



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA

etsinf

Escola Tècnica  
Superior d'Enginyeria  
Informàtica

Escuela Técnica Superior de Ingeniería Informática  
Universitat Politècnica de València

Big data para codificar diagnósticos  
clínicos de manera automática.

Trabajo Fin de Máster

Máster Big Data Analytics

Autor: José Manuel Marín Noguera

Director: Dr. Francisco M. Rangel Pardo

Abril 2017

*Agradecimientos.*

*Auroritas Consulting por aportar una valiosa fuente de datos y a su alma Kico por su inestimable ayuda y ser mi director, a Jon Ander por su esfuerzo en organizar este máster, a Germán Moltó y al programa AWS Educate por su colaboración, a mi hijo Dani y a mi hijo Sergio por su paciencia, y a mi chica por su amor.*

# Resumen

---

Extraer valor de diagnósticos clínicos es un problema dentro del ámbito de la salud. Los textos de diagnóstico clínicos expresan un juicio sobre el estado psicofísico de una persona, identificar y clasificar la enfermedad subyacente es el hito prioritario para obtener información útil. Los documentalistas, la interoperabilidad entre sistemas de información, la medicina traslacional son los principales beneficiarios de los avances sobre este campo.

El cometido de este trabajo consiste en resolver como primer problema, con tecnología big data, la categorización y correspondencia de una clasificación internacional de enfermedades como CIE-9 con los textos diagnósticos.

El segundo problema a resolver se centra en el área del aprendizaje automático (ML), cuyo trabajo reside en encontrar maneras efectivas de combinar y ensamblar un conjunto de clasificadores para mejorar el resultado individual de cualquiera de ellos, siendo este un tema importante en los procesos de extracción de conocimiento de datos.

Como método a este trabajo se va a realizar un recorrido práctico completo de un proceso Data Science aplicando conceptos big data a los problemas anteriormente mencionados. En este camino nos encontramos con: decisiones a tomar y dificultades a resolver, la aplicación de técnicas ya conocidas y consolidadas, la investigación y aplicación de técnicas emergentes, y la aportación de nuevo conocimiento. Así, el trabajo está centrado fundamentalmente en el ¿cómo? sin olvidarse del ¿qué?

El punto de partida es un conjunto existente de clasificadores débiles de muy diversa naturaleza. En el primer paso, se profundiza en como calibrar un meta-clasificador que fusiona los clasificadores, apoyándose en la tecnología big data. Como segundo paso, se crean dos clasificadores para TextMining con técnicas emergentes como Low Dimensionality Representation (LDR) y Distributed Representations of Sentences and Documents (Doc2vec) y modelos Support Vector Machine (SVM), valorando sus resultados y su aportación. Previo a estos dos pasos, se ha planificado, decidido y provisto los recursos necesarios para este recorrido; Aquí es donde aparecen Spark, Amazon Web Service (AWS), Pyspark, Docker, Jupyter, Python, Scikit-learn, Pandas y Gensim.

Como conclusión aplicable está la técnica de calibrado de meta-clasificadores que mejora su propia efectividad, que conforme aumenta el tamaño de problema aplicar Spark (o similar) se hace totalmente necesaria. Además, con la técnica de LDR se obtiene resultados muy eficientes y que el Doc2Vec es una técnica prometedora. Por tanto, podemos concluir que disponemos de excelentes soluciones basadas en tecnología big data que resuelven la problemática de la clasificación de diagnósticos clínicos. Así pues, big data es una tecnología con amplio espectro y que puede ofrecer a la comunidad médica un gran valor en la toma de decisiones clínicas, asistenciales y de gestión.

**Palabras clave:** diagnóstico clínico, big data analytics, meta-clasificador, clasificador, doc2vec, LDR, CIE9, SVM, Spark, Amazon Web Service (AWS), Pyspark, Docker, Jupyter, Python, Scikit-learn, Pandas, Gensim, TextMinig, aprendizaje automático, supervisado, no supervisado, DeepLearning.

# Tabla de contenidos

---

Tabla ilustraciones .....	6
1. INTRODUCCION .....	8
1.1    Presentación del problema.....	8
1.1.1    Diagnóstico clínico.....	9
1.1.2    CIE-9 .....	9
1.1.3    El Caso de Uso .....	9
1.1.4    Clasificación de diagnóstico.....	10
1.2    Objetivos.....	11
1.3    Estructura.....	11
2 METODOLOGIA .....	12
2.1    Clasificación de textos .....	12
2.2    Punto de partida .....	14
2.3    Propuesta.....	14
2.4    TF-IDF .....	15
2.5    LDR.....	16
2.6    Doc2Vec.....	17
2.7    Otras aproximaciones.....	18
2.8    Aprendizaje automático.....	18
2.8.1    Algoritmos de clasificación.....	18
2.8.2    Combinación de clasificadores.....	20
2.8.3    Método de nivel de medidas.....	21
2.9    Big Data Analytics.....	22
2.9.1    Definición.....	22
2.9.2    Proceso KDD.....	22
2.9.3    Data Science .....	24
2.9.4    Conclusión .....	24
2.10    Recursos .....	25
2.10.1    Infraestructura .....	25
2.10.2    Software .....	26
2.10.3    Datos .....	26

<b>3</b>	<b>EXPERIMENTOS.....</b>	<b>27</b>
3.1	Recursos .....	27
3.1.1	Python .....	27
3.2	Local: IPython Notebook sobre Docker .....	28
3.2.1	Cluster: Pyspark + AWS Educate.....	28
3.3	Clasificadores Texto.....	30
3.3.1	Datos para codificar .....	32
3.3.2	TF-IDF. ....	37
3.3.3	Clasificador LDR .....	40
3.3.4	Doc2Vec.....	43
3.3.5	Resultados comparación de clasificadores .....	49
3.4	Propuesta de combinación de clasificadores .....	51
3.4.1	Datos .....	52
3.4.2	Local .Python .....	55
3.4.3	Tamaño de problema.....	58
3.4.4	Cluster Spark .....	61
3.4.5	Spark EMR AWS .....	66
<b>4</b>	<b>CONCLUSIONES.....</b>	<b>71</b>
4.1	Futuro.....	73
	Bibliografía .....	75

## Tabla ilustraciones.

Ilustración 1. Flujo de trabajo de clasificación.....	13
Ilustración 2. Representación del vector de un documento.....	13
Ilustración 3. Fundamentos TF-IDF .....	15
Ilustración 4. Planteamientos Doc2vec.....	17
Ilustración 5. Mapa de algoritmos de clasificación.....	19
Ilustración 6. Vector soporte con dos clases.....	19
Ilustración 7. Combinación de clasificadores.....	20
Ilustración 8. Fusión de múltiples clasificadores débiles.....	21
Ilustración 9. Reglas de fusión.....	21
Ilustración 10. Disciplinas del Data Science .....	24
Ilustración 11. Ecosistema Big Data .....	25
Ilustración 12. Recursos locales.....	28
Ilustración 13. Ecosistema Spark .....	29
Ilustración 14. Librerías empleadas.....	31
Ilustración 15. Selección clasificador .....	32
Ilustración 16. Carga datos + descripción.....	33
Ilustración 17. Histograma de diagnósticos.....	33
Ilustración 18. CIE.9 capítulos enfermedades.....	34
Ilustración 19. Separación en palabras .....	34
Ilustración 20. Palabras más usadas.....	35
Ilustración 21. Box-plot Equilibrio de clases ajustadas.....	35
Ilustración 22. Descripción conjunto ajustado.....	36
Ilustración 23. Función de tokenizado .....	37
Ilustración 24. Código Python clasificador TF-IDF.....	37
Ilustración 25. Código extendido clasificador TF-IDF .....	38
Ilustración 26. Funciones que implementan LDR.....	40
Ilustración 27. Solución clasificador LDR .....	41
Ilustración 28. Top 10 de probabilidades.....	42
Ilustración 29. Integración meta-clasificador.....	42
Ilustración 30. Incorporación de catálogos.....	43
Ilustración 31. Preparación Doc2vec .....	44
Ilustración 32. Descripción del corpus entrenamiento Doc2vec.....	44
Ilustración 33. Construcción espacio vectorial Doc2Vec .....	45
Ilustración 34. Ejemplo de palabras y documentos similares.....	46
Ilustración 35. Vectorización con Doc2vec.....	46
Ilustración 36. Solución clasificador Doc2vec .....	47
Ilustración 37. Top probabilidades Doc2vec.....	47
Ilustración 38. Evolución efecto tamaño del corpus en Doc2vec .....	48
Ilustración 39. Solución train_test_split con Doc2vec .....	48
Ilustración 40. Comparativa de resultados.....	49
Ilustración 41. Comparativa línea base Top de clasificadores.....	50
Ilustración 42. Comparativa Top tipo extendido.....	50
Ilustración 43. Resultados ponderación clasificadores original.....	52
Ilustración 44. Visión de datos para ponderación de clasificadores.....	52
Ilustración 45. Carga de predicciones de clasificadores.....	53

Ilustración 46. Fragmentos de código. Descripción del conjunto de datos de ponderación.....	53
Ilustración 47. Comparación de los histogramas real y predicción.....	54
Ilustración 48. Equilibrio entre clasificadores.....	55
Ilustración 49. Solución local fusión clasificadores.....	56
Ilustración 50. Comprobación de la validez del código de fusión. ....	57
Ilustración 51. Aprendizaje de pesos.....	57
Ilustración 52. Segundos en función de n° combinaciones de pesos .....	58
Ilustración 53. Duración prevista de la solución local.....	58
Ilustración 54. Ponderación fina.....	59
Ilustración 55. Accuray del espacio vectorial testeado. ....	60
Ilustración 56. Gestores Spark .....	61
Ilustración 57. Planificación tareas Spark .....	61
Ilustración 58. Definición del área de Contexto. ....	62
Ilustración 59. Función para pivotado. ....	62
Ilustración 60. Función calculo de combinaciones. ....	63
Ilustración 61. Join de clasificaciones y combinaciones. ....	63
Ilustración 62. Solución distribuida fusión de clasificadores. ....	64
Ilustración 63. Representación Grafica de tareas con gestor Standalone Spark.....	65
Ilustración 64. Creación de cluster en EMR de AWS en modo web.....	66
Ilustración 65. Cluster manager en Spark.....	67
Ilustración 66. Creación cluster Spark modo comando. ....	68
Ilustración 67. Trabajos Spark sobre EMR .....	69
Ilustración 68. Salida de la solución de calibrado de clasificadores en Spark.....	69
Ilustración 69. Solución fusión clasificadores en AWS. ....	70

## 1. INTRODUCCION.

En este capítulo se identifica el problema y se describe la estructura del documento.

### 1.1 Presentación del problema.

Dentro del área de salud, el cómo obtener valor de los "diagnósticos clínicos" es un tema ampliamente abordado tanto, en la sanidad asistencial, en salud pública y en las investigaciones clínicas. Como muestra de esa preocupación no hay más que fijarse en la variedad de clasificaciones de enfermedades, lesiones y demás entidades relacionadas con la salud, adoptadas y consensuadas por organismos nacionales e internacionales, como una forma de estandarizar. Como por ejemplo, CIE-9 Clasificación Internacional de Enfermedades 9<sup>a</sup> Revisión, la más extendida hoy en día en España y su evolución el CIE-10; o la propuesta actual, SNOMED-CT (Systematized Nomenclature of Medicine – Clinical Terms) terminología clínica integral multilingüe. Imaginemos pues un texto donde se expresa el diagnóstico clínico sobre una cierta patología o dolencia de un paciente, es un texto normalmente amplio, variado y con un lenguaje bastante técnico, dentro del cual se encuentra descrita la enfermedad o trastorno. Como encontrar dentro del texto la correspondencia de la enfermedad del diagnóstico principal con una clasificación como CIE-9, donde hay más de 9.000 posibilidades, no es tarea fácil, ni manualmente y ni mucho menos por medios automáticos.

Por todo lo anterior, la propuesta debe construir modelos (aprendizaje automático) que capturen el comportamiento (clasificación) de unos datos (textos diagnósticos clínicos) para extraer valor y aplicarlo (predicción). Aunque complejo y útil, el problema de clasificación de diagnósticos clínicos fue abordado ya por la empresa cedente de su experiencia (Autoritas Consulting <http://www.autoritas.es>), es esta experiencia el referente y punto de partida del estudio. Se tiene pues tres modelos de clasificación de diversa naturaleza, dos basados en la recuperación de información y otro que proviene del aprendizaje automático, el cómo combinarlos se convierte en un reto a enfrentarse para conseguir los objetivos propuestos. Averiguar cómo abordar el tamaño de problema es una cuestión importante a resolver a la hora de aplicar cualquier solución en un entorno big data.

Cuando se habla de clasificar textos y explorar nuevas técnicas, construir modelos propios se convierte en un instrumento que permite profundizar y obtener mejores soluciones. Además de obtener conocimiento aplicable en otros problemas similares.

Para poder llegar a una solución se necesita disponer de medios y recursos. Por un lado, hay que seleccionar y disponer una infraestructura hardware, que cuando se habla de big data es un punto especialmente complicado. Por otro lado, se debe seleccionar productos software dentro de ese bosque que rodea al aprendizaje automático y al big data. Es este punto lo suficientemente importante para ser una parte relevante del estudio.

### 1.1.1 Diagnóstico clínico.

"En medicina, el diagnóstico o propedéutica clínica es el procedimiento por el cual se identifica una enfermedad, entidad nosológica, síndrome, o cualquier estado de salud o enfermedad." [[https://es.wikipedia.org/wiki/Diagn%C3%B3stico\\_medico](https://es.wikipedia.org/wiki/Diagn%C3%B3stico_medico)]

### 1.1.2 CIE-9

"El CIE-9 es el acrónimo de la Clasificación Internacional de Enfermedades, novena edición, publicada en 1977 por la Organización Mundial de la Salud -OMS- y cuyo fin es clasificar las enfermedades, afecciones y causas externas de enfermedades y traumatismos, con objeto de recopilar información sanitaria útil relacionada con defunciones, enfermedades y traumatismos (mortalidad y morbilidad). Aunque la versión vigente es la décima edición, CIE-10, la CIE-9 se sigue usando en algunos países entre ellos España. El Centro nacional para la estadística de la salud de EEUU añadió una sección de códigos de los procedimientos que dio lugar a la versión denominada CIE-9-MC, MC corresponde a la descripción de Modificación Clínica". [<https://es.wikipedia.org/wiki/CIE-9>]

A través de la codificación se pretende indexar toda la información clínica que contienen las historias para facilitar su almacenamiento y su recuperación. La codificación se realiza solo en las altas hospitalarias y el codificador debe extraer de ellas los elementos sustanciales para luego clasificarlas siguiendo una clasificación homologada que asigna un código.

Se consideran los siguientes elementos sustanciales:

- Diagnóstico Principal
- Diagnóstico Secundario
- Procedimientos quirúrgicos
- Procedimientos no quirúrgicos
- Codificación de enfermedades

El propósito de la CIE es permitir el registro sistemático, el análisis, la interpretación y la comparación de los datos de mortalidad y morbilidad recolectados en diferentes países o áreas, y en diferentes momentos. La clasificación permite la conversión de los términos diagnósticos y de otros problemas de salud, de palabras a códigos alfanuméricos que facilitan su almacenamiento y posterior recuperación para el análisis de la información. CIE puede utilizarse para clasificar enfermedades y otros problemas de salud consignados en muchos tipos de registros vitales y de salud.

### 1.1.3 El Caso de Uso

El caso de uso explicativo del proceso de codificación se enmarca en un hospital de España y el reto es cumplir con el requisito que marca la OMS de codificar la enfermedad del diagnóstico principal en las altas hospitalarias. La situación actual es que es una tarea manual a cargo de los documentalistas y en muchos casos la tarea recae en personal administrativo no capacitado para ello. El objetivo es pues automatizar del modo más eficiente y eficaz posible. El resolver este caso concreto no debe impedir que se pueda extrapolar la solución propuesta a otros problemas similares.

#### 1.1.4 Clasificación de diagnóstico.

En el problema hay un conjunto de textos en lenguaje natural (corpus) que pertenecen a un cierto dominio (diagnósticos clínicos) y se pretende hacer una asignación automática a unas categorías (Enfermedades clasificación CIE-9). Se aborda este problema aplicando técnicas de Procesamiento del Lenguaje Natural (PLN) y de Aprendizaje Automático (en inglés Machine Learning, ML). Hay otros enfoques que trabajan en resolver este tipo de problemas: el uso de ontologías, búsqueda por reglas, uso de Grafos etc. En general no trabajaremos con ellos, aunque bajo ciertas condiciones se pueden integrar en la propuesta del mismo modo que se hace con los dos clasificadores basados en recuperación de la información proporcionados por Autoritas. El número de categorías a clasificar (más de 9.000) enmarca el tipo de técnica de PLN a utilizar, clasificación multi-etiquetas de textos.

El PLN aborda las problemáticas desde diversa perspectivas:

- Morfológica. Como se construyen las palabras.
- Sintaxis. Las estructuras relacionales de las palabras.
- Semántica. El significado de las palabras.
- Discurso. Como afecta el contexto al significado de las palabras.

El PLN se apoya en recursos léxicos y lingüísticos: diccionarios, redes semánticas, ontologías y corpus. Por ejemplo para medicina: \*GENIA, SNOMED-CT, MeSH, DeCS, UMLs, ICD. (La mayoría en inglés).

Principales paradigmas para resolución de problemas:

- Modelos Ocultos de Markov.
- Modelos de Máxima Entropía.
- Memory based Learning.
- Transformation based Learning.
- Métodos de clasificación.
- Métodos combinados.

Algunos referentes de técnicas PLN:

- Recuperación de información [Castells, Fernández y Vallet, 2007; Song et al. 2007],
- Clasificación de documentos [Yun, Guiyi y Jun, 2006; Weng, 2006]
- Extracción de información [Endres-Niggmeyer et al., 2006; Cimiano, Reyle y Saric, 2005].

\*GENIA [<http://www.nactem.ac.uk/meta-knowledge>]

\*MeSH [<https://www.ncbi.nlm.nih.gov/mesh/>]

\*SNOMED-CT [<http://www.snomed.org/snomed-ct>]

\*UMLs [<https://www.ncbi.nlm.nih.gov/research/umls/>]

\*DeCS [<http://decs.bvs.br/E/decsweb2016.htm>]

\*ICD [<https://www.cdc.gov/nchs/icd/index.htm>]

## 1.2 Objetivos.

El propósito general del trabajo es descubrir conocimiento sobre datos con tecnología Big Data Analytics. Partiendo de un conjunto de datos hacer un recorrido en la búsqueda de valor, aplicando los conocimientos dentro del campo de Big Data Analytics.

Como objetivos concretos definimos:

-Descubrir conocimiento en diagnósticos clínicos. Obtener los modelos que clasifiquen diagnósticos clínicos para ser aplicables en el beneficio de la mejora asistencial y por tanto de la salud de las personas.

-Descubrir conocimiento y uso de meta-clasificadores. Obtener una manera efectiva para ensamblar un conjunto de clasificadores.

-Descubrir conocimiento y uso de clasificadores de texto emergentes. Saber cómo aplicar nuevos métodos para la clasificación de textos y ver sus resultados.

-Descubrir conocimiento y uso de recursos necesarios. Saber el qué cómo, porqué, de los elementos hardware y software empleados.

La misión del trabajo es:

Construir dos clasificadores que junto a los tres proporcionados por Autoritas, se ensamblan y se calibran para obtener un meta-clasificador débil de CIE-9 para diagnósticos clínicos en español.

Los objetivos son amplios y ambiciosos. Los modelos y productos resultantes deben ser una propuesta que sirva como claro ejemplo de cómo se puede extraer valor de un conjunto de datos sobre diagnósticos clínicos.

## 1.3 Estructura

La memoria está estructurada en tres partes:

Una primera para definir y enmarcar el problema, capítulos 1 y 2. Una segunda parte central para exponer la propuesta objetivo del proyecto, capítulo 3. Y una parte final para presentar y aplicar conclusiones, capítulo 4. Los capítulos se corresponden con las fases que se aplican en el desarrollo del estudio:

1. Definición del problema. En esta fase se identifican, describen y relacionan, los elementos que conforman el problema. Y con los objetivos a perseguir clarificados y con la evaluación inicial realizada se establece un plan de acción.
2. Metodología. En esta fase se identifica el conocimiento aplicable para el tipo de problema presentado y necesario para desarrollar la propuesta. Se seleccionan las técnicas, los modelos de comportamiento y los criterios de evaluación, además de la infraestructura necesaria para llevar a cabo el estudio.

3. Experimentos. En esta fase central y con los recursos adecuados, se realizan los experimentos y se desarrollan las propuestas para solucionar los problemas planteados. Es donde se realizan las experiencias del proceso analítico de datos.
4. Conclusiones. En esta fase se identifica las conclusiones y productos aplicables, que forman parte de la propuesta de solución a los problemas identificados y que cumplen los objetivos perseguidos.

## 2 METODOLOGIA.

En este capítulo, se recopila conceptos previos y fundamentos necesarios para desarrollar la propuesta. Para conseguir los objetivos se emplean metodologías, técnicas, y herramientas relacionadas con estadística, matemática, análisis de datos, aprendizaje automático, ciencia de datos y visualización datos, todas ellas forman parte del Big Data Analytics.

### 2.1 Clasificación de textos.

Dentro de PLN y con el paradigma métodos de clasificación (estadísticos), podemos definir la clasificación de textos como:

Entrada:

- Un documento d
- Un conjunto fijo de clases  $C = \{c_1, c_2, \dots, c_j\}$
- Un conjunto de entrenamiento de m
- Documentos etiquetados en clases  $(d_1, c_1), \dots, (d_m, c_m)$

Salida:

- Un clasificador aprendido,  $\gamma: d \rightarrow c$

En la clasificación de documentos habitualmente siempre hay dos partes principales:

- Módulo que procesa los documentos y transforma los textos en representaciones más manejables, como por ejemplo espacios vectoriales de características.
- Módulo que mediante algoritmos de aprendizaje automático con entrenamiento y evaluación, obtienen los modelos de clasificación.

Las soluciones y técnicas a aplicar en la clasificación de textos son muy parecidas a las empleadas en el problema ampliamente tratado del análisis de sentimientos (SA), que sirve de referencia sobre todo en la parte de extracción de características.

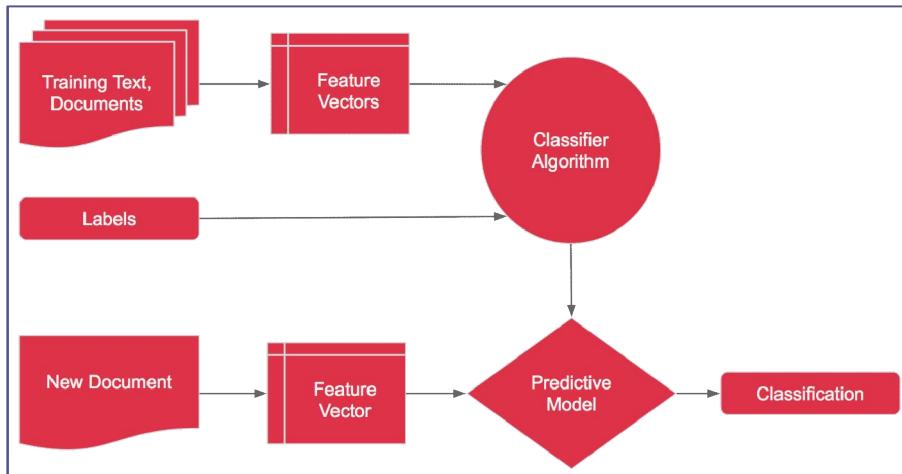


Ilustración 1. Flujo de trabajo de clasificación.

[Tony Ojeda, et al. (2017) Applied Text Analysis with Python ]

Dentro de la parte de extracción de características hay que segmentar el texto separándolo en las unidades de interés (tokenizar). Entonces seleccionamos características: palabras, n-gramas, etiquetas POS, bolsa de palabras, extracción de entidades, análisis morfológico y análisis sintáctico. Se puede también incorporar lógica de algún corpus sobre el dominio. Para finalizar en base a las características se cuantifica y se construyen los vectores: frecuencias relativas TF, frecuencia “binarias”, frecuencias inversas IDF están dentro de las más utilizadas. Una forma más común de transformar es la de asignar pesos a cada término de interés.

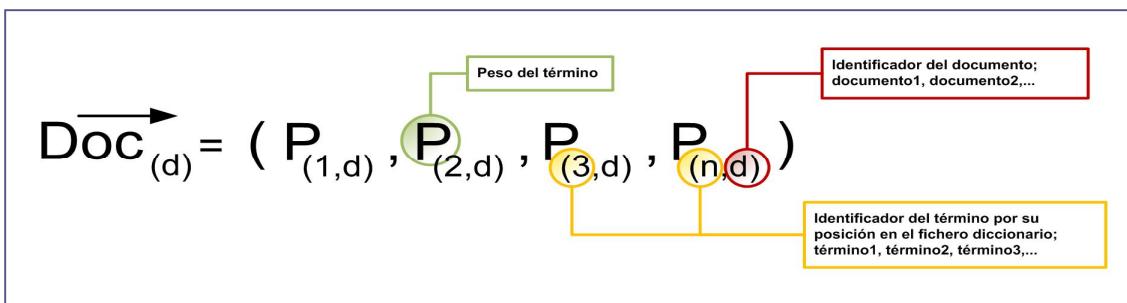


Ilustración 2. Representación del vector de un documento

[Manuel Blázquez Ochando <http://ccdoc-tecnicasrecuperacioninformacion.blogspot.com.es> ]

Transformamos, seleccionamos y relacionamos de muy diversas maneras, según el tipo de clasificador a usar y de la naturaleza del estudio. Hay que tomar decisiones: que se “tokeniza”, se hace PostTag y se lematiza, cuales “stop\_words” se usan, que límites de frecuencias, como seleccionamos características, como evaluar los modelos. Un punto interesante a la hora de la construcción de vectores es ver cómo se traslada las características seleccionadas de los documentos, a los conjuntos de entrenamiento y de test, ¿debemos tratarlos igual? Con todo esto queremos significar que una parte importante de este tipo de procesos es el ajuste.

Después de la transformación de los textos, ya se puede extraer los modelos de comportamiento según el objetivo perseguido. Para la clasificación de textos, en función del número de etiquetas (clases) y del número de características se buscan los algoritmos adecuados. Algunos algoritmos validos: Naive-Bayes basado en probabilidades, Random Forests, Arboles de decisión, Mejores Vecinos y Maquinas Vectores soporte. Cada modelo tendrá su conjunto de "hiperparametros" a ajustar. Además hay que ver como validar; separar en conjunto de entrenamiento y test, o validación cruzada por ejemplo. Trabajar con un clasificador o combinar varios de ellos es otra de las decisiones a tomar. El fin último de todo es a partir de nuevos documentos, transformarlos para así poder aplicar los modelos y predecir su clasificación.

## 2.2 Punto de partida.

Como antecedente se resume el punto de partida aportado por Autoritas, que es fruto de un amplio proceso de I+D+I sobre un caso de uso similar.

Una parte del estudio comprendía codificar historias clínicas de un hospital. Los clasificadores propuestos fueron tres que se combinaron con un meta-clasificador débil que los pondera:

- Un clasificador basado en Machine Learning con los términos de mayor ganancia de información combinada con el sexo y la edad.
- Un clasificador basado en recuperación de información sobre el diccionario CIE9 oficial.
- Un clasificador basado en recuperación de información sobre los propios diagnósticos.

Los dos clasificadores basados en recuperación de información efectúan expansión de acrónimos.

Los conjuntos de datos usados en el presente estudio forman parte de este antecedente. Algunos de sus resultados obtenidos se utilizan como una referencia donde compararse.

## 2.3 Propuesta.

Como aproximación al problema se propone:

- Construir un clasificador como línea base donde compararse.
- Construir dos clasificadores débiles de naturaleza diferenciada.
- Optimizar la combinación de un conjunto de clasificadores débiles.

De esa manera codificar diagnósticos clínicos en base a un meta-clasificador que fusiona ponderadamente los dos clasificadores propios y los tres proporcionados por Autoritas.

## 2.4 TF-IDF

El Tf-Idf es especialmente útil cuando se quiere saber cuáles son las palabras clave de un documento, desde el punto de vista estadístico. Muy utilizado en minería de textos, buscadores y en clasificación de textos.

La idea fundamental es combinar en un único cuantificador, lo frecuente del término en el documento y lo excepcional de un término para el conjunto de documentos. Entonces podemos caracterizar cada documento con los pesos tf-idf de cada término. Además también nos sirve para poder seleccionar e identificar las palabras frecuentes pero de escaso aporte (stop-words). En la figura siguiente se expone claramente los fundamentos.

$$tf(n) = \sum_{(n)} D1$$

La frecuencia de aparición de un término ( $n$ ) en un documento ( $D1$ ) es la suma de las ocurrencias de dicho término

$$IDF_{(n)} = \log_{10} \frac{N}{DF_{(n)} + 1}$$

También suele utilizarse el logaritmo en base 2, su función es conseguir un coeficiente bajo, fácil de manejar

N es el número total de documentos de la colección.

DF (Document Frequency) es el número documentos en los que aparece el término ( $n$ ) a lo largo de toda la colección

+1 Factor correctivo

$$TF-IDF_{(n,d)} = TF_{(n,d)} \times IDF_{(n)}$$

Peso de un término ( $n$ ) en un documento ( $d$ )

Frecuencia de aparición de un término ( $n$ ) en un documento ( $d$ )

Factor IDF de un término ( $n$ )

Ilustración 3. Fundamentos TF-IDF

[Manuel Blázquez Ochando <http://ccdoc-tecnicasrecuperacioninformacion.blogspot.com.es> ]

Los pesos de los términos explican tanto características del documento como características del conjunto de todos documentos. Esto implica que es necesario recorrer todo el conjunto de datos para la transformación.

## 2.5 LDR.

Low Dimensionality Representation (LDR), [Rangel, F., et al. (2016)] es un modelo de representación que permite utilizar el conjunto completo del vocabulario de un corpus, a la par que reducir de manera drástica el número de características necesarias para modelar los textos. Esta representación de baja dimensionalidad permite mantener la competitividad con el estado del arte en diferentes tareas de clasificación de textos y su aplicación a entornos big data. El concepto clave es el peso que representa la probabilidad de pertenencia de cada término a cada una de las clases. La distribución de pesos para un documento debería estar más cercana a la de los documentos de su correspondiente clase que al resto.

LDR está basado en tf-idf e intenta salvar las limitaciones de esta técnica sin perder representatividad. Cada documento se transforma en características sobre la distribución por clase de los tf-idf de cada término.

El primer paso es calcular los tf-idf para cada documento.

$$\Delta = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1m} & \delta(d_1) \\ w_{21} & w_{22} & \dots & w_{2m} & \delta(d_2) \\ \dots & \dots & \dots & \dots & \dots \\ w_{n1} & w_{n2} & \dots & w_{nm} & \delta(d_n) \end{bmatrix}$$

(Dimensiones de la matriz = términos X documentos)

- Cada columna es un término.
- Cada registro es un documento.
- Cada  $W_{i,j}$  es el peso tf-idf del término  $j$  en el documento  $i$ .
- La última columna representa la clase asignada al documento  $i$ .

Segundo hay que calcular las proporciones del tf-idf por término dentro de cada clase

$$W(t, c) = \frac{\sum_{d \in D / c=\delta(d)} w_{dt}}{\sum_{d \in D} w_{dt}}, \forall d \in D, c \in C$$

(Dimensiones de la matriz = términos X clases)

Tercero en base a los pesos  $W(t,c)$  por cada documento y para cada clase se calcula

$$F(c_i) = \{avg, std, min, max, prob, prop\}$$

avg	Media $W(t,c)$ de todos sus términos
std	Desviación típica de $W(t,c)$ de todos sus términos
min	Mínimo $W(t,c)$ de todos sus términos
max	Máximo $W(t,c)$ de todos sus términos
prob	Suma $W(t,c)$ de todos sus términos dividido total de palabras del documento.
prop	Total de términos del documento dividido total de palabras del documento.

Último paso, por cada documento juntar el vector  $F(c)$  de cada clase.

$$d = \{F(c_1), F(c_2), \dots, F(c_n)\} \sim \forall c \in C,$$

(Dimensiones de la matriz = (6\*clases) X documentos)

## 2.6 Doc2Vec

Este método de representación utiliza modelos neuronales para transformar documentos a espacio vectorial, de modo que documentos similares sean representadas por vectores con poca distancia entre sí.

Como dicen sus autores muchos algoritmos de aprendizaje automático requieren que su entrada sea representada como un vector de características de longitud fija. Cuando se trata de textos, una de las características más comunes de longitud fija es la bolsa de palabras. A pesar de su popularidad, las características de la bolsa de palabras tienen dos debilidades principales: pierden el orden de las palabras y también ignoran la semántica de las palabras. Por ejemplo, 'poderoso', 'fuerte' y 'París' están igualmente distantes. Doc2vec [Tomas Mikolov, et al. (2014)] es un algoritmo no supervisado que aprende las representaciones de características de longitud fija sobre partes de textos de longitud variable, como oraciones, párrafos y documentos. El algoritmo representa cada documento por un vector denso que está entrenado para predecir palabras dentro del documento. Su construcción le da al algoritmo el potencial de superar las debilidades de los modelos de bolsas de palabras.

Muy parecido a Word2Vec, pero utiliza además información de sentencias, oraciones o párrafos. Los planteamientos propuestos:

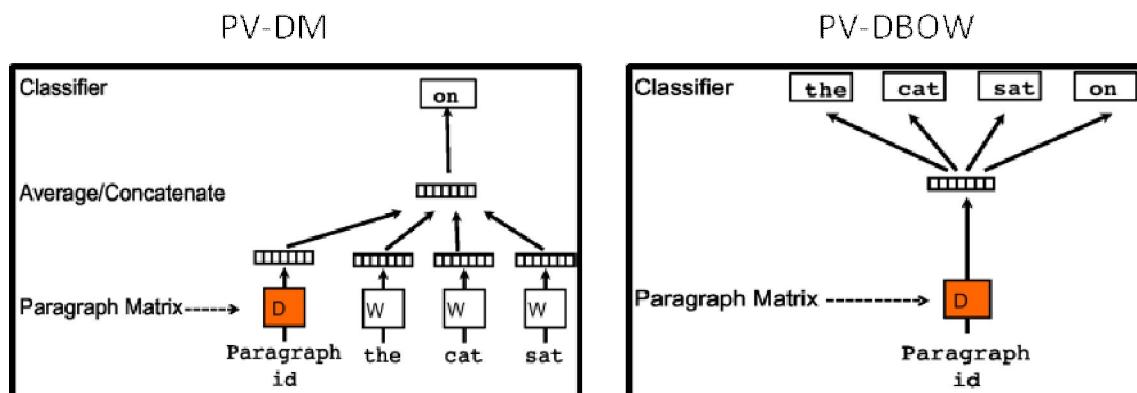


Ilustración 4. Planteamientos Doc2vec.

[Quoc V. Le y Tomas Mikolov (2014)]

### 1-Modelo de memoria distribuida de vectores de párrafos. (PV-DM)

- Cada párrafo u oración se la asigna un identificador único.
- Se intenta predecir la próxima palabra con una concatenación de las anteriores más el vector correspondiente al párrafo.

- El vector del párrafo expresa el contexto restante.
- Los vectores de palabras se comparten para todos los párrafos.
- Para predecir se entrena los vectores de párrafo con SGD descenso de gradiente estocástico con backpropagation.

## 2-Versión Bolsa de palabras distribuidas de vector de párrafo (PV-DBOW)

- Parecido a Skipgram.
- Intenta predecir las palabras de un párrafo dado su vector.
- Modelo más simple.

## 2.7 Otras aproximaciones.

La mayoría de esfuerzos para resolver el problema de automatizar la codificación CIE.9 de diagnósticos clínicos se centran en soluciones con buscadores de términos a modo de asistentes online. Hay varios trabajos de codificación CIE.9 de diagnósticos clínicos con técnicas de Machine Learning casi todos para diagnósticos para lenguaje inglés y alguno para lenguaje castellano, todos ellos con el enfoque clásico de que partiendo de un conjunto intentar obtener el mejor resultado.

Hay un denominador común en todos ellos, se transforma con Bolsa de palabras y uno de los clasificadores evaluados es SVM. Otros clasificadores utilizados son RandomForest, Arboles de decisión [Zhang & Patrick (2008)] [Nerea Aguirre, Estibaliz Amillano (2014)]. Solo uno aplica combinación de clasificadores SVM, KNN, y un buscador de patrones [Aronson & et al. (2007)]. Algunos construyen un modelo clasificador por clase [J.Medori (2010)] [Zhang (2006)]. Hay uno con diagnósticos en francés [J.Medori (2010)] y otro en castellano [Nerea Aguirre, Estibaliz Amillano (2014)]. Hay uno que codifica SNOMED en vez de CIE.9 y aplica TF-IDF de N-Gramm [Zhang & Patrick (2008)]. Casi todos preparan datos: tildes, minúsculas, stop-words y algunos de ellos stemming [Zhang & Patrick (2008)] [J.Medori (2010)].

El enfoque propuesto aquí es diferente ya que se buscar el valor que aporta Big Data a problemas como este. Además se sondan técnicas de representación no antes aplicadas al problema de la codificación de diagnósticos clínicos.

## 2.8 Aprendizaje automático.

En el Aprendizaje Automático (Machine Learning en inglés) se utilizan algoritmos que aprenden a partir de los datos para crear modelos de comportamiento.

### 2.8.1 Algoritmos de clasificación.

El Aprendizaje supervisado se aplica cuando se dispone de los valores de la variable de respuesta. Los modelos se estiman aprendiendo las reglas que nos llevan a la variable respuesta desde las variables de entrada. La Clasificación es cuando la variable respuesta divide a los ejemplos en dos o más clases o categorías. Entonces los modelos etiquetan los elementos de entrada que no se conoce a que clase o categoría pertenece.

Un clasificador infiere unas reglas de decisión durante el entrenamiento e intente predecir sobre un conjunto de clases prefijadas. Vemos la visión de la clasificación que tiene scikit-learn, un referente en ML para Python.

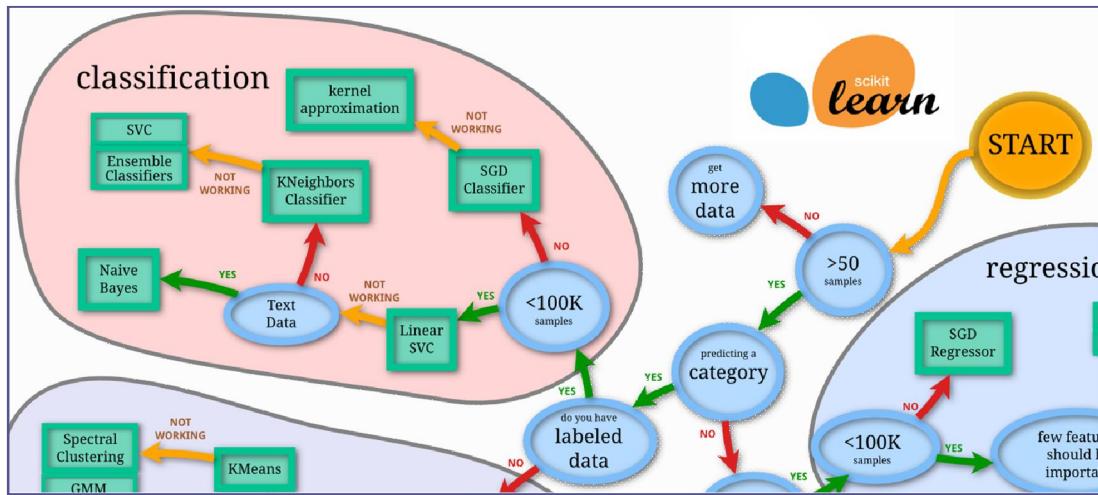


Ilustración 5. Mapa de algoritmos de clasificación.

[[http://scikit-learn.org/stable/tutorial/machine\\_learning\\_map/](http://scikit-learn.org/stable/tutorial/machine_learning_map/)]

**SVM.** Algoritmo Support vector machines (SVM) se introdujo por Vapnik en los años noventa, aplicado para problemas de clasificación y regresión. Es una técnica basada en funciones kernel, utiliza un subconjunto de puntos de entrenamiento en la función de decisión (llamados vectores de soporte). Eficaz para los espacios de alta dimensionalidad y resistente al sobreajuste. Es necesario definir las fronteras entre las diferentes clases hay que construir una función que minimice el error en la separación entre objetos (Error en clasificación) y maximice el margen de separación (mejora de la generalización). Para el caso de más de dos clases ( $k$ ), o se construyen  $k(k-1)/2$  modelos o cada clase es dividida en otros y todas son combinadas. No proporcionan directamente estimaciones de probabilidad, se calculan utilizando una validación cruzada.

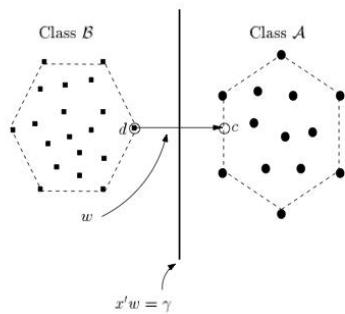


Ilustración 6. Vector soporte con dos clases.

**Clasificador débil** es un clasificador el cual está solo débilmente correlacionado con la clasificación correcta. Predice una clase o varias clases acompañada de una probabilidad. Es aplicable en muchos métodos de aprendizaje automático. Se utiliza para combinar clasificadores.

### 2.8.2 Combinación de clasificadores

Un clasificador asigna a cada ejemplo una predicción de la clase a que pertenece. La combinación de clasificadores es una combinación de las observaciones individuales.

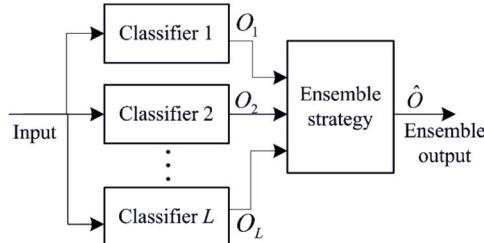


Ilustración 7. Combinación de clasificadores

Normalmente se obtienen mejores resultados que los clasificadores individuales. La combinación es independiente del tipo de clasificador. En muchos casos reduce el sobre ajuste de los clasificadores individuales. Puede dar resultados difíciles de entender. Puede necesitar alto nivel de cómputo.

Tipos de combinaciones:

- Diferentes variables.
- Diferentes ejemplos.
- Diferentes clasificadores.
- Diferentes parámetros.

Cuando se quiere combinar diferentes clasificadores sobre un mismo conjunto de datos, hay muchos métodos de ensamblaje de clasificadores.

Métodos básicos:

- Fusión.
- Voto mayoría simple.
- Voto ponderado.
- Voto bayesiano.
- Stacking.
- En cascada.

Métodos Avanzados:

- Bagging. (bootstrap y aggregation)
- Random Forest.
- Boosting.
- Adaboost.

Además se puede combinar y hacer métodos híbridos, de hecho muchos de los nombrados son mezcla, variación o evolución de algún otro u otros.

### 2.8.3 Método de nivel de medidas.

El método de combinación “ fusión por voto ponderado” está basado en otorgar pesos a las predicciones de los diferentes clasificadores, siendo la predicción final la de mejor ranking al fusionar. Se basa en la metáfora de que hay expertos mejores que otros y su opinión es más valorada en la toma de decisiones.

Qué pasa si estas predicciones son un vector con el nivel de confianza de cada una de sus clases. Entonces hay que buscar la manera de combinarlas para obtener la nueva predicción. Vemos el enfoque de F.Rolin en 2003.

*F. Roli / A Gentle Introduction to Fusion of Multiple Pattern Classifiers*

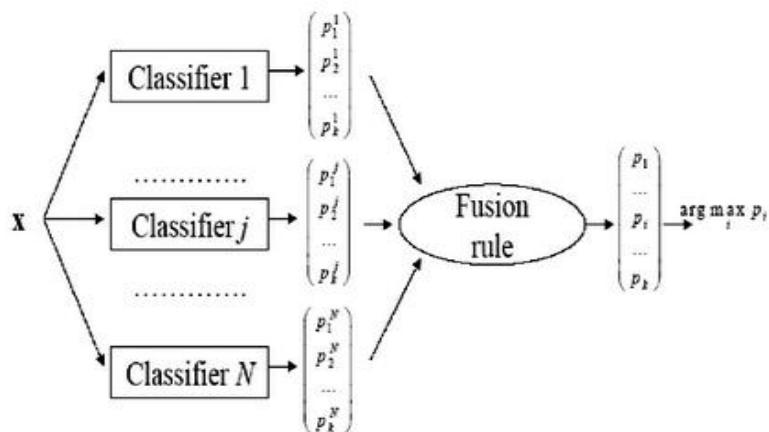


Ilustración 8. Fusión de múltiples clasificadores débiles.

Para la combinación de las salidas se establece unas reglas de fusión que son en definitiva operadores estadísticos sobre los vectores  $v$ .

Promedio simple	$\alpha = \sum_{j=1}^m \frac{v_j}{m}$	Suma: $\alpha = \sum_{j=1}^m v_j$
		Mediana: $\alpha = mediana_{i \leq j < m}(v_j)$
		Máximo: $\alpha = máximo_{i \leq j < m}(v_j)$
Producto	$\alpha = \prod_{j=1}^m v_j$	Mínimo: $\alpha = mínimo_{i \leq j < m}(v_j)$

Ilustración 9. Reglas de fusión.

Vemos un ejemplo explicativo de la aplicación de los operadores estadísticos mencionados. Si se tienen tres clasificadores individuales ( $m=3$ ) para resolver un problema de 4 clases ( $k=4$ ) con los valores  $\{a, b, c, d\}$  y las predicciones de los clasificadores para cada clase son:

$$\begin{aligned} v1 &= \{0.25, 0.55, 0.10, 0.10\} \\ v2 &= \{0.50, 0.00, 0.05, 0.45\} \\ v3 &= \{0.15, 0.20, 0.30, 0.45\} \end{aligned}$$

Entonces los resultados serían:

- Suma :  $\{0.90, 0.75, 0.45, 1.00\} \rightarrow d$
- Promedio:  $\{0.30, 0.25, 0.15, 0.33\} \rightarrow d$
- Producto :  $\{0.018, 0.00, 0.0015, 0.02\} \rightarrow d$
- Mediana :  $\{0.25, 0.20, 0.10, 0.45\} \rightarrow d$
- Máximo :  $\{0.50, 0.55, 0.30, 0.45\} \rightarrow b$
- Mínimo :  $\{0.15, 0.00, 0.05, 0.10\} \rightarrow a$

Si se quiere trabajar con " fusión voto ponderado" se puede aplicar una regla como la del promedio ponderado, sea  $w_j$  el peso del clasificador  $j$ .

$$p_i = \sum_{j=1}^N w_j p_i^j$$

## 2.9 Big Data Analytics.

En este apartado se describe el proceso para descubrir conocimiento de un conjunto de datos con el objetivo de extraer valor de los mismos.

### 2.9.1 Definición

El Big Data Analytics es como una amalgama de conceptos, técnica y visiones, poco clara y difícil de concretar. No hay una definición formal, no hay un ámbito definido, hoy en día parece que todo es big data. Simplificando, un ecosistema Big Data consiste en definir un problema, coleccionar fuentes de datos y marcar un objetivo: Queremos tomar mejores decisiones, mejores modelos que expresen el comportamiento de una realidad, queremos convertir datos en conocimiento.

### 2.9.2 Proceso KDD

Descubrimiento de Conocimiento a partir de Bases de Datos (KDD, del inglés Knowledge Discovery from Databases).

"KDD se refiere al proceso general de descubrimiento de conocimiento útil a partir de datos" "KDD es una integración de múltiples tecnologías para la gestión de datos, tales como la gestión de base de datos y almacenamiento de datos, aprendizaje automático estadístico, apoyo a las decisiones, y otros, como la visualización y computación paralela." [De Martino, 2002] "proceso no trivial de identificar patrones válidos, novedosos, potencialmente útiles y en última instancia comprensibles a partir de los datos". "KDD se refiere al proceso general de descubrimiento de conocimiento útil a partir de datos, y minería de datos se refiere a un paso en particular en este proceso." [Fayyad, 1996]

KDD se divide en las siguientes fases.

## Definición del problema

- Tipo de conocimiento: dirigido, no dirigido
- Toma requisitos
- Objetivos: de negocio, refinado, de minería de datos

## Integración de datos

- Crear fuente de datos específica para este caso o general que se pueda reutilizar.
- Base de datos BBDD interna o externa.
- Históricos suficientes y de calidad
- Nivel de granularidad, formatos.
- Volumen de datos.
- Vista Minable.

## Preparación de Datos

- Comprensión de los datos: características descriptivas.
- Visualización de los datos: scatterplot de características.
- Limpieza: valores redundantes, valores despreciables, valores anómalos.
- Transformación: discretizar, numerar, normalizar, atributos derivados, reducción de atributos, intercambio de dimensiones.
- Selección: vertical, horizontal muestreo.

## Modelado

- Descriptivo: correlaciones, estudios factoriales, reglas de asociación, agrupamiento.
- Predictivo: regresión, clasificación y categorización.

## Evaluación de modelos.

- Clasificación: %Acierto, %Error, Curva ROC, desequilibrio
- Regresión: Error cuadrático medio, Error absoluto.
- Reglas de asociación: Soporte confianza
- Agrupación: Distancias, matriz de confusión.
- Otros criterios: comprensibilidad, sencillez, interés, aplicabilidad.

## Aplicación de Modelos

- Interpretarlo y comprenderlo.
- Contrastarlo con lo conocido.
- Intercambio y difusión.
- Integrarlo.
- Adaptarlo y combinarlo.

## Monitorización y Revisión de Modelos

- Evaluación periódica del modelo.
- Evaluación del cambio.

- Revisión parcial o total

La etapa de minería de datos es el hecho diferencial de los procesos KDD frente a los métodos estadísticos o los sistemas OLAP. Las técnicas de minería de datos son las que permiten extraer conocimiento de los datos.

### 2.9.3 Data Science

Como un paso más allá del KDD emerge el Data Science .

"Data Science es el estudio generalizable de la extracción del conocimiento de los datos" [Tom Fawcett,2013.]

"La ciencia de los datos se refiere a un área de trabajo emergente que se ocupa de la recopilación, preparación, análisis, visualización, gestión y conservación de grandes colecciones de información" [Jeffrey Stanton (2012)]

El científico de datos analiza, interpreta y comunica datos trasladando conocimiento a la empresa para que haga uso del mismo y adapte sus productos y servicios, creando nuevas oportunidades de negocio.

Los 4 pasos del proceso de Data Science según Allen G. Grimm

- Identificar los problemas
- Modelizar
- Integrar
- Crecer



Ilustración 10. Disciplinas del Data Science

<http://berkeleysciencereview.com/how-to-become-a-data-scientist-before-you-graduate/>

### 2.9.4 Conclusión

La minería de datos como parte del proceso de KDD aporta la base para enfrentarnos al problema. El Data Science la actitud para revolver a este tipo de problema. Y el Big Data Analytics engloba todo lo necesario para enfrentarnos a los nuevos desafíos.

## 2.10 Recursos

En este capítulo se expone la selección de los recursos a utilizar.

Y como muestra de la dificultad de realizar tal selección solo hay que mirar la amalgama de productos que se muestra en la figura adjunta a este punto, en ella vemos que no solo hay diversidad de marcas comerciales, productos e iconos, sino también que hay diversas agrupaciones. Y como esta figura seguramente podemos encontrar otras tantas, con maneras diferentes de clasificarlas. Esto denota la variabilidad cambiante cuando hablamos de tecnologías Big Data.

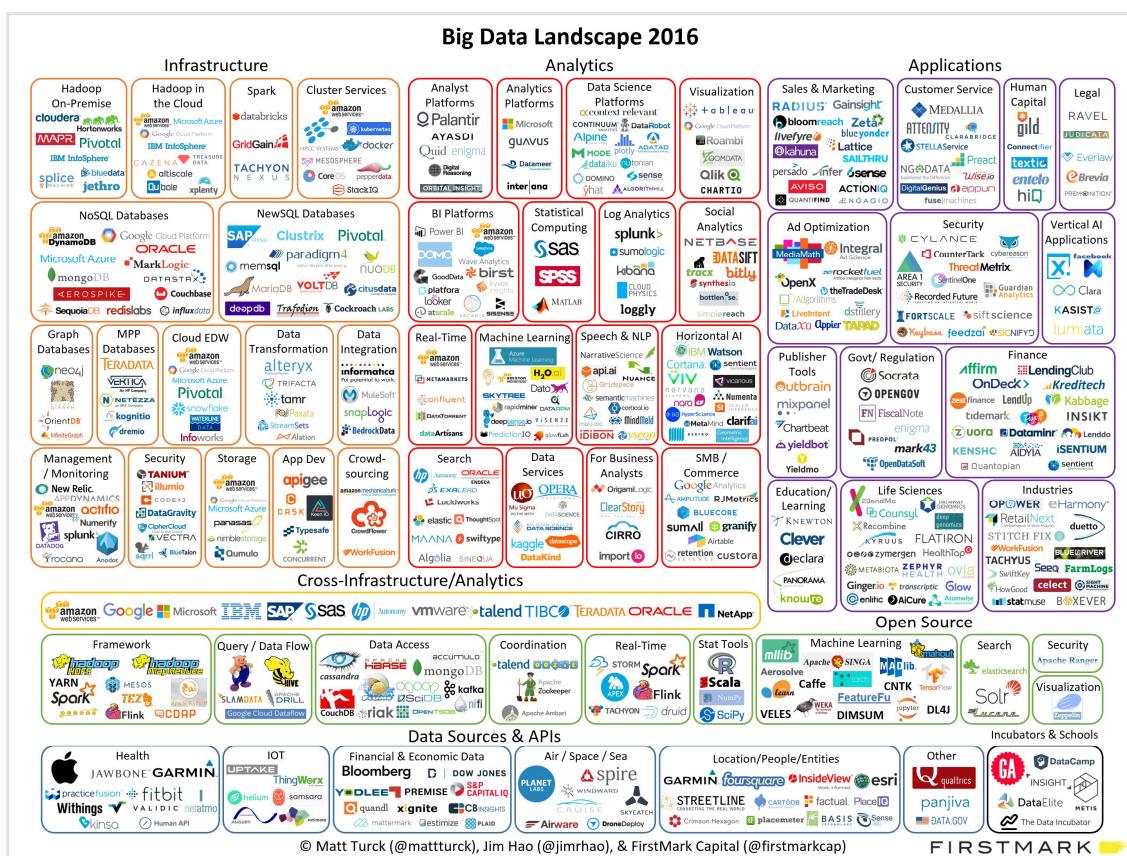


Ilustración 11. Ecosistema Big Data.

### 2.10.1 Infraestructura.

En cuanto a infraestructura hay multitud de opciones, se nombran algunas de las que se consideran aplicables y adecuadas al problema. Cuando se dice adecuado se quiere decir: viable, valido, eficaz y eficiente.

Se puede hablar de varios escenarios:

- Local. Adecuado para un tamaño de problema pequeño o mediano, digamos que no requieren un cluster ni para almacén de datos ni para el procesamiento de datos. Opciones a considerar: Linux, Windows, VMWare, Docker y Vagrant.

- Cluster. Adecuado cuando el tamaño del problema crece y no se puede abordar "Localmente", y se hace necesario un cluster de procesamiento paralelo o de almacenamiento distribuido. Opciones a considerar: Spark, Hadoop, Storm y Flink.
- Web. El pago por uso es la clave de este escenario. Opciones a considerar: Amazon Web Service (IaaS), Microsoft Azure, IBM y Google son los referentes.

#### 2.10.2 Software.

La elección del software en parte viene determinada por la infraestructura seleccionada, y presenta además también una problemática similar y muchas opciones validas.

Con un problema como el propuesto, los escenarios orientados a procesos de ciencia de datos son:

- Local. Cuando se puede trabajar con herramientas localmente. Opciones a considerar: Python, R, Weka, RapidMiner, Java y Tableau.
- Cluster. Cuando es necesario un cluster de procesamiento paralelo o de almacenamiento distribuido. Opciones a considerar: PySpark, Java y Scala.
- Web. Cuando el pago por uso es la clave de este escenario. Opciones a considerar: BigML, Amazon Web Service (PaaS), IBM y Microsoft.

#### 2.10.3 Datos.

Sin datos no hay análisis posible, su obtención se convierte pues en una tarea primordial. La calidad de los datos es determinante, en gran medida, para obtener buenas conclusiones y resultados.

Para el estudio se dispone de dos conjuntos de datos reales proporcionados por Autoritas:

- Un conjunto de 10.000 predicciones de tres clasificadores débiles de las altas de historias clínicas de un cierto hospital. No se dispone de los textos diagnósticos. Se dispone también de la evaluación de varias ponderaciones al fusionar los tres clasificadores.
- Un conjunto de 2.954 diagnósticos clínicos anónimos del servicio de medicina interna de un cierto hospital. Si se dispone de los textos diagnósticos.

### 3 EXPERIMENTOS

En este capítulo se exponen los experimentos realizados, las propuestas para solucionar el problema y las experiencias del proceso analítico desarrollado.

#### 3.1 Recursos.

Para poder alcanzar los objetivos perseguidos hay que elegir los medios adecuados. Se marcan los requisitos a cumplir, por las infraestructuras y por las herramientas software. Toda elección debe cumplir ser:

- Útil. Debe servir para estudios de minería de datos y de extracción de conocimiento.
- Possible. Debe ser económicamente viable y ser una opción real alcanzable.
- Flexible. Debe poder enfrentarse a distintos tamaños del problema y entornos.
- Visible. Debe poder transmitir el qué y el cómo, mientras más directo y visual sea, mejor se comunican las ideas.
- Futuro. Debe ser opciones con buena proyección.

##### 3.1.1 Python.

Teniendo en cuenta los requisitos marcados y los objetivos del estudio, se valora como una excelente opción el lenguaje Python. Los argumentos a su favor son muchos: Es uno de los principales referentes en los entornos "Analytics" y es un producto "community". Tiene una buena curva de aprendizaje, y es esta una característica especialmente atractiva para cuando se quiere realizar un estudio como medio de crecer en conocimiento. Python se aplica tanto para entornos local como distribuidos. Además, tiene muchas librerías que facilitan y simplifican el trabajo.

Python presenta una desventaja en rendimiento frente a los lenguajes compilados y esto puede plantear dudas sobre su validez, sobre todo en entornos de producción. Actualmente ya hay notables avances al reescribir en lenguaje C muchas de sus librerías.

La opción JAVA parece también bastante lógica, Weka, Hadoop, Spark se basan en él, además es un lenguaje muy versátil y sobre todo con un excelente rendimiento. Pero supone una curva de aprendizaje muy alta para la coyuntura de este estudio.

Como consecuencia de lo expuesto la elección es trabajar con Python. Esta decisión es un nuevo requisito a tener en cuenta.

### 3.2 Local: IPython Notebook sobre Docker

Cuando el tamaño del problema permite trabajar localmente se hace altamente fructífero, olvidándose de las complicaciones que conllevan los entornos distribuidos.

Hay que decidir cómo se va a trabajar con Python. Hay muchos framework para trabajar con Python y casi todos válidos. Pero entre todos ellos llama mucho la atención los Notebook, su facilidad para transmitir lo que hace el código y el cómo se resuelve el problema, es determinante a la hora de su elección. El Notebook no solo sirve para contar la historia, sino también es especialmente útil cuando se está aprendiendo y probando las diferentes maneras de abordar un problema concreto de un punto preciso del código. El ver explícitamente lo que se ha hecho antes y poder probar diferentes soluciones sin afectar al resto da mucha flexibilidad. Por todo ello IPython Notebook es la elección y un nuevo requisito.

Hay que disponer de una estación de trabajo que sirva como laboratorio para el desarrollo de las propuestas y los experimentos. Hay que decidir sobre el sistema operativo y donde montar el entorno de desarrollo. Hay que buscar la manera de ser independiente y limitar las interferencias con otros usos de la estación de trabajo. La opción de virtualizar parece entonces una consecuencia lógica. Y dentro de las muchas maneras de virtualizar llama la atención Docker, cuyo fundamento es crear contenedores con lo básico y necesario para ejecutar una determinada aplicación. Es una opción de futuro que cada vez se utiliza más para entornos de desarrollo y test e incluso para entornos de producción. Es la base para el paradigma de los Micro Servicios. Entonces se selecciona Docker como la mejor opción.

Se destaca <https://github.com/jupyter/docker-stacks> como un contenedor con todo lo necesario para la realización de las propuestas.

**IP[y]:** IPython  
Interactive Computing



Ilustración 12. Recursos locales

#### 3.2.1 Cluster: Pyspark + AWS Educate

Cuando el tamaño del problema crece aparece la necesidad de alguna solución para obtener resultados en un tiempo razonable; la opción de un cluster de procesamiento distribuido siempre debe de estar sobre la mesa.

Big Data es una solución para abordar problemas de procesamiento distribuido pero hay que concretar entre todas sus múltiples opciones.

La opción de  se convierte en el framework de procesamiento paralelo distribuido seleccionado.

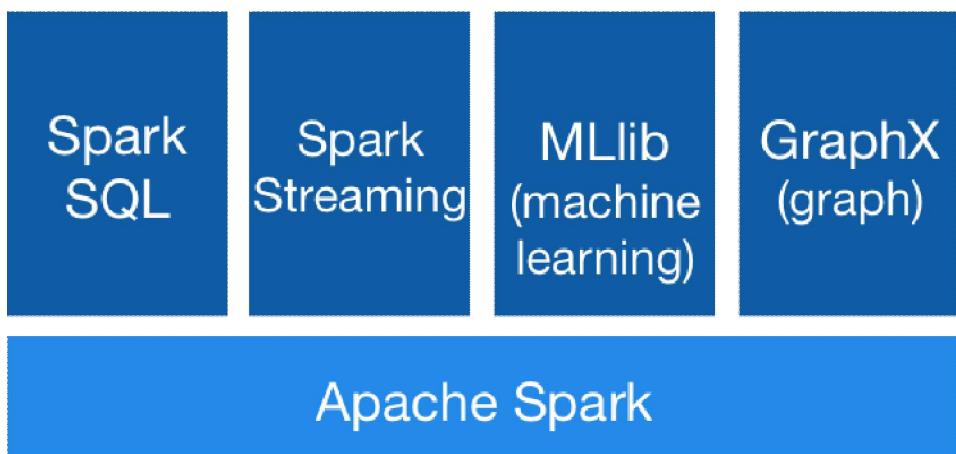


Ilustración 13. Ecosistema Spark

<http://spark.apache.org/>

¿Por qué Apache Spark? De manera general porque: es open-source, es rápido, es presente y futuro, la industria lo está adoptando como motor dinamizador de Big Data, tiene amplias facilidades para los científicos de datos, además de por su habilidad para conectarse con muchos tipos de base de datos. Su hecho diferencial es que busca maximizar el procesamiento apoyado en memoria y minimizar la utilización de disco.

Si se busca una experiencia real Big Data, no hay que conformarse en desarrollar código valido para entornos distribuidos sin ver todo lo que conlleva. Cuando se habla de entorno distribuido se habla de varias maquinas interconectadas y esto implica un salto cuantitativo económico y en número de problemas. Encontrar pues un cluster real, donde poder desarrollar y probar las propuestas y experimentos, es una tarea complicada. Descartado la inversión se sondea entornos libres en la web, (DataBrick y Amazon Web Service), pero la parte libre de costes económicos está muy limitada y orientada a un entorno de formación más que a uno real, solo se permiten ejecuciones con poco consumo de memoria (característica muy determinante en Spark). Gracias a Germán Moltó y al programa AWS Educate se tiene la oportunidad de trabajar con un cluster real de Spark con el servicio Amazon EMR. Con la ventaja añadida de poder desarrollar código Pyspark (Python para Spark) en un entorno local, válido para entornos distribuidos. Todo esto hace de Amazon EMR la elección.



<http://spark.apache.org/>  
<https://aws.amazon.com/es/education/awseducate>  
<https://aws.amazon.com/es/emr>

"Apache Spark en Amazon EMR. Apache Spark en Hadoop YARN dispone de compatibilidad nativa con Amazon EMR. Además, puede crear de forma rápida y sencilla clusteres de Apache Spark administrados a través de la consola de administración de AWS, la CLI de AWS o la API de Amazon EMR. Asimismo, puede utilizar características adicionales de Amazon EMR, incluida la conectividad rápida con Amazon S3 mediante el sistema de archivos de Amazon EMR (EMRFS), la integración con el mercado de capacidad puntual de Amazon EC2 y Auto Scaling para agregar o eliminar instancias de su cluster. Además, puede utilizar Apache Zeppelin para crear cuadernos interactivos y colaborativos para explorar los datos con Apache Spark." [<https://aws.amazon.com/es/emr/details/spark/>]

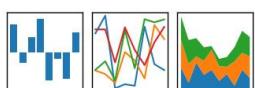
### 3.3 Clasificadores Texto.

Hay que estudiar como clasificar diagnósticos clínicos y también que puede aportar el Big Data Analytics. La construcción de un clasificador es la base para la resolución de este tipo de problemas, por lo que construir clasificadores propios es una manera de aprender como extraer conocimiento de diagnósticos clínicos. Se decide hacer dos clasificadores (LDR y Doc2Vec), de naturaleza y técnicas diferenciadas y que no hayan sido empleados aun en casos similares a ser posible. Notar también que, la decisión de construir dos clasificadores surge como consecuencia de buscar los límites al calibrado cuando se combinan diferentes clasificadores. Una vez tomada la decisión de su construcción, hay que ver la manera de evaluarlos, por eso construimos un clasificador más con una técnica ya contrastada (TF-IDF), que se convierte en la línea base de referencia donde compararse. Hay que tener en cuenta algunas consideraciones para entender el porque de cierta decisiones. En el estudio realizado no se busca maximizar la precisión de las predicciones, se busca saber si hay alguna aportación significativa de las propuestas investigadas y cuál es el camino para sacarles mejor provecho a cada clasificador. Muchas veces en la obsesión de buscar un mejor resultado, se toman decisiones y se hacen ajustes de hiperparametros, o se eligen opciones de preprocesado, que enmascaran el funcionamiento o validez real de una propuesta. Se escogerá siempre las opciones más comunes y habituales, con pequeñas adaptaciones naturales a cada tipo de clasificador. Es un equilibrio difícil de conseguir pero necesario si se quiere poder comparar adecuadamente.

La elección de trabajar con un IPython Notebook es una buena manera de ver la traza del pensamiento deductivo pero a la vez demasiado extensa para plasmarlo por completo en este estudio, por lo que solo se pondrán algunos extractos como ejemplo cuando lo se considere necesario. El Notebook de Python empiezan con las librerías en las se basa el código, y entre ellas destacamos especialmente Sklearn y Pandas, dos librerías que han sido muy útiles para el desarrollo de este trabajo. Pandas y sus dataframe son excelentes para la manipulación de datos, y Sklearn es un referente a la hora de algoritmos de aprendizaje.



Utilidades para análisis de datos y minería de datos.  
<http://scikit-learn.org>



Utilidades para estructuras de datos y análisis de datos.  
<http://pandas.pydata.org>

Otras librerías a destacar: NLTK para trabajar con textos, Gensim que implementa el algoritmo Doc2vec y Matplotlib o Seaborn para visualización.



<https://radimrehurek.com/gensim/>



<http://www.nltk.org/>



<https://matplotlib.org/>



<https://seaborn.pydata.org/>

```
# -*- coding: utf-8 -*-
# Importar la libreria NLTK para manejo de texto en python
from nltk.tokenize import sent_tokenize
from nltk.tokenize import word_tokenize
from nltk.stem.porter import PorterStemmer
from nltk.stem import SnowballStemmer
from nltk.corpus import stopwords
from nltk import FreqDist

from string import punctuation #Importar simbolos puntuacion del ingles
import unicodedata #Para quitar tildes

#Importar lo las funciones, clases y algoritmos que utilizaremos de Scikit-Learn
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer, TfidfTransformer
from sklearn.linear_model import SGDClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import StandardScaler, LabelEncoder, normalize
from sklearn.model_selection import StratifiedShuffleSplit, KFold, cross_val_score, cross_val_predict, train_test_split
#from sklearn import decomposition, pipeline, metrics, grid_search
from sklearn.externals import joblib
from sklearn import metrics
from sklearn.utils import shuffle
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

#import gensim
#from gensim.models import word2vec, Phrases ,doc2vec
#from gensim.models import doc2vec
#from gensim.models.doc2vec import TaggedDocument,LabeledSentence

#Importar la libreria PANDAS para el manejo de datos
import pandas as pd
pd.set_option('max_colwidth',1000) #Para que los textos se vean enteros

from scipy.sparse import hstack #Importar la funcion de matriz no densa de scipy

import numpy as np #Importar para trabajar con matrices que utiliza el pandas.

#Importar la libreria para manipulacion de XML
import xml.etree.ElementTree as ET

#Importar el modulo de expresiones regulares y de strings
import re
import string

#Importar para sacar tiempos
import time
import datetime

import gc #Para gestionar la memoria

import logging #Importar gestion de logs.

import os # Import para interaccion con el sistema

#Importar para visualizar
import seaborn as sns
sns.set(font="serif")
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')
```

Ilustración 14. Librerías empleadas

En el caso de uso de estudio se construyen varios clasificadores individuales que se combinan con otros ya existentes. La manera de ensamblar se hace por votación ponderada de clasificadores débiles. Por lo tanto un requisito a cumplir es que los algoritmos y modelos deben poder predecir con probabilidades.

Para hacer la comparación de diferentes clasificadores lo más homogénea posible se decide seleccionar un único algoritmo clasificador para las tres técnicas del estudio. Con el apoyo de la utilidad "GridSearchCV" se comparan tres algoritmos usuales en la clasificación de textos, resultando ganador el SVM. El algoritmo Support Vector Machine (SVM), está considerado como uno de los mejores dentro del estado del arte y se encuentra en muchos estudios y casi siempre con buenos resultados. Además cumple el requisito de poder hacer predicciones con probabilidades. Por todo ello SVM es seleccionado como el clasificador único en la construcción de todos los clasificadores.

```
Selección clasificador

csvc = SVC(probability = True,decision_function_shape='ovo')
cmul = MultinomialNB()
csgd = SGDClassifier(loss='log', penalty='l2',alpha=1e-3, n_iter=5, random_state=42)

text_clf = Pipeline([
    ('tfidf', TfidfVectorizer(analyzer = 'word',min_df=1,tokenizer = my_tokenize)),
    ('scaler', StandardScaler(with_mean=False)),
    ('clf', cmul ),
])
parameters = {'clf':(cmul, csvc,csgd), 'tfidf_min_df': [1,3,6,9]}
clf = GridSearchCV(text_clf, parameters, n_jobs=-1, verbose=1)
clf.fit(corpus_textos_df.texto, corpus_textos_df.clase)
print("Best score: %0.3f" % clf.best_score_)
print("Best parameters set:")
best_parameters = clf.best_estimator_.get_params()
for param_name in sorted(parameters.keys()):
    print("\t%s: %r" % (param_name, best_parameters[param_name]))
#clf.cv_results_
#clf.grid_scores_

Fitting 3 folds for each of 12 candidates, totalling 36 fits
[Parallel(n_jobs=-1)]: Done 36 out of 36 | elapsed: 6.7s finished
Best score: 0.402
Best parameters set:
clf: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
decision_function_shape='ovo', degree=3, gamma='auto', kernel='rbf',
max_iter=-1, probability=True, random_state=None, shrinking=True,
tol=0.001, verbose=False)
tfidf_min_df: 6
```

Ilustración 15. Selección clasificador

### 3.3.1 Datos para codificar.

Para obtener modelos hay que poder entrenar y para entrenar hacen falta ejemplos. Para extraer pautas de comportamiento de un conjunto de datos es fundamental conocer como son esos ejemplos.

Para el estudio se dispone de un conjunto de datos anonimizados en formato csv que corresponde a los diagnósticos clínicos del Servicio de Medicina Interna de un hospital. Los diagnósticos han pasado por el filtro de los documentalistas del hospital. Se dispone además de un fichero en formato csv del catálogo de codificación de enfermedades CIE.9. A continuación se muestra cómo se cargan los datos, que aspecto tienen y alguna información descriptiva.

jupyter TFM\_Datos\_BUENO Last Checkpoint: 20 hours ago [autosaved] Logout

File Edit View Insert Cell Kernel Widgets Help Python 3 CellToolbar

### DESCRIPCION

```
In [13]: df = pd.read_csv(fichero_dat, encoding='utf-8', header=0,infer_datetime_format=True)
df.head(5)
```

	sexo	fechanac	clie9	diag textual
0	V	14/08/1963	3	GEA POR SALMONELLA
1	V	07/09/1971	3	GEA SEVERA CON PRODUCTOS PATHOLOGICOS
2	V	07/04/1928	3	- GASTROENTERITIS AGUDA POR SALMONELLA SP
3	V	10/02/1965	4.9	DIARREA ETIOLOGIA INFECCIOSA PROBABLE VIH
4	M	28/11/1938	5.9	VOMITOS. DESHIDRATACION LEVE. BRONCOESPASMO. INSUF. SUPRARRENAL

```
In [14]: #Cambiamos nombres de columnas
df=df.rename(columns = {'clie9':'clase'})
df=df.rename(columns = {'diag textual':'texto'})
print ('Conjunto datos N° de registros',df.shape[0], 'clases ',len(df.clase.unique()))
df.describe(include=['O'])
```

Conjunto datos N° de registros 2954 clases 551

	sexo	fechanac	clase	texto
count	2954	2954	2954	2943
unique	2	2141	551	2603
top	V	01/03/1922	428	INSUFICIENCIA CARDIACA DESCOMPENSADA
freq	1571	8	222	14

```
In [11]: df.sexo.value_counts(normalize=True)
```

```
Out[11]: v    0.531821
M    0.468179
Name: sexo, dtype: float64
```

Ilustración 16. Carga datos + descripción.

Del análisis de los datos se desprende que hay 2.954 ejemplos, evidentemente no son muchos y esto es una dificultad añadida a tener en cuenta. Hay 551 clases diferentes de 14.733 posibles, que aunque no cubren todas las posibles clases, hay que tener en cuenta como se realizan diagnósticos clínicos en un Servicio de un hospital. Los servicios están separados por especialidades y esto implica que el número de posibles diagnósticos se reduce. El estudio asume este supuesto. Si se considera el conjunto de clases como un conjunto ordenado se puede hacer un histograma para ver su distribución.

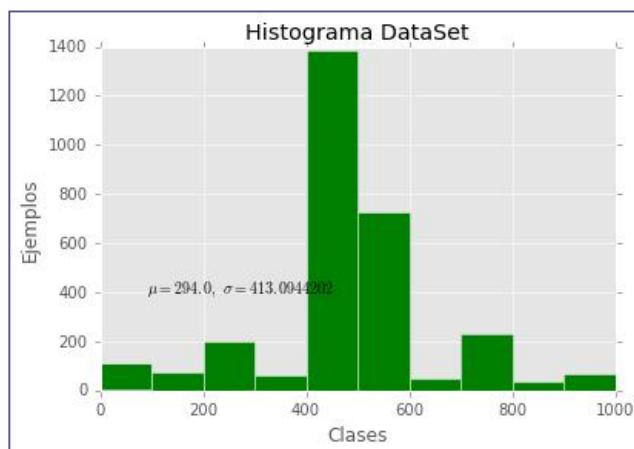


Ilustración 17. Histograma de diagnósticos.

## Big data para codificar diagnósticos clínicos de manera automática.

La mayoría de ejemplos se concentran entre los códigos 400 y 600 que corresponden a enfermedades del aparato circulatorio, respiratorio, digestivo y urinario (Medicina Interna).

CLASIFICACION DE ENFERMEDADES Y LESIONES	
1. ENFERMEDADES INFECCIOSAS Y PARASITARIAS (001-139)	
2. NEOPLASIAS (140-239)	
3. ENFERMEDADES ENDOCRINAS, DE LA NUTRICION Y METABOLICAS Y TRASTORNOS DE LA INMUNIDAD (240-279)	
4. ENFERMEDADES DE LA SANGRE Y DE LOS ORGANOS HEMATOPOYETICOS (280-289)	
5. TRASTORNOS MENTALES, DEL COMPORTAMIENTO Y EL DESARROLLO NEUROLÓGICO (290-319)	
6. ENFERMEDADES DEL SISTEMA NERVIOSO Y DE LOS ÓRGANOS DE LOS SENTIDOS (320-389)	
7. ENFERMEDADES DEL SISTEMA CIRCULATORIO (390-459)	
8. ENFERMEDADES DEL APARATO RESPIRATORIO (460-519)	
9. ENFERMEDADES DEL APARATO DIGESTIVO (520-579)	
10. ENFERMEDADES DEL APARATO GENITOURINARIO (580-629)	
11. COMPLICACIONES DEL EMBARAZO, PARTO Y PUEPERIO (630-679)	
12. ENFERMEDADES DE LA PIEL Y DEL TEJIDO SUBCUTÁNEO (680-709)	
13. ENFERMEDADES DEL SISTEMA OSTEO-MIOARTICULAR Y TEJIDO CONJUNTIVO (710-739)	
14. ANOMALIAS CONGÉNITAS (740-759)	
15. CIERTAS ENFERMEDADES CON ORIGEN EN EL PERÍODO PERINATAL (760-779)	
16. SÍNTOMAS, SIGNOS Y ESTADOS MAL DEFINIDOS (780-799)	
17. LESIONES Y ENVENENAMIENTOS (800-999)	
CODIGOS V	
CLASIFICACIÓN SUPLEMENTARIA DE FACTORES QUE INFLUEN EN EL ESTADO DE SALUD Y CONTACTO CON SERVICIOS SANITARIOS (V01-V91)	
CODIGOS E	
CLASIFICACION DE PROCEDIMIENTOS	
CODIGOS M	

Ilustración 18. CIE.9 capítulos enfermedades.

[http://eciemaps.msssi.gob.es/ecieMaps/browser/index\\_9\\_mc.html](http://eciemaps.msssi.gob.es/ecieMaps/browser/index_9_mc.html)

Se puede separar los textos por palabras para describir características cuantitativas.

<pre>corpus_textos_df['words'] = corpus_textos_df.texto.map(my_tokenize) corpus_textos_cat['words'] = corpus_textos_cat.texto.map(my_tokenize) shuffle(corpus_textos_df).head()</pre>	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Out[25]:</th><th style="text-align: center;">sexo</th><th style="text-align: center;">fechanac</th><th style="text-align: center;">clase</th><th style="text-align: center;">texto</th><th style="text-align: center;">words</th></tr> </thead> <tbody> <tr> <td></td><td style="text-align: center;">1053</td><td style="text-align: center;">M</td><td style="text-align: center;">24/08/1926</td><td style="text-align: center;">453.4</td><td>PARADA CARDIORESPIRATORIA. TROMBOSIS VENOSA PROFUNDA DE EID</td><td>[parada, cardiorespiratoria, trombosis, venosa, profunda, eid]</td></tr> <tr> <td></td><td style="text-align: center;">2175</td><td style="text-align: center;">M</td><td style="text-align: center;">25/08/1918</td><td style="text-align: center;">519.8</td><td>INSUFICIENCIA RESPIRATORIA AGUDA NO HIPERCAPNICA</td><td>[insuficiencia, respiratoria, aguda, hipercapnica]</td></tr> <tr> <td></td><td style="text-align: center;">1024</td><td style="text-align: center;">M</td><td style="text-align: center;">10/06/1927</td><td style="text-align: center;">434.91</td><td>SD. CONFUSIONAL AGUDO EN PACIENTE CON DETERIORO COGNITIVO SUBYACENTE DE CAUSA VASCULAR</td><td>[sd, confusional, agudo, paciente, deterioro, cognitivo, subyacente, causa, vascular]</td></tr> <tr> <td></td><td style="text-align: center;">2873</td><td style="text-align: center;">M</td><td style="text-align: center;">05/08/1923</td><td style="text-align: center;">850.5</td><td>TCE CON CONTUSION COSTAL IZQ POR CAIDA CASUAL</td><td>[tce, contusion, costal, izq, caida, casual]</td></tr> <tr> <td></td><td style="text-align: center;">1518</td><td style="text-align: center;">V</td><td style="text-align: center;">19/04/1920</td><td style="text-align: center;">486</td><td>NEUMONIA. INSUFICIENCIA RESPIRATORIA</td><td>[neumonia, insuficiencia, respiratoria]</td></tr> </tbody> </table>	Out[25]:	sexo	fechanac	clase	texto	words		1053	M	24/08/1926	453.4	PARADA CARDIORESPIRATORIA. TROMBOSIS VENOSA PROFUNDA DE EID	[parada, cardiorespiratoria, trombosis, venosa, profunda, eid]		2175	M	25/08/1918	519.8	INSUFICIENCIA RESPIRATORIA AGUDA NO HIPERCAPNICA	[insuficiencia, respiratoria, aguda, hipercapnica]		1024	M	10/06/1927	434.91	SD. CONFUSIONAL AGUDO EN PACIENTE CON DETERIORO COGNITIVO SUBYACENTE DE CAUSA VASCULAR	[sd, confusional, agudo, paciente, deterioro, cognitivo, subyacente, causa, vascular]		2873	M	05/08/1923	850.5	TCE CON CONTUSION COSTAL IZQ POR CAIDA CASUAL	[tce, contusion, costal, izq, caida, casual]		1518	V	19/04/1920	486	NEUMONIA. INSUFICIENCIA RESPIRATORIA	[neumonia, insuficiencia, respiratoria]
Out[25]:	sexo	fechanac	clase	texto	words																																					
	1053	M	24/08/1926	453.4	PARADA CARDIORESPIRATORIA. TROMBOSIS VENOSA PROFUNDA DE EID	[parada, cardiorespiratoria, trombosis, venosa, profunda, eid]																																				
	2175	M	25/08/1918	519.8	INSUFICIENCIA RESPIRATORIA AGUDA NO HIPERCAPNICA	[insuficiencia, respiratoria, aguda, hipercapnica]																																				
	1024	M	10/06/1927	434.91	SD. CONFUSIONAL AGUDO EN PACIENTE CON DETERIORO COGNITIVO SUBYACENTE DE CAUSA VASCULAR	[sd, confusional, agudo, paciente, deterioro, cognitivo, subyacente, causa, vascular]																																				
	2873	M	05/08/1923	850.5	TCE CON CONTUSION COSTAL IZQ POR CAIDA CASUAL	[tce, contusion, costal, izq, caida, casual]																																				
	1518	V	19/04/1920	486	NEUMONIA. INSUFICIENCIA RESPIRATORIA	[neumonia, insuficiencia, respiratoria]																																				
<pre>In [26]: #np.mean(len(corpus_textos_df.lab)) print("Longitud media de frase: %0.2f (+/- %0.2f)\n" % (np.mean(corpus_textos_df['words'].apply(len)),  np.std(corpus_textos_df['words'].apply(len)) * 2)) fdist = acumula_fdist(corpus_textos_df.words) print ('Nº de palabras del corpus:', fdist.N()) print ('Nº de palabras del vocabulario:', fdist.B()) print ('Nº de palabras frecuencia 1=', len(fdist.hapaxes())) fdist.plot(30)  Longitud media de frase: 4.62 (+/- 5.45)  Nº de palabras del corpus= 13597 Nº de palabras del vocabulario= 2713 Nº de palabras frecuencia 1= 1641</pre>																																										

Ilustración 19. Separación en palabras

Las frases son cortas, longitud media 4.62 y el vocabulario es bastante específico. Las palabras más usadas son: respiratorio, insuficiencia e infección.

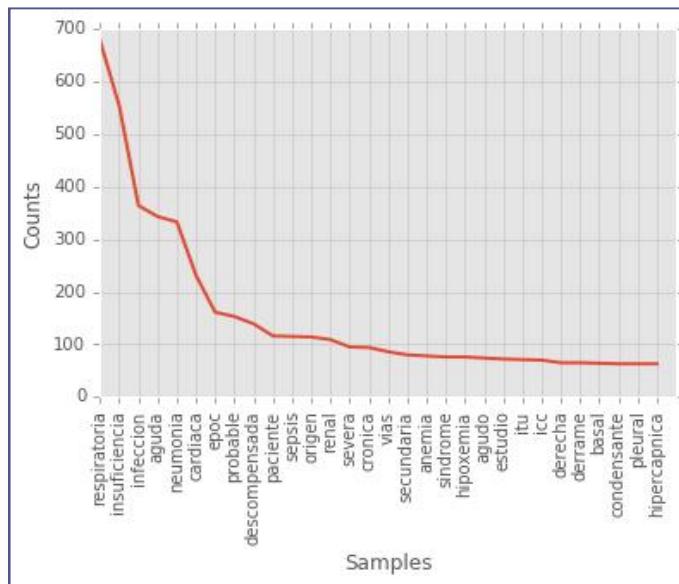


Ilustración 20. Palabras más usadas.

Se hace una limpieza bastante simple, se quitan ejemplos sin texto y se quita algún ejemplo que no corresponde con una enfermedad. Se eliminan también las características de sexo y edad por dos motivos: por homogenizar con los estudios anteriores y por valorar los resultados solo por que aporta los textos, si se obtienen buenos resultados es debido a que las soluciones de clasificación de textos son buenas.

Un estudio más detallado descubre un problema al que hay que enfrentarse. Se ve que hay 312 clases con un solo ejemplo y 535 clases con menos de 30 ejemplos, por lo que su valor estadístico es limitado y difícilmente se pueden encontrar modelos de comportamiento. El conjunto de datos está bastante desbalanceado. Hay algunas maneras de enfrentarse a este problema, una es llenar las clases con reposición aleatoria de ejemplos, otra es eliminar los ejemplos con escaso valor estadístico e incluso si se completa con ejemplos de las definiciones del propio catálogo CIE9. Para no caer en demasiado en el problema del sobreajuste se elige la opción de centrar el problema principalmente en las clases de más de 30 ejemplos. El conjunto de datos una vez hecha el ajuste quedaría de la siguiente manera.

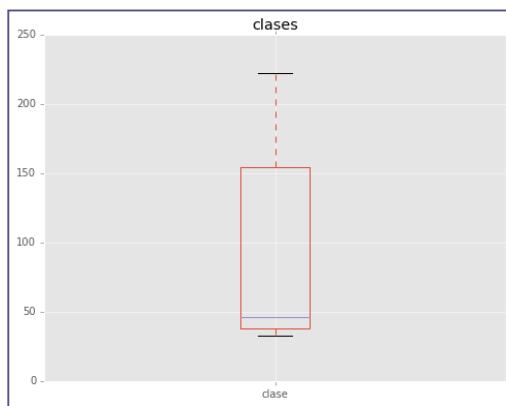


Ilustración 21. Box-plot Equilibrio de clases ajustadas.

**Reducción con punto de corte**

```
In [28]: #Ejemplo de preparación
#df_balance = prepara_ejemplos(corpus_textos_df)
#Quitamos los no significativos
corte = 30
df = frecuentes(corpus_textos_df,corte)

In [29]: df.clase.describe(include=['O'])

Out[29]: count    1516
unique     16
top      428
freq     222
Name: clase, dtype: object

In [30]: #np.mean(len(corpus_textos_df.labs))
print("Longitud media de frase: %.2f (+/- %.2f)\n" % (np.mean(df['words'].apply(len)),
                                                       np.std(df['words'].apply(len)) * 2))
fdist = acumula_fdist(df.words)
print ('Nº de palabras del corpus=', fdist.N())
print ('Nº de palabras del vocabulario=', fdist.B())
print ('Nº de palabras frecuencia 1=', len(fdist.hapaxes()))
fdist.plot(30)

Longitud media de frase: 4.52 (+/- 5.35)

Nº de palabras del corpus= 6852
Nº de palabras del vocabulario= 1284
Nº de palabras frecuencia 1= 812
```

Ilustración 22. Descripción conjunto ajustado.

De este modo se reduce el tamaño del dataset a 1.516 ejemplos y 16 clases, la distribución de palabras es muy parecida y las clases están algo más equilibradas. Este será el conjunto de datos de referencia. No obstante parece útil ver como varían los resultados en función del punto de corte (mínimo n° de ejemplos por clase), entonces se prueba con varios puntos de cortes (5, 10, 20, 30) que definen otros tantos datasets.

El hecho de tener un conjunto pequeño de datos dificulta entre otras muchas cosas, el como entrenar y el como validar los modelos. Hay que ver como evaluar y qué medidas se seleccionan para comparar. Se determina que para evaluar se utiliza como línea base la validación-cruzada de 5 pliegues. También es útil ver las tendencias de las otras alternativas ya que ayudan a entender la manera de comportarse y por eso también se hace una separación aleatoria del conjunto de datos en 80% para entrenamiento y un 20% para test. La medida a comparar será el % de acierto global (Accuracy).

### 3.3.2 TF-IDF.

El clasificador basado en la técnica tf-idf es la línea base de referencia donde comparar los demás clasificadores. Los principales motivos de su elección son: que hay muchos otros estudios que trabajan con esta técnica para transformar textos obteniendo buenos resultados y que tf-idf se emplea en alguna fase de la construcción del clasificador LDR.

Siguiendo con la intención de ser lo más generalista posible, se eligen las opciones y los parámetros más típicos para este tipo de clasificador. Se decide trabajar con bolsa de palabras y se quitan del vocabulario las "stop-word" para castellano, acentos y se pasa a minúsculas. A continuación se muestra el proceso de tokenizado utilizado en la construcción de todos los clasificadores.

#### Tokenizado

```
# Funciones para adecuacion de los tokenizadores
non_words = list(punctuation)
#Añadimos puntuacion castellano + letras sueltas + numeros
non_words.extend(['¿', '¡', '...', '.', '<br />', '<t />'])#puntuacion en español
non_words.extend(['a','b','c','d','e','f','g','h','i','j','k','l','m','n','ñ','o','p','q','r','s','t','u','v','w','x',''])
#non_words.extend(map(str,range(10000)))
non_words.extend(['de', 'por', 'con', 'en', 'la', '1.'])
spanish_stopwords = stopwords.words('spanish')
non_words.extend(spanish_stopwords)
def my_tokenize(text):
    text = text.lower()# pasa a minusculas
    text = ''.join([c for c in unicodedata.normalize('NFD',text) if unicodedata.category(c) != 'Mn'])#quita acentos
    tk = word_tokenize(text,language='spanish')# tockeniza castellano
    tk = [c for c in tk if c not in non_words] # Quitamos non_words
    return tk
```

Ilustración 23. Función de tokenizado

Tf-idf es un algoritmo de los llamados de conteo y como consecuencia el orden de las palabras no importa y no será sensible, ni a la morfología ni a la sintaxis de los textos. Gracias a la librería Sklearn de Python es muy sencillo transformar y clasificar con tf-idf. Una forma básica de hacerlo quedaría así:

#### Algoritmos estilo compacto

```
In [23]: # Pasos para clasificar
text_clf = Pipeline([
    ('tfidf', TfidfVectorizer(analyzer = 'word',min_df=1,tokenizer = my_tokenize)),
    ('scaler', StandardScaler(with_mean=False)),
    ('clf', SVC(probability = True,decision_function_shape='ovo') ),
])
```

#### Division en Train y Test

```
In [24]: # Separacion en Train y Test
dx_train, dx_test, dy_train, dy_test = train_test_split(corpus_textos_df.texto, corpus_textos_df.clase,
                                                       test_size=0.2, random_state=3)
text_clf.fit(dx_train, dy_train)
scores = text_clf.score(dx_test, dy_test)
print("Train-Split. Global accuracy : %0.2f \n" % (scores))
```

Train-Split. Global accuracy : 0.54

#### Validacion cruzada

```
In [25]: #Validacion cruzada
scores=cross_val_score(text_clf, corpus_textos_df.texto, corpus_textos_df.clase, cv=5)
print("Validacion cruzada. Global accuracy: %0.2f (+/- %0.2f)\n" % (np.mean(scores), np.std(scores) * 2))
Validacion cruzada. Global accuracy: 0.45 (+/- 0.05)
```

Ilustración 24. Código Python clasificador TF-IDF.

Es un claro ejemplo de lo mucho que facilita el trabajar con este tipo de librerías. Con unas simples líneas se ha transformado, normalizado, y clasificado con las dos posibles evaluaciones con división del conjunto y con validación cruzada. Se ha incluido una fase opcional de normalizar los vectores antes de pasarlos por el clasificador dado que en este caso al SVM le sienta bien esta normalización. También se han seleccionado los hiperparámetros adecuados para una salida de clasificador débil.

No se limitan las palabras por su frecuencia máxima y se hace un ajuste para las frecuencias mínimas. Para este caso particular tampoco se limita el máximo de palabras de la bolsa, dado que como máximo tenemos 1284 que es el número total de palabras diferentes del corpus. Esto supone eliminar un inconveniente que presenta este tipo de algoritmos, ya que en muchas ocasiones no se puede trabajar con todas las palabras o características, por problemas de potencia de cálculo.

Esta solución de código aunque muy compacta no la se puede utilizar en los otros clasificadores al no estar incluidos los algoritmos de transformación en la librería de Sklearn. Para equiparse la solución a los otros clasificadores se implementa otra versión de código algo más extendida pero aun así bastante reducida que sirve de plantilla en la construcción de los demás clasificadores.

```
Algoritmos estilo extendido 1
```

```
#Clases vectoriza y clasificador
vec = TfidfVectorizer(analyzer = 'word',min_df=1,tokenizer = my_tokenize,max_features = 4000 )
cls = SVC(probability = True,decision_function_shape='ovo')

Validacion cruzada

#Validacion cruzada
tiempo_ini = time.time()

scores =[]
kf = KFold(n_splits=5, shuffle=True, random_state=0)
print("Seleccion corte > ",corte," \n Clases : ",len(clases)," \n")
print("Modelo ",cls," \n Dataset : ",corpus_textos_df.shape," \n")

for train_index, test_index in kf.split(corpus_textos_df):

    # Separamos los conjuntos de Entrenamiento y Test
    df_train = corpus_textos_df.iloc[train_index]
    df_test = corpus_textos_df.iloc[test_index]

    X_train_denso = vec.fit_transform(df_train.texto)# Vectorizar
    X_train = X_train_denso.toarray()
    y_train = df_train['clase']#Encoding

    scaler = StandardScaler().fit(X_train)
    X_train = scaler.transform(X_train)# Normaliza
    cls.fit( X_train, y_train)# Entrena

    #Vectoriza tfidf textos con las caracteristicas del Train
    tfidf_test_denso = vec.transform(df_test.texto.values)
    #Vectoriza LDR textos con los pesos del train
    X_test = tfidf_test_denso.toarray()#Vectorizar
    X_test = scaler.transform(X_test) # Normaliza
    y_test = df_test['clase']#Encoding

    scores.append(cls.score(X_test, y_test))# Evalua

print("Vectoriza con tf-idf, Validacion Cruzada .Global Accuracy: %0.2f (+/- %0.2f) . %0.10f segundos.\n"
    % (np.mean(scores), np.std(scores) * 2,(time.time() - tiempo_ini))

Seleccion corte > 30
Clases : 16

Modelo SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
decision_function_shape='ovo', degree=3, gamma='auto', kernel='rbf',
max_iter=-1, probability=True, random_state=None, shrinking=True,
tol=0.001, verbose=False)
Dataset : (1516, 3)

Vectoriza con tf-idf, Validacion Cruzada .Global Accuracy: 0.47 (+/- 0.03) . 78.4069571495 segundos.
```

Ilustración 25. Código extendido clasificador TF-IDF

Se hace validación cruzada de 5 pliegues dado que se dispone de pocos datos y si se hacen muchos pliegues se reduce demasiado el conjunto de entrenamiento.

Aunque solo se expone la versión extendida para la validación cruzada se puede hacer de manera similar para la división train / test. Y como después hay que poder juntar todos los clasificadores en la implementación para la construcción del meta-clasificador final, hay que poder predecir con probabilidades.

El resultado de 47% aciertos de la versión extendida es la línea base donde comparar los demás clasificadores.

### 3.3.3 Clasificador LDR.

La propuesta de este clasificador viene porque puede aportar buenos resultados y porque es un clasificador emergente que no se ha aplicado antes a casos de uso como este. Además hay que sumar el hecho de que se busca una combinación de clasificadores de distinta naturaleza, seleccionando pues al LDR como el representante de tipo estadístico.

La base del LDR es el tf-idf que aporta todas sus bondades, pero el tf-idf tiene también desventajas. Cuando crece el corpus, crece el vocabulario y crece las dimensiones del vector y esto implica problemas de capacidad de proceso. Para evitar este problema se suele reducir la bolsa de palabras con el consiguiente riesgo de dejarse palabras que pueden ser significativas. El LDR hace una reducción de la dimensionalidad con lo que aguanta bien cuando los corpus crecen, pero además en la transformación tiene en cuenta todo el vocabulario para no perder poder de representación. Otra diferencia del LDR respecto a tf-idf, es que al transformar se tiene en cuenta las frecuencias de los términos en clases a predecir, por lo que los vectores representan además el comportamiento de las clases a predecir.

El conjunto de datos es pequeño con lo que no se aprecia mucho la ventaja de la reducción de dimensionalidad, pero seguramente le da más valor a los resultados obtenidos en caso de obtener mejoría. La implementación de las funciones particulares de esta técnica quedaría así:

```
Transformacion LDR

#Funcion que transforma al vector de caracteristicas por clase Fci(W)
def Vcc(vec,i,v_Td):
    r=np.zeros(6)
    r[0]=np.mean(vec) # Media
    r[1]=np.std(vec) # Desviación típica
    r[2]=np.min(vec) # Mínimo
    r[3]=np.max(vec) # Mínimo
    r[4]=np.mean(vec)/v_Td[i] # Prob
    r[5]=np.sum(len(vec))/v_Td[i] # Prop
    return r
#Definimos el numero de características
num_medidas = 6

# Funcion Vector LDR
def transform_ldr(a_tfidf,a_wtc,v_Td):
    #Matriz con los términos del vocabulario por documento
    a_vocablos= a_tfidf>0
    #Inicializamos un matriz vacía.
    a_ldr = np.zeros((a_vocablos.shape[0],num_medidas*a_wtc.shape[0]))
    for index,v_vocablos in enumerate(a_vocablos):
        if np.any(v_vocablos):
            v = np.empty(0)
            for v_wtc in a_wtc:
                v = np.hstack((v,Vcc(v_wtc[v_vocablos],index,v_Td)))
        else:
            v = np.zeros(num_medidas*a_wtc.shape[0])
        a_ldr[index]=v
    return a_ldr
    
$$d = \{F(c_1), F(c_2), \dots, F(c_n)\} \sim \forall c \in C,$$


# Funcion Transforms LDR
def fit_transform_ldr(dfx,a_tfidf):
    Wtc = Wtc_ldr(dfx,a_tfidf) # Calculo de la matriz de pesos W(t,c)
    X_vec = transform_ldr(a_tfidf ,Wtc ,dfx.Td.values ) # Vectoriza(reduce dimensionalidad) con LDR
    return X_vec,Wtc

# Funcion Transforma LDR
le = LabelEncoder()
def Wtc_ldr(dfx,a_tfidf):
    l_clases = le.fit_transform(dfx.clase)
    Y_clases = le.transform(dfx.clase)
    def Wtermino(a):
        return np.bincount(Y_clases, weights=a)/np.sum(a)
    Wtc = np.apply_along_axis(Wtermino, 0, a_tfidf) #Matriz con los pesos Wtc por término.
    return Wtc


$$W(t, c) = \frac{\sum_{d \in D / c = \delta(d)} w_{dt}}{\sum_{d \in D} w_{dt}}, \forall d \in D, c \in C$$

```

Ilustración 26. Funciones que implementan LDR

Para implementar el modelo LDR se sigue el esquema tipo planteado en la versión extendida del modelo TF-IDF añadiendo las particularidades del LDR y añadiendo lo necesario para poder predecir con probabilidades. Entonces hay que calcular previamente los tf-idf de todos los términos de los textos y ya se ha visto lo fácil y flexible que se hace con las librerías Sklearn. Posteriormente la función "fit\_transform\_ldr" calcula la matriz (Wtc), con las proporciones tf-idf por clase de cada término del conjunto de entrenamiento, que posteriormente se aplicara en la transformación del conjunto de test, y en base al Wtc se hace la trasformación al vector con las características por clase, tanto del conjunto de entrenamiento como del conjunto de test. Se normaliza el vector, se construye el modelo SVC, se evalúa con el número de aciertos y se registra las probabilidades de la predicción. Todo esto se hace con cada uno de los 5 pliegues. Se calcula un accuracy global como la media de los accuracy de cada pliegue. Se registra la predicción con probabilidades de todos los ejemplos. El código Python versión extendida quedaría:

```
Validacion cruzada

tiempo_ini = time.time()

df_probas = pd.DataFrame()
scores = []
kf = KFold(n_splits=5, shuffle=True, random_state=3)
print("Seleccion corte > ",corte," , N°Ejemplos : ",corpus_textos_df.shape[0]," ,N° Clases : ",len(corpus_textos_df.clase.unique))
print("Modelo ",cls," \n")

for train_index, test_index in kf.split(corpus_textos_df):
    # Separamos los conjuntos de Entrenamiento y Test
    df_train = corpus_textos_df.iloc[train_index]
    df_test = corpus_textos_df.iloc[test_index]

    tfidf_train_denso = vec.fit_transform(df_train.texto)# Vectorizar TF-IDF
    X_train,Wtc = fit_transform_ldr(df_train,tfidf_train_denso.toarray())# Vectorizar LDR

    y_train = df_train.clase
    scaler = StandardScaler().fit(X_train)
    X_train = scaler.transform(X_train)# Normaliza
    cls.fit( X_train, y_train)# Entrena

    tfidf_test_denso = vec.transform(df_test.texto.values)#Vectoriza tfidf textos con las caracteristicas del Train
    X_test = transform_ldr(tfidf_test_denso.toarray(),Wtc ,df_test.Td.values )# Vectoriza(reduce dimensionalidad) con LDR
    X_test = scaler.transform(X_test)# Normaliza
    y_test = df_test.clase
    scores.append(cls.score(X_test, y_test))# Evalua

    pred_prob = cls.predict_proba(X_test)# Prediccion con probabilidades
    df_probas = pd.concat([df_probas, (pd.DataFrame(pred_prob, columns = clases, index= test_index ))])

print("LDR, Validacion Cruzada .Nº caracteristicas %d. .Global Accuracy: %0.2f (+/- %0.2f) . %0.10f segundos.\n"
     "% (X_train.shape[1],np.mean(scores), np.std(scores) * 2,(time.time() - tiempo_ini))"

Seleccion corte > 30 , N°Ejemplos : 1516 ,N° Clases : 16

Modelo SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
decision_function_shape='ovo', degree=3, gamma='auto', kernel='rbf',
max_iter=-1, probability=True, random_state=None, shrinking=True,
tol=0.001, verbose=False)

LDR, Validacion Cruzada .Nº caracteristicas 96. .Global Accuracy: 0.55 (+/- 0.05) . 13.0064313412 segundos.
```

Ilustración 27. Solución clasificador LDR

El resultado con punto de corte de 30 ejemplos mínimo y la versión extendida de código, se obtiene un accuracy 55% mejorando los resultados de nuestro base line de 47%.

Otra manera de evaluar los resultados es, partiendo de las predicciones con probabilidades se calcula el top de aciertos según el número de las clases más probables. Por ejemplo un Top 3 77% significa que, dentro de los 3 CIE.9 etiquetados más probables de cada diagnóstico clínico se encuentra el 77% de las veces el CIE.9 correcto.

```

Top de Probabilidades

#Probabilidad con division
df_top = probabilidades(df_probas,corpus_textos_df)

TOP DE PROBABILIDADES DE PRDICCIONES
1.0      54.617414
2.0      71.306069
3.0      77.836412
4.0      82.387863
5.0      85.686016
6.0      87.730871
7.0      89.643799
8.0      91.886544
9.0      93.205805
10.0     94.986807
Name: top, dtype: float64

def probabilidades(df_probas,dfx):
    df_probabilidad=df_probas.copy()
    corpus_textos_test=dfx.copy()
    #Pivotamos para hacer el join
    df_probabilidad = df_probabilidad.unstack().reset_index()
    df_probabilidad.columns=['prediccion','id_diag','confianza']
    #df_probabilidad.query('confianza>0.0')
    #Preparamos la matriz de textos
    corpus_textos_test = corpus_textos_test.filter(items=['clase','texto'])
    #corpus_textos_test = corpus_textos_test.drop('ejemplo',axis = 1)
    corpus_textos_test = corpus_textos_test.reset_index()
    corpus_textos_test = corpus_textos_test.rename(columns = {'index':'ejemplo'})
    corpus_textos_test = corpus_textos_test.reset_index()
    corpus_textos_test = corpus_textos_test.rename(columns = {'index':'id_diag'})
    #Hacemos el Join de las probabilidades con los textos
    df_result = pd.merge(corpus_textos_test, df_probabilidad, how='inner', on=['id_diag', 'id_diag'])
    df_result = df_result.filter(items=['id_diag','texto','clase','prediccion','confianza'])
    df_result['top'] = df_result.groupby(['id_diag'])['confianza'].rank(ascending=False)
    total=len(df_result['id_diag'].unique())
    df_top=df_result.query('clase == prediccion')
    df_top=df_top['top'].value_counts()
    df_top.sort_index(inplace=True)
    df_top=(df_top.cumsum()/total)*100
    df_top=df_top.head(10)
    print ('TOP DE PROBABILIDADES DE PRDICCIONES \n',df_top)
    #DATOS GUARDAR TOP
    df_top=df_top.reset_index()
    df_top = df_top.rename(columns = {'index':'top_acc'})
    df_top = df_top.rename(columns = {'top':'accuracy'})
    return df_top

```

Ilustración 28. Top 10 de probabilidades

Con los DataFrame de la librería Pandas de Python es bastante fácil trabajar con los conjuntos de resultados registrados y el conjunto de diagnósticos iniciales. Es fácil fusionar, ordenar, filtrar e incluso grabar para usar posteriormente en el meta-clasificador de diagnósticos clínicos.

df_result.head(5)						
	id_diag	texto	clase	prediccion	confianza	top
0	0	NEUMONIA EN PACIENTE VIH	42	411.1	0.009699	15.0
1	0	NEUMONIA EN PACIENTE VIH	42	415.19	0.015960	11.0
2	0	NEUMONIA EN PACIENTE VIH	42	42	0.326240	1.0
3	0	NEUMONIA EN PACIENTE VIH	42	428	0.010783	14.0
4	0	NEUMONIA EN PACIENTE VIH	42	428.1	0.014541	13.0

**Exportar datos**

```

#Grabamos el fichero
df_result['clasificador'] = 'idc.ml.ldr'
df_result.to_csv('ldr.csv',columns=['clasificador','id_diag','clase','prediccion','confianza'],index=False)

```

Ilustración 29. Integración meta-clasificador.

### 3.3.4 Doc2Vec.

Las razones de la elección de este clasificador son parecidas a las del anterior clasificador, puede aportar buenos resultados, es relativamente nuevo y no es empleado todavía para clasificar diagnósticos clínicos. De naturaleza claramente diferenciada de los otros clasificadores del estudio, ya que se basa en la construcción de un espacio vectorial con técnicas de conocimiento profundo.

La gran diferencia con los otros clasificadores, TF-IDF y LDR, es que en Doc2vec hace un aprendizaje no supervisado para obtener un espacio vectorial. Este aprendizaje se basa en textos sobre el dominio del problema, diagnósticos clínicos en este estudio. Entonces se trata de incorporar otras fuentes de información que contengan textos de diagnósticos clínicos. En este caso se incorporan dos versiones diferentes del catalogo CIE9 (solo diagnósticos), una versión del catalogo CIE10 (solo diagnósticos), y una parte de la terminología SNOMED-CT (solo trastornos), todos ellos en su versión castellano. Vemos un resumen de cómo se implementa la incorporación de estos textos y con el apoyo de Pandas.

```

df_cie10 = df_cie10[df_cie10.index<80590]#Quita capitulos 20 y 21
df_cie10 = df_cie10[df_cie10['clase'].str.contains('Cap')==False]#Quita titulos capitulos
df_cie10 = df_cie10[df_cie10['clase'].str.contains('Tab')==False]#Quita titulo data set
df_cie10 = df_cie10[df_cie10['clase'].str.contains('-')==False]
df_cie10.head()

```

	clase	texto
3	A00	Cólera
4	A00.0	Cólera debido a Vibrio cholerae 01, biotipo cholerae
5	A00.1	Cólera debido a Vibrio cholerae O1, biotipo El Tor
6	A00.9	Cólera, no especificado
7	A01	Fiebres tifoidea y paratifoidea

```

# Terminología SNOMED-CT
df_SnomedCT = pd.read_csv('DescriptoresSnomedCT.csv', encoding='utf-8', sep=';', index_col=False, low_memory=False)
df_SnomedCT = df_SnomedCT.drop('LANGUAGECODE', axis = 1)
df_SnomedCT = df_SnomedCT[df_SnomedCT.DESCRIPTIONSTATUS!=8]
df_SnomedCT = df_SnomedCT[df_SnomedCT['TERM'].str.contains('enfermedad|trastorno')==True]#Quita titulos capitulos
df_SnomedCT['TERM'] = df_SnomedCT['TERM'].str.replace("\(\.\*\)\)", " ")
df_SnomedCT = df_SnomedCT.filter(items=['DESCRIPTIONID','TERM'])
df_SnomedCT = df_SnomedCT.rename(columns = {'DESCRIPTIONID':'clase'})
df_SnomedCT = df_SnomedCT.rename(columns = {'TERM':'texto'})
df_SnomedCT.texto.astype(str)
df_SnomedCT.head()

```

	clase	texto
21	845800010	enfermedad de Morquio
22	845797013	enfermedad de Morquio - Ullrich
49	845847018	necrobacirosis interdigital
93	845912013	lesión traumática superficial del tobillo, sin infección
101	845922019	hipertrofia del escroto

```

#catalogos
print ('Catalogo CIE9 Version 1 N° de registros',df_cat.shape[0])
print ('Catalogo CIE9 Version 2 N° de registros',df_cie9.shape[0])
print ('Catalogo CIE10 N° de registros',df_cie10.shape[0])
print ('Catalogo SNOMED-CT N° de registros',df_SnomedCT.shape[0])

Catalogo CIE9 Version 1 N° de registros 14733
Catalogo CIE9 Version 2 N° de registros 14376
Catalogo CIE10 N° de registros 80329
Catalogo SNOMED-CT N° de registros 90669

```

Ilustración 30. Incorporación de catálogos.

Se ve una gran diferencia de numero de registros entre el catalogo CIE.9 14 mil etiquetas y el CIE.10 80 mil o SNOMED-CT con 90 mil. Aunque parece que hay muchos ejemplos, son insuficientes ya que este algoritmo suele necesitar entrenar con gran cantidad de textos, del orden de millones de ejemplos.

Ahora se utiliza la librería de Gensim (from gensim.models import doc2vec) que tiene una implementación completa del algoritmo Doc2vec. Previamente es necesario un pre-procesado para aplicar esta técnica. El preprocessado consiste en etiquetar de manera única cada diagnóstico y en tokenizar los textos.

Preproceso para vectorizar					
	iejemplo	clase	texto	labs	words
72567	72824	T42.2X4D	Envenenamiento por succinimidas y oxazolidindionas, intencionalidad sin determinar, contacto sucesivo	[0_T42.2X4D_72824]	[envenenamiento, succinimidas, oxazolidindionas, intencionalidad, determinar, contacto, sucesivo]
73961	74218	T46.5X6D	Infradosisificación de otros fármacos antihipertensivos, contacto sucesivo	[0_T46.5X6D_74218]	[infradosisificacion, farmacos, antihipertensivos, contacto, sucesivo]
4038	43222	918395010	trastorno metabólico generalizado	[S_918395010_43222]	[trastorno, metabolico, generalizado]
75089	75346	T50.A15A	Efecto adverso de vacuna contra los ferina, incluyendo combinaciones con componente pertussis, contacto inicial	[0_T50.A15A_75346]	[efecto, adverso, vacuna, tos, ferina, incluyendo, combinaciones, componente, pertussis, contacto, inicial]
69137	69390	T23.599D	Corrosión de primer grado de localizaciones múltiples de muñeca y mano no especificadas, contacto sucesivo	[0_T23.599D_69390]	[corrosion, primer, grado, localizaciones, multiples, muneca, mano, especificadas, contacto, sucesivo]

Ilustración 31. Preparación Doc2vec

La descripción del conjunto de entrenamiento aporta información para poder hacer mejores ajustes.

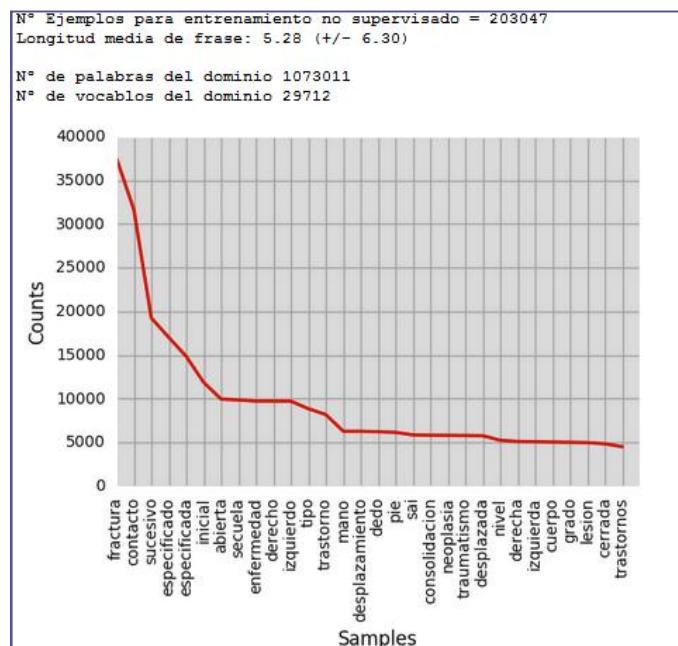


Ilustración 32. Descripción del corpus entrenamiento Doc2vec.

Para la construcción del modelo se hacen algunos ajustes de hiperparámetros como con el "min\_count" (frecuencia mínima de palabra) y con el "size" (número de características), pero también intentando mantener los valores más o menos comunes. Según sus autores se obtienen mejores resultados si se realizan varias pasadas y barajando los textos de ejemplo en cada pasada. Los tiempos de ejecución son en general bastante aceptables.

```
Vectoriza

1: def labs_d2v(X):
    for r in X.itertuples():
        yield doc2vec.LabeledSentence(r.words,r.labs)
num_medidas = 300

1: #Modelo para el campo vectorial
tiempo_ini = time.time()
df_vocabulario = shuffle(df_vocabulario)
train_voca = labs_d2v(df_vocabulario)
model = doc2vec.Doc2Vec(min_count=5, dm=1, dm_concat=1, size=num_medidas, window=16, iter=30
                       , negative=0, hs=1, workers=6, alpha=0.025, min_alpha=0.025)
model.build_vocab(train_voca)
model.train_words = True
model.train_lbls = True
for epoch in range(20):
    df_vocabulario = shuffle(df_vocabulario)
    train_voca = labs_d2v(df_vocabulario)
    model.train(train_voca)
    model.alpha -= 0.001 # decrease the learning rate
    model.min_alpha = model.alpha # fix the learning rate, no decay
print("Espacio Vectorial Doc2Vec %0.10f segundos.\n\n" % (time.time() - tiempo_ini))

Espacio Vectorial Doc2Vec 312.1826989651 segundos.

1: #Modelo para el campo vectorial
tiempo_ini = time.time()
df_vocabulario = shuffle(df_vocabulario)
train_voca = labs_d2v(df_vocabulario)
model_bow = doc2vec.Doc2Vec(min_count=5, dm=0, dm_concat=1, size=num_medidas, window=16, iter=30
                           , negative=0, hs=1, workers=6, alpha=0.025, min_alpha=0.025)
model_bow.build_vocab(train_voca)
model_bow.train_words = True
model_bow.train_lbls = True
for epoch in range(20):
    df_vocabulario = shuffle(df_vocabulario)
    train_voca = labs_d2v(df_vocabulario)
    model_bow.train(train_voca)
    model_bow.alpha -= 0.001 # decrease the learning rate
    model_bow.min_alpha = model_bow.alpha # fix the learning rate, no decay
print("Espacio Vectorial Doc2Vec %0.10f segundos.\n\n" % (time.time() - tiempo_ini))

Espacio Vectorial Doc2Vec 169.4236922264 segundos.

1: model_bow.save('Doc2Vec_cbow_c_d300_n5_w16_mc5_i30B.doc2vec')
#model.save('Doc2Vec_cdm_c_d300_n5_w16_mc5_i30B.doc2vec')
#model = doc2vec.Doc2Vec.load('Doc2Vec_cdm_c_d300_n5_w16_mc5_i30.doc2vec')
```

Ilustración 33. Construcción espacio vectorial Doc2Vec.

Se construyen los modelos con los dos planteamientos del Doc2Vec, el DM y DBOW (parámetro dm), según sus autores el DM es más costoso y más efectivo (es el valor por defecto), pero para sondar más posibilidades se incluyen además el DBOW. Los modelos se pueden guardar para usos posteriores o incluso para poder continuar el entrenamiento de modo incremental.

Una manera de comprobar lo ajustado del espacio vectorial resultante es sondar la similitud de palabras o diagnósticos. Gensim dispone del método "most\_similar" que indica las palabras o etiquetas más próximas dentro del espacio vectorial.

```

model.most_similar('tbc')
[('tuberculoma', 0.48186957836151123),
 ('tuberculosis', 0.43104490637779236),
 ('tuberculosa.bacilo', 0.3400164842605591),
 ('lobulo', 0.3354746103286743),
 ('ultima', 0.3276470899581909),
 ('centinela', 0.325431227684021),
 ('debajo', 0.3225518465042114),
 ('melicoidosis', 0.32137173414230347),
 ('or', 0.316356331100006),
 ('desmeliinizante', 0.3155888319015503)]

model.docvecs.most_similar(["D_550.9_2236"])
#model.docvecs.most_similar([inferred_vector], topn=len(model.docvecs))

[('D_188.9_170', 0.987333357334137),
 ('D_550.9_2234', 0.9844403266906738),
 ('D_491.21_1709', 0.983745276927948),
 ('D_550.9_2233', 0.9811376333236694),
 ('D_550.9_2235', 0.970273494720459),
 ('0_T50.Z92A_75486', 0.9575631618499756),
 ('S_1372335019_261571', 0.9541568756103516),
 ('C_642.93_7982', 0.9514926671981812),
 ('S_1360672012_742189', 0.9514385461807251),
 ('0_T42.6X4A_72923', 0.9496539831161499)]

```

Ilustración 34. Ejemplo de palabras y documentos similares.

Una vez que tenemos el modelo que representa el espacio vectorial, Gensim tiene la posibilidad de inferir sus vectores, ya sea partiendo de los términos de un diagnóstico o de sus etiquetas. Se implementa varias versiones de la función "transformación\_dos2vec" para probar varias opciones de vectores.

```

# Funcion Tranforma Doc2Vec a vector de etiquetas
def transform_doc2vec(dfx):
    #Inicializamos un matriz vacia.itertuples().iteritems() iterrows()
    X_vec = np.zeros((dfx.shape[0],num_medidas*4))
    for index, row in enumerate(zip(dfx.labs,corpus_textos_df.words)):
        X_vec[index]=np.concatenate((model.docvecs[row[0]][0],model.infer_vector(row[1])
                                    ,model_bow.docvecs[row[0]][0],model_bow.infer_vector(row[1]))
                                    ,axis=0)
    return X_vec

# Funcion Tranforma Doc2Vec a vector de etiquetas
def transform_doc2vec(dfx):
    nmedidas = (model_bow.docvecs[dfx.labs.iloc[0]].shape[1])*1
    X_vec = np.zeros((dfx.shape[0],nmedidas))
    for index, row in enumerate(zip(dfx.labs,dfx.words)):
        X_vec[index]=model_bow.docvecs[row[0]][0]
    return X_vec

```

Ilustración 35. Vectorización con Doc2vec.

Con la representación de los textos en vectores ya se puede aplicar el algoritmo de clasificación basado en SVM al conjunto preparado de diagnósticos. Se realizan experimentos con las diferentes versiones de código, con diferentes opciones de transformaciones para los diferentes puntos de corte.

### Algoritmos estilo extendido

```
cls = SVC(probability = True,decision_function_shape='ovo')

Validacion cruzada

tiempo_ini = time.time()
clases = corpus_textos_df.clase.unique()
df_probas = pd.DataFrame()
scores = []
kf = KFold(n_splits=5, shuffle=True, random_state=3)
print("Seleccion corte > ",corte," , N°Ejemplos : " ,corpus_textos_df.shape[0]," ,N° Clases :" ,len(corpus_textos_df))
print("Modelo ",cls," \n")

for train_index, test_index in kf.split(corpus_textos_df):
    # Separamos los conjuntos de Entrenamiento y Test
    df_train = corpus_textos_df.iloc[train_index]
    #df_train = prepara_ejemplos(corpus_textos_df.iloc[train_index])
    df_test = corpus_textos_df.iloc[test_index]

    X_train = transform_doc2vec(df_train)# Vectorizar Doc2Vec
    #scaler = StandardScaler().fit(X_train)
    X_train = scaler.transform(X_train)# Normaliza
    cls.fit( X_train, df_train.clase)# Entrena

    X_test = transform_doc2vec(df_test )#Vectorizar
    X_test = scaler.transform(X_test )# Normaliza

    scores.append(cls.score(X_test, df_test.clase))# Evalua

    pred_prob = cls.predict_proba(X_test )# Prediccion con probabilidades
    df_probas = pd.concat([df_probas, (pd.DataFrame(pred_prob, columns = clases, index= test_index ))])

print("Doc2Vec, Validacion Cruzada .N° caracteristicas ",df_probas.shape[1],".Global Accuracy: ",np.mean(scores), "+/- ",np.std(scores) * 2,(time.time() - tiempo_ini)))
print(" ",X_train.shape[1],np.mean(scores), np.std(scores) * 2,(time.time() - tiempo_ini)))
print(" ")

Seleccion corte > 30 , N°Ejemplos : 1516 ,N° Clases : 16

Modelo SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovo', degree=3, gamma='auto', kernel='rbf',
           max_iter=-1, probability=True, random_state=None, shrinking=True,
           tol=0.001, verbose=False)

Doc2Vec, Validacion Cruzada .N° caracteristicas 300. .Global Accuracy: 0.51 (+/- 0.03) . 17.2437276840 segundos.
```

Ilustración 36. Solución clasificador Doc2vec.

Los resultados para punto de corte 30 ejemplos como mínimo, con DM y concatenación de vector de palabras con el vector de documento, se obtiene un accuracy del 29%.

Con DBOW y con solo el vector de documento ,el resultado con punto de corte de 30 ejemplos mínimo y la versión extendida de código, se obtiene un accuracy 51% mejorando los resultados de nuestro base line de 47% y por debajo de LDR 55%.

También se puede evaluar el top de las predicciones con probabilidades.

```
#Probabilidad con division
df_top = probabilidades(df_probas,corpus_textos_df,'extendido')

TOP DE PROBABILIDADES DE PRDICCIONES
1.0      48.812665
2.0      64.907652
3.0      72.163588
4.0      76.846966
5.0      79.947230
6.0      82.255937
7.0      85.092348
8.0      87.664908
9.0      89.116095
10.0     91.094987
Name: top, dtype: float64
```

Ilustración 37. Top probabilidades Doc2vec.

Para comprobar el efecto que tiene el volumen de ejemplos empleados en el entrenamiento de los modelos de representación, se repite el experimento de modo incremental con los distintos conjuntos CIE.9, CIE.10 y SNOMED-CT, con la transformación de mejor resultado (DBOW y vector documento).

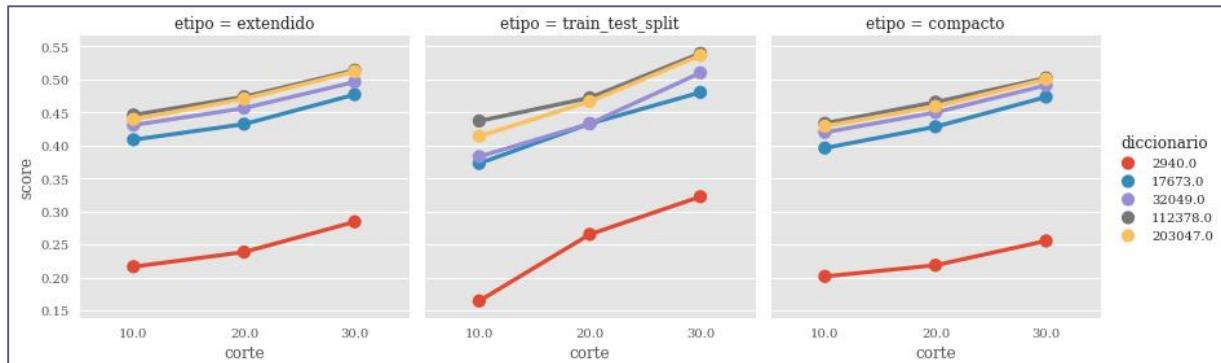


Ilustración 38. Evolución efecto tamaño del corpus en Doc2vec.

Se aprecia que si solo se entrena con los diagnósticos del conjunto de datos de color rojo con 2940 ejemplos o documentos, los resultados son muy pobres. También se ve que cuando se añaden los catálogos de CIE.9 (azul y violeta) con un total de 17.673 documentos, muestra una gran mejoría. Al añadir el catalogo CIE.10, de color gris con un total de 112.378 documentos, da un pequeño salto. Pero al añadir SNOMED-CT, de color amarillo con un total de 203.047 documentos no mejoran los resultados. Todo esto confirma que para trabajar con Doc2vec es necesario entrenar con gran numero de textos sobre el domino del problema.

En el supuesto de tener gran cantidad de diagnósticos la construcción del modelo se realizaría con la división del conjunto en entrenamiento y test. Por eso se plantea una versión de código con `train_test_split` de Sklearn.

```

Division en Train y Test
: dx_train, dx_test, dy_train, dy_test = train_test_split(corpus_textos_df, corpus_textos_df
                                                       , test_size=0.2, random_state=3)
X_train = transform_doc2vec(dx_train) # Vectorizar Doc2Vec
scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train) # Normaliza
cls.fit( X_train, dy_train.clase) # Entrena
X_test = transform_doc2vec(dx_test )#Vectorizar
X_test = scaler.transform(X_test) # Normaliza
scores = cls.score(X_test, dy_test.clase) # Evalua
print("Train-Split .Nº características %d. Global accuracy : %0.2f \n" % (X_train.shape[1],scores))
Train-Split .Nº características 300. Global accuracy : 0.53

: #Probabilidad con division
predicted = cls.predict_proba(X_test)
df_probabilidad = pd.DataFrame(predicted, columns=clases)
df_top = probabilidad(df_probabilidad,corpus_textos_df,'train_test_split')

TOP DE PROBABILIDADES DE PRDICCIONES
1.0      16.776316
2.0      25.986842
3.0      31.907895
4.0      38.486842
5.0      46.381579
6.0      50.000000
7.0      56.578947
8.0      62.171053
9.0      68.421053
10.0     74.013158
Name: top, dtype: float64

```

Ilustración 39. Solución `train_test_split` con Doc2vec.

### 3.3.5 Resultados comparación de clasificadores

Para la comparación de los diferentes clasificadores se realizan los diferentes experimentos con los tres clasificadores. Doc2Vec color rojo, TF-IDF color azul y LDR color violeta. Se evalúan tres tipos de código:

- Extendido. Validación cruzada con bucle sobre KFold de sklearn
- Compacto. Validación cruzada con cross\_val\_score y cross\_val\_predict de sklearn.
- Train\_test\_split. Separación del conjunto con 80% train y 20% test.

Se evalúan 4 puntos de cortes (5, 10, 20, 30) diferentes que determinan el número mínimo de ejemplos que debe de tener una clase para que sus ejemplos sean tenidos en cuenta.

La línea base para la comparación se fija con la versión extendida de código y con punto de corte 30. La versión train\_test\_split no está indicada para evaluar dado el bajo número de ejemplos, pero si sirve para ver las tendencias. El código tipo compacto no está disponible en todos los algoritmos e igualmente refleja las tendencias.

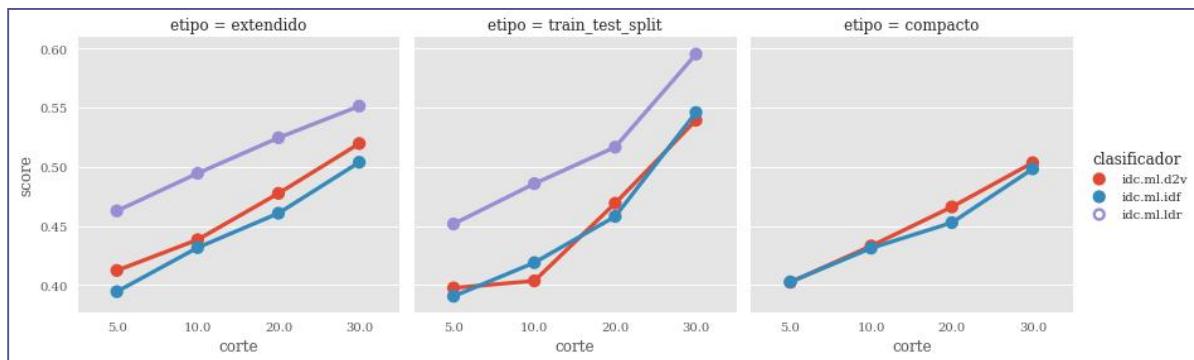


Ilustración 40. Comparativa de resultados.

Todos los clasificadores empeoran cuando aumentamos el número de clases a predecir y esto pasa a pesar de incrementar también el número de ejemplos. Esto sucede por dos motivos, por el desequilibrio entre clases y por el pequeño tamaño del conjunto de datos disponible para entrenar. El LDR se comporta siempre como el mejor con un paralelismo al resto de modelos especialmente al TF-IDF. El Doc2vec se comporta ligeramente mejor que la línea base TF-IDF.

También se puede comparar con la evolución de las curvas top de los clasificadores para el corte 30 que implica 16 clases a predecir. Siguen los tres la misma tendencia.

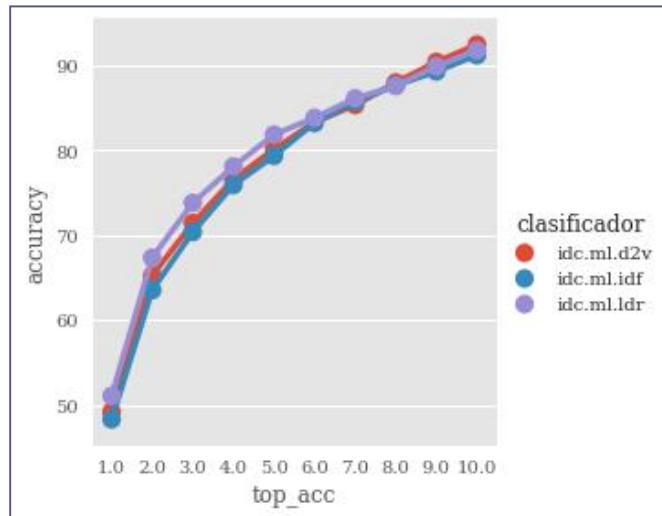


Ilustración 41. Comparativa línea base Top de clasificadores.

Se puede hacer la comparativa con todos los puntos de corte.



Ilustración 42. Comparativa Top tipo extendido.

Con el apoyo de la librería Seaborn se ve algo que no se ha podido ver en la comparativa de resultados (fig.40) y es que para el punto de corte 5, el cual maximiza el número de ejemplos, su capacidad de predicción es muy baja. Esto confirma la imposibilidad de obtener modelos representativos con el conjunto original completo. Los algoritmos de construcción de modelos ajustan con la predicción de la clase mayoritaria y así maximiza su porcentaje de aciertos. Cuando se predice con probabilidades ya no se puede realizar el ajuste expresando así mejor la capacidad real predictiva de los modelos.

### 3.4 Propuesta de combinación de clasificadores

El cómo combinar un conjunto de clasificadores de diagnósticos clínicos es una problemática que interesa resolver. Como en este estudio, en muchas ocasiones hay la disyuntiva de ver cómo combinar unos clasificadores ya existentes con otros creados nuevos, incluso dándose la circunstancia de tener que combinar algún modelo que no proviene de técnicas de aprendizaje automático. El método de combinación, el número de ejemplos y el número de clasificadores marcan el tamaño del problema. La combinación de un conjunto de clasificadores puede llegar a necesitar grandes capacidades de cálculo, y por eso pensamos que las técnicas Big Data nos puede ayudar en este asunto. Siendo esta circunstancia una de las causas determinantes en la selección del caso de uso para nuestro estudio.

Se ha definido el requisito de que todos los clasificadores a combinar sean débilmente correlacionados, lo que implica predecir con probabilidades por clase. Aplicamos el método de nivel de medidas con la regla de fusión de suma ponderada. La motivación de la elección viene porque se obtiene una mejora respecto a las predicciones individuales, se evita el sobre ajuste y principalmente porque es la manera de poder compararse con algo. En este caso se puede comparar con los resultados del estudio antecedente punto de partida de este.

Hay que codificar diagnósticos clínicos. Esta codificación se hace por medio de los tres clasificadores iniciales ya comentados en los antecedentes de este estudio. Son clasificadores de principios variados, uno basado en aprendizaje automático con ganancia de términos, y los otros dos basados en recuperación de información. En un estudio inicial se constató que la eficacia de estos codificadores es también variada, dos funciona bastante bien y el otro acierta poco aunque muy discriminante. Para equilibrar las decisiones se opta por ponderar con pesos su aportación a la codificación final. La ponderación se puede hacer: de manera heurística y elegir los pesos por algún criterio, por ejemplo, su proporción en una matriz de confusión, o se puede hacer algún tipo aprendizaje, o incluso hacer aprendizaje por fuerza bruta. En cualquier caso el objetivo sería el mismo, conseguir los pesos adecuados que hagan funcionar mejor al conjunto de codificadores.

Pero que hagan funcionar mejor ¿cuándo? y ¿bajo qué circunstancias?, los diagnósticos clínicos no siempre se comportan igual a lo largo del tiempo, cada médico puede expresar el diagnóstico de manera diferente, hay temporalidad de enfermedades o trastornos y hay cambios en los medios diagnósticos. Una manera de adaptarnos al cambio es recalcular los pesos o los modelos periódicamente. Cada cuanto se repite el calibrado, si se hace calibrado de pesos, o si se hace recálculo de modelos, vendrá determinado por la viabilidad de sus procesos asociados. Es aquí donde las técnicas Big Data puede ser un gran aliado, abriendo un abanico de nuevas posibilidades. Se puede marcar un objetivo de calibrado diario a modo de referencia.

Gracias al Big Data y sus grandes capacidades de cálculo podemos usar sin miedo la técnica de ponderación de aprendizaje por fuerza bruta, probando la mayor cantidad de combinaciones posibles de pesos y seleccionando la combinación que de mejor resultado es decir el mejor porcentaje de aciertos (Accuracy).



### 3.4.1 Datos

Hay que conocer el punto de partida y el conjunto de datos. Primero se ve la información disponible sobre el antecedente que sirve de referencia.

Con un conjunto de 10.000 historias clínicas y 3 clasificadores se obtiene los siguientes resultados:

Combinación	<i>icd.ml.WekaClassifier</i>	<i>icd.nlp.SimilarDiagnoses</i>	<i>icd.search.Searcher</i>	Accuracy
<i>Voto simple</i>	1	1	1	65,28%
<i>icd.ml.WekaClassifier</i>	1	0	0	65,18%
<i>icd.nlp.SimilarDiagnoses</i>	0	1	0	59,98%
<i>icd.search.Searcher</i>	0	0	1	1,10%
<i>LÍNEA BASE</i>	1	0,3	0,3	68,88%

Ilustración 43. Resultados ponderación clasificadores original.

Se confirman varios hechos esperados:

- El codificar con buscador sobre el catálogo CIE.9 (*icd.search.Searcher*) es muy ineficiente.
- El codificador aprendizaje automático (*icd.ml.WekaClassifier*) es el mejor de los tres.
- El codificar con buscador sobre los propios diagnósticos (*icd.nlp.SimilarDiagnoses*) tiene un buen rendimiento.
- La combinación de clasificadores por voto simple mejora a cualquier clasificador.

Hay una combinación por voto ponderado que mejora al voto simple. Esta será la línea base referencia donde compararse 68,88%.

Los datos disponibles son las 10.000 historias clínicas del antecedente debidamente anonimizados.

```
# Vision general del fichero
with open(fichero_ini) as fh:
    id_list = [x.split() for x in fh.read().split('\n')[:-1]]
id_list[0:10]
```

```
[['CLASE:', '003.0'],
 ['icd.ml.WekaClassifier', '558.9', '0.4886313'],
 ['icd.ml.WekaClassifier', '003.0', '0.43409505'],
 ['icd.ml.WekaClassifier', '008.43', '0.026652254'],
 ['icd.ml.WekaClassifier', '599.0', '0.010536384'],
 ['icd.ml.WekaClassifier', '787.91', '0.029289527'],
 ['icd.ml.WekaClassifier', '519.8', '0.0107954545'],
 ['icd.search.Searcher', '003.1', '0.2451126'],
 ['icd.search.Searcher', '003.0', '1.0'],
 ['icd.search.Searcher', '58.4', '0.25488737']]
```

Ilustración 44. Visión de datos para ponderación de clasificadores.

Transformamos, mediante una función de carga, la información del fichero original a un DataFrame de pandas con un formato más adecuado a nuestra forma de trabajar. Al mismo tiempo etiquetamos los diferentes diagnósticos con un identificador único.

Funcion Carga	
<pre>import pandas as pd import string #Funcion para la carga de datos def carga_datos(filename):     ##### CARGA #####     tiempo_ini = time.time()     df = pd.DataFrame(columns=('id_diag','clasificador', 'prediccion','confianza','clase'))     inputFile = open(filename, "r")     list_d = []     id_d = 0     aux=[]     for line in inputFile.read().split('\n')[:-1]:         aux = line.split(" ")         if aux[0].startswith('CLASE'):             id_d+=1             clase= id_d, aux[1]         else:             list_d.append ({'id_diag': clase[0],                            'clasificador': aux[0],                            'prediccion' : aux[1],                            'confianza' : float(aux[2]),                            'clase' : clase[1]})      inputFile.close()     df = pd.DataFrame(list_d)     print ('Fichero:',filename)     print ('Total de ids',len(df))     print("Tiempo Carga "+filename+": %.10f segundos.\n\n" % (time.time() - tiempo_ini))     return df</pre>	
CARGA DATOS	
<pre># Carga de los datos en un data frame de pandas clasificaciones = carga_datos(fichero_ini)  Fichero codificacion.txt Total de ids 492790 Tiempo Carga codificacion.txt: 1.2839467525 segundos.</pre>	

Ilustración 45. Carga de predicciones de clasificadores.

Después de cargar y transformar el fichero, vemos el aspecto del DataFrame y algunos datos relevantes, descriptivos estadísticos y visuales.

clasificaciones.head(2)	clasificaciones['confianza'].describe()																		
<table border="1"> <thead> <tr> <th>clase</th> <th>clasificador</th> <th>confianza</th> <th>id_diag</th> <th>prediccion</th> </tr> </thead> <tbody> <tr> <td>0 003.0</td> <td>id3ml.WekaClassifier</td> <td>0.483631</td> <td>1</td> <td>558.9</td> </tr> <tr> <td>1 003.0</td> <td>id3ml.WekaClassifier</td> <td>0.434095</td> <td>1</td> <td>003.0</td> </tr> </tbody> </table>	clase	clasificador	confianza	id_diag	prediccion	0 003.0	id3ml.WekaClassifier	0.483631	1	558.9	1 003.0	id3ml.WekaClassifier	0.434095	1	003.0	<pre>count      492790.000000 mean       0.189877 std        0.289719 min        0.000231 25%        0.012441 50%        0.050000 75%        0.200564 max        1.000000 Name: confianza, dtype: float64</pre>			
clase	clasificador	confianza	id_diag	prediccion															
0 003.0	id3ml.WekaClassifier	0.483631	1	558.9															
1 003.0	id3ml.WekaClassifier	0.434095	1	003.0															
clasificaciones.describe(include=['O'])	<pre>#quitamos lo que no son diagnosticos df = clasificaciones.query('clase != "PENTALODAR"') df = df.query('clase != "INFECCION"') df = df.query('clase != "")'</pre>																		
<table border="1"> <thead> <tr> <th>clase</th> <th>clasificador</th> <th>prediccion</th> </tr> </thead> <tbody> <tr> <td>count</td> <td>492790</td> <td>492790</td> </tr> <tr> <td>unique</td> <td>299</td> <td>3</td> </tr> <tr> <td>top</td> <td>486</td> <td>id3lp.SimilarDiagnoses</td> </tr> <tr> <td>freq</td> <td>51060</td> <td>221945</td> </tr> <tr> <td></td> <td></td> <td>16005</td> </tr> </tbody> </table>	clase	clasificador	prediccion	count	492790	492790	unique	299	3	top	486	id3lp.SimilarDiagnoses	freq	51060	221945			16005	<pre>#funcion histograma dataset def histograma(df):     n, bins, patches = plt.hist(dfy.astype(float),bins=100,range=[0,1000], facecolor='blue', alpha=0.75)     plt.xlabel('Clases')     plt.ylabel('ejemplos')     plt.ylim(0, 100000)     plt.title('Histograma DataSet')     plt.text(20, 60000, r'f(x)=\mu+\sigma(np.mean(n))+/\sqrt{np.std(n)}*t\$')     plt.grid(True)     plt.show()</pre>
clase	clasificador	prediccion																	
count	492790	492790																	
unique	299	3																	
top	486	id3lp.SimilarDiagnoses																	
freq	51060	221945																	
		16005																	
<pre>#clasificaciones.info() clasificaciones.info(memory_usage='deep')</pre>	<pre>['PENTALODAR', 'INFECCION', '510.0', '510.11', '510.00', '510.04', '510.03']</pre>																		

Ilustración 46. Fragmentos de código. Descripción del conjunto de datos de ponderación.

Los diagnósticos corresponden a 299 clases diferentes pero se han clasificado en 1.742 clases diferentes, esto implica que tenemos de entrada más de 1.400 diagnósticos erróneos como mínimo. La confianza media es del 18% y esto indica que se clasifica con confianzas bajas. El DataFrame ocupa 102.9 Mb con lo que no tendremos problema de volumen. Hay clases que no son códigos CIE.9 y esto debería implicar un proceso de limpieza, pero para poder compararse se mantienen los datos como son originalmente. En las gráficas siguientes se muestra como están de equilibrados los ejemplos respecto a los 3 clasificadores. Además, se muestra la distribución de las clases tanto en las reales como en las clasificadas.

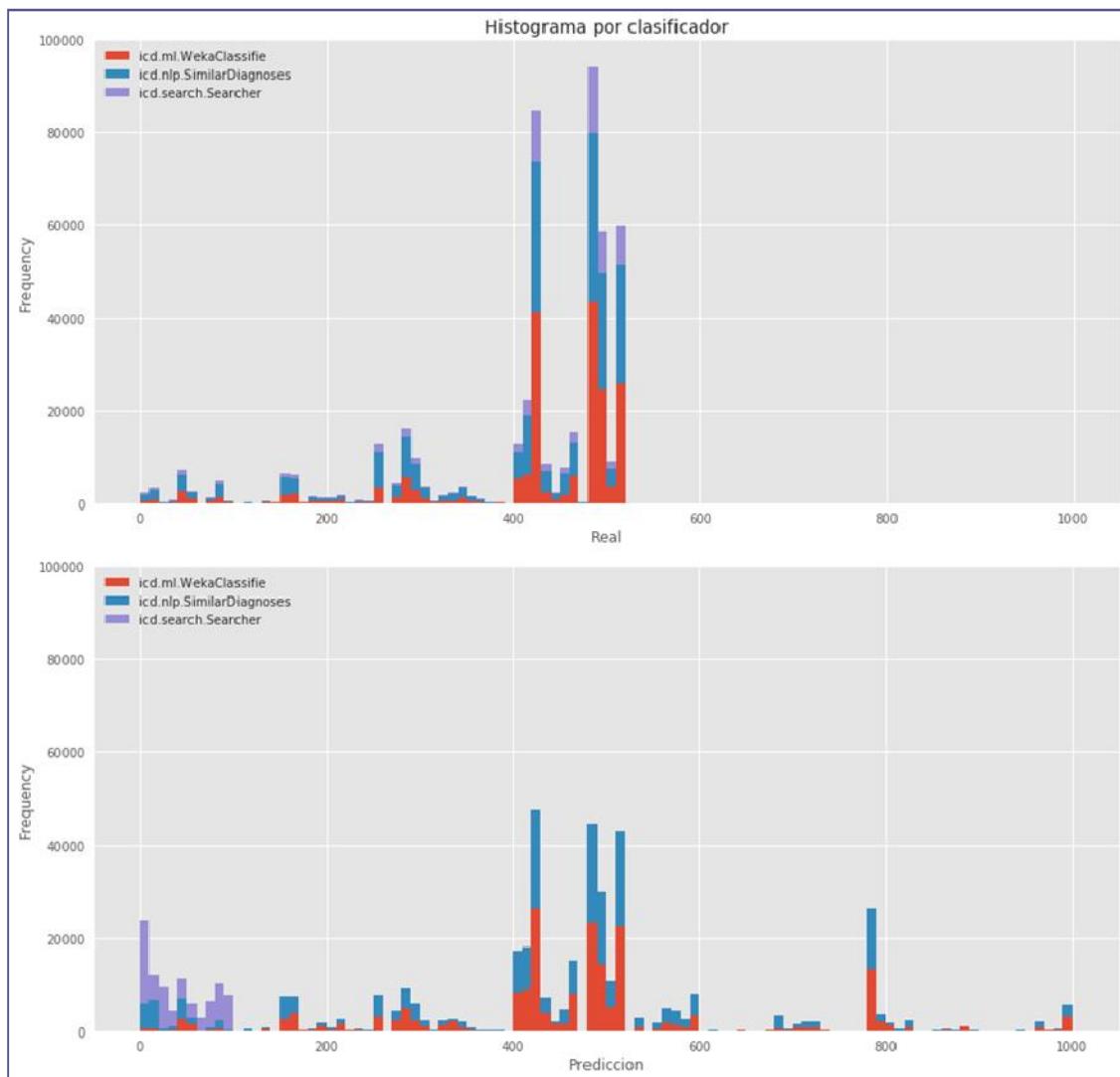


Ilustración 47. Comparación de los histogramas real y predicción.

Visualmente se ve un desequilibrio en la forma de codificar, el buscador sobre CIE.9 (icd.search.Searcher) casi siempre predice en los diagnósticos de enfermedades infecciosas, códigos CIE.9 menores de 100. Hay predicciones por encima del código 500 y en realidad no hay ningún diagnóstico con esa codificación.

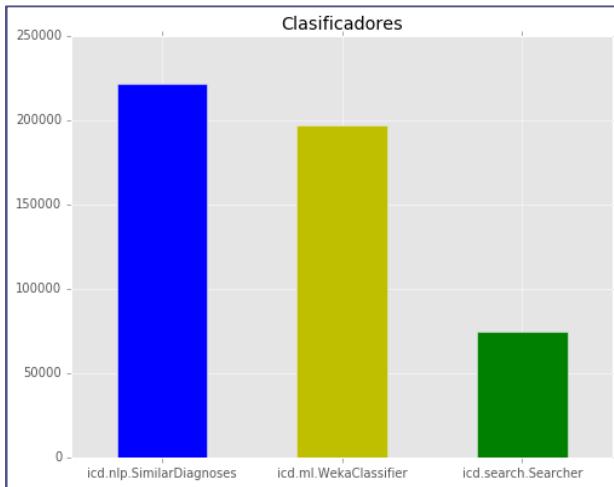


Ilustración 48. Equilibrio entre clasificadores.

Respecto al número de ejemplos el conjunto de datos presenta desequilibrios. Hay pocos ejemplos del clasificador `icd.search.Searcher` respecto a los otros dos clasificadores. Una vez que ya se conoce como son los datos ya se puede trabajar en las posibles soluciones al problema de la combinación de clasificadores.

### 3.4.2 Local .Python

Buscar inicialmente una solución local, que no implique necesariamente tener un cluster, es conveniente por varias razones. Por un lado, es una manera de empezar con una versión simplificada del problema y puede servir también de guía para la versión distribuida. Además hay que justificar la necesidad de la opción distribuida en función del tamaño del problema. Por ello, hay que medir, evaluar y encontrar los límites de la opción "Local" y comparar con la opción distribuida.

Para este apartado se sigue trabajando con Notebook-Python y con las librerías de Pandas, Numpy y Matplot. En la medida de lo posible, se busca una programación funcional y evitando el recorrer todo el conjunto de datos por medio de bucles. De esta manera, se consigue una transición más fácil hacia el código Pyspark. Ahora, se muestra la función principal que aplica la función de fusión para unos pesos concretos obteniendo el porcentaje de aciertos.

```
#Funcion para calculo de accuracy para unos pesos concretos
def accuracy_peso(peso):

    ##### CARGA #####
    #tiempo_ini = time.time()
    df2 = clasificaciones.copy()
    ac = 0.0
    # Aplicamos los pesos a cada clasificador
    df2['clasificador'] = df2.clasificador.map({'icd.ml.WekaClassifier': peso[0],
                                                'icd.nlp.SimilarDiagnoses': peso[1],
                                                'icd.search.Searcher': peso[2]})

    # Multiplicamos por su peso
    df2['confianza'] = df2.confianza * df2.clasificador
    #quitar fila clasificador
    df2 = df2.drop(['clasificador'], axis=1)
    # Agrupamos por la predicción sumando sus confianzas
    df2 = df2.groupby(['id_diag','clase','prediccion']).sum() 
$$\alpha = \sum_{j=1}^m v_j w_{ji}$$

    # Quitamos los que no tienen confianza alguna
    df2 = df2.query('confianza>0')
    # Quitamos el multi-index para poder agrupar por donde nos interesa
    df2 = df2.reset_index()
    # Ordenamos para que quede la predicción de mayor confianza la primera del grupo
    df2 = df2.sort_values(['id_diag','confianza'],axis=0 , ascending=[True, False])
    # Agrupamos seleccionando la predicción de mejor confianza para cada diagnóstico
    df2 = df2.groupby(['id_diag','clase'])[['prediccion']].first()
    # Quitamos el multi-index
    df2 = df2.reset_index()
    # Calcula el accuracy para un peso determinado
    ejemplos = len(df2)
    aciertos = len (df2.query( 'clase == prediccion'))
    if ejemplos >0 :
        ac = aciertos/ejemplos
    else:
        ac = 0.0
    #print ('Pesos ',peso,' Accuracy=',ac*100 ,'%')
    #print("Tiempo calculo : %0.10f segundos.\n\n" % (time.time() - tiempo_ini))
    return ac * 100.0
```

Ilustración 49. Solución local fusión clasificadores.

Se comprueba una vez más las ventajas de Python que de una manera clara y concisa resuelve el algoritmo. Para mejorar la eficiencia del algoritmo se trabaja con la función de fusión de suma ponderada. Primero, para ver la validez del código se compara con los resultados válidos, ya conocidos del antecedente. Posteriormente, se realiza una aproximación del aprendizaje por fuerza bruta. Para ello, se evalúa con 3 clasificadores con pesos desde cero hasta uno, décima a décima, de este modo hay 11 diferentes. Finalmente, se hace el producto cartesiano de los pesos de tres clasificadores, con lo que hay 1.331 combinaciones a probar.

## Comprobar la validez del algoritmo

```

pesos_test=[[1.0,1.0,1.0]
            ,[1.0,0.0,0.0]
            ,[0.0,1.0,0.0]
            ,[0.0,0.0,1.0]
            ,[1.0,0.3,0.3]
            ,[0.0,0.0,0.0]]
```

```

tiempo_ini = time.time()
eval=[]
for x in pesos_test:
    eval.append([x[0],x[1],x[2],(accuracy_peso(x))])

print("Tiempo evaluacion : %0.10f segundos.\n\n" * (time.time() - tiempo_ini))

Tiempo evaluacion : 2.3885340691 segundos.
```

## Los datos coinciden bastante con los aportados como ciertos

```

evaluacion = pd.DataFrame(eval, columns=clasicadores+['confianza'])
#evaluacion2 = evaluacion.set_index(clasicadores)
evaluacion
```

	icd.ml.WekaClassifier	icd.nlp.SimilarDiagnoses	icd.search.Searcher	confianza
0	1.0	1.0	1.0	64.45000
1	1.0	0.0	0.0	65.15000
2	0.0	1.0	0.0	59.95000
3	0.0	0.0	1.0	1.11336
4	1.0	0.3	0.3	68.90000
5	0.0	0.0	0.0	0.00000

Ilustración 50. Comprobación de la validez del código de fusión.

## Aprendizaje

```

divisiones = 10
a=np.linspace( 0, 1, (divisiones+1) )
pesos=cartesian((a, a, a))
df_evaluacion = pd.DataFrame(pesos, columns=clasicadores)
df_evaluacion.tail(3)
```

	icd.ml.WekaClassifier	icd.search.Searcher	icd.nlp.SimilarDiagnoses
1328	1.0	1.0	0.8
1329	1.0	1.0	0.9
1330	1.0	1.0	1.0

```

# Esta forma clarifica a pesar que no es lo mas eficiente . Vectorizar siempre es mas eficiente
tiempo_ini = time.time()
df_evaluacion["confianza"] = df_evaluacion.apply(accuracy_peso, axis=1)
df_evaluacion = df_evaluacion.sort_values(['confianza'],axis=0 , ascending=[False])
print("Tiempo evaluacion : %0.10f segundos.\n\n" * (time.time() - tiempo_ini))
df_evaluacion.head(5)
```

Tiempo evaluacion : 570.9389340878 segundos.

	icd.ml.WekaClassifier	icd.search.Searcher	icd.nlp.SimilarDiagnoses	confianza
1245	1.0	0.3	0.2	69.00
1112	0.9	0.2	0.1	69.00
1113	0.9	0.2	0.2	68.95
1244	1.0	0.3	0.1	68.95
870	0.7	0.2	0.1	68.90

Ilustración 51. Aprendizaje de pesos.

Como mejor resultado se obtiene un accuracy de 69.0% que supera ya nuestra línea base de referencia. El consumo de tiempo ha sido de aproximadamente 10 minutos.

### 3.4.3 Tamaño de problema.

Se puede pensar que con el algoritmo Python ya se tiene resuelto el problema, pero ¿hasta dónde se puede mejorar?, ¿Se puede llegar a la centésima o a la milésima en los pesos?, ¿qué pasa si hay que combinar mas clasificadores?, ¿qué pasa si hay más de 10.000 historias clínicas para evaluar? Hay que intentar responder a todas estas cuestiones. Para ello, se realizan mediciones de las divisiones desde 2 hasta 10 y se registran los tiempos y se obtiene la grafica siguiente.

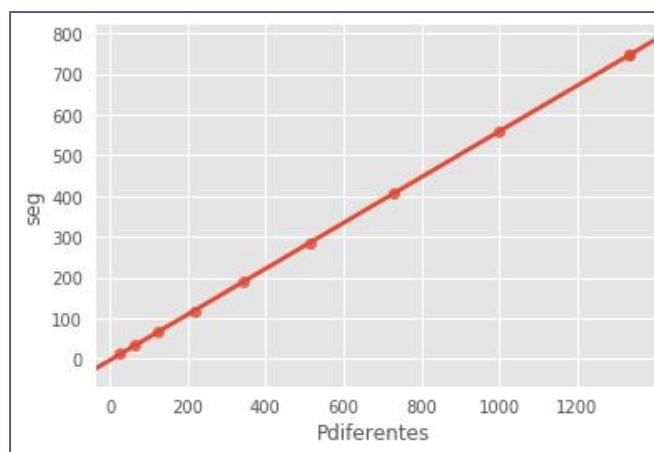


Ilustración 52. Segundos en función de nº combinaciones de pesos

El número de combinaciones frente al tiempo sigue una relación lineal. Entonces se calcula la pendiente de la recta y se predice cuanto tiempo tardaría en horas con mayores numero de combinaciones y si se extiende a 4 y 5 clasificadores. El resultado es que la extensión en número de diagnósticos es lineal y el doble de diagnósticos implica el doble de tiempo. Véase la tabla resumen:

Pendiente Recta	Espaciado	Combinaciones	Horas con 3Clasi.	Horas con 4Clasi.	Horas con 5Clasi.
0,5588	0,500	27	0,004	0,013	0,038
	0,333	64	0,010	0,040	0,159
	0,250	125	0,019	0,097	0,485
	0,200	216	0,034	0,201	1,207
	0,167	343	0,053	0,373	2,609
	0,143	512	0,079	0,636	5,086
	0,125	729	0,113	1,018	9,166
	0,111	1.000	0,155	1,552	15,522
<b>Decima</b>	<b>0,100</b>	<b>1.331</b>	<b>0,207</b>	<b>2,273</b>	<b>24,999</b>
	0,050	9.261	1,438	30,188	
	0,033	29.791	4,624		
	0,025	68.921	10,698		
	0,020	132.651	20,590		
	0,019	157.464	24,442		
<b>Centesima</b>	<b>0,010</b>	<b>1.030.301</b>	<b>159,926</b>		<b>1.631.401</b>

Ilustración 53. Duración prevista de la solución local.

De la tabla se desprende que con los tres clasificadores se llega rápidamente a un coste de cálculo de un día de duración, con cinco clasificadores la duración de 24 horas ya se alcanza con un espaciado en decimas. Por lo que se concluye que con este procedimiento trabajar con la centésima tiene un alto coste computacional.

Se propone una alternativa basada en el algoritmo Python ya expuesto. La alternativa parte del supuesto de que la combinación optima estaría en una única zona del espacio vectorial a probar. La idea consiste volver hacer 10 divisiones el espacio vectorial pero en esta ocasión de la zona delimitada por los mejores resultados obtenidos con el nivel a la de decima. El coste computacional es el doble del algoritmo con 1.331 combinaciones, unos 20 minutos según resultados. Se muestra como quedaría:

#### Afinamos por la zona de mas éxito

```
divisiones = 10
a=np.linspace( 0.9, 1.0, (divisiones+1) )
b=np.linspace( 0.2, 0.3, (divisiones+1) )
c=np.linspace( 0.1, 0.2, (divisiones+1) )
pesos=cartesian((a, b, c))
df_evaluacion = pd.DataFrame(pesos, columns=clasificadores)
df_evaluacion.tail(3)
```

	icd.ml.WekaClassifier	icd.search.Searcher	icd.nlp.SimilarDiagnoses
1328	1.0	0.3	0.18
1329	1.0	0.3	0.19
1330	1.0	0.3	0.20

```
# Esta forma clarifica a pesar que no es lo mas eficiente . Vectorizar siempre es mas eficiente
tiempo_ini = time.time()
df_evaluacion["confianza"] = df_evaluacion.apply(accuracy_peso, axis=1)
df_evaluacion = df_evaluacion.sort_values(['confianza'],axis=0 , ascending=[False])
print("Tiempo evaluacion : %0.10f segundos.\n\n" % (time.time() - tiempo_ini))
df_evaluacion.head(10)
```

Tiempo evaluacion : 604.3277890682 segundos.

	icd.ml.WekaClassifier	icd.search.Searcher	icd.nlp.SimilarDiagnoses	confianza
17	0.90	0.21	0.16	69.20
1130	0.99	0.23	0.18	69.20
513	0.94	0.22	0.17	69.20
139	0.91	0.21	0.17	69.20
138	0.91	0.21	0.16	69.20
635	0.95	0.22	0.18	69.20
634	0.95	0.22	0.17	69.20
18	0.90	0.21	0.17	69.20
1009	0.98	0.23	0.18	69.20
11	0.90	0.21	0.10	69.15

Ilustración 54. Ponderación fina.

El resultado obtenido del 69,2 % de aciertos, mejora la línea base 68,88%

Podemos repetir de nuevo el algoritmo llegando a la milésima en los pesos, pero no se mejora los resultados. Hemos llegado ya a la mejor ponderación para el conjunto de diagnósticos clínicos.

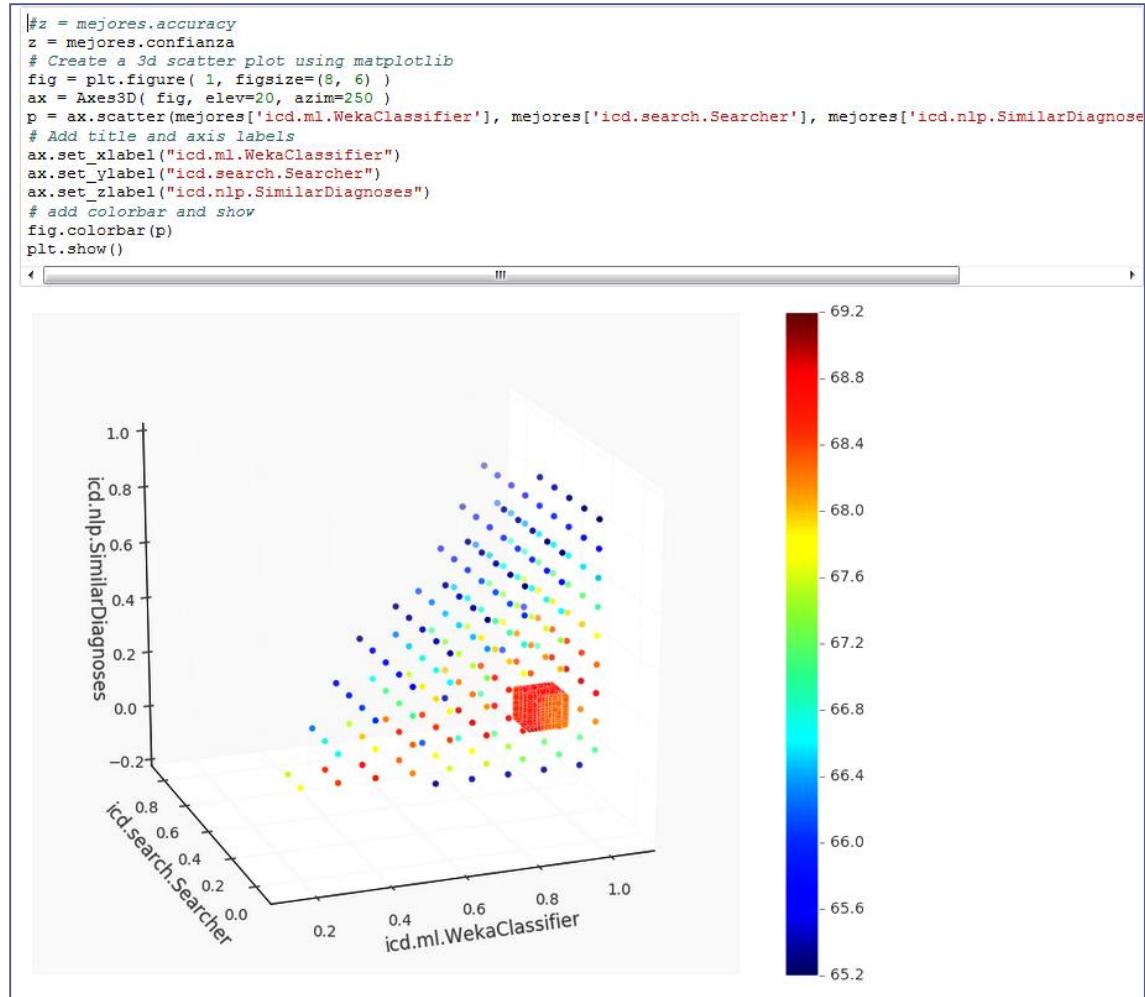


Ilustración 55. Accuray del espacio vectorial testeado.

La técnica de refinamientos sucesivos puede ser una buena solución, aunque mejorable, es fácil de aplicar, con buenos resultados y con una gran reducción de coste computacional. Con todo, si se aumenta el número de clasificadores a 5, el coste de cálculo en la primera ronda ya es grande, siendo conveniente explorar alguna otra alternativa, tal vez alguna opción para entornos distribuidos.

### 3.4.4 Cluster Spark.

Una vez justificada la necesidad, llega el momento para las técnicas Big Data. Hay que pensar en el caso de uso llevado a máximos. Es factible tener que trabajar con grandes volúmenes de datos, por ejemplo en la Comunitat Valenciana los diagnósticos clínicos supera los 10.000 diarios. Adicionalmente es factible querer combinar 5 clasificadores, por ejemplo los tres de nuestro punto de partida más los dos nuevos construidos con Doc2vec y LDR. Se puede marcar como el nuevo problema hipotético a resolver. La opción de un cluster de procesamiento y almacenamiento es una alternativa excelente para enfrentarse a este tipo de problemas.

Y es aquí donde aparece  que resulta especialmente ventajoso, al facilitar la traslación de las soluciones de las propuestas del entorno local al distribuido. En Spark se puede programar en código Python, llamado Pyspark. Existen los dataframe conceptualmente similares a los dataframe de Pandas, además, existen librerías equivalentes de aprendizaje automático con implementaciones de los algoritmos SVM y Doc2Vec. También se puede desarrollar código con IPython Notebook localmente y ese mismo código ser válido para un entorno distribuido.

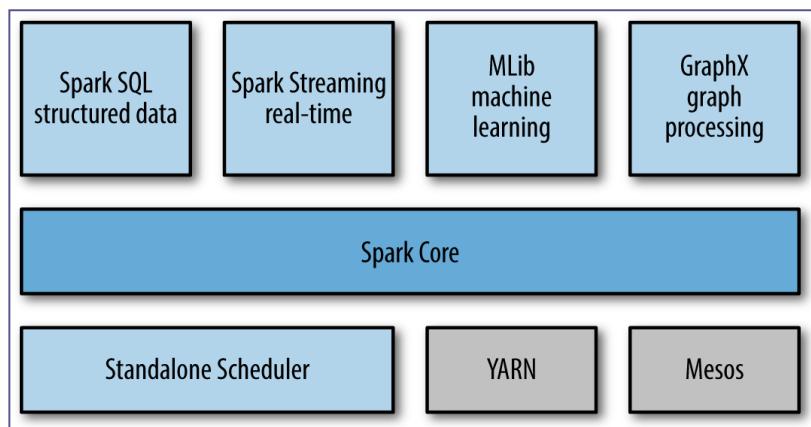


Ilustración 56. Gestores Spark

<http://www.hadooppoint.com/>

Para desarrollar código PySpark debemos tener muy en cuenta que se trabaja en memoria y entender dos de los pilares de Spark, los RDD (Resilient Distributed Dataset) y DAG (Directed Acyclic Graph).

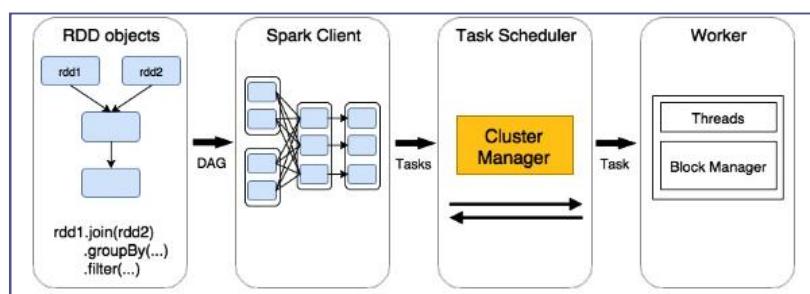


Ilustración 57. Planificación tareas Spark

<http://datastrophic.io>

A partir de una variable de entorno llamada "Context" se crea un objeto RDD leyendo datos de fichero, bases de datos o cualquier otra fuente de información. Una vez creado el RDD inicial se realizan transformaciones para crear más objetos RDD a partir del primero. Dichas transformaciones se expresan en términos de programación funcional, los RDD son inmutables no eliminan el RDD original, sino que crean uno nuevo. Tras realizar las acciones y transformaciones necesarias sobre los datos, los objetos RDD deben converger para crear en el RDD final. Este RDD puede ser almacenado en fichero y bases de datos. Los RDD se dividen en particiones que serán distribuidas por los nodos.

El DAG primero se analiza para determinar el orden de las transformaciones; con el fin de minimizar el mezclado de datos, se realizan las transformaciones narrow (filter, map) en cada RDD. Finalmente, se realiza la transformación wide (reduce, shuffle) a partir de los RDD sobre los que se han realizado las transformaciones narrow. Hay que favorecer las operaciones dentro de un mismo nodo ejecutor (Worker) y acercar el dato al cómputo.

```
Carga de los datos en un DF pandas
```

```
[1]: clasificaciones = carga_datos(fichero_ini)

Tiempo Carga codificacion.txt: 1.2591390610 segundos.Total de registros 492790
```

```
Crear el area de los contextos
```

```
[1]: from pyspark import SparkContext, SparkConf
from pyspark.sql import SQLContext
from pyspark.sql import HiveContext
tiempo_ini = time.time()
# HiveContext Hace falta para utilizar ventanas deslizantes. Es un superconjunto de SQLContext
conf = SparkConf().setMaster("local[8]")
conf = conf.setAppName('Eval_meta2')
sc = SparkContext(conf=conf)
sqlContext = HiveContext(sc)
print("Tiempo Crea SparkConf %.10f segundos.\n\n" % (time.time() - tiempo_ini))

Tiempo Crea SparkConf 2.0955872536 segundos.
```

Ilustración 58. Definición del área de Contexto.

Véase como se planifica el código para este caso concreto. Se está diseñando un algoritmo de aprendizaje por fuerza bruta en código PySpark que prueba muchas combinaciones de pesos con los tres clasificadores. La idea es que en cada Worker se realicen todas las operaciones sobre una o varias combinaciones de pesos, de tal manera que solo se necesite consolidar las diferentes particiones cuando se ordena para encontrar las mejores combinaciones de pesos. Para eso se pivota las combinaciones y se hace un "join" con las predicciones de los clasificadores que están en memoria de todos.

```
#Funcion para calculo combinaciones
def pivota_combinaciones(combinaciones):
    # Pivota a formato columna
    combinaciones = combinaciones.unstack().reset_index()
    combinaciones.columns=['clasificador','combinacion','peso']
    return combinaciones
```

Ilustración 59. Función para pivotado.

```

#Funcion para calculo combinaciones
def carga_combinaciones(divisiones,clasicadores):
    ##### COMBINACIONES Maximo 7 Clasificadores #####
    a=np.linspace( 0, 1, (divisiones+1) )
    nar=(a,a,a,a,a,a)
    pesos = cartesian((nar[:len(clasicadores)]))
    #pesos=np.array(np.meshgrid(nar[:len(clasicadores)])).T.reshape(-1,len(clasicadores)))
    if PESOS_PROPORCIONES:
        #Proporciones diferentes OPCION A
        pesos = pesos[~np.all(pesos == 0, axis=1)]
        proporciones = pesos.sum(axis=1)
        proporciones = proporciones[:, np.newaxis]
        proporciones = (pesos / proporciones)
    else:
        proporciones = pesos
    combinaciones = pd.DataFrame(proporciones, columns=clasicadores)
    combinaciones = pd.DataFrame.drop_duplicates(combinaciones)
    combinaciones = combinaciones.dropna(how='any')
    combinaciones = combinaciones.reset_index()
    print(" %i Combinaciones %i No repetidas.\n\n" % (pesos.shape[0],combinaciones.shape[0]) )
    return combinaciones

```

Ilustración 60. Función calculo de combinaciones.

El RDD del join en números aproximados seria, para la división del espacio en 10 divisiones y los tres clasificadores, unos 650 millones de registros, una manera de reducir esta cifras es no evaluar los pesos que representan las mismas proporciones entre clasificadores, ya que los resultados son los mismos.

#### Crea los DataFrame sql.pyspark

De las clasificaciones (los ejemplos) y de las combinaciones (los pesos a evaluar).

```

]: from pyspark.sql import DataFrame
tiempo_ini = time.time()
s_df = sqlContext.createDataFrame(clasificaciones).cache()
c_df = sqlContext.createDataFrame(combinaciones).repartition(40,'combinacion')
#c_df = sqlContext.createDataFrame(combinaciones).cache()
print("Tiempo Crea DF  clasificaciones y combinaciones %.10f segundos.\n\n" % (time.time() - tiempo_ini))

Tiempo Crea DF  clasificaciones y combinaciones 29.3321292400 segundos.

```

#### Join clasificaciones y combinaciones

```

]: tiempo_ini = time.time()
t2_df = c_df.join(broadcast(s_df), c_df.clasificador == s_df.clasificador).drop('clasificador')
print("Tiempo Crea DF  Join de todo %.10f segundos.\n\n" % (time.time() - tiempo_ini))

Tiempo Crea DF  Join de todo 0.1288392544 segundos.

```

```

]: tiempo_ini = time.time()
t2_df.show()
print("Tiempo Show (El DAG ejecuta ahora) %.10f segundos.\n\n" % (time.time() - tiempo_ini))

+-----+-----+-----+-----+-----+
|combinacion|peso|clase| confianza|id_diag|prediccion|
+-----+-----+-----+-----+-----+
|      26| 0.0|519.8|0.0010640819| 10000|   995.91|
|      26| 0.0|519.8|3.4965036E-4| 10000|   995.90|
|      26| 0.0|519.8|0.0025301764| 10000|   907.01|

```

Ilustración 61. Join de clasificaciones y combinaciones.

En la creación de los dataframe se puede indicar en un particionado para adecuarlo al tipo de workers y en función de la memoria y procesadores de cada nodo ajustarlo (40 en nuestro caso) cuando no se dice nada es el gestor el que decide con un valor por defecto de 200. También se puede forzar el uso en memoria con la opción cache. En el join con la opción broadcast de las clasificaciones mejora bastante su eficiencia.



Ahora se muestra como es la parte central del algoritmo en PySpark.

```
tiempo_ini = time.time()
# Multiplicamos por su peso y quitar columna clasificador
df = t2_df.withColumn("nconfianza", (t2_df.confianza * t2_df.peso) )
# Agrupamos por la prediccion sumando sus confianzas
df = df.groupby(['combinacion','id_diag','clase','prediccion']).agg(F.sum('nconfianza').alias('sconfianza'))
# Quitamos los que no tienen confianza alguna
df = df.filter(df.sconfianza > 0)
# Definimos la ventana de desplazamiento Ojo! como puede haber empates gana la mayor prediccion
windowSpec = Window.partitionBy(df['combinacion'],df['id_diag']).orderBy(F.desc('sconfianza'),'prediccion')
# Se calcula los ranking por diagnostico
df = df.select('combinacion','id_diag', 'clase', 'prediccion','sconfianza', F.dense_rank().over(windowSpec).alias('rnk'))
# Se selecciona el de mejor confianza
df = df.filter(df['rnk'] == 1).drop('rnk')
# Se seleccionan los aciertos
df = df.filter( 'clase = prediccion')
# Agrupamos por la prediccion sumando sus confianzas
df = df.groupby(['combinacion']).count()
# Multiplicamos por su peso y quitar columna clasificador
df = df.withColumn('accuracy',(df['count'] / ejemplos) .drop('count')
# Ordenar para sacar los maximos
df = df.orderBy(F.desc('accuracy')).head(50)
#df.show(30)
print("Tiempo evaluacion : %0.10f segundos.\n\n" % (time.time() - tiempo_ini))

Tiempo evaluacion : 0.3835554123 segundos.
```

Ilustración 62. Solución distribuida fusión de clasificadores.

Gracias a los DF de PySpark se mantiene la facilidad y claridad vista en Python. Se multiplica cada clasificación por el peso. Se agrupa y suman todas las confianzas de una misma clase y predicción. Se eliminan los que no tienen confianza. Se hace una ventana con las sumas ordenadas de confianzas por combinación y diagnostico. Se establece un ranking de confianzas y se filtra por el ganador. Se filtra cuando el ganador coincide con la predicción y se calcula el porcentaje de aciertos.

Spark puede mostrar de manera grafica el resultado de las operaciones del algoritmo.

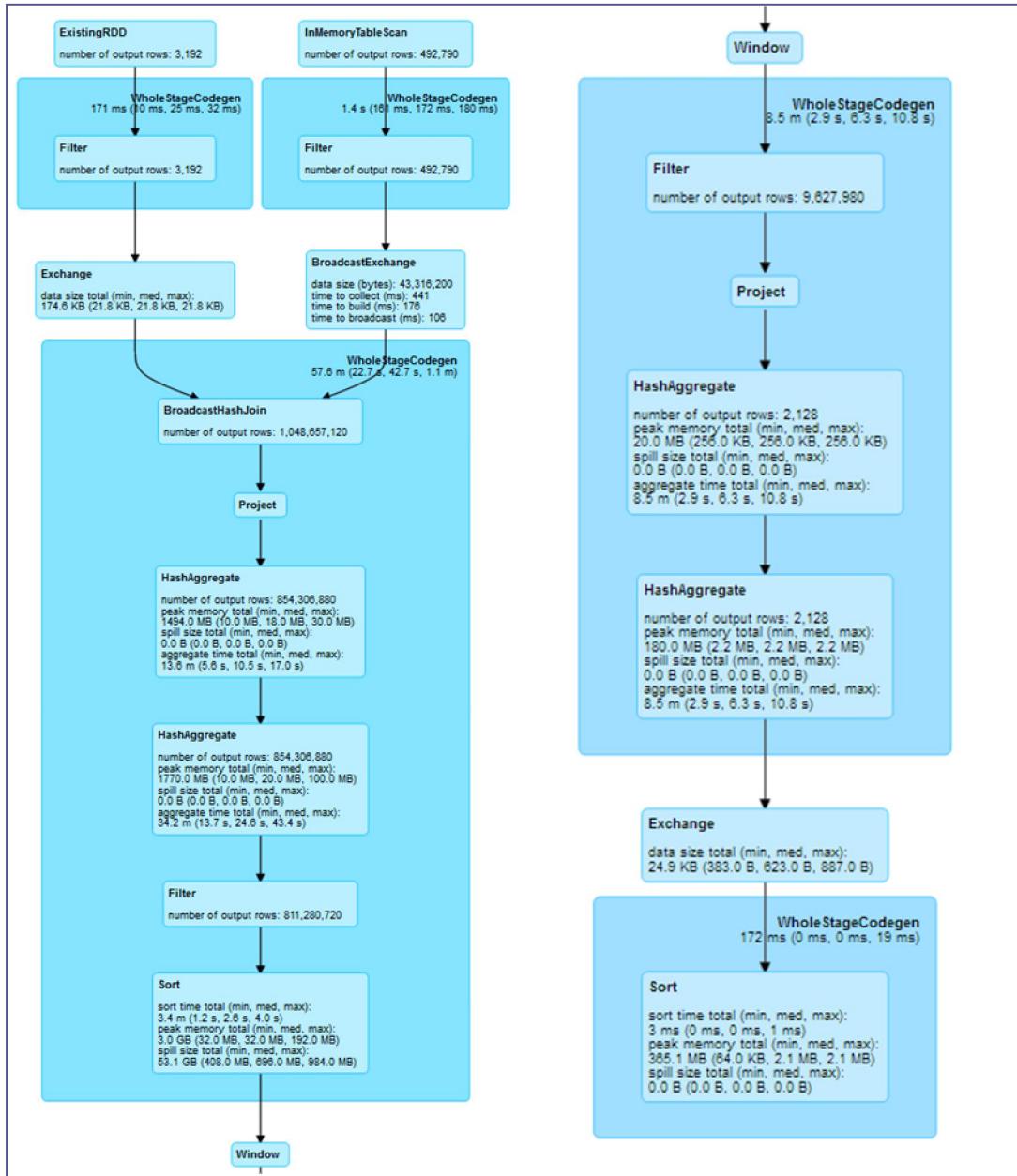


Ilustración 63. Representación Grafica de tareas con gestor Standalone Spark.

Se ve claramente los costes computacionales de cada operación, el flujo de trabajo y la planificación. Un punto clave para ajustar es la repartición del RDD de las combinaciones y evitar los habituales problemas con la gestión de memoria de las maquinas Java.

Con el código claro se procede de la misma manera que localmente, primero se comprueba la validez del código evaluando con pesos de resultados ya conocidos. Despues, cada eje del espacio vectorial total se divide en 10, se eliminan los de proporciones equivalentes, y entonces evaluamos. Por ultimo, se repite esto ultimo pero para el espacio temporal delimitado por la zona de más éxito. Los resultados coinciden con los obtenidos en la versión local Python, tanto en valor como en tiempo.

Tenemos código Python valido para ponderar un conjunto de clasificadores en un Cluster Spark. Ahora hay que ver todo lo que conlleva su aplicación en un Cluster Real.

### 3.4.5 Spark EMR AWS

Uno de los objetivos es tener una experiencia real en un Cluster Big Data. Una vez que ya se tiene el qué (nuestro algoritmo Pyspark) hay que ver el cómo. Ya se explico el porque de la elección de AWS EMR gracias al programa AWS Educate.

Una vez ya en AWS hay dos posibilidades claras:

- EC2 . Hay que hacer un despliegue y configuracion completa sobre maquinas EC2.
- EMR. De una manera facil se indica el hardware y los elementos software del ecosistema Big Data. El AWS hace el despliegue y configuracion.

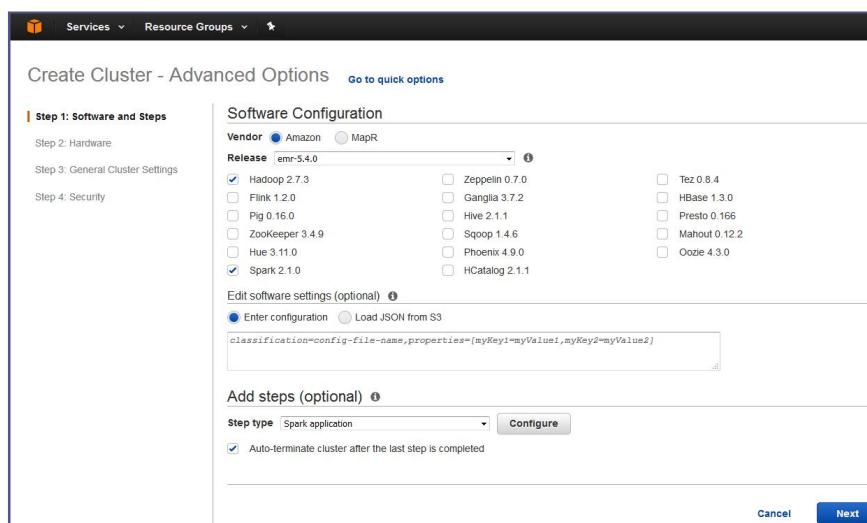


Ilustración 64. Creación de cluster en EMR de AWS en modo web.

Se ha seleccionado la alternativa EMR por ser una manera de tener una vision global del ecosistema y porque abstrae de la problemática de la configuracion para poder fijarnos en la esencia de los Cluster Big Data.

Cuando se esta en un Cluster real hay que tener muy presente como se reparte el calculo y los datos. Vease una manera muy simplista de cómo trabaja en Spark.

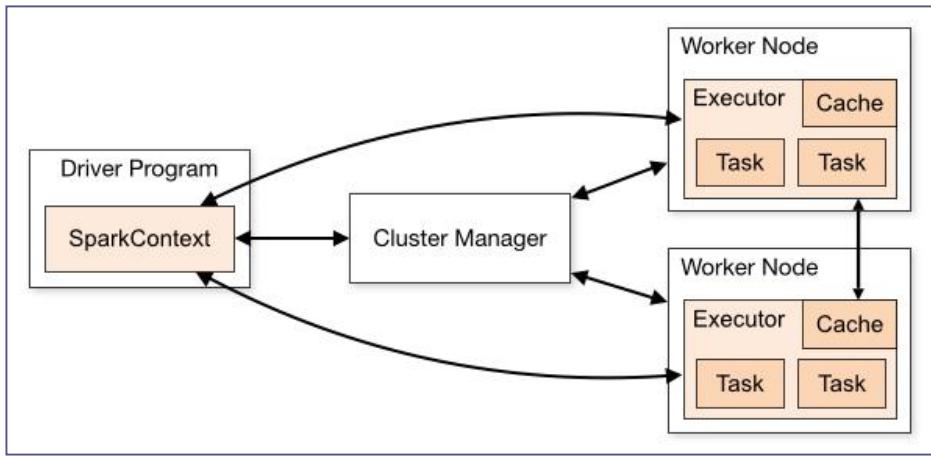


Ilustración 65. Cluster manager en Spark

<http://spark.apache.org/docs/latest/cluster-overview.html>

El “Driver” es donde esta el “context” con el código, puede estar en un nodo concreto cuando se despliega en modo “client” o distribuido cuando se despliega en modo “cluster”. El “Manager” distribuye, gestiona y controla los tareas (Task). Puede ser de 3 tipos :

- Local. Todo se hace en un unico nodo. Es como se ha hecho para desarrollar el código de este estudio.
- Apache Mesos. Un gestor de cluster general de código abierto que también puede ejecutar Hadoop MapReduce y aplicaciones.
- Hadoop YARN. Es el gestor de recursos de Hadoop 2.0. Es como trabaja AWS EMR.

Todo esto se tiene reflejo en la forma de cómo lanzar un trabajo en Spark:

```

bin/spark-submit \
  --class <main-class>
  --master <master-url> \
  --deploy-mode <deploy-mode> \
  --conf <key>=<value> \
  ... # other options
  <application-jar> \
  [application-arguments]

```

AWS es un servicio de pago por uso, lo más conveniente para disminuir los costes, ajustar y minimizar el consumo de recursos. Una manera para minimizar el tiempo es lanzar los trabajos desde nuestro sistema de manera desatendida, se levanta el Cluster se lanza el trabajo y cuando este termine la ejecución se apaga el Cluster. El problema estriba donde se almacena los datos necesarios, el trabajo en sí mismo, sus logs y los resultados. En AWS es normal usar S3 como lugar de almacenamiento persistente. En AWS EMR se trabaja por defecto con Hadoop HDFS como almacén de datos distribuido para apoyo de los trabajos.

Para este caso concreto se hace una versión del algoritmo pensada para AWS EMR, que interactúa tanto con S3 como con HDFS según convenga. En un Bucket de

S3 se deja el código del trabajo, el conjunto de datos inicial, se habilita donde dejar los resultados, un Shell script (bootstrap.sh) para añadir particularidades de configuración a los nodos. En AWS EMR para ejecutar código Python 3 hay que indicárselo explícitamente, y esto se hace por medio de otro fichero de configuración inicial, myConfig.json en nuestro caso, que también lo dejaremos en S3. Véase el contenido bootstrap.sh:

```
#!/bin/bash
#instalar paquetes Python que no vienen por defecto
sudo python3.4 -m pip install -U pandas
sudo python3.4 -m pip install -U scikit-learn
sudo python3.4 -m pip install -U boto3
#Copia de S3 lo necesario
aws s3 cp 's3://alucloud36/app/CalibraMeta.py' /home/hadoop/
aws s3 cp 's3://alucloud36/input/codificacion.txt' /home/hadoop/
```

Una precondición para trabajar con AWS EMR es la de configurar los "Security Group" de AWS para poder acceder a los nodos EC2 del Cluster. Para ello, hay que abrir las conexiones SSH y habilitar el acceso al manager de YARN y el acceso al manager de Spark.

La primera aproximación es crear un Cluster de Spark lo más sencillo posible con el asistente web en AWS y ejecutar un trabajo en modo cliente. Se observa que para levantar y tener acceso al Cluster se tarda unos 10 minutos, dato a tener en cuenta en los costes. Cuando hablamos de costes el nivel de maquina es un elemento donde hay que prestar especial atención. Spark es un nato consumidor de memoria y en AWS para encontrar maquinas EC2 con algo de memoria hay que ir hasta el nivel "large", teniendo un coste económico más elevado. En este caso, para la ejecución del algoritmo se utiliza instancias m1.large (2 vCPU, 7,5GB), 1 instancia drvier, 3 instancias cores (Spark+Hadoop), 1 instancia task (Solo Spark).

Véase como lanzar la petición desatendida que utiliza el paquete "awscli" (toda instancia EC2 lo tiene ya):

```
aws emr create-cluster \
--applications Name='Spark' \
--tags 'Name=SparkClusteralucloud36' \
--ec2-attributes '{"KeyName":"alucloud36-keypair","InstanceProfile":"EMR_EC2_DefaultRole","AvailabilityZone":"us-east-1d" \
,"EmrManagedSlaveSecurityGroup":"sg-fe018b96","EmrManagedMasterSecurityGroup":"sg-fc018b94" \
,"AdditionalMasterSecurityGroups":["sg-74a57f62"],"AdditionalSlaveSecurityGroups":["sg-74a57f62"]}' \
--release-label emr-5.4.0 \
--log-uri 's3n://alucloud36/output/' \
--instance-groups '[{"InstanceCount":1,"InstanceGroupType":"MASTER","InstanceType":"m1.large","Name":"Master - 1"} \
,{"InstanceCount":3,"InstanceGroupType":"CORE","InstanceType":"m1.large","Name":"Core - 2"} \
,{"InstanceCount":2,"InstanceGroupType":"TASK","InstanceType":"m1.large","Name":"Task - 3"}]' \
--configurations file:s3://alucloud36/app/myConfig.json \
--bootstrap-actions '[{"Path":"s3://alucloud36/app/bootstrap.sh","Name":"Custom action"}]' \
--steps Type=Spark,Name="Spark Program" \
    ,ActionOnFailure=CONTINUE \
    ,Args=[="--deploy-mode","cluster","s3://alucloud36/app/CalibraMeta.py","s3://alucloud36/input/codificacion.txt","s3://alucloud36/output/resultados.csv",10] \
--service-role EMR_DefaultRole \
--enable-debugging \
--name 'SparkCluster36' \
--region us-east-1 \
--auto-terminate
```

Ilustración 66. Creación cluster Spark modo comando.

Con esta simple llamada desde cualquier equipo se puede lanzar el algoritmo contra un cluster Spark en Amazon con 5 workers Spark 3 de ellos con Hadoop y un Manager con Yarn. Se ve como en la opción --steps están como argumento los buckets de S3 donde

está el código, los datos y donde dejar los resultados, además del parámetro con el número de divisiones deseadas.

El trabajo se lanza modo cluster, esto dificulta el donde obtener los resultados ya que YARN hace un contenedor con la aplicación. Véase un esquema de cómo sería:

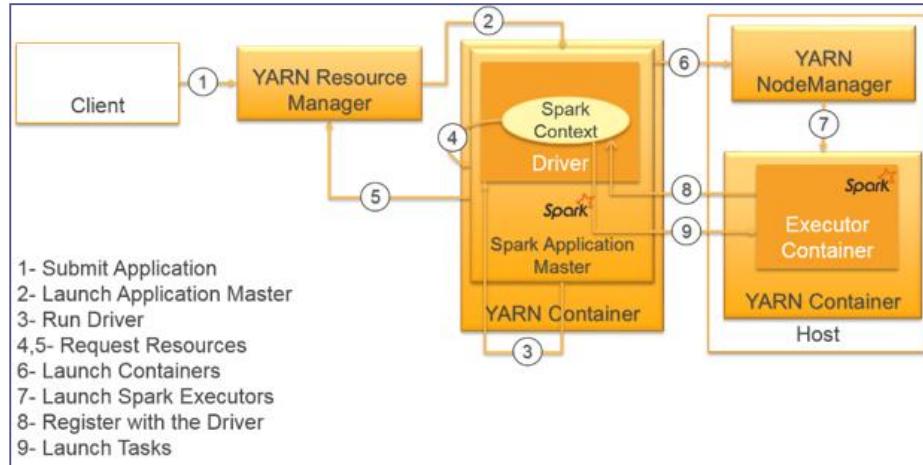


Ilustración 67. Trabajos Spark sobre EMR

Y después de 14 minutos se obtienen los resultados en S3 y la siguiente salida.

```
Calibra Metaclasificador 10 Divisiones.

-----
Tiempo crear context Spark 2.1413145065 segundos.

Tiempo Carga codificacion.txt: 4.0227975845 segundos.Total de registros 492790

10000 Ejemplos a clasificar , 3 Clasificadores .

1330 Combinaciones 1064 No repetidas.

Particiones tabla combinaciones 48.

Tiempo Crea DF clasificaciones y combinaciones 23.0223066807 segundos.

Tiempo Calculo 433.3195245266 segundos.

Tiempo Total 466.6702482700 segundos.
```

Ilustración 68. Salida de la solución de calibrado de clasificadores en Spark.

El tiempo de ejecución del propio algoritmo es algo menor a la simulación Pyspark sobre un único Server, 700 segundos local frente a 466 segundos AWS, a esto hay que sumar los trece minutos que tarda en levantar el cluster, este inconveniente se minimiza para tamaños de mayores. En el supuesto de fusionar 5 clasificadores según cálculos se puede tardar 24 horas, pero este mismo código con más nodos bajaría sustancialmente esos tiempos, solo es cuestión de añadir nodos. Con lo que ya se tiene el algoritmo para ponderar la combinación de clasificadores en Big Data.

## Big data para codificar diagnósticos clínicos de manera automática.

```
# -*- coding: utf-8 -*-
"""
Created on (Oct 2016)
Author: Jose Manuel Marin Noguera
"""

## Imports
from __future__ import print_function
import sys
#Importar la libreria Pyspark manejo de Spark
from pyspark import SparkContext, SparkConf
from pyspark.sql import SQLContext
from pyspark.sql import Row
from pyspark.sql import DataFrame
from pyspark.sql import HiveContext
from pyspark.sql import functions as F
from pyspark.sql.types import *
from pyspark.sql.functions import udf
from pyspark.sql.functions import broadcast
from pyspark.sql.window import Window
#Importar la libreria PANDAS para el manejo de datos
import pandas as pd
#Importar lo las funciones, clases y algoritmos que utilizaremos de Scikit-Learn
from sklearn.utils.extmath import cartesian
#Importar para sacar tiempos
import time
#Importar para trabajar con matrices que utiliza el pandas.
import numpy as np

#Funcion para la carga de datos
def carga_datos(filename):

#Funcion para calculo combinaciones
def carga_combinaciones(divisiones,clasificadores):

#Funcion para calculo combinaciones
def pivota_combinaciones(combinaciones):

## Main functionality
def main(sqlContext):
    clasificaciones = carga_datos(text_file)# Carga de los datos en un data frame de pandas
    clasificadores=clasificaciones['clase'].unique().tolist() # Enumera los clasificadores para poder trabajar con mas clasificadores
    ejemplos=len(clasificaciones['id_diag'].unique())# Totaliza los ejemplos para la evaluacion
    print("% Ejemplos a clasificar , % Clasificadores .\n% % (ejemplos, len(clasificadores)) )
    combinaciones = carga_combinaciones(divisiones,clasificadores)# Carga de los datos en un data frame de pandas
    cc_df = sqlContext.createDataFrame(combinaciones).cache()# Crea el DF de sql.pyspark para las combinaciones en formato linea
    combinaciones = pivota_combinaciones(combinaciones)# Pivotla las combinaciones para hacer un Join con las clasificaciones
    clasificaciones = clasificaciones.sort_values(['id_diag','prediccion'],axis=0 , ascending=[True, True])# Ordena para facilitar el join
    combinaciones = combinaciones.sort_values(['combinacion','clase'],axis=0 , ascending=[True, True])# Crea el DF de sql.pyspark
    tiempo_ini = time.time()
    s_df = sqlContext.createDataFrame(clasificaciones).cache()# Crea el DF de sql.pyspark para las clasificaciones (los ejemplos)
    c_df = sqlContext.createDataFrame(combinaciones).repartition(PARTICIONES_COMBINACIONES,'combinacion')# Crea el DF de sql.pyspark p
    print("Tiempo Crea DF clasificaciones y combinaciones %0.10f segundos.\n% % (time.time() - tiempo_ini))
    tiempo_ini = time.time()
    ##### MAP-REDUCE
    #t2_df = c_df.join(broadcast(s_df), c_df.clasificador == s_df.clasificador).drop('clasificador').repartition(100,'combinacion')
    t_df = c_df.join(broadcast(s_df), c_df.clasificador == s_df.clasificador).drop('clasificador')# Join el DF Todo
    t_df = t_df.withColumn('nconfianza',(t_df.confianza * t_df.peso))# Multiplicamos por su peso
    t_df = t_df.groupby(['combinacion','id_diag','clase','prediccion']).agg(F.sum('nconfianza').alias('sconfianza'))# Agrupamos por la combinacion
    t_df = t_df.filter(t_df.sconfianza > 0)# Quitamos los que no tienen confianza alguna
    # Definimos la ventana de desplazamiento Ojo! como puede haber empates gana la mayor prediccion alfabetica
    windowSpec = Window.partitionBy(t_df['combinacion'],t_df['id_diag']).orderBy(F.desc('sconfianza'),'prediccion')
    t_df = t_df.select('combinacion','id_diag', 'clase', 'prediccion','sconfianza', F.dense_rank().over(windowSpec).alias('rnk'))# Se ordenan
    t_df = t_df.filter(t_df['rnk'] == 1).drop('rnk')# Se selecciona el de mejor confianza
    ### Evaluacion
    t_df = t_df.filter('clase = prediccion')# Se seleccionan solo los aciertos
    t_df = t_df.groupby(['combinacion']).count()# Agrupamos por combinacion sumando sus aciertos
    t_df = t_df.withColumn('accuracy',(t_df['count'] / ejemplos) ).drop('count')# Calculamos el accuracy aciertos / total ejemplos
    t_df = t_df.orderBy(F.desc('accuracy'))# Ordenamos y sacamos los resultados a json (en la version 2.0.2 a csv)
    f_df = t_df.join(broadcast(cc_df), cc_df.index == t_df.combinacion)
    f_df.write.csv(out_file)
    #t_df.explain()
    print("Tiempo Calculo %0.10f segundos.\n% % (time.time() - tiempo_ini))
    ##### END MAP-REDUCE

## Module Constants
APP_NAME = "CalibraMeta"
divisiones = 10
PESOS_PROPORCIONES = True
PARTICIONES_COMBINACIONES = 40
if __name__ == "__main__":
    """
    Usage: calibra file_input file_outup [slice]
    """
    tiempo_ini_t = time.time()
    print(" Calibra Metaclasificador % Divisiones % Particiona combinaciones.\n% % (divisiones,PARTICIONES_COMBINACIONES) )
    print("-----")
    text_file = sys.argv[1]
    out_file = sys.argv[2]
    divisiones = int(sys.argv[3]) if len(sys.argv) > 3 else divisiones
    # Configure Spark
    tiempo_ini = time.time()
    conf = SparkConf().setAppName(APP_NAME)
    sc = SparkContext(conf=conf)
    sqlContext = HiveContext(sc)
    print("\n Tiempo crear context Spark %0.10f segundos.\n% % (time.time() - tiempo_ini))
    # Execute Main functionality
    main(sqlContext)
    sc.stop()
    print("Tiempo Total %0.10f segundos.\n% % (time.time() - tiempo_ini_t))
```

Ilustración 69. Solución fusión clasificadores en AWS.

## 4 CONCLUSIONES.

En este apartado se exponen los argumentos y afirmaciones derivados del estudio. Para ello hacemos un recorrido por sus objetivos.

Descubrir conocimiento con diagnósticos clínicos. Se identifica el problema de la codificación a CIE.9 como un problema actual y donde la búsqueda de soluciones aportaría beneficios de un gran impacto en la normalización y calidad de los sistemas de salud. Los textos diagnósticos en lenguaje natural y en castellano tienen un corpus caracterizado por su particular lenguaje, la corta longitud de los textos y con muchos acrónimos. La codificación CIE.9 se caracteriza por su amplio número de etiquetas (más de 14.000), dividido en capítulos y ordenados por criterios anatómicos o tipo de afección. En el entorno hospitalario hay un gran número de altas clínicas por codificar, nuestra propuesta ayudaría a automatizar la codificación con Big Data, combinando clasificadores de Aprendizaje Automático con otros ya existentes, aportando flexibilidad, robustez y garantizando alta capacidad tanto de modelado como de predicción. Entonces se concluye que hay una propuesta de valor con diagnósticos clínicos.

Descubrir conocimiento y uso de meta-clasificadores. La fusión de clasificadores es útil para un gran número de problemas. La propuesta aplicada, mejora los resultados tanto, de los clasificadores individuales, como de la fusión por voto simple. El modelo resultante es más robusto evitando el problema del sobre ajuste generalizando al calibrar las bondades de cada clasificador. Nuestra propuesta de refinamientos sucesivos mejora la eficiencia de la solución. El Big Data añade robustez eliminando problemas derivados del volumen o del rendimiento. Por lo tanto, se puede afirmar que la combinación de clasificadores por ponderación con refinamiento para Big Data es una excelente solución para combinar clasificadores débiles.

Descubrir conocimiento y uso de clasificadores de texto emergentes. La construcción de clasificadores multi-etiquetas de textos es un problema ampliamente estudiado sobre el cual salen constantes avances. Comprobar y entender estos avances nos dan la oportunidad de crecer en el uso de los clasificadores como medio de aprendizaje.

Low Dimensionality Representation (LDR) tiene la ventaja de no perder expresividad en su reducción y tiene en cuenta la distribución de clases en su entrenamiento, mostrándose adecuado para aplicaciones Big Data. Le presuponemos una predisposición al sobre ajuste ya que caracteriza ampliamente el conjunto de entrenamiento. En todos los experimentos ha superado el estado del arte a pesar de no aprovechar todas sus ventajas, obteniendo un buen comportamiento predictivo tanto a una clase y como a probabilidades por clase. Por todo ello el Low Dimensionality Representation es una opción muy válida para la clasificación de textos.

Document to Vector (Doc2Vec) [Quoc V. Le y Tomas Mikolov (2014)] como una evolución del Word to Vector tiene la ventaja de su entrenamiento no supervisado para representación vectorial de documentos. Al no ser necesario conocer sus clases se puede entrenar con otros corpus de un dominio parecido al problema. Para aprovechar esta ventaja es necesario disponer de un gran número de textos y esta es la causa. Doc2vec tiene muchas opciones tanto para el entrenamiento del espacio vectorial como en la transformación de los ejemplos, esto nos indica que tiene mucho margen de

mejora con los ajustes de hiperparametros. El algoritmo es aplicable ha superado también el estado del arte actual. En consecuencia el Doc2Vec es buena técnica para representación de textos con altas capacidades para problemas Big Data.

Descubrir conocimiento y uso de recursos necesarios. Sin los medios adecuados no es posible obtener buenos resultados. La noción obtenida sobre las herramientas software empleadas es conocimiento reutilizable, aportando gran valor al estudio. Se puede decir que el conjunto de herramientas seleccionadas corresponde a la visión personal del futuro del Big Data.

Linux +Docker + Python + Notebook + Spark. Es como nuestro framework todo terreno donde las piezas encajan, integrándose como un todo. Linux libre y con máximo rendimiento. Docker solo lo necesario, aislado y transportable. Python valido para todo, para describir un conjunto pequeño de 2900 ejemplos y también para operaciones sobre RDD de Spark de 1000 millones de registros. Python bien acompañado con Pandas, Scikit-learn, SNS, Matplot, Pyspark y Sqlcontext. Notebook-Jupyter permite programar contando historias visuales lo que es una gran ventaja. Y Spark, por supuesto, la nueva tecnología que va a referenciar a los ecosistemas Big Data. Afirmamos que Linux +Docker + Python + Notebook + Spark es una excelente plataforma para los proyectos de ciencia de datos.

Hay que decir que el conjunto herramientas seleccionadas corresponde a una apuesta personal sobre el futuro del Big Data.

Construir dos clasificadores que junto a los 3 conocidos se ensamblan y se calibran para obtener un meta-clasificador de CIE-9 para diagnósticos clínicos en español. Sin un código valido aplicable es imposible integrar la solución. En este caso los productos software aplicables a destacar son:

- Clasificador multi-etiqueta con LDR para Python
- Clasificador multi-etiqueta con Doc2Vec para Python.
- Calibrado de la Fusión clasificadores débiles multi-etiqueta para Python.
- Calibrado de la Fusión clasificadores débiles multi-etiqueta para Spark.
- Utilidades para investigar, presentar y visualizar datos en Python.

El propósito general del trabajo es descubrir conocimiento sobre datos con tecnología Big Data Analytics. Transformar los datos en valor y conseguir conocimiento es un reto muy presente en la sociedad actual. ¿Cuál es el proceso para conseguirlo? Obtener datos, limpiar datos, preparar datos, modelar datos, visualizar y presentar datos, datos, datos y más datos. Si solo se mira los datos solo consigues información, para conseguir conocimiento es necesario sumar la actitud. Gracias a la constante formalización de las dudas, las nuevas ideas, la visión total y la alineación con el negocio, se consigue el paso de la información a conocimiento. En el proceso de este estudio en particular, no se ha bastado con una simple aplicación de un algoritmo a un conjunto de datos para obtener un determinado resultado. Se ha enfocado en los porqués de las cosas, la exploración de los límites y la investigación de nuevas ideas. Con la aplicación de ese espíritu se puede afirmar que este estudio es sobre todo un recorrido completo de una vivencia en Big Data Analytics con capacidad de dar valor y conocimiento.

#### 4.1 Futuro.

Como en muchas cosas este estudio no es el final, sino el comienzo. Siempre hay ideas que se quedan por el camino, dudas pendientes por resolver, mejoras posibles, y la intención de convertirse en realidad. En este apartado se hace una visión de futuro hacia dónde dirigirse.

La visión de futuro pasa lógicamente por ver la idea de este estudio hecha realidad. Esto implica que se dispondría de la información necesaria para fijar objetivos más exigentes respecto a sus resultados. Se podría profundizar en los ajustes de hiperparametros, se podría investigar otras ideas como agrupaciones de clases, o modelos separados por clase, los N-gramas de textos, el Post\_tag y análisis morfológico de textos. Se podrían investigar otros algoritmos de clasificación y su manera de combinarlos. Y además la satisfacción de comprobar la eficacia real de LDR, Doc2Vec y la fusión de clasificadores en contextos Big Data.

Lo que se puede mejorar de la solución está siempre presente en todo el estudio y es una referencia para señalar trabajos futuros. Por ejemplo hacer la implementación de fusión de clasificadores en APIs para facilitar su uso, y aplicar algoritmos de agrupaciones para las zonas de mejora. El hacer programación orientada a objetos en los códigos de LDR y Doc2Vec para Python y además hacer versión Spark para ellos. Y en cuanto a su uso clínico, como mejora para los diagnósticos, pensamos que codificar en la jerarquía SNOMED-CT en lugar de CIE.9 puede aportar beneficios añadidos a la solución.

Se plantean varias dudas encontradas en el recorrido de este estudio: ¿Hay solo una zona de mejora en la ponderación de clasificadores?, ¿vale la pena el stemming de los textos? y ¿seguirá LDR como el mejor clasificador al crecer el tamaño del problema?

Además, añadimos retos que nos gustaría investigar en un futuro. ¿Qué puede aportar el lenguaje R en este tipo de problemas?, ¿tiene sentido una solución Real-Time? y ¿qué implica trabajar con Mesos como manager de Cluster Manager?

A corto plazo, nos proponemos la difusión del estudio. Y sobre todo el deseo es, crecer en la solución y en los conocimientos que han llevado a ella.



Big data para codificar diagnósticos clínicos de manera automática.

Dedicado a ti papa...

## Bibliografía

- [Baeza-Yates, R. (2004)]. Challenges in the Interaction of Information Retrieval and Natural Language Processing. in Proc. 5 th International Conference on Computational Linguistics and Intelligent Text Processing (CICLing 2004), Seoul , Corea. Lecture Notes in Computer Science vol. 2945, pages 445-456, Springer
- [Baeza-Yates, R. and Ribeiro-Neto, Berthier (1999)]. Modern information retrieval. Addison-Wesley Longman
- [Sebastiani,( 2002)] Automática de Textos
- [Joachims, 1998], (Friedman, Geiger, y Goldszmidt, 1997), (Lewis et al., 1996)
- [Martín Valdivia (2003)] Redes neuronales (Martín Valdivia, García Vega, y Ureña López, 2003; Li et al., 2002),
- [Bauer y Kohavi, 1999]. Técnicas de votos.
- [Rangel, F., et al. (2016)]. Rangel, F., P. Rosso, y M. Franco-Salvador. (2016).A low dimensionality representation for language variety identification. En 17th International Conference on Intelligent Text Processing and Computational Linguistics, CICLing. Springer-Verlag, LNCS.
- [Roli, F. (2003)]: Fusión de Multiple Pattern Classifiers. 8th National Conference of the Italian Association on Artificial Intelligence, September, Pisa, Italy,
- [Tom M. Mitchell] "Machine Learning", McGraw-Hill International Editions.
- [Ting, K.M., Witten, I.H. (1999)]: Issues in stacked generalization. Journal of Artificial Intelligence Research, 10, 271-289
- [Hernández-Orallo, J., Ramírez, M.J., Ferri, C. (2004)]: Introducción a la minería de datos. Pearson Educación, S.A., Madrid,
- [Aronson & et al. (2007)] Aronson, A. & et al., O. B. (2007), Indexing the Biomedical Literature to Coding Clinical Text: Experience with MTI and MachineLearning Approaches.
- [Zhang (2006)] Zhang, Y. (2006), «A Hierarchical Approach to Encoding Medical Concepts for Clinical Notes».
- [J.Medori(2010)] J.Medori, C. (2010), «Machine learning and features selection for semi-automatic ICD-9-CM encoding», Proceedings of the NAACLHLT, vol. Second Louhi Workshop on Text and DataMining of Health Documents, p84–89, 2010.
- [Zhang, W. H. & Patrick, J. (2008)], Automatic Classification of Pathology Reports into SNOMED Codes Automatic Classification of Pathology Reports into SNOMED Codes Automatic classification of pathology reports into SNOMED codes, Degree Memory, University of Sidney.

Big data para codificar diagnósticos clínicos de manera automática.

[Nerea Aguirre, Estibaliz Amillano (2014)]. Aportaciones de las técnicas de aprendizaje automático a la clasificación de partes de alta hospitalarios reales en castellano"

[Ferrao, J. C., M. D. Oliveira, F. Janel, and H.M.G. Martins. (2012)]. Clinical coding Support based on structured data stored in electronic health records. In Bioinformatics and Biomedicine Workshops, 2012 IEEE International Conference on, pages 790,797.

[Kim, Jin-Dong, Tomoko Ohta, Yuka Teteisi and Jun'ichi Tsujii. (2003)] GENIA corpus a semantically annotated corpus for bio-textmining. Bioinformatics. 19(suppl. 1). pp. i180-i182, Oxford University Press, 2003. ISSN 1367-4803

[Tony Ojeda, Rebecca Bilbro, Benjamin Bengfort (2017)] Applied Text Analysis with Python.

[Quoc V. Le y Tomas Mikolov (2014)] Distributed Representations of Sentences and Documents.

[Tom Fawcett (2013)] Foster Provost and Tom Fawcett, Data Science for Business: Fundamental principles of data mining and data analytic thinking, O'Reilly Media, 2013

[Dhar, V. (2013)]. "Data science and prediction"

[Jeffrey Stanton (2012)], "Data Science", 2012.