que almacenarán todos los txt leídos para cada una de las temáticas. Seguidamente creamos tres DataFrames uno para cada temática, los cuales tienen una columna texto que será el texto plano y una columna clase que será de 0, 1 o 2 según si la temática es salud, política o deportes respectivamente. El siguiente paso realizado consiste en mezclar de manera aleatoria cada una de las filas de las tres temáticas, esto es debido a que cada uno de los miembros del grupo se encargo de obtener 30 documentos para cada temática, y debido a la fuente de información elegida por cada uno, podríamos obtener un sesgo. Este detalle lo hemos tenido en cuenta debido a que más adelante los glosarios se construirán a partir de los 45 primeros documentos de cada temática, y queremos que dicho glosario sea representativo sobre el total de documentos que después debamos clasificar. Por último concatenamos los tres DataFrames creados anteriormente, teniendo uno final con los 180 documentos y correctamente etiquetados todos ellos. Este DataFrame lo pasamos a un csv para no tener que volver a hacer estas transformaciones ni lecturas tan lentas, ya que podremos leer directamente el csv como DataFrame.
2. **Textos a minúsculas:** Dado que ahora podemos trabajar directamente con el DataFrame, podemos aplicar operaciones de manera muy simple, la primera de ellas es pasar a minúsculas todo el texto.
3. **Tokenizar:** La tokenización consiste en transformar el texto en tokens, es decir, en el conjunto de palabras y símbolos por el que están compuestos.
4. **Eliminación de signos de puntuación:** Dado que los signos de puntuación como exclamaciones, interrogaciones, comas, puntos, comillas y otros por si solos no aportan información adicional, los eliminaremos de nuestros textos. Sin embargo mantendremos las palabras con los acentos, ya que a veces dos palabras se diferencian por dicho acento.
5. **Eliminación de números:** Los números por si solos al igual que los signos de puntuación, no aportan información, por ello se ha decidido eliminarlos también.
6. **Eliminación de palabras vacías:** Las palabras vacías son las palabras en cualquier idioma que no agregan mucho significado a una oración. Pueden ignorarse con seguridad sin sacrificar el significado de la oración.
7. **Restauración de texto:** Dado que ya hemos aplicado todas las funciones de preprocesado de datos que hemos creído convenientes, deshacemos la tokenización.

Todo el código asociado a este proceso se puede ver en el notebook [transformacion.ipynb](#)

Capítulo 3

Obtención de glosario

Una vez hemos preprocesado todos los textos, el siguiente paso es extraer las palabras más importantes para cada una de las temáticas, es decir, obtener el glosario. Para ello, hemos decidido utilizar 45 de los 60 documentos de cada temática lo que corresponde a un 75 % de los datos, dejando el 25 % restante destinado a evaluar el comportamiento del clasificador. Cabe decir que hemos realizado esta división debido a que creemos que es una aproximación más realista, es decir, en una aproximación real, se entrenarán los modelos con mayor cantidad de documentos que los que luego clasificará el mismo. Una vez realizada la división de datos se ha extraído las palabras más relevantes convirtiendo la colección de documentos en una matriz de características TF-IDF, la cual se expresa formalmente como:

$$tf(t) = \frac{t}{NTD} \quad (3.1)$$

Dónde

- **tf(Term Frequency)** es el número de apariciones de un término en un determinado documento.
- **NTD**: Es el número de términos de un documento.
- **t**: Es el término del que se quiere obtener la frecuencia.

$$idf(t) = \log_{10} \left[\frac{1 + N}{1 + df(t)} \right] + 1 \quad (3.2)$$

Dónde

- **IDF(Inverse Document Frequency)**: Número de apariciones de un término sobre el total de documentos.

Conociendo estos dos términos, TF-IDF se define como:

$$tf - idf(t, d) = tf(t, d) \cdot idf(t) \quad (3.3)$$

Por tanto a nivel conceptual, el aplicar esta fórmula sobre el conjunto de términos dado por el conjunto de 45 documentos por temática, nos permitirá determinar una puntuación para cada uno de esos términos. Seguidamente tenemos que determinar con cuantos nos queremos quedar, es decir, a partir de cuantas palabras va a estar conformado cada uno de los tres glosarios pertenecientes a cada temática. En nuestro caso tras varias aproximaciones, hemos decidido quedarnos con los 100 primeros. Para ello simplemente hemos ordenado las puntuaciones asociadas a cada palabra de manera descendente y hemos seleccionado las 100 primeras. El resultado de estos glosarios iniciales se puede ver en Figura 2.

Sin embargo, si tras hacer un análisis de las palabras que aparecen, se ha decidido eliminar varias de ellas:

1. **Salud**: Si, ser, puede, pueden, cada, mejor, bien, según, hacer, manera, día, hoy, también, días, así, hace, solo, dos, siempre, ejemplo, incluso, mismo, muchas. Es decir se han eliminado 23 palabras.
2. **Política**: Dos, ahora, hace, ser, tras, siempre, sido, pasado, meses, solo, bien, según, aunque, mismo, año. Suponiendo una eliminación de 15 palabras.

3. Deportes: Dos, tres, si, sido, ser, tras, después, solo, ahora, así, hace, siempre, tan, aunque. Reduciendo en 14 palabras el glosario para la temática deportes.

Además, tras ello, se ha aplicado otro filtro. Este consiste en hacer la diferencia simétrica entre los glosarios resultantes, es decir, se eliminan las palabras que aparezcan en dos glosarios o más. Creemos que esta operación puede ser útil debido a que si una palabra aparece en varias temáticas, significa que no es representativa únicamente de una temática. En Figura 3.1 podemos ver visualmente esta operación.

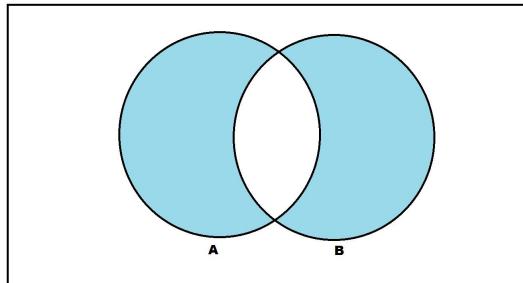


Figura 3.1: Diferencia simétrica.

De esta manera el glosario resultante conformado por 71 palabras de las temáticas salud y política y 70 por deportes, siendo el glosario total 212 palabras se puede ver en Figura 3.

Además hemos querido incluir otra aproximación para realizar comparativas con el glosario extraído de esta manera. Esta consiste en obtener una matriz TFIDF mediante todo el conjunto de entrenamiento (45 documentos por temática), y tras ello aplicar una simplificación Single Value Decomposition (SVD) a un número limitado de 100 elementos. SVD es una técnica de reducción de dimensión para matrices que funciona muy bien con datos dispersos, es decir, datos con muchos valores a cero. Este es justo el caso en el que nos encontramos ya que debido a que muchas de las palabras no aparecen en gran cantidad de documentos de nuestro corpus, al hacer *TFIDF* siendo IDF 0, se obtienen gran cantidad.

Al hacer una comparativa entre estas dos aproximaciones la simplificación SVD ha resultado ganadora, sin embargo, como veremos más adelante en el modelo Naïve Bayes, no siempre es posible aplicarla. Además cabe aclarar que seguramente sea mejor porque no somos expertos en formación de glosarios, y que en un caso real se intuye que daría peores resultados.

Capítulo 4

Clasificadores

Una vez hemos obtenido el glosario, ahora un clasificador recibirá la entrada de o bien la matriz TF-IDF asociado al vocabulario del glosario para cada documento, o bien la matriz TF-IDF de todo el vocabulario con una simplificación SVD a cien elementos tal y como expusimos en el apartado anterior.

Respecto a los clasificadores, hemos realizado múltiples aproximaciones dos de ellas basadas en SVM o máquinas de vector soporte con glosario y con SVD. Otra mediante Naïve Bayes también con el glosario. Además incluimos otra aproximación con random forest y otra mediante redes neuronales.

En cada sección encontramos cada una de las aproximaciones y porqué se ha querido probar dicho clasificador.

4.1. Máquinas de vector soporte (SVM)

Tal y como se explica en [2], las máquinas de vectores soporte tienen su origen en los trabajos sobre la teoría del aprendizaje estadístico y fueron introducidas en los años 90 en los artículos [3, 4]. Aunque originariamente las SVMs fueron pensadas para resolver problemas de clasificación binaria, actualmente se utilizan para resolver otros tipos de problemas como regresión, agrupamiento o multicasificación. Dentro de la tarea de clasificación, las SVMs pertenecen a la categoría de los clasificadores lineales, puesto que inducen separadores lineales, también llamados hiperplanos, ya sea en el espacio original de los ejemplos de entrada, si éstos son linealmente separables o cuasiseparables (ruido), o en un espacio transformado (espacio de características) si los ejemplos no son linealmente separables en el espacio original. La búsqueda del hiperplano de separación en estos espacios transformados, normalmente de muy alta dimensión, se hará de forma implícita utilizando las denominadas funciones kernel.

Mientras la mayoría de los métodos de aprendizaje se centran en minimizar los errores cometidos por el modelo generado a partir de los ejemplos de entrenamiento (error empírico), el sesgo inductivo asociado a las SVMs radica en la minimización del denominado riesgo estructural. La idea es seleccionar un hiperplano de separación que equidista de los ejemplos más cercanos de cada clase para, de esta forma, conseguir lo que se denomina un margen máximo a cada lado del hiperplano. Además, a la hora de definir el hiperplano, sólo se consideran los ejemplos de entrenamiento de cada clase que caen justo en la frontera de dichos márgenes. Estos ejemplos reciben el nombre de vectores soporte. Desde un punto de vista práctico, el hiperplano separador de margen máximo ha demostrado tener una buena capacidad de generalización, evitando en gran medida el problema del sobreajuste a los ejemplos de entrenamiento. En Figura 4.1 se puede ver un ejemplo de como SVM funciona para la clasificación binaria, básicamente permite separar lo máximo posible dos clases con el objetivo de generalizar de manera óptima.

En nuestro caso, tal y como hemos mostrado en 4.1 utilizaremos un kernel lineal, es decir, el clasificador se encargará de introducir líneas de separación entre clases con el objetivo de que haya la máxima distancia posible entre ellas. Además utilizaremos la aproximación one vs one, es decir, el clasificador separará todas las clases dos a dos mediante una línea y después la intersección de dichas líneas definirá la región de cada una de las clases. En Figura 4.2 podemos ver un ejemplo de como funciona la aproximación one vs one.

4.1.1. Clasificador SVM con glosario supervisado

El entrenamiento del SVM se hará sobre la puntuación TFIDF dada a las palabras que extrajimos de nuestro glosario. Por tanto recibirá una matriz de 135 documentos (45 de cada clase) junto con las



(a) Posibles hiperplanos que logran separar las dos clases. (b) Hiperplano de separación óptimo.

Figura 4.1: Margen de un hiperplano de separación.

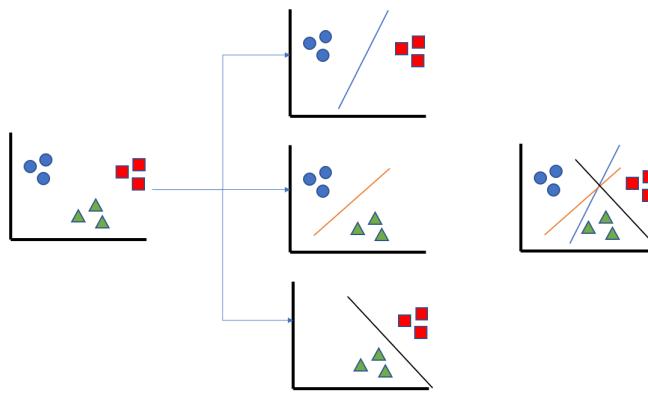


Figura 4.2: Aproximación One vs One.

212 palabras pertenecientes al glosario para cada documento con el TFIDF asociado. A partir de estos datos tendrá que trazar líneas rectas por el método one vs one descrito.

Antes del entrenamiento del modelo, primeramente se llevo un validación cruzada k-Fold. Este método tiene como objetivo tener una estimación más realista de las métricas o bondad del modelo. Cuando contamos con pocos datos (como es el caso), es muy conveniente llevar a cabo una validación cruzada. La validación cruzada k-fold consiste en dividir el conjunto de datos en k divisiones no superpuestas. De esta manera se crean K modelos, cada uno de ellos con diferentes datos de entrenamiento y de prueba y se obtiene el resultado medio de estos modelos. Una vez hechas varias pruebas sobre el glosario, se obtuvo una precisión del 91 %, por lo que se decidió entrenar el modelo y ver que tal se comportaba sobre los datos de test. Si se quiere entender con más detalle el modelo llevado a cabo, recomendamos visitar el notebook [Modelo_SVM.ipynb](#).

4.1.2. Clasificador SVM con SVD

Esta aproximación es muy semejante a la anterior, sin embargo, ahora no buscaremos obtener un glosario de las palabras más relevantes, sino que, dejaremos esta responsabilidad en manos del método SVD. SVD o Single Value Decomposition es una técnica de reducción de dimensión para matrices. Además este método funciona mejor con datos dispersos, es decir, datos con muchos valores a cero. Este es justo el caso en el que nos encontramos, ya que muchas de las palabras que conforman el vocabulario de la matriz TFIDF no aparecen en gran cantidad de documentos de nuestro corpus, por lo que al hacer $TF \cdot IDF$ siendo $IDF = 0$, se obtienen gran cantidad.

Este método permite introducir un valor para que en caso de que su simplificación no consiga reducir en más que el tamaño establecido, se queda en dicho límite. Por tanto, ahora las características a partir de las cuales nuestro mismo modelo SVM realiza predicciones, es el vocabulario total de los documentos de entrenamiento aplicando a los mismos el método SVD limitado a 100 elementos. En esta aproximación también hemos aplicado una validación cruzada k-fold, dándonos como resultado una precisión de 93,8 % siendo mejor que la aproximación anterior. Nuevamente, si se quiere obtener más información sobre esta aproximación, recomendamos visitar el notebook [Modelo_SVM_SVD.ipynb](#).

4.2. Naïve bayes con glosario

En un sentido amplio, los modelos de naïve Bayes son una clase especial de algoritmos de clasificación de Aprendizaje Automático. Estos modelos son llamados algoritmos “naïve”, o “Inocentes” en español. En ellos se asume que las variables predictoras son independientes entre sí. En otras palabras, que la presencia de una cierta característica en un conjunto de datos no está en absoluto relacionada con la presencia de cualquier otra característica. Proporcionan una manera fácil de construir modelos con un comportamiento muy bueno debido a su simplicidad. Esto es debido a la forma de calcular la probabilidad ‘posterior’ de que ocurra un cierto evento A, dadas algunas probabilidades de eventos ‘anteriores’.

Entre las ventajas de este modelo destacan:

1. Ofrece una manera fácil y rápida de predecir clases, para problemas de clasificación binarios y multiclase.
2. En los casos en que sea apropiada una presunción de independencia, el algoritmo se comporta mejor que otros modelos de clasificación, incluso con menos datos de entrenamiento.
3. El desacoplamiento de las distribuciones de características condicionales de clase significan que cada distribución puede ser estimada independientemente como si tuviera una sola dimensión. Esto ayuda con problemas derivados de la dimensionalidad y mejora el rendimiento.

Sin embargo, también tiene puntos débiles como:

1. La presunción de independencia naïve muy probablemente no reflejará cómo son los datos en el mundo real.
2. Cuando el conjunto de datos de prueba tiene una característica que no ha sido observada en el conjunto de entrenamiento, el modelo le asignará una probabilidad de cero y será inútil realizar predicciones.

En nuestro caso, pese a que los modelos entrenados con una aproximación TFIDF y SVD nos ofrecen mejores resultados, el modelo naïve Bayes al no interpretar datos negativos nos deja con dos posibles opciones. La primera de ellas consiste en aplicar SVD y después una normalización de datos como podría ser Min-Max scaler para así transformar los datos en positivos sin perder su distribución. La segunda consiste en utilizar la matriz TFIDF con las palabras reflejadas en el glosario. Tras investigar esta problemática se ha leído que un normalizado de datos tras la aplicación del método SVD reduce los algoritmos de aprendizaje automático hasta en un 20%, por lo que hemos decidido optar por la segunda aproximación. Al realizar la validación del modelo nos ofreció una precisión del 91,7%.

Si se quiere ver con más detalle este modelo, recomendamos mirar el notebook [Modelo_NB.ipynb](#).

4.3. Random forest TFIDF-SVD

Antes de empezar a hablar de los bosques aleatorios, debemos entender el método de ensamblaje (Bagging). El ensamblaje es un método simple y muy potente. Es un procedimiento general que puede utilizarse para reducir la varianza de nuestro modelo. Esto es relevante en el sentido de que una mayor varianza significa el sobreajuste del modelo. Algunos algoritmos, como los árboles de decisión, suelen tener una varianza elevada. Por otra parte, los árboles de decisión son extremadamente sensibles a los datos sobre los que han sido entrenados. Si los datos subyacentes cambian aunque sea un poco, el árbol de decisión resultante puede ser muy diferente, y, como resultado, las predicciones de nuestro modelo cambiarán drásticamente. El ensamblado ofrece una solución al problema de la alta varianza ya que permite reducir sistemáticamente el sobreajuste tomando una media de varios árboles de decisión. El ensamblaje utiliza el muestreo bootstrap y, finalmente, agrega los modelos individuales mediante un promedio para obtener las predicciones finales.

El muestreo bootstrap consiste en muestrear filas al azar del conjunto de datos de entrenamiento con reemplazo. Por tanto, con el método de ensamblaje es posible que se extraiga un mismo ejemplo de entrenamiento más de una vez, lo que da lugar a una versión modificada del conjunto de entrenamiento en la que algunas filas se representan varias veces y otras están ausentes. Esto permite crear nuevos datos, que son similares a los datos con los que se comenzó, de este modo, puede ajustar muchos modelos diferentes pero similares.

El modelo random forest utiliza un algoritmo de aprendizaje de árboles modificado que inspecciona, en cada división del proceso de aprendizaje, un subconjunto aleatorio de las características. Esto se hace con el propósito de evitar la correlación entre los árboles. Supongamos que tenemos un predictor muy fuerte en el conjunto de datos junto con otros predictores moderadamente fuertes, entonces en la colección de árboles ensamblados, la mayoría o todos los árboles de decisión utilizarán el predictor más fuerte para la primera división, lo que resultará en que los árboles ensamblados sean similares. Por lo tanto, todas las predicciones de los árboles ensamblados estarán altamente correlacionadas, lo que no ayudan a mejorar la precisión de la predicción. Al tomar un subconjunto aleatorio de características, Random Forests evita sistemáticamente la correlación y mejora el rendimiento del modelo.

El modelo random forest es uno de los algoritmos de aprendizaje por conjuntos más utilizados. La razón es que al utilizar múltiples muestras del conjunto de datos original, reducimos la varianza del modelo final. El sobreajuste se produce cuando nuestro modelo intenta explicar pequeñas variaciones en el conjunto de datos porque nuestro conjunto de datos es sólo una pequeña muestra de la población de todos los ejemplos posibles del fenómeno que intentamos modelar. Si no hemos tenido suerte con el muestreo de nuestro conjunto de entrenamiento, podría contener algunos artefactos indeseables (pero inevitables): ruido, valores atípicos y ejemplos sobrerepresentados o infrarrepresentados. Al crear múltiples muestras aleatorias con reemplazo de nuestro conjunto de entrenamiento, reducimos el efecto de estos artefactos.

Por tanto, este algoritmo se empleará con el objetivo de compararlo con el que mejor resultado nos había ofrecido hasta ese momento (SVM con SVD) recibirá la matriz TFIDF con la simplificación SVD. Al someter este modelo al mismo proceso de validación, ofreció una precisión media del 95,1 %, la mejor hasta ahora. Se puede ver el proceso y más detalles de la implementación en el notebook [Modelo_RF.ipynb](#).

4.4. Redes neuronales con SVD

Una red neuronal profunda es una red neuronal artificial con varias capas ocultas entre las capas de entrada y salida. Las redes neuronales artificiales deben su nombre a que tratan de imitar el funcionamiento de las redes neuronales biológicas que constituyen los cerebros biológicos. Estas redes se basan en un conjunto de unidades interconectadas denominadas neuronas. Cada conexión entre estas neuronas, permite transmitir una señal a otras neuronas, cuando esto ocurre, la neurona que recibe la señal procesa dicha información y puede enviarla a una nueva (o a varias). De esta manera, ante un conjunto de datos de entrada en la red neuronal, producirá un conjunto de datos de salida, modelando funciones no lineales complejas. En Figura 4.3 se muestra un ejemplo.

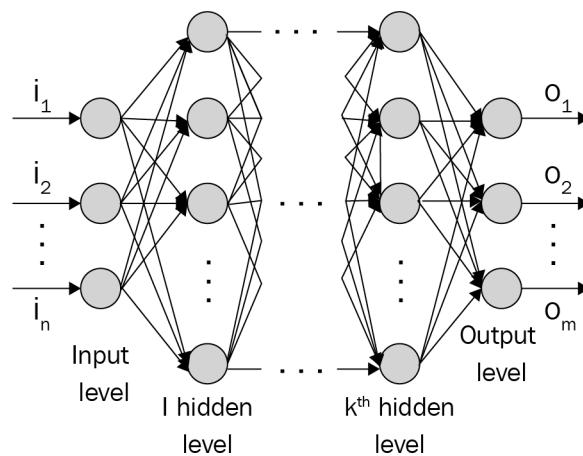


Figura 4.3: Arquitectura de una red neuronal profunda.

Las redes neuronales artificiales son capaces de aprender de la experiencia y generalizarla, realizando modelizaciones no lineales sin un conocimiento previo sobre las relaciones entre las variables de entrada y de salida. De esta forma, las redes neuronales aproximan de una manera más general y flexible que los métodos estadísticos tradicionales.

Las redes neuronales, debido a que el proceso de aprendizaje se basa en ejemplos, es aplicable a todos aquellos campos de los que se disponga de un gran conjunto de ejemplos de gran calidad, es decir, que sean representativos. A continuación se exponen sus principales campos de aplicación en la actualidad:

1. **Procesamiento de lenguaje natural:** En este campo podemos destacar el traductor DeepL [5], el cual detecta el idioma del texto de manera automática y traduce de una manera muy eficaz según el contexto. En Figura 4.4 podemos ver un ejemplo de traducción automática de este software.
2. **Asistencia sanitaria:** Las redes neuronales profundas permiten ayudar al diagnóstico temprano, preciso y rápido de enfermedades potencialmente mortales. Por ejemplo, permite detectar células cancerígenas a través de imágenes [7].
3. **Detección de fraude:** Se ha demostrado que las redes neuronales también pueden detectar el fraude en transacciones con tarjetas de crédito [8]. Estas redes tratan de conocer el comportamiento "normal" del propietario de la tarjeta, y en caso de encontrar alguna anomalía puede congelar la tarjeta para que no se lleven a cabo más transacciones hasta que el propietario determine si se trata de un fraude o no.
4. **Predicciones:** Podemos ver múltiples ejemplos de predicciones en todo tipo de campos, por ejemplo, para estimar el precio de una casa [9] según su ubicación, tamaño, y otro tipo de atributos.

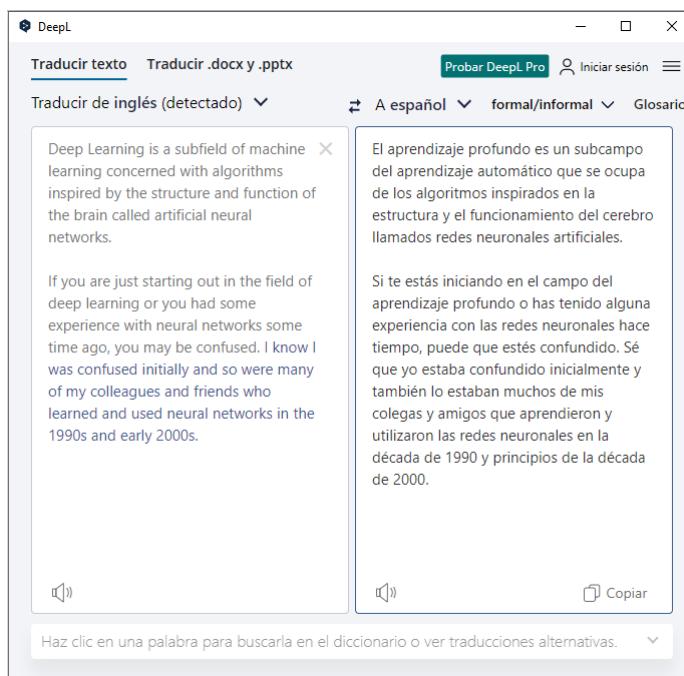


Figura 4.4: Ejemplo de traducción automática de DeepL.

Una vez vista la importancia de estos métodos de computación y sus principales aplicaciones, veamos los diferentes componentes de los que disponemos para su creación.

El primer componente que explicaremos y que nos ayudará a entender los siguientes serán las funciones de activación y sus principales tipos. Cada modelo tiene un conjunto de capas, cada capa tiene una función de activación que se aplica sobre todas sus neuronas. Para comprender esto de manera más clara, podemos ver en Figura 4.5 que para la primera capa se utiliza la función de activación uno, por lo que sus dos neuronas usarán dicha función de activación. En la capa oculta se utiliza la función de activación dos por lo que sus cuatro neuronas utilizarán dicha función de activación. Por último, la capa de salida utiliza la función de activación tres, por lo que sus dos neuronas utilizarán esta.

En Figura 4.6 podemos ver las funciones de activación utilizadas para nuestra red. Utilizaremos ReLU sobre todas las capas ocultas de la red y softmax sobre la capa de salida, ya que queremos que el modelo nos de una probabilidad (rango [0-1]) asociada a cada una de las clases. Además añadiremos inicialización de pesos he uniforme y GlorotNormal sobre las capas con activación ReLU y Softmax respectivamente ya que normalmente se adaptan mejor al problema con dicha inicialización.

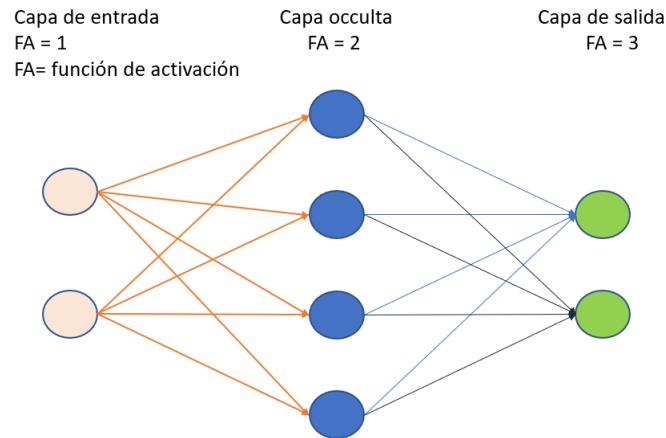


Figura 4.5: Ejemplo de como se pueden distribuir las funciones de activación.

Función de activación	Ecuación	Gráfico
ReLU (Rectified Linear Unit)	$S(X_i) = \max(0, X_i)$	
Softmax	$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$	

Figura 4.6: Funciones de activación utilizadas en el modelo.

Las redes neuronales como hemos podido ver en Figura 4.3 constan de tres capas necesarias:

1. **Capa de entrada:** Esta es la capa que debe permitir introducir los datos a la red, por ello deberá tener el mismo formato que los datos a introducir en la red.
2. **Capa/s oculta/s:** En esta parte de la red, podremos introducir un número indeterminado de capas con un número indeterminado de neuronas en cada una de ellas.
3. **Capa de salida:** Esta capa, al igual que la de entrada, debe tener un tamaño específico, que corresponderá con el número de variables que queremos predecir.

A continuación explicaremos cada una de las capas utilizadas en nuestro modelo:

1. **Capa Dense:** Esta capa se utiliza cuando pueden existir asociaciones entre diferentes variables introducidas en el modelo. Dada su utilidad, el número de neuronas de la capa anterior (C_{n-1}) estará totalmente interconectada con las neuronas de esta capa (C_n), ya que pueden existir $n1*n2$ relaciones entre ellas.
2. **Dropout:** Este capa permite eliminar una serie de neuronas de una determinada capa determinado por un porcentaje que se pasa como parámetro para la construcción de la misma. Esto permite que, al no contar con esa neurona la capa, se distribuya el peso entre el resto de neuronas que la conforma, de esta manera permite que los pesos de las diferentes neuronas se distribuya mejor y no exista una con un valor muy elevado o bajo. Esta capa se utiliza para evitar el sobreentrenamiento¹ de los modelos.

Por tanto, utilizaremos dos tipos de capas, una de ellas destinadas a evitar el sobreentrenamiento (Dropout). Esto es debido a que se cuentan con muy pocos datos, por lo que es importante que intentemos generalizar lo máximo posible el conocimiento que se puede extraer de estos.

Respecto a la función de pérdida, utilizaremos la función categorical_crossentropy ya que buscamos minimizar el número de errores cometidos por el modelo a la hora de clasificar los documentos.

Finalmente, utilizaremos las mismas características que el modelo SVM que nos dio mejor precisión al realizar la validación cruzada, es decir, una matriz TFIDF sobre los documentos de entrenamiento junto con el método SVD. La salida del modelo por su parte será un one-hot-encode es decir, un grupo de bits con un solo bit a 1 y los demás a 0. Esto es debido a que los modelos basados en NN funcionan mejor con los rangos [0-1]. En este caso, no se ha llevado una validación cruzada a cabo debido a que el proceso de entrenamiento requiere mucha mayor capacidad que el resto de algoritmos. Nuevamente incluimos un notebook asociado el modelo, que se puede ver en [Modelo_NN.ipynb](#)

¹Sobreentrenamiento: Capacidad que tiene un modelo de predecir con exactitud un conjunto particular de datos, siendo poco preciso en valores no conocidos por el mismo.

Capítulo 5

Evaluación de los clasificadores

Una vez que hemos determinado que modelos vamos a utilizar para clasificar los documentos, es importante hacer una evaluación de los mismos, es decir, cuantificar su bondad. Para ello es posible utilizar diferentes métricas. A continuación mostramos cuales hemos utilizado en nuestro caso:

1. **Precisión:** La precisión es intuitivamente la capacidad del clasificador de no etiquetar como positiva una muestra que es negativa. El mejor valor es 1 y el peor es 0. Su fórmula es:

$$Precision = \frac{tp}{tp + fp} \quad (5.1)$$

Donde tp es el número de verdaderos positivos y fp el número de falsos positivos

2. **Recall:** Recall es, intuitivamente, la capacidad del clasificador para encontrar todas las muestras positivas. El mejor valor es 1 y el peor es 0. Su formula viene dada por:

$$Recall = \frac{tp}{tp + fn} \quad (5.2)$$

Donde fn es el número de falsos negativos.

3. **F1-score:** La puntuación F1 puede interpretarse como una media armónica de la precisión y recall, donde la puntuación F1 alcanza su mejor valor en 1 y la peor puntuación en 0. La contribución relativa de la precisión y la recuperación a la puntuación F1 son iguales. La fórmula de la puntuación F1 es:

$$F1 = 2 \cdot \frac{(precision \cdot recall)}{(precision + recall)} \quad (5.3)$$

4. **Support:** Número de ocurrencias de cada clase que se quiere predecir. Si por ejemplo entre los datos que disponemos tenemos 10 documentos que pertenecen a la clase Deportes, el support de Deportes será 10.

5. **Matriz de confusión:** Por definición, una matriz de confusión C es aquella cuyo $C_{i,j}$ es igual al número de observaciones que se sabe que están en el grupo i y que se predice que están en el grupo j .

Así, en la clasificación binaria, el número de verdaderos negativos es $C_{0,0}$, el de falsos negativos es $C_{1,0}$, el de verdaderos positivos es $C_{1,1}$ y el de falsos positivos es $C_{0,1}$. Esto es aplicable a N clases, en nuestro caso 3. Al final es una matriz que permite expresar el número de verdaderos positivos de cada clase, frente a la predicción que se ha hecho sobre ese determinado dato (documento).

Hemos elegido estas métricas debido a que son las más ampliamente extendidas y además permiten evaluar de una manera clara el comportamiento de los clasificadores.

A continuación se ofrece un informe detallado de los resultados obtenidos para cada uno de los clasificadores desarrollados al tratar de clasificar el 25 % de los documentos de los cuales no se han extraído ningún tipo de información de manera directa.

5.1. Evaluación del clasificador basado en SVM con glosario

Primeramente se ofrece en Figura 5 la probabilidad con la que se predicen cada una de las clases. En el apéndice en Figura 6 se ofrece la probabilidad asignada a cada uno de los documentos sobre los que se está evaluando el modelo. Pese a que algunos documentos son clasificados erróneamente, ofrece casi una probabilidad del 100 % para la clasificación, es decir, está muy seguro de que pertenece a dicha clase (erróneamente). Esto puede ser debido a que durante el entrenamiento se ha asociado de manera equivocada el vocabulario asociado a una determinada clase para otra, por lo que al trazar las líneas de separación entre clases, dos clases que pareciese que están muy separadas, resultan no estarlo. En Figura 5.1 se ofrecen las métricas Precisión, Recall, F1 y Support. Finalmente en 5.2 podemos ver la matriz de confusión asociada.

SVM-Glosario	Precision	Recall	F1-score	Support
Salud	1,000	1,000	1,000	15,000
Política	1,000	0,867	0,929	15,000
Deportes	0,882	1,000	0,938	15,000
Macro avg	0,961	0,956	0,955	45,000
Weighted avg	0,961	0,956	0,955	45,000

Figura 5.1: Informe de métricas del modelo SVM con glosario.

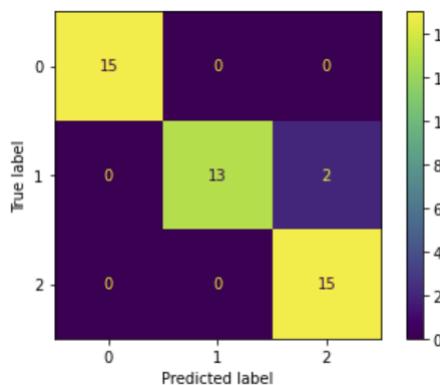


Figura 5.2: Matriz de confusión del modelo SVM con glosario.

La matriz de confusión nos muestra que:

1. De 15 documentos pertenecientes a la temática salud, 15 han sido correctamente clasificados.
2. De 15 documentos pertenecientes a la temática política, 13 han sido correctamente clasificados.
3. De 15 documentos pertenecientes a la temática deportes, 15 han sido correctamente clasificados.

5.2. Evaluación del clasificador basado en SVM con SVD

Este clasificador ha resultado ser el mejor a nivel clasificar documentos, ya que clasifica el total de documentos adecuadamente. Esto puede ser debido a que SVD ha conseguido representar mejor las características para el modelo que mediante el glosario supervisado a mano. Algo curioso a destacar es que pese a que clasifica mejor que el modelo SVM con glosario manual, no aporta tanta probabilidad a la clase en la que se acaba clasificando. Se puede ver en Figura 6. Esto a nivel conceptual creemos que se debe a que SVD representa mejor el espacio entre clases, lo que demuestra que no existe tanto espacio entre las mismas como se creía con el glosario manual. En Figura 5.1 se ofrecen las métricas Precisión, Recall, F1 y Support. Finalmente en 5.2 podemos ver la matriz de confusión asociada.

SVM + SVD	Precision	Recall	F1-score	Support
Salud	1	1	1	15
Política	1	1	1	15
Deportes	1	1	1	15
Macro avg	1	1	1	45
Weighted avg	1	1	1	45

Figura 5.3: Informe de métricas del modelo SVM con SVD.

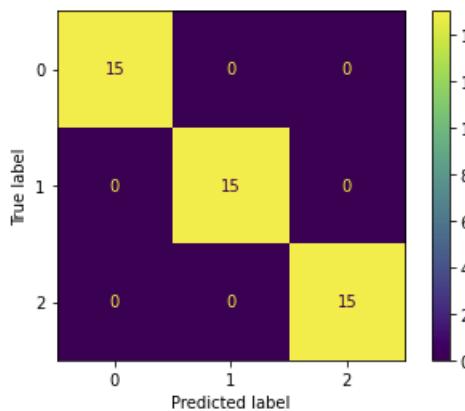


Figura 5.4: Matriz de confusión del modelo SVM con SVD.

La matriz de confusión nos muestra que:

1. De 15 documentos pertenecientes a la temática salud, 15 han sido correctamente clasificados.
2. De 15 documentos pertenecientes a la temática política, 15 han sido correctamente clasificados.
3. De 15 documentos pertenecientes a la temática deportes, 15 han sido correctamente clasificados.

5.3. Evaluación del clasificador basado en redes neuronales

Este clasificador ha logrado obtener un gran desempeño, ya que solo clasificó mal uno de los documentos. Además a nivel probabilidad asociada a cada clase, aporta el 100 % a una determinada clase, aunque sea erróneo. En Figura 5.5 se ofrecen las métricas Precisión, Recall, F1 y Support. Finalmente en 5.6 podemos ver la matriz de confusión asociada.

Red neuronal + SVD	Precision	Recall	F1-score	Support
Salud	1,000	1,000	1,000	15,000
Política	1,000	0,933	0,966	15,000
Deportes	0,938	1,000	0,968	15,000
Macro avg	0,979	0,978	0,978	45,000
Weighted avg	0,979	0,978	0,978	45,000

Figura 5.5: Informe de métricas del modelo NN.

La matriz de confusión nos muestra que:

1. De 15 documentos pertenecientes a la temática salud, 15 han sido correctamente clasificados.
2. De 15 documentos pertenecientes a la temática política, 14 han sido correctamente clasificados.
3. De 15 documentos pertenecientes a la temática deportes, 15 han sido correctamente clasificados.

5.4. Evaluación del clasificador naïve Bayes

Los resultados de este clasificador resultan bastante curiosos. Ha logrado clasificar bien todos los documentos, sin embargo la probabilidad asociada a ellos no dista apenas tal y como se puede ver en

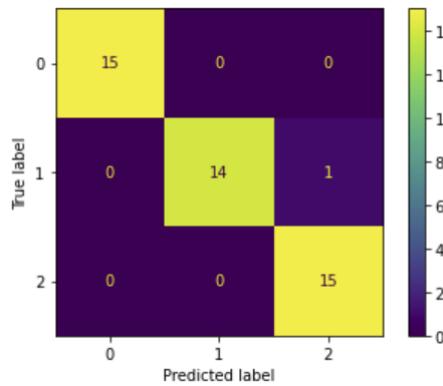


Figura 5.6: Matriz de confusión del modelo NN.

Figura 8. Por lo que si se tuviese que elegir entre SVM con SVD o este, se elegiría SVM con SVD pese a que ambos clasifiquen todos los documentos correctamente. En Figura 5.7 se ofrecen las métricas Precisión, Recall, F1 y Support. Finalmente en 5.6 podemos ver la matriz de confusión asociada.

NB-Glosario	precision	recall	f1-score	support
Salud	1	1	1	15
Política	1	1	1	15
Deportes	1	1	1	15
macro avg	1	1	1	45
weighted avg	1	1	1	45

Figura 5.7: Informe de métricas del modelo naïve Bayes.

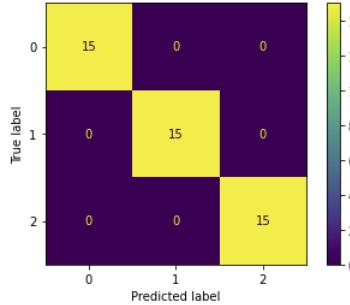


Figura 5.8: Matriz de confusión del modelo naïve Bayes.

La matriz de confusión nos muestra que:

1. De 15 documentos pertenecientes a la temática salud, 15 han sido correctamente clasificados.
2. De 15 documentos pertenecientes a la temática política, 15 han sido correctamente clasificados.
3. De 15 documentos pertenecientes a la temática deportes, 15 han sido correctamente clasificados.

5.5. Evaluación del clasificador Random Forest

Este clasificador ofreció el mejor rendimiento en la fase de validación cruzada. Sin embargo, su desempeño final sobre el conjunto de test es equitativo a SVM con glosario manual. Además la probabilidad asignada a las clases no siempre dista claramente, siendo incluso en un caso concreto un 1 % de diferencia entre clases. En Figura 5.9 se ofrecen las métricas Precisión, Recall, F1 y Support. Finalmente en 5.10 podemos ver la matriz de confusión asociada.

La matriz de confusión nos muestra que:

1. De 15 documentos pertenecientes a la temática salud, 15 han sido correctamente clasificados.

NB-Glosario	precision	recall	f1-score	support
Salud	1	1	1	15
Política	1	1	1	15
Deportes	1	1	1	15
macro avg	1	1	1	45
weighted avg	1	1	1	45

Figura 5.9: Informe de métricas del modelo Random Forest.

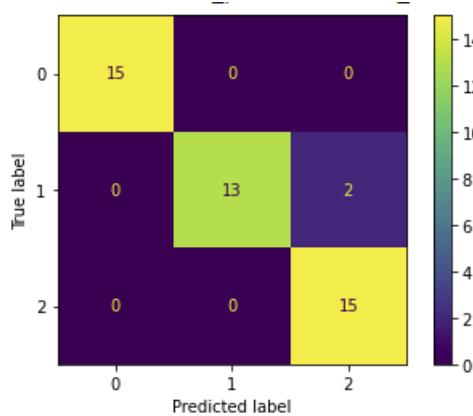


Figura 5.10: Matriz de confusión del modelo Random Forest.

2. De 15 documentos pertenecientes a la temática política, 13 han sido correctamente clasificados.
3. De 15 documentos pertenecientes a la temática deportes, 15 han sido correctamente clasificados.

5.6. Comparativa

Dado que hemos llevado a cabo 5 aproximaciones y puede resultar un poco tedioso el comparar una a una, mostramos en Figura 5.11 una comparativa entre todas ellas. Como podemos ver, las aproximaciones que mejores resultados han dado han sido Naïve Bayes con glosario y SVM junto con SVD. Sin embargo, todas ellas ofrecen métricas muy buenas.

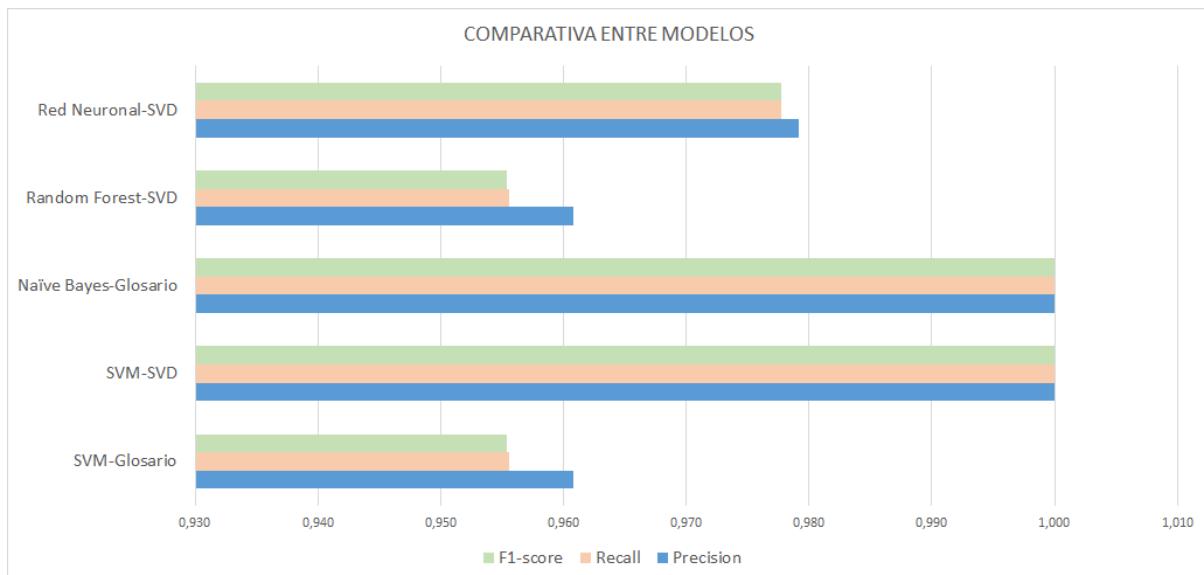


Figura 5.11: Comparativa de las puntuaciones medias de los modelos para las métricas seleccionadas.

Finalmente también se expone la fase de validación, aplicada a todos los clasificadores menos al

basado en redes neuronales, ya que requiere de mayor carga computacional. Esta comparativa se puede ver en Figura 5.12

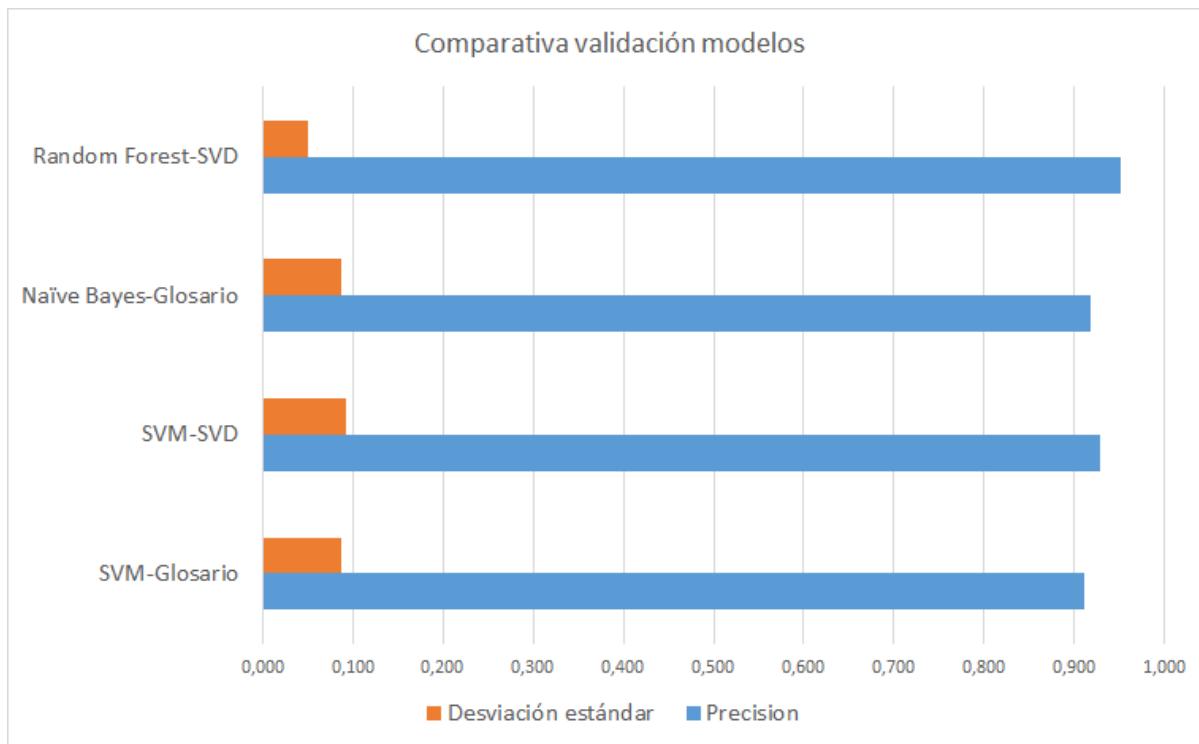


Figura 5.12: Comparativa de las puntuaciones de la precisión y desviación típica de las validaciones cruzadas aplicadas a los clasificadores.

Capítulo 6

Conclusiones

Primeramente queríamos incluir un resumen visual de lo hecho. Para ello hemos creado un esquema que se puede ver en 6.1.

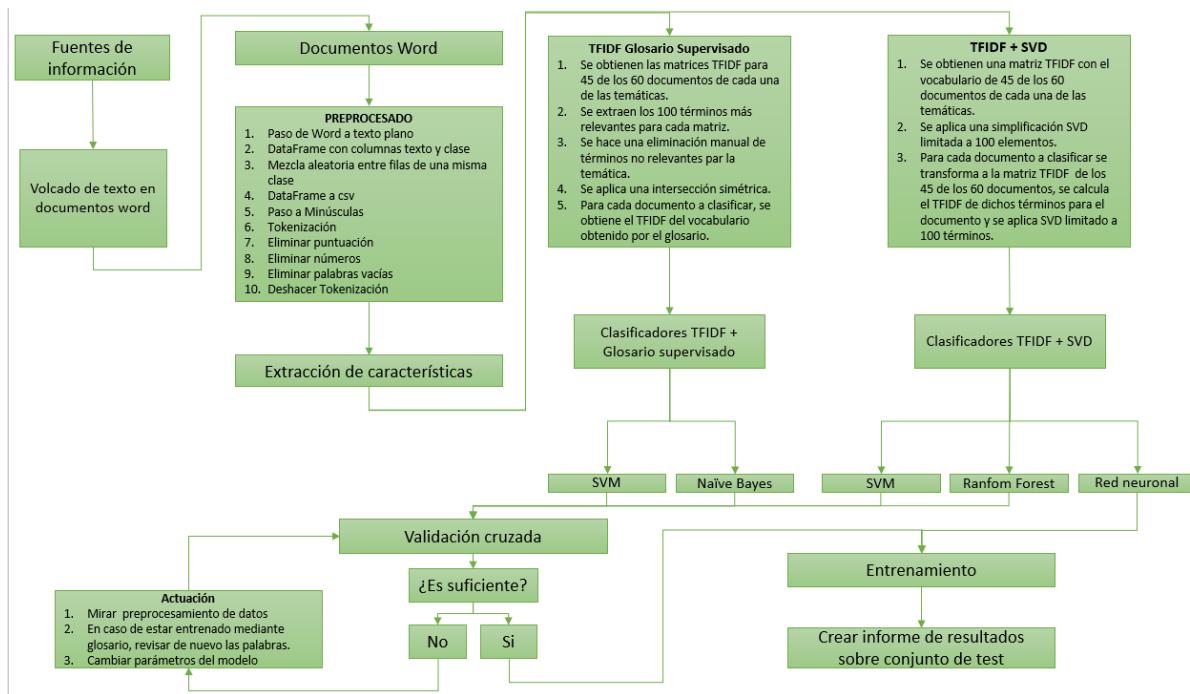


Figura 6.1: Resumen de los procedimientos llevados a cabo para la realización de la práctica.

Por último nos gustaría exponer las conclusiones más relevantes que hemos extraído durante el desarrollo de la práctica.

1. **Extracción de términos:** Hemos visto una manera sencilla y muy útil de extraer términos de textos mediante la matriz TFIDF. También la importancia de SVD, el cual permite establecer una primera aproximación cuando no se cuenta con el glosario o si no es un experto en la materia.
2. **Preprocesado de textos:** El tratamiento de los datos anterior a la extracción de términos ofrece muchas posibilidades. Por ejemplo, podríamos eliminar textos en negrita rodeados por la expresión regular sin embargo, esto también se podría interpretar como que el texto rodeado entre la misma es más relevante. También ocurre lo mismo con tamaños de letras u otras características propias del texto digital.
3. **SVM:** Hemos visto que los clasificadores SVM juegan un papel fundamental en el objetivo de la práctica, es decir, clasificar documentos, obteniendo los mejores resultados mediante este enfoque. Además no requieren de mucho esfuerzo computacional.

4. **NN:** Hemos visto también que las redes neuronales son una posible aproximación al problema, y que, con una cantidad más elevada de datos quizás se obtuviese mayor rendimiento que los enfoques basados en SVM.
5. **Naïve Bayes:** El clasificador Naïve Bayes es una primera aproximación al problema y puede servir para saber si el glosario ha sido creado correctamente, sin embargo, creemos que no es buen modelo debido a que existe poca separación entre clases (dada por la probabilidad asociada a cada una de ellas).
6. **Random Forest:** Es el clasificador que parece más robusto, ya que es el que mejor precisión ha dado en la validación cruzada. Pese a que consideramos que es mejor aproximación que Naïve Bayes, creemos que tampoco deja la suficiente separación entre clases. Quizás una posible solución era aumentar la profundidad del mismo.
7. **Importancia del glosario:** Durante el desarrollo de la práctica hemos comprobado como se veía afectado directamente el modelo SVM mediante cambios en la selección de palabras del glosario de manera supervisada. Se ha comprobado que el eliminar las palabras que aparecían en varias temáticas mediante la diferencia simétrica de conjuntos ayudaba a ofrecer mayor precisión a la hora de clasificar.
8. **Creación del espacio entre clases:** Se ha obtenido una intuición de como los clasificadores a partir del entrenamiento tratan de crear un espacio de clasificación para predecir a que clase pertenecen los diferentes documentos.
9. **Métricas:** Se ha aprendido como es posible medir la bondad de clasificadores de una manera sencilla, comprensible y gráfica.

Bibliografía

- [1] S. Raschka, “Activation functions for artificial neural networks,” http://rasbt.github.io/mlxtend/user_guide/general_concepts/activation-functions/.
- [2] E. J. C. Suárez, “Tutorial sobre máquinas de vectores soporte (svm),” *Tutorial sobre Máquinas de Vectores Soporte (SVM)*, pp. 1–12, 2014.
- [3] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A training algorithm for optimal margin classifiers,” in *Proceedings of the fifth annual workshop on Computational learning theory*, 1992, pp. 144–152.
- [4] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [5] L. GmbH, “Traductor automático deepl,” <https://www.deepl.com/es/translator>.
- [6] L. Floridi and M. Chiriatti, “Gpt-3: Its nature, scope, limits, and consequences,” *Minds and Machines*, vol. 30, no. 4, pp. 681–694, 2020.
- [7] Z. Hu, J. Tang, Z. Wang, K. Zhang, L. Zhang, and Q. Sun, “Deep learning for image-based cancer detection and diagnosis- a survey,” *Pattern Recognition*, vol. 83, pp. 134–149, 2018.
- [8] A. Roy, J. Sun, R. Mahoney, L. Alonzi, S. Adams, and P. Beling, “Deep learning detecting fraud in credit card transactions,” in *2018 Systems and Information Engineering Design Symposium (SIEDS)*. IEEE, 2018, pp. 129–134.
- [9] P.-Y. Wang, C.-T. Chen, J.-W. Su, T.-Y. Wang, and S.-H. Huang, “Deep learning model for house price prediction using heterogeneous data analysis along with joint self-attention mechanism,” *IEEE Access*, vol. 9, pp. 55 244–55 259, 2021.

Apéndice

Recomendamos el abrir los notebook para visualizarlos correctamente, pero en caso de no disponer de tiempo y querer echarle un vistazo rápido se han incluido en este apéndice.

.1. Figuras

SALUD	POLÍTICA	DEPORTES
salud	años	madrid
vida	política	mundial
si	pp	equipo
puede	gobierno	español
pacientes	tribunal	partido
personas	podemos	real
ser	presidente	laliga
tratamiento	sánchez	años
pueden	españa	jugador
gripe	efe	selección
cuerpo	país	tres
ms	casado	bale
mundo	foto	dos
cada	partido	campeón
piel	si	si
hospital	generalitat	sido
importante	madrid	hazard
tabaco	psoe	ser
mejor	año	repsca
ejercicio	así	temporada
riesgo	jóvenes	tras
tipo	dos	gran
dosis	congreso	club
asegura	político	después
forma	cuentas	español
correr	ahora	jugadores
doctora	políticos	foto
alimentos	fiscalia	márquez
enfermedades	ley	carrera
explica	portavoz	sol
años	vivienda	qatar
vacunación	hace	mundo
bien	ayuntamiento	ahora
diabetes	partidos	así
evitar	ser	estadio
arginina	tras	año
queratitis	caso	partidos
debe	redondo	primera
además	nadal	puntos
fructosa	europeo	hacé
consumo	ue	española
enfermedad	parte	mclaren
tratamientos	precios	minutos
año	líder	mejor
según	rey	primer
agua	presupuestos	final
tiempo	monarquía	tiempo
casos	delito	bien
uso	serra	título
hacer	dia	garbiñe
manera	social	cartuja
bata	cup	historia
paciente	momento	último
día	tiempo	sevilla
dolor	nacional	odegaard
zapatos	diaz	rossi
hoy	prisión	mercedes
especialista	cambio	momento
tambin	popular	gales
problemas	alcaldesa	valentino
ictus	vox	tenista
ácido	además	ancelotti
mayor	audiencia	pilotos
dr	siempre	ciudad
sueño	sido	dia
estética	supremo	siempre
células	ciudadanos	juego
comer	constitucional	jugar
familia	comunidad	martínez
medicamentos	pasado	hamilton
atención	consejo	parte
días	haití	deportiva
niños	meses	pista
tener	acuerdo	vez
doctor	sol	fútbol
servicio	sentencia	wta
aunque	millones	tan
peso	bien	balón
efectos	andalucía	noche
así	según	piloto
hace	pedro	domingo
cáncer	aunque	horner
beneficios	montero	fin
síntomas	pablo	mientras
solo	medio	segundo
lesiones	después	hoy
persona	mismo	aunque
metabolismo	euros	italia
dos	pensiones	smartbank
profesionales	diputados	deporte
protector	aprobación	remy
siempre	aragones	clasificación
calidad	europa	sánchez
incluso	agentes	menos
ejemplo	fondos	millones
menos	hecho	wolff
pandemia	presidenta	marc
mismo	peticón	galahad
muchas	instituciones	británico
lactancia	presupuesto	grupo

Figura 2: Glosario inicial con 100 términos por temática.

Deportes - 70	Política - 71	Salud - 71
mundial	política	salud
equipo	pp	vida
real	gobierno	pacientes
laliga	tribunal	personas
jugador	podemos	tratamiento
selección	presidente	gripe
bale	efe	cuerpo
campeón	país	ms
hazard	casado	piel
repesca	sí	hospital
temporada	generalitat	importante
gran	psoe	tabaco
club	así	ejercicio
español	jóvenes	riesgo
jugadores	congreso	tipo
márquez	político	dosis
carrera	cuentas	asegura
qatar	políticos	forma
estadio	fiscalía	correr
primera	ley	doctora
puntos	portavoz	alimentos
española	vivienda	enfermedades
mcclaren	ayuntamiento	explica
minutos	caso	vacunación
mejor	redondo	diabetes
primer	nadal	evitar
final	europeo	arginina
título	ue	queratitis
garbiñe	precios	debe
cartuja	líder	fructosa
historia	rey	consumo
último	presupuestos	enfermedad
sevilla	monarquía	tratamientos
odegaard	delito	agua
rossi	serra	casos
mercedes	social	uso
gales	cup	bata
valentino	nacional	paciente
tenista	díaz	dolor
ancelotti	prisión	zapatos
pilotos	cambio	especialista
ciudad	popular	problemas
juego	alcaldesa	ictus
jugar	vox	ácido
martinez	audiencia	mayor
hamilton	supremo	dr
deportiva	ciudadanos	sueño
pista	constitucional	estética
vez	comunidad	células
fútbol	consejo	comer
wta	haití	familia
balón	acuerdo	medicamentos
noche	sentencia	atención
piloto	andalucía	niños
domingo	pedro	tener
horner	montero	doctor
fin	pablo	servicio
mientras	medio	aunque
segundo	euros	peso
hoy	pensiones	efectos
italia	diputados	cáncer
smartbank	aprobación	beneficios
deporte	aragones	síntomas
remy	europa	lesiones
clasificación	agentes	persona
wolff	fondos	metabolismo
marc	hecho	profesionales
galahad	presidenta	protector
británico	petición	calidad
grupo	instituciones	pandemia
	presupuesto	lactancia

Figura 3: Glosario por temática supervisado.

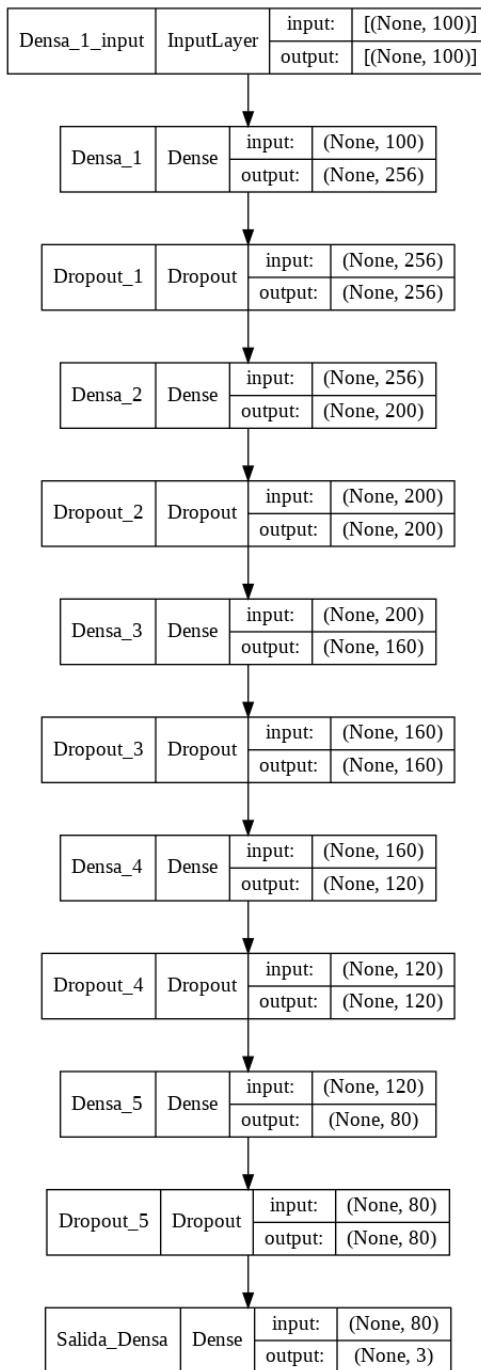


Figura 4: Arquitectura del modelo basado en redes neuronales.

SVM-Glosario	Salud	Politica	Deportes	Clase_real
0	0,883	0,021	0,096	0,000
1	0,999	0,001	0,001	0,000
2	0,993	0,004	0,004	0,000
3	0,848	0,115	0,037	0,000
4	0,727	0,151	0,121	0,000
5	0,999	0,001	0,000	0,000
6	0,915	0,045	0,040	0,000
7	0,999	0,001	0,000	0,000
8	0,994	0,004	0,002	0,000
9	0,999	0,001	0,000	0,000
10	0,995	0,003	0,001	0,000
11	0,998	0,001	0,001	0,000
12	0,926	0,059	0,015	0,000
13	0,889	0,062	0,049	0,000
14	0,997	0,002	0,001	0,000
15	0,017	0,948	0,035	1,000
16	0,000	1,000	0,000	1,000
17	0,050	0,913	0,037	1,000
18	0,110	0,637	0,253	1,000
19	0,007	0,984	0,009	1,000
20	0,254	0,404	0,342	1,000
21	0,132	0,772	0,096	1,000
22	0,009	0,979	0,013	1,000
23	0,005	0,990	0,005	1,000
24	0,022	0,963	0,015	1,000
25	0,217	0,449	0,334	1,000
26	0,004	0,989	0,007	1,000
27	0,098	0,807	0,095	1,000
28	0,008	0,985	0,007	1,000
29	0,026	0,926	0,048	1,000
30	0,000	0,000	1,000	2,000
31	0,000	0,003	0,996	2,000
32	0,003	0,006	0,991	2,000
33	0,008	0,018	0,974	2,000
34	0,001	0,006	0,992	2,000
35	0,016	0,160	0,824	2,000
36	0,004	0,010	0,986	2,000
37	0,028	0,031	0,941	2,000
38	0,001	0,003	0,997	2,000
39	0,000	0,000	1,000	2,000
40	0,058	0,076	0,866	2,000
41	0,005	0,021	0,974	2,000
42	0,002	0,006	0,992	2,000
43	0,003	0,006	0,991	2,000
44	0,000	0,000	1,000	2,000

Figura 5: Predicciones probabilísticas del modelo SVM con glosario.

SVM+SVD	Salud	Politica	Deportes	Clase_real
0	0,840	0,060	0,100	0,000
1	1,000	0,000	0,000	0,000
2	0,960	0,030	0,010	0,000
3	0,920	0,050	0,030	0,000
4	0,800	0,060	0,140	0,000
5	1,000	0,000	0,000	0,000
6	0,970	0,020	0,020	0,000
7	1,000	0,000	0,000	0,000
8	0,990	0,000	0,000	0,000
9	1,000	0,000	0,000	0,000
10	0,990	0,000	0,000	0,000
11	0,990	0,000	0,010	0,000
12	0,980	0,010	0,010	0,000
13	0,970	0,010	0,010	0,000
14	0,990	0,000	0,000	0,000
15	0,010	0,960	0,030	1,000
16	0,000	1,000	0,000	1,000
17	0,040	0,930	0,030	1,000
18	0,040	0,730	0,230	1,000
19	0,010	0,990	0,010	1,000
20	0,050	0,680	0,270	1,000
21	0,020	0,950	0,030	1,000
22	0,020	0,950	0,030	1,000
23	0,020	0,960	0,010	1,000
24	0,050	0,910	0,040	1,000
25	0,150	0,730	0,120	1,000
26	0,010	0,980	0,010	1,000
27	0,070	0,880	0,050	1,000
28	0,000	0,980	0,010	1,000
29	0,000	0,990	0,010	1,000
30	0,000	0,000	0,990	2,000
31	0,010	0,010	0,980	2,000
32	0,010	0,020	0,970	2,000
33	0,010	0,010	0,980	2,000
34	0,020	0,070	0,920	2,000
35	0,010	0,190	0,800	2,000
36	0,020	0,040	0,940	2,000
37	0,040	0,040	0,920	2,000
38	0,000	0,010	0,990	2,000
39	0,010	0,010	0,990	2,000
40	0,020	0,040	0,940	2,000
41	0,000	0,010	0,980	2,000
42	0,000	0,010	0,980	2,000
43	0,010	0,030	0,960	2,000
44	0,000	0,010	0,990	2,000

Figura 6: Predicciones probabilísticas del modelo VM con SVD.

NN+SVD	Salud	Politica	Deportes	Clase_real
0	1,000	0,000	0,000	0,000
1	1,000	0,000	0,000	0,000
2	0,000	0,000	1,000	0,000
3	1,000	0,000	0,000	0,000
4	1,000	0,000	0,000	0,000
5	0,000	0,000	1,000	0,000
6	0,000	0,000	1,000	0,000
7	0,000	1,000	0,000	0,000
8	0,000	1,000	0,000	0,000
9	1,000	0,000	0,000	0,000
10	1,000	0,000	0,000	0,000
11	1,000	0,000	0,000	0,000
12	0,000	0,000	1,000	0,000
13	1,000	0,000	0,000	0,000
14	0,000	1,000	0,000	0,000
15	0,000	0,000	1,000	1,000
16	1,000	0,000	0,000	1,000
17	0,000	0,000	1,000	1,000
18	1,000	0,000	0,000	1,000
19	0,000	0,000	1,000	1,000
20	0,990	0,000	0,010	1,000
21	0,000	1,000	0,000	1,000
22	0,000	0,000	1,000	1,000
23	0,000	0,000	0,990	1,000
24	0,000	0,000	1,000	1,000
25	1,000	0,000	0,000	1,000
26	1,000	0,000	0,000	1,000
27	0,000	0,000	1,000	1,000
28	1,000	0,000	0,000	1,000
29	1,000	0,000	0,000	1,000
30	0,000	1,000	0,000	2,000
31	0,000	0,000	1,000	2,000
32	0,000	1,000	0,000	2,000
33	0,000	0,000	1,000	2,000
34	1,000	0,000	0,000	2,000
35	0,000	0,000	1,000	2,000
36	0,000	1,000	0,000	2,000
37	1,000	0,000	0,000	2,000
38	0,000	1,000	0,000	2,000
39	0,000	0,000	1,000	2,000
40	0,000	0,000	1,000	2,000
41	0,000	1,000	0,000	2,000
42	1,000	0,000	0,000	2,000
43	0,000	0,000	1,000	2,000
44	0,000	0,000	1,000	2,000

Figura 7: Predicciones probabilísticas del modelo basado en redes neuronales.

NB-Glosario	Salud	Politica	Deportes	Clase_real
0	0,4	0,28	0,33	0,000
1	0,49	0,24	0,27	0,000
2	0,45	0,27	0,28	0,000
3	0,38	0,32	0,3	0,000
4	0,37	0,31	0,31	0,000
5	0,5	0,24	0,26	0,000
6	0,4	0,29	0,3	0,000
7	0,51	0,25	0,24	0,000
8	0,46	0,27	0,27	0,000
9	0,51	0,25	0,24	0,000
10	0,48	0,26	0,26	0,000
11	0,49	0,25	0,26	0,000
12	0,41	0,31	0,28	0,000
13	0,4	0,3	0,3	0,000
14	0,49	0,25	0,26	0,000
15	0,29	0,41	0,3	1,000
16	0,23	0,52	0,24	1,000
17	0,31	0,39	0,3	1,000
18	0,31	0,36	0,33	1,000
19	0,27	0,45	0,28	1,000
20	0,33	0,34	0,33	1,000
21	0,32	0,37	0,3	1,000
22	0,28	0,43	0,29	1,000
23	0,28	0,44	0,28	1,000
24	0,3	0,42	0,28	1,000
25	0,33	0,34	0,33	1,000
26	0,28	0,44	0,29	1,000
27	0,32	0,38	0,31	1,000
28	0,28	0,45	0,27	1,000
29	0,29	0,41	0,3	1,000
30	0,25	0,24	0,51	2,000
31	0,26	0,27	0,47	2,000
32	0,29	0,28	0,44	2,000
33	0,28	0,29	0,43	2,000
34	0,27	0,28	0,45	2,000
35	0,27	0,33	0,4	2,000
36	0,28	0,28	0,44	2,000
37	0,31	0,29	0,4	2,000
38	0,27	0,27	0,46	2,000
39	0,21	0,22	0,57	2,000
40	0,31	0,31	0,38	2,000
41	0,28	0,3	0,42	2,000
42	0,28	0,28	0,44	2,000
43	0,28	0,28	0,44	2,000
44	0,25	0,26	0,5	2,000

Figura 8: Predicciones probabilísticas del modelo Naive Bayes.

RF + SVD	Salud	Politica	Deportes	Clase_real
0	0,75	0,11	0,14	0,000
1	0,82	0,08	0,1	0,000
2	0,79	0,08	0,13	0,000
3	0,64	0,23	0,13	0,000
4	0,59	0,14	0,27	0,000
5	0,85	0,07	0,08	0,000
6	0,77	0,07	0,16	0,000
7	0,8	0,09	0,11	0,000
8	0,81	0,09	0,1	0,000
9	0,8	0,09	0,11	0,000
10	0,85	0,06	0,09	0,000
11	0,83	0,1	0,07	0,000
12	0,86	0,09	0,05	0,000
13	0,77	0,12	0,11	0,000
14	0,75	0,13	0,12	0,000
15	0,18	0,47	0,35	1,000
16	0,11	0,55	0,34	1,000
17	0,19	0,48	0,33	1,000
18	0,12	0,38	0,5	1,000
19	0,16	0,62	0,22	1,000
20	0,15	0,38	0,47	1,000
21	0,22	0,43	0,35	1,000
22	0,18	0,47	0,35	1,000
23	0,12	0,59	0,29	1,000
24	0,15	0,54	0,31	1,000
25	0,27	0,37	0,36	1,000
26	0,18	0,52	0,3	1,000
27	0,19	0,44	0,37	1,000
28	0,13	0,58	0,29	1,000
29	0,15	0,55	0,3	1,000
30	0,03	0,05	0,92	2,000
31	0,05	0,12	0,83	2,000
32	0,01	0,03	0,96	2,000
33	0,11	0,15	0,74	2,000
34	0,09	0,15	0,76	2,000
35	0,06	0,19	0,75	2,000
36	0,12	0,14	0,74	2,000
37	0,06	0,04	0,9	2,000
38	0,04	0,04	0,92	2,000
39	0,01	0,18	0,81	2,000
40	0,1	0,12	0,78	2,000
41	0,02	0,08	0,9	2,000
42	0,08	0,13	0,79	2,000
43	0,07	0,14	0,79	2,000
44	0,06	0,11	0,83	2,000

Figura 9: Predicciones probabilísticas del modelo Random Forest.

.2. Notebooks

.2.1. Preprocesamiento - transformacion.ipynb

Preprocesamiento de los textos

1-Importamos librerías y cargamos drive

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/drive/')
```

Mounted at /content/drive/

2-Transformamos los textos de word a texto plano

In [2]:

```

!pip install pypandoc
import pypandoc

ruta_salud_origen      = '/content/drive/MyDrive/Ignieria_Linguistica/Documentos/SALUD/'
ruta_salud_destino     = '/content/drive/MyDrive/Ignieria_Linguistica/Documentos/salud_p
lano/'

ruta_politica_origen   = '/content/drive/MyDrive/Ignieria_Linguistica/Documentos/POLITIC
A/'
ruta_politica_destino  = '/content/drive/MyDrive/Ignieria_Linguistica/Documentos/politic
a_plano/'

ruta_deportes_origen   = '/content/drive/MyDrive/Ignieria_Linguistica/Documentos/DEPORTE
S/'
ruta_deportes_destino  = '/content/drive/MyDrive/Ignieria_Linguistica/Documentos/deporte
s_plano/'


for documento_salud in range(30):
    docx = ruta_salud_origen + str(documento_salud + 1) + '.docx'
    txt = pypandoc.convert_file(docx, 'plain', outputfile=ruta_salud_destino + str(docum
ento_salud+1) + '.txt')

for documento_politica in range(30):
    docx = ruta_politica_origen + 'POLITICA_' + str(documento_politica + 1) + '.docx'
    txt = pypandoc.convert_file(docx, 'plain', outputfile=ruta_politica_destino + str(do
cumento_politica+1) + '.txt')

for documento_deportes in range(30):
    docx = ruta_deportes_origen + 'DEPORTES_' + str(documento_deportes + 1) + '.docx'
    txt = pypandoc.convert_file(docx, 'plain', outputfile=ruta_deportes_destino + str(do
cumento_deportes+1) + '.txt')

```

Collecting pypandoc

```

  Downloading pypandoc-1.6.4.tar.gz (26 kB)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist
-packages (from pypandoc) (57.4.0)
Requirement already satisfied: pip>=8.1.0 in /usr/local/lib/python3.7/dist
-packages (from pypandoc) (21.1.3)
Requirement already satisfied: wheel>=0.25.0 in /usr/local/lib/python3.7/d
ist-packages (from pypandoc) (0.37.0)
Building wheels for collected packages: pypandoc
  Building wheel for pypandoc (setup.py) ... done
  Created wheel for pypandoc: filename=pypandoc-1.6.4-py3-none-any.whl siz
e=17427 sha256=a063ec70e17fc82cdb88f449034292adbb81a69d27cd8263af24f3eb6c7
1427c
  Stored in directory: /root/.cache/pip/wheels/9a/d8/4b/4e1fb6b7b685395364
021848b77a6fc061283b56c3998ff78c
Successfully built pypandoc
Installing collected packages: pypandoc
Successfully installed pypandoc-1.6.4

```

2.1-Leemos todos los txt

In [3]:

```

documentos_salud      = []
documentos_politica = []
documentos_deportes = []

for documento in range(1,31):
    txt_deporte = open('/content/drive/MyDrive/Ignieria_Linguistica/Documentos/deportes_plano/'+str(documento)+'.txt')
    txt_politica = open('/content/drive/MyDrive/Ignieria_Linguistica/Documentos/politica_plano/'+str(documento)+'.txt')
    txt_salud = open('/content/drive/MyDrive/Ignieria_Linguistica/Documentos/salud_plano/'+str(documento)+'.txt')

    content_deportes = txt_deporte.read()
    content_politica = txt_politica.read()
    content_salud = txt_salud.read()

    documentos_deportes.append(content_deportes)
    documentos_politica.append(content_politica)
    documentos_salud.append(content_salud)

for documento in range(31, 61):
    with open('/content/drive/MyDrive/Ignieria_Linguistica/Documentos/deportes_plano/'+'DEPORTES_'+str(documento)+'.txt', encoding="utf8", errors='ignore') as txt_deporte:
        content_deportes = txt_deporte.read()

    with open('/content/drive/MyDrive/Ignieria_Linguistica/Documentos/politica_plano/'+'POLITICA_'+str(documento)+'.txt', encoding="utf8", errors='ignore') as txt_politica:
        content_politica = txt_politica.read()

    with open('/content/drive/MyDrive/Ignieria_Linguistica/Documentos/salud_plano/'+'SALUD_'+str(documento)+'.txt', encoding="utf8", errors='ignore') as txt_salud:
        content_salud = txt_salud.read()

    documentos_deportes.append(content_deportes)
    documentos_politica.append(content_politica)
    documentos_salud.append(content_salud)

print(len(documentos_salud), len(documentos_politica), len(documentos_deportes))

```

60 60 60

2.3-Creamos un Dataframe por lista dando una clase para cada uno de ellos

1. Salud
2. Política
3. Deportes

In [4]:

```
df_salud      = pd.DataFrame(documentos_salud, columns=['texto'])
df_salud['topic'] = 0

df_politica = pd.DataFrame(documentos_politica, columns=['texto'])
df_politica['topic'] = 1

df_deportes = pd.DataFrame(documentos_deportes, columns=['texto'])
df_deportes['topic'] = 2

df_salud[:2]
```

Out[4]:

	texto	topic
0	CORONAVIRUS HOY, ÚLTIMAS NOTICIAS ALEMANIA B...	0
1	CUENTA ATRÁS PARA VACUNAR CONTRA EL COVID A ME...	0

2.4-Mezclamos de manera aleatoria los documentos Los 30 primeros han sido extraídos por un miembro del grupo y los otros 30 por el otro, lo que debido a la fuente de información, podría introducir un sesgo, ya que más adelante utilizaremos los 45 primeros para crear el glosario.

In [5]:

```
df_salud      = df_salud.sample(frac=1).reset_index(drop=True)
df_politica = df_politica.sample(frac=1).reset_index(drop=True)
df_deportes = df_deportes.sample(frac=1).reset_index(drop=True)

df_salud[:5]
```

Out[5]:

	texto	topic
0	\n\nZAPATOS CON ANCHO ESPECIAL Y PARA PERSONAS...	0
1	EL ACEITE DE PALMA PROMUEVE LA METÁSTASIS DEL ...	0
2	\n\nBENEFICIOS DE LA CIRUGÍA ESTÉTICA PARA LA ...	0
3	\n\nLA REHABILITACIÓN CARDIACA DISMINUYE LA MO...	0
4	Dieta sana en la desescalada: ojo con los hidr...	0

2.5-Unimos todos los DF

In [6]:

```
df = pd.concat([df_salud, df_politica, df_deportes], ignore_index=True)
df
```

Out[6]:

	texto	topic
0	\n\nZAPATOS CON ANCHO ESPECIAL Y PARA PERSONAS...	0
1	EL ACEITE DE PALMA PROMUEVE LA METÁSTASIS DEL ...	0
2	\n\nBENEFICIOS DE LA CIRUGÍA ESTÉTICA PARA LA ...	0
3	\n\nLA REHABILITACIÓN CARDIACA DISMINUYE LA MO...	0
4	Dieta sana en la desescalada: ojo con los hidr...	0
...
175	\n\nCUÁNDO ES EL MUNDIAL DE QATAR 2022\n\nLA...	2
176	\n\nUN GOL DE SERBIA EN EL ÚLTIMO MINUTO CONDE...	2
177	\n\n KIKO 'SENSACIÓN' MARTÍNEZ, CAMPEÓN CON UN...	2
178	\n\nGARBIÑE MUGURUZA Y PAULA BADOSA, SEMIFINAL...	2
179	\n\nLUIS ENRIQUE: "AHORA TOCA BAILAR PEGADITOS...	2

180 rows × 2 columns

2.6-Pasamos los datos a csv para no tener que volver a hacer esta transformación

In [7]:

```
df.to_csv('/content/drive/MyDrive/Ignieria_Linguistica/datos.csv', index=False)
```

2.7-Leemos los datos para asegurarnos de que se han guardado correctamente

In []:

```
df = pd.read_csv('/content/drive/MyDrive/Ignieria_Linguistica/datos.csv')
```

3-Preprocesamos el texto

3.1-Pasamos a minúsculas el texto

3.2-Tokenizamos el texto, es decir, lo transformamos en una lista de palabras individuales

3.3-Eliminamos signos de puntuación como exclamaciones, interrogaciones, comas o puntos.

3.4-Eliminamos los números que puedan existir en el texto.

3.5-Eliminamos las palabras que son comunes del lenguaje español y que no aportan significado, como artículos o preposiciones.

3.6-Finalmente tenemos todo el texto preprocessado.

In []:

```
import string
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
def preprocesar_textos(df):
    nltk.download('punkt')
    nltk.download('stopwords')
    df = df.copy()
    ## 1) Lower
    df['texto'] = df['texto'].str.lower()

    ## 2) Tokenize
    df['texto_tokenizado'] = df.apply(lambda x: word_tokenize(x['texto']), axis=1)

    ## 3) Remove punctuation
    nltk.download('punkt')
    table = str.maketrans('', '', string.punctuation)
    df['texto_tokenizado'] = df.apply(lambda x: [w.translate(table)
                                                for w in x['texto_tokenizado']], axis=1)

    ## 4) Remove non-alpha
    df['texto_tokenizado'] = df.apply(lambda x:
                                       [w for w in x['texto_tokenizado'] if w.isalpha()],
                                       axis=1)

    ## 5) Remove stop-words
    nltk.download('stopwords')
    stop_words = set(stopwords.words('spanish'))
    df['texto_tokenizado'] = df.apply(lambda x:
                                       [w for w in x['texto_tokenizado'] if not w in stop_
                                         words], axis=1)

    ## 6) Reformat to have a single text.
    df['texto'] = df.apply(lambda x: ' '.join(x['texto_tokenizado']), axis=1)

    ## 7) Eliminamos la columna que ya no es necesaria
    df.drop('texto_tokenizado', inplace=True, axis=1)
    return df
df = preprocesar_textos(df)
df
```

3.7-Con el objetivo de que, en todas las **aproximaciones** sean igualmente **justas**, es decir, se usen los **mismos documentos para extraer las características** y los **mismos documentos para test**, vamos a hacer una división de los datos en dos DataFrames. El primero de ellos serán los documentos a través de los cuales vamos a extraer las características para predecir los documentos. El segundo de ellos será el destinado a medir la bondad del clasificador.

Determinamos 45 documentos de cada temática para extracción de características y 15 para hacer un test sobre el modelo. Esta división es debida a que creemos que una aproximación más realista en una experimentación real, podría llegar a faltar el 25% del vocabulario respecto de los documentos con los que se está haciendo el entrenamiento, ya que creemos que que falte el 50% es algo desmesurado.

In []:

```
doc_salud_train      = df.iloc[:45]
doc_politica_train = df.iloc[60:105]
doc_deportes_train = df.iloc[120:165]
dfs_train = [doc_salud_train, doc_politica_train, doc_deportes_train]
df_train = pd.concat(dfs_train)
df_train.to_csv('/content/drive/MyDrive/Ignieria_Linguistica/entrenamiento.csv', index=False)

doc_salud_test      = df.iloc[45:60]
doc_politica_test = df.iloc[105:120]
doc_deportes_test = df.iloc[165:]
dfs_test = [doc_salud_test, doc_politica_test, doc_deportes_test]
df_test = pd.concat(dfs_test)
df_test.to_csv('/content/drive/MyDrive/Ignieria_Linguistica/test.csv', index=False)
```

3.8-A partir de estos textos preprocesados, comenzaremos a tratar nuestros modelos y a extraer características.

.2.2. Modelo SVM con glosario - Modelo_SVM.ipynb

Clasificador basado en SVM con glosario personalizado

1-Montamos y cargamos los datos

In [12]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/drive/')

df_train = pd.read_csv('/content/drive/MyDrive/Ignieria_Lingustica/entrenamiento.csv')
df_test = pd.read_csv('/content/drive/MyDrive/Ignieria_Lingustica/test.csv')
```

Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/content/drive/", force_remount=True).

2-Obtención de glosario

2.1-Extraemos las palabras más relevantes convirtiendo la colección de documentos en una matriz de características TF-IDF.

Supongamos N=Número de documentos, T=apariciones de un termino en un documento, df=Número de apariciones de un término (t) en todos los documentos, NTD= Número de términos de un documento. Entonces:

1. TF(Term Frequency): Número de apariciones de un término en un determinado documento.

$$tf(t) = \frac{t}{NTD}$$

2. IDF(Inverse Document Frequency): Numero de apariciones de un término sobre el total de documentos.

$$idf(t) = \log_{10}\left[\frac{1+N}{1+df(t)}\right] + 1$$

3. TF-IDF: Permite conocer la relevancia de una palabra clave en un documento.

$$tf - idf(t, d) = tf(t, d) \cdot idf(t)$$

Debido al preprocessado del notebook **Transformacion_inicial.ipynb** sabemos que df_train tiene 45 documentos de cada tipo de manera ordenada, es decir, los [0-44] pertenecen a salud, [45-89] pertenecen a política y [90:134] pertenecen a deportes. El primer paso para obtener el glosario será crear una matriz TFIDF por temática.

In []:

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_salud = TfidfVectorizer()
tfidf_deportes = TfidfVectorizer()
tfidf_politica = TfidfVectorizer()

df_salud = df_train.iloc[:45, 0]
df_politica = df_train.iloc[45:90, 0]
df_deportes = df_train.iloc[90:, 0]

tfidf_salud_train = tfidf_salud.fit_transform(df_salud)
tfidf_politica_train = tfidf_politica.fit_transform(df_politica)
tfidf_deportes_train = tfidf_deportes.fit_transform(df_deportes)
print(tfidf_salud_train.shape, tfidf_deportes_train.shape, tfidf_politica_train.shape)
```

(45, 5456) (45, 6852) (45, 6678)

2.2-Creación de glosario según temática: Obtenemos los 100 términos más importantes según la métrica TFIDF para cada temática

In []:

```
def obtener_glosario(tfidf, tfidf_train, topico, nterms=100):
    """Dada una matriz tfidf se devuelven los n keywords más relevantes"""
    terms = tfidf.get_feature_names_out()

    # sum tfidf frequency of each term through documents
    sums = tfidf_train.sum(axis=0)

    # connecting term to its sums frequency
    data = []
    for col, term in enumerate(terms):
        data.append( (term, sums[0,col] ) )

    df_ranking = pd.DataFrame(data, columns=['palabra','ranking'])
    df_ranking = df_ranking.sort_values('ranking', ascending=False)
    print("======" + topico + "=====")
    print(df_ranking[:10])
    df_ranking.drop('ranking', inplace=True, axis=1)
    df_ranking = df_ranking.iloc[:nterms]
    return df_ranking

glosario_salud      = obtener_glosario(tfidf_salud, tfidf_salud_train, "SALUD")
glosario_politica  = obtener_glosario(tfidf_politica, tfidf_politica_train, "POLITICA")
glosario_deportes  = obtener_glosario(tfidf_deportes, tfidf_deportes_train, "DEPORTES")
```

=====SALUD=====

	palabra	ranking
4633	salud	1.436722
5332	vida	1.277392
4760	si	1.175304
4203	puede	1.147999
3670	pacientes	1.129328
3824	personas	1.065938
4738	ser	1.044038
5136	tratamiento	1.020511
4204	pueden	0.946392
2457	gripe	0.934924

=====POLITICA=====

	palabra	ranking
714	años	1.393596
4855	política	1.304164
4911	pp	1.243645
3066	gobierno	1.232390
6336	tribunal	1.213392
4829	podemos	1.103156
4987	presidente	1.102825
6127	sánchez	1.063895
2525	españa	1.059743
2226	efe	1.029939

=====DEPORTES=====

	palabra	ranking
3907	madrid	1.350337
4279	mundial	1.272425
2467	equipo	1.253648
2537	españa	1.182940
4653	partido	1.155746
5364	real	1.141463
3693	laliga	1.072634
667	años	1.062380
3614	jugador	1.060628
5900	selección	1.000111

2.3-Volvemos los glosarios a un excel para realizar una revisión manual de cada uno de los glosarios.

A continuación se muestra una lista de las palabras eliminadas según temática:

1. Salud: si, ser, puede, pueden, cada, mejor, bien, según, hacer, manera, día, hoy, tambin, días, así, hace, solo, dos, siempre, ejemplo, incluso, mismo, muchas. :23
2. Política: dos, ahora, hace, ser, tras, siempre, sido, pasado, meses, solo, bien, según, aunque, mismo, año. :15
3. Deportes: dos, tres, si, sido, ser, tras, después, solo, ahora, así, hace, siempre, tan, aunque. :14

In []:

```
glosario_salud.to_excel('/content/drive/MyDrive/Ignieria_Linguistica/glosarios/salud.xlsx', index=False)
glosario_politica.to_excel('/content/drive/MyDrive/Ignieria_Linguistica/glosarios/politica.xlsx', index=False)
glosario_deportes.to_excel('/content/drive/MyDrive/Ignieria_Linguistica/glosarios/deportes.xlsx', index=False)
```

2.3.1-Cargamos los excel revisados manualmente a un dataframe, le añadimos la clase a la que pertenece y vemos el número de palabras por glosario resultante

In [13]:

```
glosario_salud      = pd.read_excel('/content/drive/MyDrive/Ignieria_Lingustica/glosarios/salud_revisado.xlsx')
glosario_politica = pd.read_excel('/content/drive/MyDrive/Ignieria_Lingustica/glosarios/politica_revisado.xlsx')
glosario_deportes = pd.read_excel('/content/drive/MyDrive/Ignieria_Lingustica/glosarios/deportes_revisado.xlsx')

glosario_salud['clase'] = 0
glosario_politica['clase'] = 1
glosario_deportes['clase'] = 2

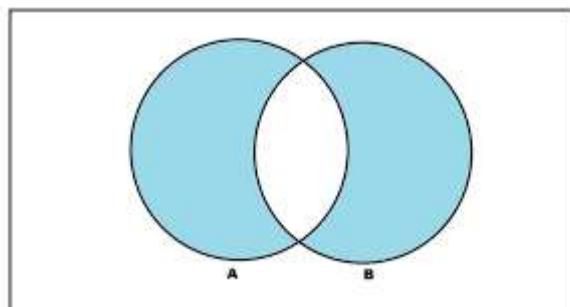
print(glosario_salud.shape, glosario_politica.shape, glosario_deportes.shape)
```

(78, 2) (86, 2) (87, 2)

2.5-Obtención de glosario global:Hacemos la intersección simétrica entre los glosraios revisados, ya que nos interesan las palabras más diferentes posibles, y si son comunes a dos tópicos preferimos no tenerlas. Para ello primeramente los unimos y luego eliminamos los elementos duplicados.

En este caso, al hacer esta operación hemos pasado de tener un glosario de 251 palabras (87+86+78) a uno de 212, es decir, existían 39 palabras que ser repetían en al menos dos temáticas.

Una vez hecha esta operación, lo volcamos sobre un csv, ya que utilizaremos este mismo glosario para otras aproximaciones.



In [14]:

```

glosarios = [glosario_salud, glosario_politica, glosario_deportes]

glosario = pd.concat(glosarios)
glosario = glosario.drop_duplicates(subset='palabra', keep=False)
glosario.reset_index(drop=True, inplace=True)
glosario.to_csv('/content/drive/MyDrive/Ignieria_Linguistica/glosario.csv', index=False)
)
print('Salud:', len(glosario[(glosario['clase']==0)]), '| Politica:', len(glosario[(glosario['clase']==1)]), '| Deportes:', len(glosario[(glosario['clase']==2)]))
glosario

```

Salud: 71 | Politica: 71 | Deportes: 70

Out[14]:

	palabra	clase
0	salud	0
1	vida	0
2	pacientes	0
3	personas	0
4	tratamiento	0
...
207	wolff	2
208	marc	2
209	galahad	2
210	británico	2
211	grupo	2

212 rows × 2 columns

3-Creación de una matriz TFIDF a partir del glosario global.

3.1-Obtenemos el tfidf global de todos los documentos a partir de los cuales hemos obtenido nuestros 3 glosarios. df_train

In [15]:

```

tfidf_global = TfidfVectorizer()
tfidf_global_train = tfidf_global.fit_transform(df_train.iloc[:,0])

#Transformamos el texto del conjunto de test a TFIDF
x_test = tfidf_global.transform(df_test.iloc[:,0])

```

3.1.1-Obtenemos los índices de las palabras de nuestro glosario

In [16]:

```
vocabulario_glosario = glosario['palabra'].tolist()
indices_palabras_glosario = [tfidf_global.vocabulary_[key] for key in vocabulario_glosario]
```

3.2.2-Obtenemos los valores tfidf asociados a cada palabra de nuestro vocabulario para cada uno de los documentos.

In [17]:

```
x_train = tfidf_global_train[:, indices_palabras_glosario]
x_train = x_train.todense()

x_test = x_test[:, indices_palabras_glosario]
x_test = x_test.todense()
```

3.3-Creacion de datos de entrenamiento y test (X)

3.3.1-Creacion DF con vocabulario glosario y el tfidf asociado (ENTRADA DEL MODELO)

In [20]:

```
df_x_train = pd.DataFrame(x_train)
df_x_train.columns=vocabulario_glosario
df_x_train
```

Out[20]:

	salud	vida	pacientes	personas	tratamiento	gripe	cuerpo	ms	piel	l
0	0.021774	0.019725	0.000000	0.037714	0.000000	0.0	0.220167	0.0	0.000000	0
1	0.017686	0.000000	0.000000	0.045950	0.000000	0.0	0.000000	0.0	0.000000	0
2	0.082838	0.000000	0.000000	0.023913	0.000000	0.0	0.000000	0.0	0.000000	0
3	0.036319	0.032901	0.000000	0.015727	0.021091	0.0	0.142816	0.0	0.026865	0
4	0.011650	0.031660	0.057138	0.010089	0.067654	0.0	0.000000	0.0	0.017235	0
...
130	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.0	0.000000	0
131	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.0	0.000000	0
132	0.000000	0.020340	0.000000	0.000000	0.000000	0.0	0.050452	0.0	0.000000	0
133	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.0	0.000000	0
134	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.085679	0.0	0.000000	0

135 rows × 212 columns

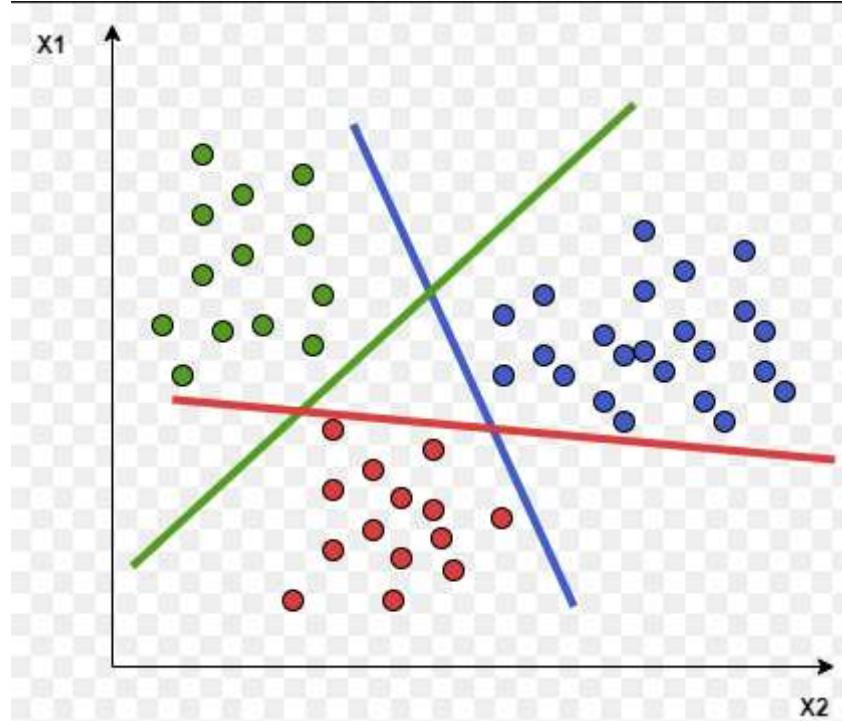
3.3.2-Creación de etiquetas asociadas a la entrada del modelo (SALIDA DEL MODELO)

In []:

```
df_y = df_train.iloc[:,1]
```

4-Creamos un clasificador SVM (Support Vector Machine).

Este clasificador tratará de trazar líneas de separación lo mejor posible para separar los diferentes puntos (puntos son cada uno de los documentos (representados por el conjunto de características extraídas)) según su clase.



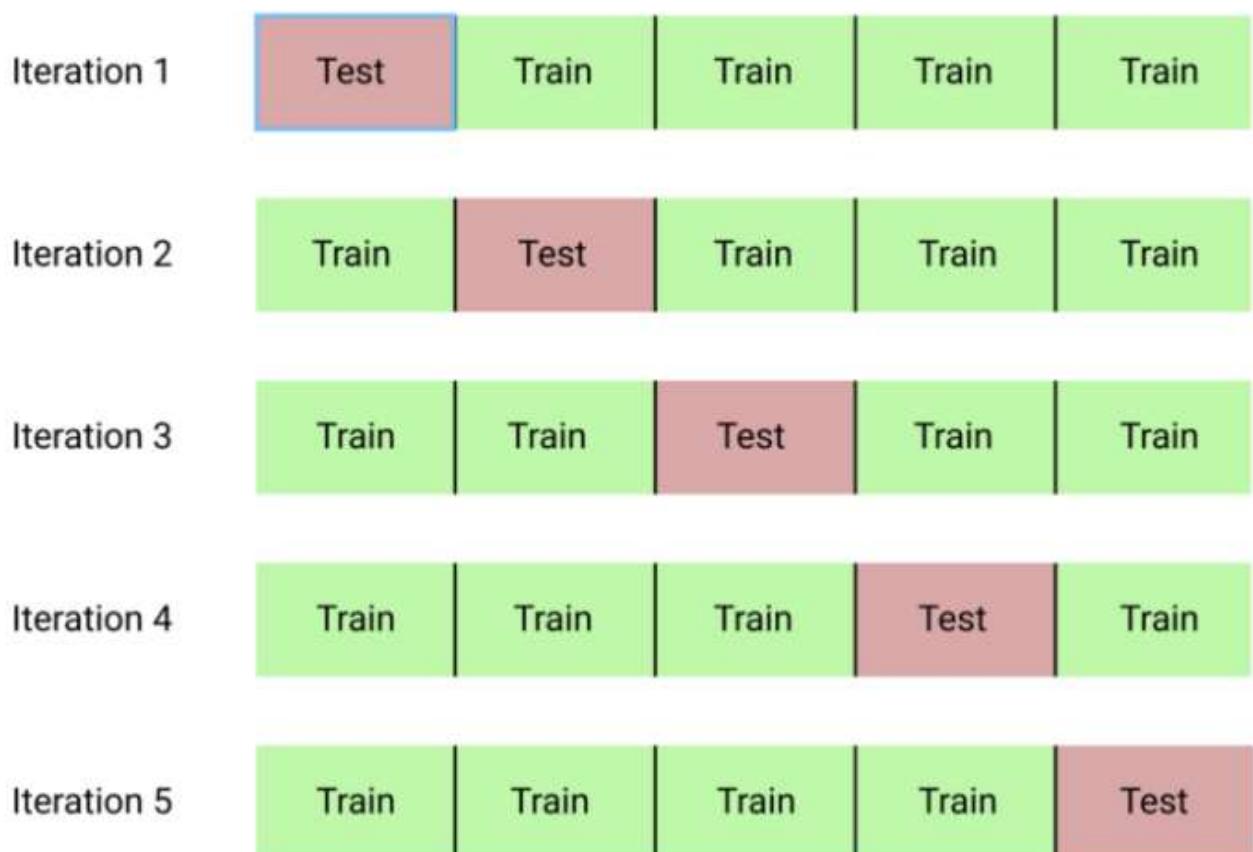
In []:

```
from sklearn import svm
model = svm.SVC(kernel='linear', C=1, decision_function_shape='ovo', probability=True)
```

4.1-Obtenemos una aproximación de la bondad del mismo mediante repeatedKfold.

Con el objetivo de tener una **estimación más realista de las métricas o bondad del modelo**, y más en este caso que tratamos con pocos datos, es muy conveniente llevar a cabo una **validación cruzada**. La validación cruzada k-fold consiste en dividir el conjunto de datos en k divisiones no superpuestas. De esta manera se crean K modelos, cada uno de ellos con diferentes datos de entrenamiento y de prueba y se obtiene el resultado medio de estos modelos. n_splits = Número de particiones para nuestro conjunto de datos. n_repeats = Número de validaciones cruzadas a repetir. Por tanto, no es lo mismo n_splits = 10 y n_repeats = 1 que n_splits = 5 y n_repeats = 2. En el primer caso, estaríamos obteniendo 10 divisiones de datos, pero dicha división aleatoria se realizaría una vez. En el segundo caso se obtendrían 10 divisiones de datos nuevamente, pero esos datos se dividirían aleatoriamente 2 veces.

k-Fold Cross Validation:



K=4 y n_repeat = 1

[1,2,3], [4,5,6], [7,8,9], [10,11,12]

k=2 y n_repeat=2

[1,2,3,4,5,6], [7,8,9,10,11,12]

[7,8,9,1,2,3], [4,5,6,10,11,12]

In []:

```
from sklearn.model_selection import cross_validate
from sklearn.model_selection import RepeatedKFold
from numpy import mean
from numpy import std
y_train = df_y.values
x_train = df_x.values

cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
metrics = cross_validate(model, x_train, y_train, scoring=['precision_macro', 'recall_macro'], cv=cv, n_jobs=-1)

print('Precision: ', str(round((mean(metrics["test_precision_macro"])),3))), '| Desviación típica: ', str( round( std(metrics["test_precision_macro"])), 3)))
```

Precision: 0.91 | Desviación típica: 0.087

Como podemos observar, los resultados parecen buenos, por lo que vamos a entrenar el modelo.

5-Entrenamiento y evaluación del modelo

5.1-Entrenamiento del modelo con los datos a partir del cual hemos obtenido el glosario

In []:

```
clasificador = svm.SVC(kernel='linear', C=1, decision_function_shape='ovo', probability=True).fit(x_train, y_train)
```

5.2-Evaluación sobre los datos de test

5.2.1-Obtenemos las predicciones en forma probabilistica y las volcamos en un excel para incluirlo en la memoria.

In []:

```
predicciones = clasificador.predict_proba(x_test)
predicciones_rounded = [np.round(x,3) for x in predicciones]

df_predicciones = pd.DataFrame(predicciones_rounded)
df_predicciones.columns=['Salud', 'Politica', 'Deportes']
df_predicciones.index.name = 'Documento'
df_predicciones['Clase_real'] = y_test
df_predicciones.to_excel('/content/drive/MyDrive/Ignieria_Linguistica/modelo_SVM_glosario/predicciones.xlsx')
```

/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:590: FutureWarning: np.matrix usage is deprecated in 1.0 and will raise a TypeError in 1.2. Please convert to a numpy array with np.asarray. For more information see: <https://numpy.org/doc/stable/reference/generated/numpy.matrix.html>

FutureWarning,

5.2.2-Obtenemos las predicciones sin probabilidad

In []:

```
predicciones = clasificador.predict(x_test_glosario_propio)
predicciones
```

/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:590: FutureWarning: np.matrix usage is deprecated in 1.0 and will raise a TypeError in 1.2. Please convert to a numpy array with np.asarray. For more information see: <https://numpy.org/doc/stable/reference/generated/numpy.matrix.html>

FutureWarning,

Out[]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 1,
       1, 1, 1, 2, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

5.3-Obtenemos un informe del clasificador

Lo guardamos en un excel para incluirlo en la memoria

In []:

```
from sklearn.metrics import classification_report
target_names = ['Salud', 'Politica', 'Deportes']
informe = classification_report(y_test, predicciones, target_names=target_names, digits=3)
print(classification_report(y_test, predicciones, target_names=target_names, digits=3))

informe = classification_report(y_test, predicciones, target_names=target_names, digits=3, output_dict=True)
df_informe = pd.DataFrame(informe).transpose()
df_informe.to_excel('/content/drive/MyDrive/Ignieria_Lingustica/modelo_SVM_glosario/informe.xlsx', index=True)
```

	precision	recall	f1-score	support
Salud	1.000	1.000	1.000	15
Politica	1.000	0.867	0.929	15
Deportes	0.882	1.000	0.938	15
accuracy			0.956	45
macro avg	0.961	0.956	0.955	45
weighted avg	0.961	0.956	0.955	45

5.4-Obtenemos una matriz de correlación para ver el número de bien y mal clasificados

In []:

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(clasificador, x_test_glosario_propio, y_test)
```

```
-----
-
NameError: name 'clasificador' is not defined
Traceback (most recent call last):
  File "<ipython-input-1-8e8c20a6ea46>", line 1, in <module>
    plot_confusion_matrix(clasificador, x_test_glosario_propio, y_test)
  File "/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py", line 100, in plot_confusion_matrix
    clasificador = check_classification_targets(y_true, y_pred)
```

NameError: name 'clasificador' is not defined

Esta matriz de confusión nos muestra que:

1. De los 15 documentos de **salud**, 15 han sido correctamente clasificado.
2. De los 15 documentos de **política**, 13 han sido correctamente clasificado.
3. De los 15 documentos de **deportes**, 15 han sido correctamente clasificado.

Como conclusión podemos extraer que parece que los glosarios son muy buenos para salud y deportes, pero que no lo es tanto para política.

.2.3. Modelo SVM con SVD - Modelo_SVM_SVD.ipynb

Clasificador basado en SVM con simplificación SVD

1-Montamos y cargamos los datos

In []:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/drive/')

df_train = pd.read_csv('/content/drive/MyDrive/Ignieria_Linguistica/entrenamiento.csv')
df_test = pd.read_csv('/content/drive/MyDrive/Ignieria_Linguistica/test.csv')
```

Mounted at /content/drive/

2-Extracción terminológica: vector TF-IDF

Una vez más vamos a crear la matriz TF-IDF sobre los datos de entrenamiento.

In []:

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
x_train_tfidf = tfidf.fit_transform(df_train.iloc[:, 0])
x_test_tfidf = tfidf.transform(df_test.iloc[:, 0])

y_train = df_train.iloc[:, 1]
y_test = df_test.iloc[:, 1]
print(x_train_tfidf.shape, x_test_tfidf.shape)
```

(135, 14347) (45, 14347)

2.1-Reducimos la dimensionalidad de nuestras características de entrada para el clasificador mediante SVD.

SVD o Single Value Decomposition es una técnica de reducción de dimensión para matrices. Además este método funciona mejor con datos dispersos, es decir, datos con muchos valores a cero. Este es justo el caso en el que nos encontramos. Esto es debido a que muchas de las palabras no aparecen en gran cantidad de documentos de nuestro corpus, por lo que al hacer $TF \cdot IDF$ siendo IDF 0, se obtienen gran cantidad.

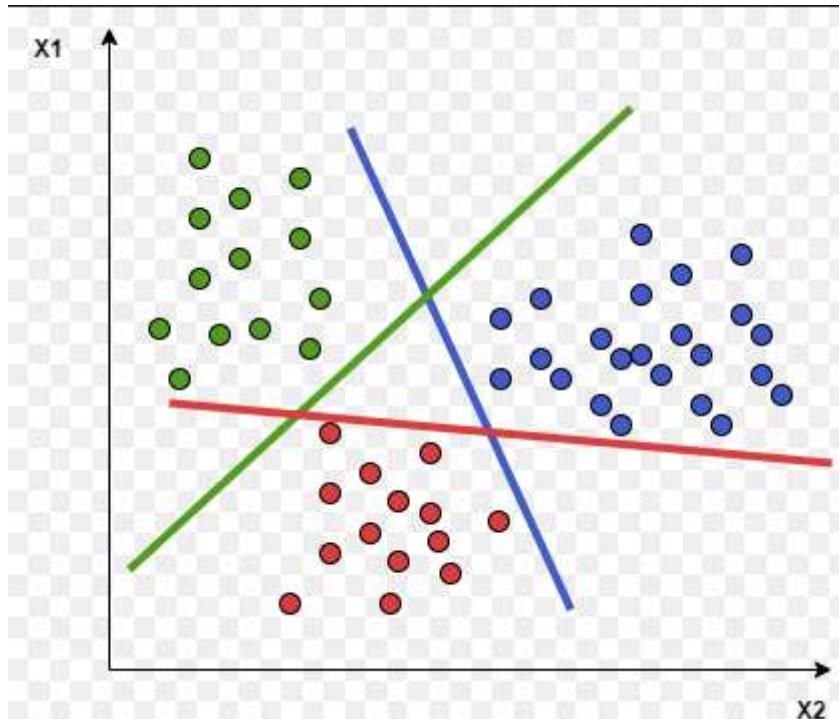
In []:

```
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=100)
x_train_svd = svd.fit_transform(x_train_tfidf)
x_test_svd = svd.transform(x_test_tfidf)
print(x_train_svd.shape, x_test_svd.shape)
```

(135, 100) (45, 100)

3-Creamos un clasificador SVM (Support Vector Machine)

Este clasificador tratará de trazar líneas de separación lo mejor posible para separar los diferentes puntos (puntos son cada uno de los documentos) según su clase.



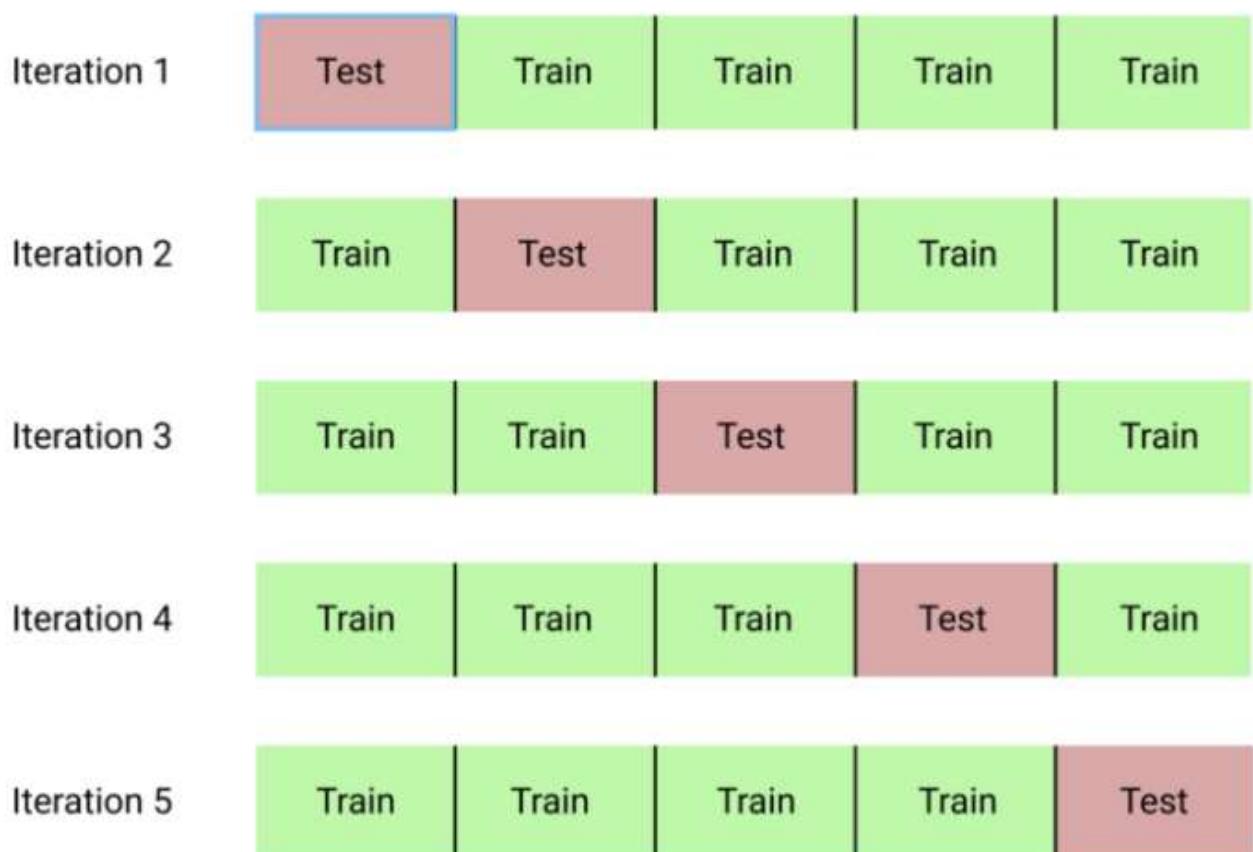
In []:

```
from sklearn import svm
model = svm.SVC(kernel='linear', C=1, decision_function_shape='ovo')
```

3.1-Obtenemos una aproximación de la bondad del mismo mediante repeatedKfold.

Con el objetivo de tener una **estimación más realista de las métricas o bondad del modelo**, y más en este caso que tratamos con pocos datos, es muy conveniente llevar a cabo una **validación cruzada**. La validación cruzada k-fold consiste en dividir el conjunto de datos en k divisiones no superpuestas. De esta manera se crean K modelos, cada uno de ellos con diferentes datos de entrenamiento y de prueba y se obtiene el resultado medio de estos modelos. n_splits = Número de particiones para nuestro conjunto de datos. n_repeats = Número de validaciones cruzadas a repetir. Por tanto, no es lo mismo n_splits = 10 y n_repeats = 1 que n_splits = 5 y n_repeats = 2. En el primer caso, estaríamos obteniendo 10 divisiones de datos, pero dicha división aleatoria se realizaría una vez. En el segundo caso se obtendrían 10 divisiones de datos nuevamente, pero esos datos se dividirían aleatoriamente 2 veces.

k-Fold Cross Validation:



K=4 y n_repeat = 1

[1,2,3], [4,5,6], [7,8,9], [10,11,12]

k=2 y n_repeat=2

[1,2,3,4,5,6], [7,8,9,10,11,12]

[7,8,9,1,2,3], [4,5,6,10,11,12]

In []:

```
from sklearn.model_selection import cross_validate
from sklearn.model_selection import RepeatedKFold
from numpy import mean
from numpy import std

cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
metrics = cross_validate(model, x_train_svd, y_train, scoring=['precision_macro', 'recall_macro'], cv=cv, n_jobs=-1)

print('Precision: ', str(round((mean(metrics["test_precision_macro"])),3)), '| Desviación típica: ', str( round( std(metrics["test_precision_macro"]), 3)))
```

Precision: 0.938 | Desviación típica: 0.083

Como podemos observar, los resultados parecen buenos, por lo que vamos a entrenar el modelo.

4-Evaluación del clasificador SVM-SVD sobre los datos de test

In []:

```
clf= svm.SVC(kernel='linear', C=1, decision_function_shape='ovo', probability=True).fit(x_train_svd, y_train)
```

4.1-Predicciones probabilísticas

In []:

```
predicciones_prob = clf.predict_proba(x_test_svd)
predicciones_rounded = [np.round(x,2) for x in predicciones_prob]

df_predicciones = pd.DataFrame(predicciones_rounded)
df_predicciones.columns=['Salud', 'Politica', 'Deportes']
df_predicciones.index.name = 'Documento'
df_predicciones.to_excel('/content/drive/MyDrive/Ignieria_Lingustica/modelo_SVM_SVD/pr
edicciones.xlsx')
predicciones_rounded

predicciones_rounded
```

Out[]:

```
[array([0.84, 0.06, 0.1]),  
 array([1., 0., 0.]),  
 array([0.96, 0.03, 0.01]),  
 array([0.92, 0.05, 0.03]),  
 array([0.8 , 0.06, 0.14]),  
 array([1., 0., 0.]),  
 array([0.97, 0.02, 0.02]),  
 array([1., 0., 0.]),  
 array([0.99, 0. , 0. ]),  
 array([1., 0., 0.]),  
 array([0.99, 0. , 0. ]),  
 array([0.99, 0. , 0.01]),  
 array([0.98, 0.01, 0.01]),  
 array([0.97, 0.01, 0.01]),  
 array([0.99, 0. , 0. ]),  
 array([0.01, 0.96, 0.03]),  
 array([0., 1., 0.]),  
 array([0.04, 0.93, 0.03]),  
 array([0.04, 0.73, 0.23]),  
 array([0.01, 0.99, 0.01]),  
 array([0.05, 0.68, 0.27]),  
 array([0.02, 0.95, 0.03]),  
 array([0.02, 0.95, 0.03]),  
 array([0.02, 0.96, 0.01]),  
 array([0.05, 0.91, 0.04]),  
 array([0.15, 0.73, 0.12]),  
 array([0.01, 0.98, 0.01]),  
 array([0.07, 0.88, 0.05]),  
 array([0. , 0.98, 0.01]),  
 array([0. , 0.99, 0.01]),  
 array([0. , 0. , 0.99]),  
 array([0.01, 0.01, 0.98]),  
 array([0.01, 0.02, 0.97]),  
 array([0.01, 0.01, 0.98]),  
 array([0.02, 0.07, 0.92]),  
 array([0.01, 0.19, 0.8 ]),  
 array([0.02, 0.04, 0.94]),  
 array([0.04, 0.04, 0.92]),  
 array([0. , 0.01, 0.99]),  
 array([0.01, 0.01, 0.99]),  
 array([0.02, 0.04, 0.94]),  
 array([0. , 0.01, 0.98]),  
 array([0. , 0.01, 0.98]),  
 array([0.01, 0.03, 0.96]),  
 array([0. , 0.01, 0.99])]
```

4.2-Generamos un informe de la precisión, recall, f1-score y support del modelo sobre los datos de test. Lo guardamos en un excel

In []:

```
predicciones = clf.predict(x_test_svd)

from sklearn.metrics import classification_report
target_names = ['Salud', 'Politica', 'Deportes']
informe = classification_report(y_test, predicciones, target_names=target_names, digits=3)
print(informe)

informe = classification_report(y_test, predicciones, target_names=target_names, digits=3, output_dict=True)
df_informe = pd.DataFrame(informe).transpose()
df_informe.to_excel('/content/drive/MyDrive/Ignieria_Lingustica/modelo_SVM_SVD/informe.xlsx', index=True)
```

	precision	recall	f1-score	support
Salud	1.000	1.000	1.000	15
Politica	1.000	1.000	1.000	15
Deportes	1.000	1.000	1.000	15
accuracy			1.000	45
macro avg	1.000	1.000	1.000	45
weighted avg	1.000	1.000	1.000	45

Obtenemos una matriz de correlación para ver el número de bien y mal clasificados

In []:

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(clasificador, x_test_svd, y_test)
```

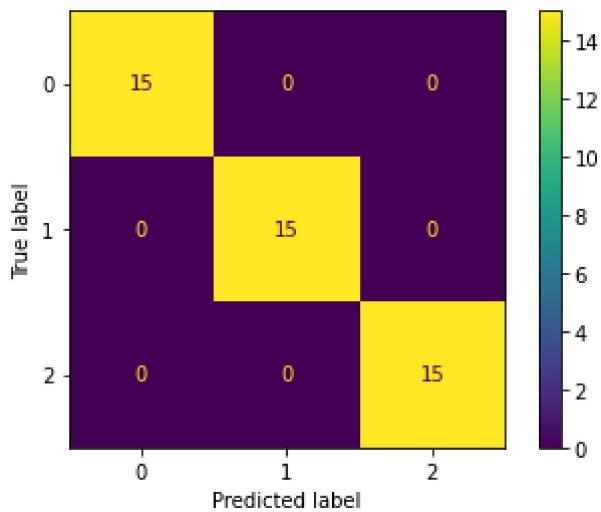
Precisión del clasificador: 1.0

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function `plot_confusion_matrix` is deprecated; Function ``plot_confusion_matrix`` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: `ConfusionMatrixDisplay.from_predictions` or `ConfusionMatrixDisplay.from_estimator`.

```
warnings.warn(msg, category=FutureWarning)
```

Out[]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f859b16b150>
```



Esta matriz de confusión nos muestra que todos los documentos han sido correctamente clasificados.

Esta aproximación por tanto, es la más eficiente.

.2.4. Modelo basado en redes neuronales con SVD Modelo_NN.ipynb

Clasificador basado en redes neuronales

1-Montamos y cargamos los datos

In []:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/drive/')
df_train = pd.read_csv('/content/drive/MyDrive/Ignieria_Linguistica/entrenamiento.csv')
df_test = pd.read_csv('/content/drive/MyDrive/Ignieria_Linguistica/test.csv')
```

Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/content/drive/", force_remount=True).

2-Extracción terminológica: vector TF-IDF

Una vez más vamos a crear la matriz TF-IDF sobre los datos de entrenamiento.

2.1-Creamos la matriz tf-idf a partir del 75% de los documentos y transformamos los 25% restantes.

In []:

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
x_train_tfidf = tfidf.fit_transform(df_train.iloc[:, 0])
x_test_tfidf = tfidf.transform(df_test.iloc[:, 0])

print(x_train_tfidf.shape, x_test_tfidf.shape)
```

(135, 14347) (45, 14347)

2.2-Reducimos la dimensionalidad de nuestras características de entrada para el clasificador mediante SVD.

SVD o Single Value Decomposition es una técnica de reducción de dimensión para matrices. Además este método funciona mejor con datos dispersos, es decir, datos con muchos valores a cero. Este es justo el caso en el que nos encontramos, se puede ver en la matriz de la celda anterior que la mayoría de valores de las palabras es 0, esto es debido a que muchas de ellas no aparecen en gran cantidad de documentos de nuestro corpus, por lo que al hacer $TF \cdot IDF$ siendo IDF 0, se obtienen gran cantidad.

In []:

```
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=100)
x_train_svd = svd.fit_transform(x_train_tfidf)
x_test_svd = svd.transform(x_test_tfidf)

print(x_train_svd.shape, x_test_svd.shape)
```

(135, 100) (45, 100)

2.3-Obtenemos y_train en formato One hot encoder, ya que las redes neuronales funcionan mejor en formatos de [0-1]

In []:

```
from sklearn.preprocessing import OneHotEncoder
onehotencoder = OneHotEncoder()
y_train_oh = onehotencoder.fit_transform(df_train.iloc[:,1].values.reshape(-1,1)).toarray()
```

3-Creación y entrenamiento del modelo

3.1-Estructura del modelo

In []:

```
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.utils import plot_model
from keras.layers.core import Activation

def build_model(x_train=x_train_svd, y_train=y_train_oh):
    model = Sequential()
    model.add(Dense(256, input_dim=x_train.shape[1], activation='relu', name='Densa_1'))
    model.add(Dropout(0.3, name='Dropout_1'))
    model.add(Dense(200, activation='relu', name='Densa_2'))
    model.add(Dropout(0.3, name='Dropout_2'))
    model.add(Dense(160, activation='relu', name='Densa_3'))
    model.add(Dropout(0.3, name='Dropout_3'))
    model.add(Dense(120, activation='relu', name='Densa_4'))
    model.add(Dropout(0.3, name='Dropout_4'))
    model.add(Dense(80, activation='relu', name='Densa_5'))
    model.add(Dropout(0.3, name='Dropout_5'))
    model.add(Dense(y_train_oh.shape[1], activation='softmax', name='Salida_Densa'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    model.summary()
    return model

#model = build_model()
#plot_model(model, to_file='/content/drive/MyDrive/Ignieria_Lingustica/estructura_modelo.png', show_shapes=True)
```

3.2-Creación y entrenamiento del modelo con las mismas condiciones que en SVM, 75% de datos de entrenamiento y 25% de test, no tenemos datos de desarrollo.

In []:

```
clf = build_model()
history = clf.fit(x_train_svd, y_train_oh, epochs=150, batch_size=1, verbose=1)
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
<hr/>		
Densa_1 (Dense)	(None, 256)	25856
Dropout_1 (Dropout)	(None, 256)	0
Densa_2 (Dense)	(None, 200)	51400
Dropout_2 (Dropout)	(None, 200)	0
Densa_3 (Dense)	(None, 160)	32160
Dropout_3 (Dropout)	(None, 160)	0
Densa_4 (Dense)	(None, 120)	19320
Dropout_4 (Dropout)	(None, 120)	0
Densa_5 (Dense)	(None, 80)	9680
Dropout_5 (Dropout)	(None, 80)	0
Salida_Densa (Dense)	(None, 3)	243
<hr/>		

Total params: 138,659

Trainable params: 138,659

Non-trainable params: 0

Epoch 1/150

135/135 [=====] - 1s 3ms/step - loss: 1.1119 - accuracy: 0.3111

Epoch 2/150

135/135 [=====] - 0s 3ms/step - loss: 1.0755 - accuracy: 0.3852

Epoch 3/150

135/135 [=====] - 0s 3ms/step - loss: 0.7252 - accuracy: 0.6667

Epoch 4/150

135/135 [=====] - 0s 3ms/step - loss: 0.1539 - accuracy: 0.9481

Epoch 5/150

135/135 [=====] - 0s 3ms/step - loss: 0.0378 - accuracy: 0.9926

Epoch 6/150

135/135 [=====] - 0s 3ms/step - loss: 0.0195 - accuracy: 0.9926

Epoch 7/150

135/135 [=====] - 0s 3ms/step - loss: 0.0312 - accuracy: 0.9778

Epoch 8/150

135/135 [=====] - 0s 3ms/step - loss: 0.0096 - accuracy: 1.0000

Epoch 9/150

135/135 [=====] - 0s 3ms/step - loss: 0.0461 - accuracy: 0.9778

Epoch 10/150

135/135 [=====] - 0s 3ms/step - loss: 0.1076 - accuracy: 0.9778

Epoch 11/150
135/135 [=====] - 0s 3ms/step - loss: 0.0353 - accuracy: 0.9852
Epoch 12/150
135/135 [=====] - 0s 3ms/step - loss: 0.0031 - accuracy: 1.0000
Epoch 13/150
135/135 [=====] - 0s 3ms/step - loss: 0.0028 - accuracy: 1.0000
Epoch 14/150
135/135 [=====] - 0s 3ms/step - loss: 3.9235e-04 - accuracy: 1.0000
Epoch 15/150
135/135 [=====] - 0s 3ms/step - loss: 3.6544e-04 - accuracy: 1.0000
Epoch 16/150
135/135 [=====] - 0s 3ms/step - loss: 2.0907e-04 - accuracy: 1.0000
Epoch 17/150
135/135 [=====] - 0s 3ms/step - loss: 2.2530e-04 - accuracy: 1.0000
Epoch 18/150
135/135 [=====] - 0s 3ms/step - loss: 3.9000e-04 - accuracy: 1.0000
Epoch 19/150
135/135 [=====] - 0s 3ms/step - loss: 0.0063 - accuracy: 0.9926
Epoch 20/150
135/135 [=====] - 0s 3ms/step - loss: 5.0860e-04 - accuracy: 1.0000
Epoch 21/150
135/135 [=====] - 0s 3ms/step - loss: 4.5177e-05 - accuracy: 1.0000
Epoch 22/150
135/135 [=====] - 0s 3ms/step - loss: 4.1110e-06 - accuracy: 1.0000
Epoch 23/150
135/135 [=====] - 0s 3ms/step - loss: 4.7160e-04 - accuracy: 1.0000
Epoch 24/150
135/135 [=====] - 0s 3ms/step - loss: 2.4180e-05 - accuracy: 1.0000
Epoch 25/150
135/135 [=====] - 0s 3ms/step - loss: 1.1140e-05 - accuracy: 1.0000
Epoch 26/150
135/135 [=====] - 0s 3ms/step - loss: 3.6664e-05 - accuracy: 1.0000
Epoch 27/150
135/135 [=====] - 0s 3ms/step - loss: 0.0492 - accuracy: 0.9852
Epoch 28/150
135/135 [=====] - 0s 3ms/step - loss: 8.7019e-05 - accuracy: 1.0000
Epoch 29/150
135/135 [=====] - 0s 3ms/step - loss: 4.1358e-05 - accuracy: 1.0000
Epoch 30/150
135/135 [=====] - 0s 3ms/step - loss: 1.0153e-04 - accuracy: 1.0000
Epoch 31/150

```
135/135 [=====] - 0s 3ms/step - loss: 1.6335e-05
- accuracy: 1.0000
Epoch 32/150
135/135 [=====] - 0s 3ms/step - loss: 9.1175e-05
- accuracy: 1.0000
Epoch 33/150
135/135 [=====] - 0s 3ms/step - loss: 3.3986e-05
- accuracy: 1.0000
Epoch 34/150
135/135 [=====] - 0s 3ms/step - loss: 0.0013 - accuracy: 1.0000
Epoch 35/150
135/135 [=====] - 0s 3ms/step - loss: 0.2366 - accuracy: 0.9630
Epoch 36/150
135/135 [=====] - 0s 3ms/step - loss: 0.0026 - accuracy: 1.0000
Epoch 37/150
135/135 [=====] - 0s 3ms/step - loss: 0.0131 - accuracy: 0.9926
Epoch 38/150
135/135 [=====] - 0s 3ms/step - loss: 0.0081 - accuracy: 1.0000
Epoch 39/150
135/135 [=====] - 0s 3ms/step - loss: 1.3519e-05
- accuracy: 1.0000
Epoch 40/150
135/135 [=====] - 0s 3ms/step - loss: 2.4385e-05
- accuracy: 1.0000
Epoch 41/150
135/135 [=====] - 0s 3ms/step - loss: 0.0608 - accuracy: 0.9852
Epoch 42/150
135/135 [=====] - 0s 3ms/step - loss: 0.0566 - accuracy: 0.9852
Epoch 43/150
135/135 [=====] - 0s 3ms/step - loss: 0.1790 - accuracy: 0.9926
Epoch 44/150
135/135 [=====] - 0s 3ms/step - loss: 0.0016 - accuracy: 1.0000
Epoch 45/150
135/135 [=====] - 0s 3ms/step - loss: 1.0736e-04
- accuracy: 1.0000
Epoch 46/150
135/135 [=====] - 0s 3ms/step - loss: 1.6847e-05
- accuracy: 1.0000
Epoch 47/150
135/135 [=====] - 0s 3ms/step - loss: 2.7215e-04
- accuracy: 1.0000
Epoch 48/150
135/135 [=====] - 0s 3ms/step - loss: 2.2969e-04
- accuracy: 1.0000
Epoch 49/150
135/135 [=====] - 0s 3ms/step - loss: 1.6111e-04
- accuracy: 1.0000
Epoch 50/150
135/135 [=====] - 0s 3ms/step - loss: 4.9509e-06
- accuracy: 1.0000
Epoch 51/150
135/135 [=====] - 0s 3ms/step - loss: 0.0016 - accuracy: 1.0000
```

```
curacy: 1.0000
Epoch 52/150
135/135 [=====] - 0s 3ms/step - loss: 4.1965e-06
- accuracy: 1.0000
Epoch 53/150
135/135 [=====] - 0s 3ms/step - loss: 4.7735e-05
- accuracy: 1.0000
Epoch 54/150
135/135 [=====] - 0s 3ms/step - loss: 5.3740e-04
- accuracy: 1.0000
Epoch 55/150
135/135 [=====] - 0s 3ms/step - loss: 4.6889e-07
- accuracy: 1.0000
Epoch 56/150
135/135 [=====] - 0s 3ms/step - loss: 0.0202 - ac
curacy: 0.9852
Epoch 57/150
135/135 [=====] - 0s 3ms/step - loss: 6.9663e-04
- accuracy: 1.0000
Epoch 58/150
135/135 [=====] - 0s 3ms/step - loss: 2.2013e-06
- accuracy: 1.0000
Epoch 59/150
135/135 [=====] - 0s 3ms/step - loss: 5.1430e-06
- accuracy: 1.0000
Epoch 60/150
135/135 [=====] - 0s 3ms/step - loss: 3.3043e-04
- accuracy: 1.0000
Epoch 61/150
135/135 [=====] - 0s 3ms/step - loss: 0.0045 - ac
curacy: 1.0000
Epoch 62/150
135/135 [=====] - 0s 3ms/step - loss: 0.1255 - ac
curacy: 0.9852
Epoch 63/150
135/135 [=====] - 0s 3ms/step - loss: 0.3670 - ac
curacy: 0.9556
Epoch 64/150
135/135 [=====] - 0s 3ms/step - loss: 5.8229e-04
- accuracy: 1.0000
Epoch 65/150
135/135 [=====] - 0s 3ms/step - loss: 0.0864 - ac
curacy: 0.9852
Epoch 66/150
135/135 [=====] - 0s 3ms/step - loss: 8.6836e-04
- accuracy: 1.0000
Epoch 67/150
135/135 [=====] - 0s 3ms/step - loss: 3.5764e-05
- accuracy: 1.0000
Epoch 68/150
135/135 [=====] - 0s 3ms/step - loss: 0.0017 - ac
curacy: 1.0000
Epoch 69/150
135/135 [=====] - 0s 3ms/step - loss: 2.2532e-04
- accuracy: 1.0000
Epoch 70/150
135/135 [=====] - 0s 3ms/step - loss: 6.6956e-05
- accuracy: 1.0000
Epoch 71/150
135/135 [=====] - 0s 3ms/step - loss: 1.7695e-05
- accuracy: 1.0000
```

Epoch 72/150
135/135 [=====] - 0s 3ms/step - loss: 0.0052 - accuracy: 1.0000
Epoch 73/150
135/135 [=====] - 0s 3ms/step - loss: 1.3441e-04 - accuracy: 1.0000
Epoch 74/150
135/135 [=====] - 0s 3ms/step - loss: 2.1837e-05 - accuracy: 1.0000
Epoch 75/150
135/135 [=====] - 0s 3ms/step - loss: 3.4443e-05 - accuracy: 1.0000
Epoch 76/150
135/135 [=====] - 0s 3ms/step - loss: 5.4422e-06 - accuracy: 1.0000
Epoch 77/150
135/135 [=====] - 0s 3ms/step - loss: 1.3857e-05 - accuracy: 1.0000
Epoch 78/150
135/135 [=====] - 0s 3ms/step - loss: 3.0511e-05 - accuracy: 1.0000
Epoch 79/150
135/135 [=====] - 0s 3ms/step - loss: 4.9096e-07 - accuracy: 1.0000
Epoch 80/150
135/135 [=====] - 0s 3ms/step - loss: 2.5861e-06 - accuracy: 1.0000
Epoch 81/150
135/135 [=====] - 0s 3ms/step - loss: 2.0701e-05 - accuracy: 1.0000
Epoch 82/150
135/135 [=====] - 0s 3ms/step - loss: 7.6505e-06 - accuracy: 1.0000
Epoch 83/150
135/135 [=====] - 0s 3ms/step - loss: 2.6932e-07 - accuracy: 1.0000
Epoch 84/150
135/135 [=====] - 0s 3ms/step - loss: 6.1281e-07 - accuracy: 1.0000
Epoch 85/150
135/135 [=====] - 0s 3ms/step - loss: 7.2562e-06 - accuracy: 1.0000
Epoch 86/150
135/135 [=====] - 0s 3ms/step - loss: 7.8381e-06 - accuracy: 1.0000
Epoch 87/150
135/135 [=====] - 0s 3ms/step - loss: 1.3863e-06 - accuracy: 1.0000
Epoch 88/150
135/135 [=====] - 0s 3ms/step - loss: 3.5856e-05 - accuracy: 1.0000
Epoch 89/150
135/135 [=====] - 0s 3ms/step - loss: 1.6882e-06 - accuracy: 1.0000
Epoch 90/150
135/135 [=====] - 0s 3ms/step - loss: 4.4151e-07 - accuracy: 1.0000
Epoch 91/150
135/135 [=====] - 0s 3ms/step - loss: 2.2607e-04 - accuracy: 1.0000
Epoch 92/150

```
135/135 [=====] - 0s 3ms/step - loss: 1.2114e-06
- accuracy: 1.0000
Epoch 93/150
135/135 [=====] - 0s 3ms/step - loss: 2.7352e-06
- accuracy: 1.0000
Epoch 94/150
135/135 [=====] - 0s 3ms/step - loss: 7.0553e-07
- accuracy: 1.0000
Epoch 95/150
135/135 [=====] - 0s 3ms/step - loss: 1.4675e-06
- accuracy: 1.0000
Epoch 96/150
135/135 [=====] - 0s 3ms/step - loss: 7.4130e-06
- accuracy: 1.0000
Epoch 97/150
135/135 [=====] - 0s 3ms/step - loss: 6.9109e-06
- accuracy: 1.0000
Epoch 98/150
135/135 [=====] - 0s 3ms/step - loss: 2.0398e-07
- accuracy: 1.0000
Epoch 99/150
135/135 [=====] - 0s 3ms/step - loss: 0.0660 - accuracy: 0.9926
Epoch 100/150
135/135 [=====] - 0s 3ms/step - loss: 3.1789e-08
- accuracy: 1.0000
Epoch 101/150
135/135 [=====] - 0s 3ms/step - loss: 2.6981e-06
- accuracy: 1.0000
Epoch 102/150
135/135 [=====] - 0s 3ms/step - loss: 4.6801e-08
- accuracy: 1.0000
Epoch 103/150
135/135 [=====] - 0s 3ms/step - loss: 5.3805e-05
- accuracy: 1.0000
Epoch 104/150
135/135 [=====] - 0s 3ms/step - loss: 8.1220e-04
- accuracy: 1.0000
Epoch 105/150
135/135 [=====] - 0s 3ms/step - loss: 1.7068e-06
- accuracy: 1.0000
Epoch 106/150
135/135 [=====] - 0s 3ms/step - loss: 1.2980e-06
- accuracy: 1.0000
Epoch 107/150
135/135 [=====] - 0s 3ms/step - loss: 1.6071e-07
- accuracy: 1.0000
Epoch 108/150
135/135 [=====] - 0s 3ms/step - loss: 1.9923e-04
- accuracy: 1.0000
Epoch 109/150
135/135 [=====] - 0s 3ms/step - loss: 1.9716e-06
- accuracy: 1.0000
Epoch 110/150
135/135 [=====] - 0s 3ms/step - loss: 2.6491e-09
- accuracy: 1.0000
Epoch 111/150
135/135 [=====] - 0s 3ms/step - loss: 4.4152e-09
- accuracy: 1.0000
Epoch 112/150
135/135 [=====] - 0s 3ms/step - loss: 6.7993e-08
```

```
- accuracy: 1.0000
Epoch 113/150
135/135 [=====] - 0s 3ms/step - loss: 1.2583e-06
- accuracy: 1.0000
Epoch 114/150
135/135 [=====] - 0s 3ms/step - loss: 0.0000e+00
- accuracy: 1.0000
Epoch 115/150
135/135 [=====] - 0s 3ms/step - loss: 1.5754e-05
- accuracy: 1.0000
Epoch 116/150
135/135 [=====] - 0s 3ms/step - loss: 8.8303e-09
- accuracy: 1.0000
Epoch 117/150
135/135 [=====] - 0s 3ms/step - loss: 2.3842e-08
- accuracy: 1.0000
Epoch 118/150
135/135 [=====] - 0s 3ms/step - loss: 3.2583e-07
- accuracy: 1.0000
Epoch 119/150
135/135 [=====] - 0s 3ms/step - loss: 1.7661e-09
- accuracy: 1.0000
Epoch 120/150
135/135 [=====] - 0s 3ms/step - loss: 5.2982e-09
- accuracy: 1.0000
Epoch 121/150
135/135 [=====] - 0s 3ms/step - loss: 8.8303e-10
- accuracy: 1.0000
Epoch 122/150
135/135 [=====] - 0s 3ms/step - loss: 1.0420e-07
- accuracy: 1.0000
Epoch 123/150
135/135 [=====] - 0s 3ms/step - loss: 1.7661e-09
- accuracy: 1.0000
Epoch 124/150
135/135 [=====] - 0s 3ms/step - loss: 2.6491e-09
- accuracy: 1.0000
Epoch 125/150
135/135 [=====] - 0s 3ms/step - loss: 8.8303e-10
- accuracy: 1.0000
Epoch 126/150
135/135 [=====] - 0s 3ms/step - loss: 7.9473e-09
- accuracy: 1.0000
Epoch 127/150
135/135 [=====] - 0s 3ms/step - loss: 4.2100e-06
- accuracy: 1.0000
Epoch 128/150
135/135 [=====] - 0s 3ms/step - loss: 2.9140e-08
- accuracy: 1.0000
Epoch 129/150
135/135 [=====] - 0s 3ms/step - loss: 0.0000e+00
- accuracy: 1.0000
Epoch 130/150
135/135 [=====] - 0s 3ms/step - loss: 7.0643e-09
- accuracy: 1.0000
Epoch 131/150
135/135 [=====] - 0s 3ms/step - loss: 8.4503e-07
- accuracy: 1.0000
Epoch 132/150
135/135 [=====] - 0s 3ms/step - loss: 2.4725e-08
- accuracy: 1.0000
```

```
Epoch 133/150
135/135 [=====] - 0s 3ms/step - loss: 2.5608e-08
- accuracy: 1.0000
Epoch 134/150
135/135 [=====] - 0s 3ms/step - loss: 1.7661e-09
- accuracy: 1.0000
Epoch 135/150
135/135 [=====] - 0s 3ms/step - loss: 1.5629e-07
- accuracy: 1.0000
Epoch 136/150
135/135 [=====] - 0s 3ms/step - loss: 7.9598e-06
- accuracy: 1.0000
Epoch 137/150
135/135 [=====] - 0s 3ms/step - loss: 0.0000e+00
- accuracy: 1.0000
Epoch 138/150
135/135 [=====] - 0s 3ms/step - loss: 1.2362e-08
- accuracy: 1.0000
Epoch 139/150
135/135 [=====] - 0s 3ms/step - loss: 0.0000e+00
- accuracy: 1.0000
Epoch 140/150
135/135 [=====] - 0s 3ms/step - loss: 4.4152e-09
- accuracy: 1.0000
Epoch 141/150
135/135 [=====] - 0s 3ms/step - loss: 2.2959e-08
- accuracy: 1.0000
Epoch 142/150
135/135 [=====] - 0s 3ms/step - loss: 6.6227e-08
- accuracy: 1.0000
Epoch 143/150
135/135 [=====] - 0s 3ms/step - loss: 0.0000e+00
- accuracy: 1.0000
Epoch 144/150
135/135 [=====] - 0s 3ms/step - loss: 1.7661e-09
- accuracy: 1.0000
Epoch 145/150
135/135 [=====] - 0s 3ms/step - loss: 0.0000e+00
- accuracy: 1.0000
Epoch 146/150
135/135 [=====] - 0s 3ms/step - loss: 6.1812e-09
- accuracy: 1.0000
Epoch 147/150
135/135 [=====] - 0s 3ms/step - loss: 0.0000e+00
- accuracy: 1.0000
Epoch 148/150
135/135 [=====] - 0s 3ms/step - loss: 2.3433e-06
- accuracy: 1.0000
Epoch 149/150
135/135 [=====] - 0s 3ms/step - loss: 0.0000e+00
- accuracy: 1.0000
Epoch 150/150
135/135 [=====] - 0s 3ms/step - loss: 1.0950e-07
- accuracy: 1.0000
```

3.3-Guardamos el modelo para no tener que volver a entrenarlo

In []:

```
clf.save('/content/drive/MyDrive/Ignieria_Linguistica/modelo_NN/modelo.h5')
```

3.3.1-En caso de querer cargar el modelo:

In []:

```
from keras.models import load_model  
clf = load_model('/content/drive/MyDrive/Ignieria_Linguistica/modelo_NN/modelo.h5')
```

3.4-Predicción probabilística, la guardamos en un excel

In []:

```
predicciones_prob = clf.predict(x_test_svd)
predicciones_rounded = [np.round(x,2) for x in predicciones_prob]
print(len(predicciones_rounded))
#predicciones_rounded
df_predicciones = pd.DataFrame(predicciones_rounded)
df_predicciones.columns=['Salud', 'Politica', 'Deportes']
df_predicciones.index.name = 'Documento'
df_predicciones.to_excel('/content/drive/MyDrive/Ignieria_Linguistica/modelo_NN/predicciones.xlsx')
predicciones_rounded
```

45

Out[]:

4-Evaluación del modelo ya entrenado

4.1-Generamos un informe de la precisión, recall, f1-score y support del modelo sobre los datos de test. Lo guardamos en un excel

In []:

```

predicciones = clf.predict(x_test_svd)
predicciones = [np.argmax(x) for x in predicciones]

from sklearn.metrics import classification_report
target_names = ['Salud', 'Politica', 'Deportes']
informe = classification_report(y_test, predicciones, target_names=target_names, digits=3)
print(classification_report(y_test, predicciones, target_names=target_names, digits=3))

informe = classification_report(y_test, predicciones, target_names=target_names, digits=3, output_dict=True)
df_informe = pd.DataFrame(informe).transpose()
df_informe.to_excel('/content/drive/MyDrive/Ignieria_Lingustica/modelo_NN/informe.xls', index=True)

```

	precision	recall	f1-score	support
Salud	1.000	1.000	1.000	15
Politica	1.000	0.933	0.966	15
Deportes	0.938	1.000	0.968	15
accuracy			0.978	45
macro avg	0.979	0.978	0.978	45
weighted avg	0.979	0.978	0.978	45

4.2-Generamos una matriz de confusión sobre los datos de test

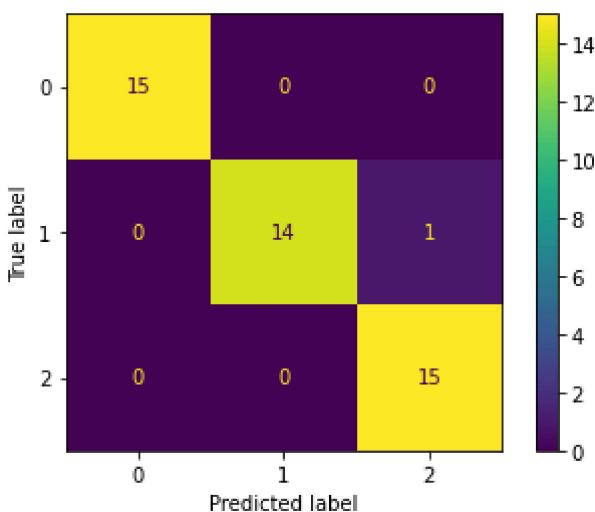
In []:

```

import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score
predicciones = clf.predict(x_test_svd)
predicciones = [np.argmax(x) for x in predicciones]
ConfusionMatrixDisplay.from_predictions(y_test, predicciones)
print("Precisión de las clasificaciones:", round(accuracy_score(y_test, predicciones),3))

```

Precisión de las clasificaciones: 0.978



Esta matriz de confusión nos muestra que:

1. De los **15** documentos de **salud**, **15** han sido correctamente clasificado.
2. De los **15** documentos de **política**, **14** han sido correctamente clasificado.
3. De los **15** documentos de **deportes**, **15** han sido correctamente clasificado.

Como conclusión podemos extraer que esta aproximación mejora ligeramente la aproximación con glasorías con supervisación.

.2.5. Modelo Naïve Bayes - Modelo_NB.ipynb

Clasificador basado en Naive Bayes con glosario supervisado

1-Montamos y cargamos los datos

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/drive/')

df_train = pd.read_csv('/content/drive/MyDrive/Ignieria_Linguistica/entrenamiento.csv')
df_test = pd.read_csv('/content/drive/MyDrive/Ignieria_Linguistica/test.csv')
```

Mounted at /content/drive/

2-Extracción terminológica: vector TF-IDF

Dado que el modelo NB no es capaz de interpretar números negativos, podemos hacer dos posibles aproximaciones:

1. Usar SVD y después aplicar una normalización de datos Max-Min scaler.
2. Usar el glosario revisado en la aproximación SVD con glosario y obtener la matriz TFIDF para ese vocabulario como entrada para el modelo.

Dado que tras cierta lectura, se ha leído que la transformación Min-Max scaler reduce en alrededor de un 20% la precisión del modelo, llevaremos a cabo la segunda aproximación.

2.1-Obtenemos la matriz TFIDF del conjunto de entrenamiento

In [3]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
x_train_tfidf = tfidf.fit_transform(df_train.iloc[:, 0])
x_test_tfidf = tfidf.transform(df_test.iloc[:, 0])

y_train = df_train.iloc[:, 1]
y_test = df_test.iloc[:, 1]
print(x_train_tfidf.shape, x_test_tfidf.shape)
```

(135, 14347) (45, 14347)

2.2-Nos quedamos únicamente con matriz TFIDF asociada al glosario supervisado.

In [4]:

```
glosario = pd.read_csv('/content/drive/MyDrive/Ignieria_Linguistica/glosario.csv')
vocabulario_glosario = glosario['palabra'].tolist()
indices_palabras_glosario = [tfidf.vocabulary_[key] for key in vocabulario_glosario]

x_train = x_train_tfidf[:, indices_palabras_glosario]
x_train = x_train.todense()

x_test = x_test_tfidf[:, indices_palabras_glosario]
x_test = x_test.todense()
```

3-Creamos un clasificador Naive Bayes

Estos modelos son llamados algoritmos “Naive”, o “Inocentes” en español. En ellos se asume que las variables predictoras son independientes entre sí. En otras palabras, que la presencia de una cierta característica en un conjunto de datos no está en absoluto relacionada con la presencia de cualquier otra característica. Proporcionan una manera fácil de construir modelos con un comportamiento muy bueno debido a su simplicidad.

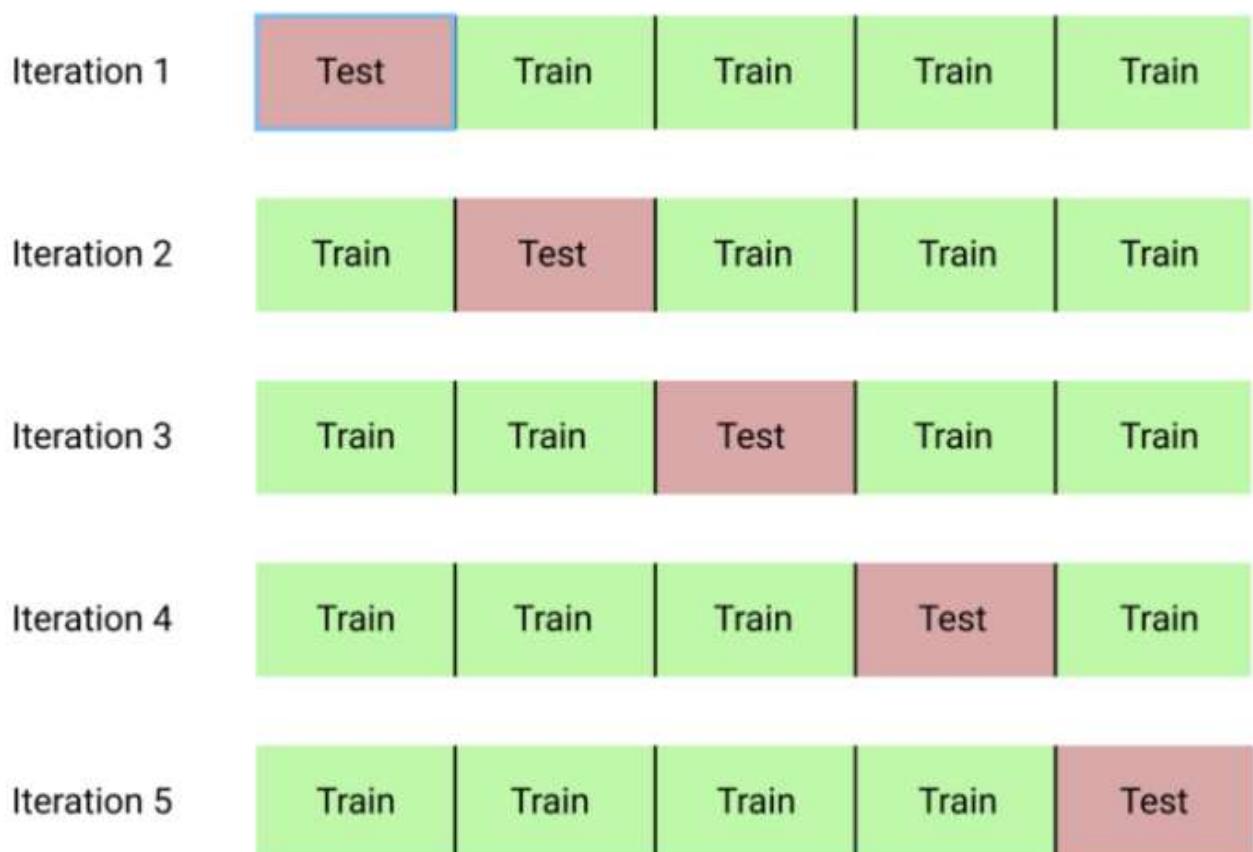
In [5]:

```
from sklearn.naive_bayes import MultinomialNB
naive_bayes = MultinomialNB()
#naive_bayes.fit(x_train, y_train)
#predicciones = naive_bayes.predict(X_test_cv)
```

3.1-Obtenemos una aproximación de la bondad del mismo mediante repeatedKfold.

Con el objetivo de tener una **estimación más realista de las métricas o bondad del modelo**, y más en este caso que tratamos con pocos datos, es muy conveniente llevar a cabo una **validación cruzada**. La validación cruzada k-fold consiste en dividir el conjunto de datos en k divisiones no superpuestas. De esta manera se crean K modelos, cada uno de ellos con diferentes datos de entrenamiento y de prueba y se obtiene el resultado medio de estos modelos. n_splits = Número de particiones para nuestro conjunto de datos. n_repeats = Número de validaciones cruzadas a repetir. Por tanto, no es lo mismo n_splits = 10 y n_repeats = 1 que n_splits = 5 y n_repeats = 2. En el primer caso, estaríamos obteniendo 10 divisiones de datos, pero dicha división aleatoria se realizaría una vez. En el segundo caso se obtendrían 10 divisiones de datos nuevamente, pero esos datos se dividirían aleatoriamente 2 veces.

k-Fold Cross Validation:



K=4 y n_repeat = 1

[1,2,3], [4,5,6], [7,8,9], [10,11,12]

k=2 y n_repeat=2

[1,2,3,4,5,6], [7,8,9,10,11,12]

[7,8,9,1,2,3], [4,5,6,10,11,12]

In [6]:

```
from sklearn.model_selection import cross_validate
from sklearn.model_selection import RepeatedKFold
from numpy import mean
from numpy import std

cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
metrics = cross_validate(naive_bayes, x_train, y_train, scoring=['precision_macro', 'recall_macro'], cv=cv, n_jobs=-1)

print('Precision: ', str(round((mean(metrics["test_precision_macro"])),3)), '| Desviación típica: ', str( round( std(metrics["test_precision_macro"])), 3)))
```

Precision: 0.917 | Desviación típica: 0.088

Como podemos observar, los resultados parecen buenos, por lo que vamos a entrenar el modelo.

4-Evaluación del clasificador NB sobre los datos de test

In [7]:

```
naive_bayes.fit(x_train, y_train)
predicciones = naive_bayes.predict(x_test)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:590: FutureWarning: np.matrix usage is deprecated in 1.0 and will raise a TypeError or in 1.2. Please convert to a numpy array with np.asarray. For more information see: https://numpy.org/doc/stable/reference/generated/numpy.matrix.html
```

```
    FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:590: FutureWarning: np.matrix usage is deprecated in 1.0 and will raise a TypeError or in 1.2. Please convert to a numpy array with np.asarray. For more information see: https://numpy.org/doc/stable/reference/generated/numpy.matrix.html
```

```
    FutureWarning,
```

4.1-Predicciones probabilísticas

In [9]:

```
predicciones_prob = naive_bayes.predict_proba(x_test)
predicciones_rounded = [np.round(x,2) for x in predicciones_prob]

df_predicciones = pd.DataFrame(predicciones_rounded)
df_predicciones.columns=['Salud', 'Politica', 'Deportes']
df_predicciones.index.name = 'Documento'
df_predicciones.to_excel('/content/drive/MyDrive/Ignieria_Lingustica/modelo_NB/predicciones.xlsx')
predicciones_rounded

predicciones_rounded
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:590: FutureWarning: np.matrix usage is deprecated in 1.0 and will raise a TypeError in 1.2. Please convert to a numpy array with np.asarray. For more information see: https://numpy.org/doc/stable/reference/generated/numpy.matrix.html
```

```
FutureWarning,
```

Out[9]:

```
[array([0.4 , 0.28, 0.33]),  
 array([0.49, 0.24, 0.27]),  
 array([0.45, 0.27, 0.28]),  
 array([0.38, 0.32, 0.3 ]),  
 array([0.37, 0.31, 0.31]),  
 array([0.5 , 0.24, 0.26]),  
 array([0.4 , 0.29, 0.3 ]),  
 array([0.51, 0.25, 0.24]),  
 array([0.46, 0.27, 0.27]),  
 array([0.51, 0.25, 0.24]),  
 array([0.48, 0.26, 0.26]),  
 array([0.49, 0.25, 0.26]),  
 array([0.41, 0.31, 0.28]),  
 array([0.4 , 0.3, 0.3]),  
 array([0.49, 0.25, 0.26]),  
 array([0.29, 0.41, 0.3 ]),  
 array([0.23, 0.52, 0.24]),  
 array([0.31, 0.39, 0.3 ]),  
 array([0.31, 0.36, 0.33]),  
 array([0.27, 0.45, 0.28]),  
 array([0.33, 0.34, 0.33]),  
 array([0.32, 0.37, 0.3 ]),  
 array([0.28, 0.43, 0.29]),  
 array([0.28, 0.44, 0.28]),  
 array([0.3 , 0.42, 0.28]),  
 array([0.33, 0.34, 0.33]),  
 array([0.28, 0.44, 0.29]),  
 array([0.32, 0.38, 0.31]),  
 array([0.28, 0.45, 0.27]),  
 array([0.29, 0.41, 0.3 ]),  
 array([0.25, 0.24, 0.51]),  
 array([0.26, 0.27, 0.47]),  
 array([0.29, 0.28, 0.44]),  
 array([0.28, 0.29, 0.43]),  
 array([0.27, 0.28, 0.45]),  
 array([0.27, 0.33, 0.4 ]),  
 array([0.28, 0.28, 0.44]),  
 array([0.31, 0.29, 0.4 ]),  
 array([0.27, 0.27, 0.46]),  
 array([0.21, 0.22, 0.57]),  
 array([0.31, 0.31, 0.38]),  
 array([0.28, 0.3 , 0.42]),  
 array([0.28, 0.28, 0.44]),  
 array([0.28, 0.28, 0.44]),  
 array([0.25, 0.26, 0.5 ])]
```

4.2-Generamos un informe de la precisión, recall, f1-score y support del modelo sobre los datos de test. Lo guardamos en un excel

In []:

```
predicciones = naive_bayes.predict(x_test)

from sklearn.metrics import classification_report
target_names = ['Salud', 'Politica', 'Deportes']
informe = classification_report(y_test, predicciones, target_names=target_names, digits=3)
print(informe)

informe = classification_report(y_test, predicciones, target_names=target_names, digits=3, output_dict=True)
df_informe = pd.DataFrame(informe).transpose()
df_informe.to_excel('/content/drive/MyDrive/Ignieria_Lingustica/modelo_NB/informe.xlsx', index=True)
```

/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:590: FutureWarning: np.matrix usage is deprecated in 1.0 and will raise a TypeError in 1.2. Please convert to a numpy array with np.asarray. For more information see: <https://numpy.org/doc/stable/reference/generated/numpy.matrix.html>

FutureWarning,

	precision	recall	f1-score	support
Salud	1.000	1.000	1.000	15
Politica	1.000	1.000	1.000	15
Deportes	1.000	1.000	1.000	15
accuracy			1.000	45
macro avg	1.000	1.000	1.000	45
weighted avg	1.000	1.000	1.000	45

Obtenemos una matriz de correlación para ver el número de bien y mal clasificados

In []:

```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(naive_bayes, x_test, y_test)
```

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function `plot_confusion_matrix` is deprecated; Function ``plot_confusion_matrix`` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: `ConfusionMatrixDisplay.from_predictions` or `ConfusionMatrixDisplay.from_estimator`.

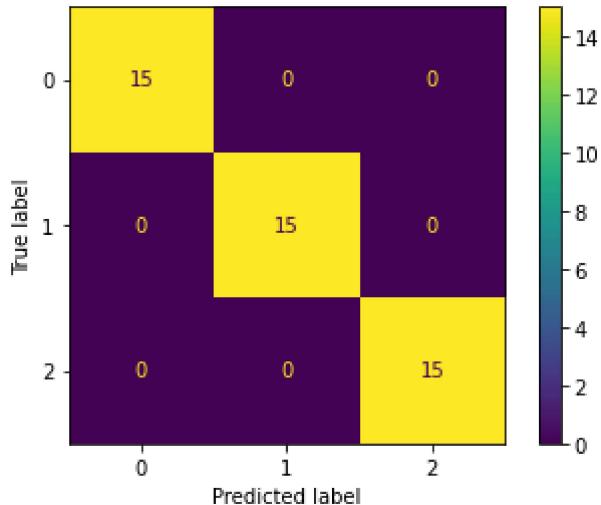
 warnings.warn(msg, category=FutureWarning)

/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:590: FutureWarning: `np.matrix` usage is deprecated in 1.0 and will raise a TypeError in 1.2. Please convert to a numpy array with `np.asarray`. For more information see: <https://numpy.org/doc/stable/reference/generated/numpy.matrix.html>

 FutureWarning,

Out[]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f3252911950>
```



Esta matriz de confusión nos muestra que todos los documentos han sido correctamente clasificados.

Esta aproximación por tanto, es la más eficiente junto con SVM con SVD.

.2.6. Modelo Random Forest - Modelo_NB.ipynb

Clasificador basado en Random Forest con simplificación SVD

1-Montamos y cargamos los datos

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/drive/')

df_train = pd.read_csv('/content/drive/MyDrive/Ignieria_Lingustica/entrenamiento.csv')
df_test = pd.read_csv('/content/drive/MyDrive/Ignieria_Lingustica/test.csv')
```

Mounted at /content/drive/

2-Extracción terminológica: vector TF-IDF

Una vez más vamos a crear la matriz TF-IDF sobre los datos de entrenamiento.

2.1-Obtenemos la matriz TFIDF del conjunto de entrenamiento

In [2]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
x_train_tfidf = tfidf.fit_transform(df_train.iloc[:, 0])
x_test_tfidf = tfidf.transform(df_test.iloc[:, 0])

y_train = df_train.iloc[:, 1]
y_test = df_test.iloc[:, 1]
print(x_train_tfidf.shape, x_test_tfidf.shape)
```

(135, 14347) (45, 14347)

2.2-Reducimos la dimensionalidad de nuestras características de entrada para el clasificador mediante SVD.

SVD o Single Value Decomposition es una técnica de reducción de dimensión para matrices. Además este método funciona mejor con datos dispersos, es decir, datos con muchos valores a cero. Este es justo el caso en el que nos encontramos. Esto es debido a que muchas de las palabras no aparecen en gran cantidad de documentos de nuestro corpus, por lo que al hacer $TF \cdot IDF$ siendo IDF 0, se obtienen gran cantidad.

In [3]:

```
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=100)
x_train_svd = svd.fit_transform(x_train_tfidf)
x_test_svd = svd.transform(x_test_tfidf)
print(x_train_svd.shape, x_test_svd.shape)
```

(135, 100) (45, 100)

3-Creamos un clasificador Random Forest

Estos modelos son llamados algoritmos “Naive”, o “Inocentes” en español. En ellos se asume que las variables predictoras son independientes entre sí. En otras palabras, que la presencia de una cierta característica en un conjunto de datos no está en absoluto relacionada con la presencia de cualquier otra característica. Proporcionan una manera fácil de construir modelos con un comportamiento muy bueno debido a su simplicidad.

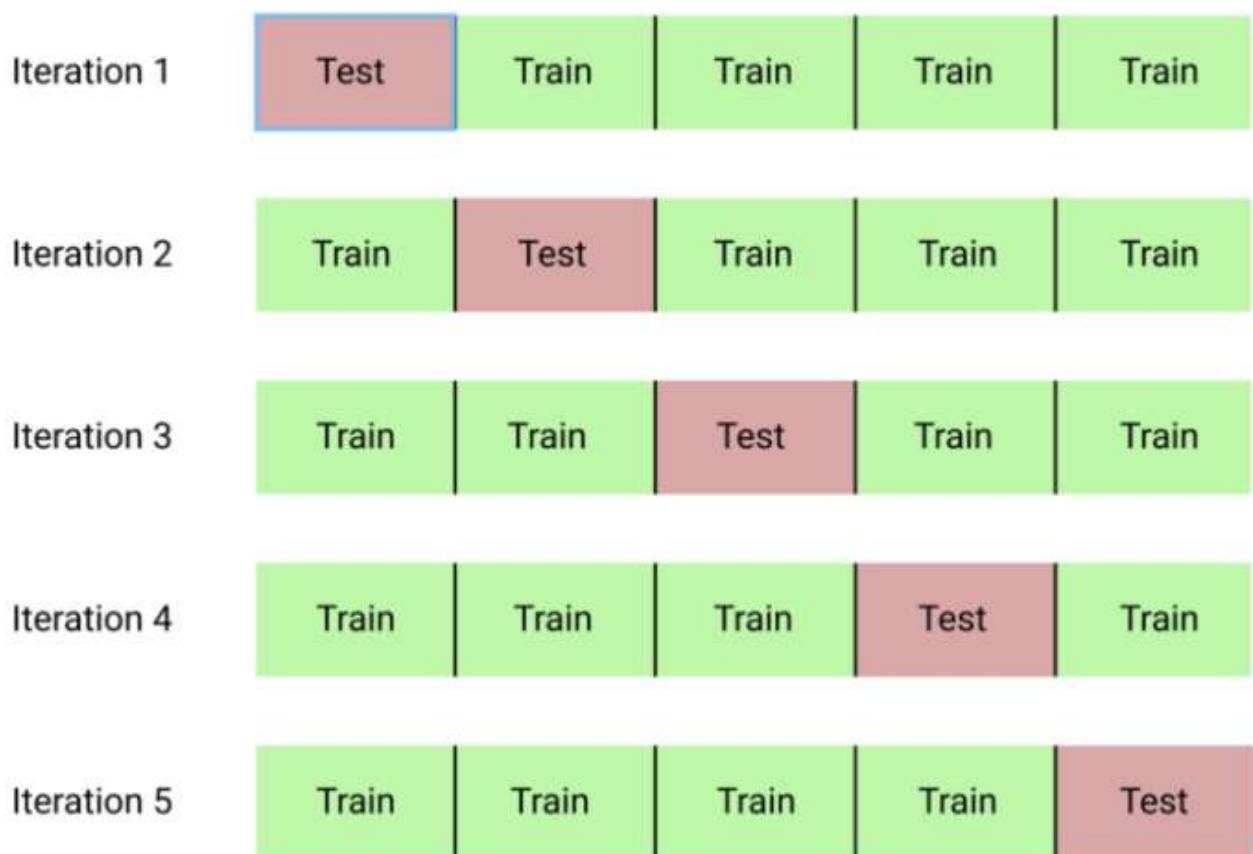
In [4]:

```
from sklearn.ensemble import RandomForestClassifier
random_forest = RandomForestClassifier(max_depth=10, random_state=0)
#naive_bayes.fit(x_train, y_train)
#predicciones = naive_bayes.predict(X_test_cv)
```

3.1-Obtenemos una aproximación de la bondad del mismo mediante repeatedKfold.

Con el objetivo de tener una **estimación más realista de las métricas o bondad del modelo**, y más en este caso que tratamos con pocos datos, es muy conveniente llevar a cabo una **validación cruzada**. La validación cruzada k-fold consiste en dividir el conjunto de datos en k divisiones no superpuestas. De esta manera se crean K modelos, cada uno de ellos con diferentes datos de entrenamiento y de prueba y se obtiene el resultado medio de estos modelos. n_splits = Número de particiones para nuestro conjunto de datos. n_repeats = Número de validaciones cruzadas a repetir. Por tanto, no es lo mismo n_splits = 10 y n_repeats = 1 que n_splits = 5 y n_repeats = 2. En el primer caso, estaríamos obteniendo 10 divisiones de datos, pero dicha división aleatoria se realizaría una vez. En el segundo caso se obtendrían 10 divisiones de datos nuevamente, pero esos datos se dividirían aleatoriamente 2 veces.

k-Fold Cross Validation:



K=4 y n_repeat = 1

[1,2,3], [4,5,6], [7,8,9], [10,11,12]

k=2 y n_repeat=2

[1,2,3,4,5,6], [7,8,9,10,11,12]

[7,8,9,1,2,3], [4,5,6,10,11,12]

In [6]:

```
from sklearn.model_selection import cross_validate
from sklearn.model_selection import RepeatedKFold
from numpy import mean
from numpy import std

cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
metrics = cross_validate(random_forest, x_train_svd, y_train, scoring=['precision_macro', 'recall_macro'], cv=cv, n_jobs=-1)

print('Precision: ', str(round((mean(metrics["test_precision_macro"])),3))), '| Desviación típica: ', str( round( std(metrics["test_precision_macro"]), 3)))
```

Precision: 0.947 | Desviación típica: 0.058

Como podemos observar, los resultados parecen buenos, por lo que vamos a entrenar el modelo.

In [9]:

```
random_forest.fit(x_train_svd, y_train)
```

4-Evaluación del clasificador RF sobre los datos de test

4.1-Predicciones probabilísticas

In [14]:

```
predicciones_prob = random_forest.predict_proba(x_test_svd)
predicciones_rounded = [np.round(x,2) for x in predicciones_prob]

df_predicciones = pd.DataFrame(predicciones_rounded)
df_predicciones.columns=['Salud', 'Politica', 'Deportes']
df_predicciones.index.name = 'Documento'
df_predicciones.to_excel('/content/drive/MyDrive/Ignieria_Lingustica/modelo_RF/predicciones.xlsx')
predicciones_rounded

predicciones_rounded
```

Out[14]:

```
[array([0.75, 0.11, 0.14]),  
 array([0.82, 0.08, 0.1]),  
 array([0.79, 0.08, 0.13]),  
 array([0.64, 0.23, 0.13]),  
 array([0.59, 0.14, 0.27]),  
 array([0.85, 0.07, 0.08]),  
 array([0.77, 0.07, 0.16]),  
 array([0.8 , 0.09, 0.11]),  
 array([0.81, 0.09, 0.1]),  
 array([0.8 , 0.09, 0.11]),  
 array([0.85, 0.06, 0.09]),  
 array([0.83, 0.1 , 0.07]),  
 array([0.86, 0.09, 0.05]),  
 array([0.77, 0.12, 0.11]),  
 array([0.75, 0.13, 0.12]),  
 array([0.18, 0.47, 0.35]),  
 array([0.11, 0.55, 0.34]),  
 array([0.19, 0.48, 0.33]),  
 array([0.12, 0.38, 0.5]),  
 array([0.16, 0.62, 0.22]),  
 array([0.15, 0.38, 0.47]),  
 array([0.22, 0.43, 0.35]),  
 array([0.18, 0.47, 0.35]),  
 array([0.12, 0.59, 0.29]),  
 array([0.15, 0.54, 0.31]),  
 array([0.27, 0.37, 0.36]),  
 array([0.18, 0.52, 0.3]),  
 array([0.19, 0.44, 0.37]),  
 array([0.13, 0.58, 0.29]),  
 array([0.15, 0.55, 0.3]),  
 array([0.03, 0.05, 0.92]),  
 array([0.05, 0.12, 0.83]),  
 array([0.01, 0.03, 0.96]),  
 array([0.11, 0.15, 0.74]),  
 array([0.09, 0.15, 0.76]),  
 array([0.06, 0.19, 0.75]),  
 array([0.12, 0.14, 0.74]),  
 array([0.06, 0.04, 0.9]),  
 array([0.04, 0.04, 0.92]),  
 array([0.01, 0.18, 0.81]),  
 array([0.1 , 0.12, 0.78]),  
 array([0.02, 0.08, 0.9]),  
 array([0.08, 0.13, 0.79]),  
 array([0.07, 0.14, 0.79]),  
 array([0.06, 0.11, 0.83])]
```

4.2-Generamos un informe de la precisión, recall, f1-score y support del modelo sobre los datos de test. Lo guardamos en un excel

In [16]:

```
predicciones = random_forest.predict(x_test_svd)

from sklearn.metrics import classification_report
target_names = ['Salud', 'Politica', 'Deportes']
informe = classification_report(y_test, predicciones, target_names=target_names, digits=3)
print(informe)

informe = classification_report(y_test, predicciones, target_names=target_names, digits=3, output_dict=True)
df_informe = pd.DataFrame(informe).transpose()
df_informe.to_excel('/content/drive/MyDrive/Ignieria_Lingustica/modelo_RF/informe.xlsx', index=True)
```

	precision	recall	f1-score	support
Salud	1.000	1.000	1.000	15
Politica	1.000	0.867	0.929	15
Deportes	0.882	1.000	0.938	15
accuracy			0.956	45
macro avg	0.961	0.956	0.955	45
weighted avg	0.961	0.956	0.955	45

Obtenemos una matriz de correlación para ver el número de bien y mal clasificados

In [19]:

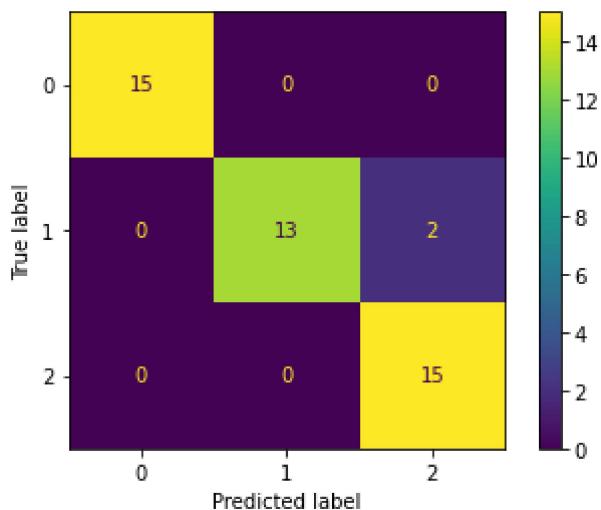
```
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(random_forest, x_test_svd, y_test)
```

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function `plot_confusion_matrix` is deprecated; Function ``plot_confusion_matrix`` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: `ConfusionMatrixDisplay.from_predictions` or `ConfusionMatrixDisplay.from_estimator`.

```
warnings.warn(msg, category=FutureWarning)
```

Out[19]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fdee7241e90>
```



Esta matriz de confusión nos muestra que:

1. De los **15** documentos de **salud**, **15** han sido correctamente clasificado.
2. De los **15** documentos de **política**, **13** han sido correctamente clasificado.
3. De los **15** documentos de **deportes**, **15** han sido correctamente clasificado.

Esta aproximación por tanto, igual de eficiente que SVM con glosario supervisado.