Universidad del País Vasco
Euskal Herriko Unibertsitatea

Proyecto: Text Mining

Parte 1: Representación de datos

Parte 2: Inferencia del clasificador

Parte 3: Predicciones

Xabier Bahillo Gil
Ruth Andrea Siles Zeballos
Nicolás Martín Celis

21/03/2020

Índice

1- Introducción a la minería de textos

La minería de datos es el conjunto de técnicas y tecnologías con las cuales hallamos patrones, anomalías y correlaciones en un conjunto de datos enorme, para así lograr comprenderlos y utilizarlos para extraer conclusiones y contribuir en la mejora de distintas áreas (empresas, medicina…). Las personas que se dedican y trabajan en este sistema se conocen como exploradores de datos. [Esther Ribas. (8 Enero 2018). ¿Qués es el Data Mining?.....]

Para clasificar estos conjuntos se utilizan algoritmos derivados de la inteligencia artificial y la elección del mejor algoritmo para un análisis de datos dependerá del problema a resolver, clasificación, segmentación, asociación, regresión y análisis de secuencias. [Esther Ribas. (8 Enero 2018). ¿Qués es el Data Mining?.....]

A la hora de realizar un análisis de datos, aunque cada caso concreto puede ser muy distinto al anterior, todos ellos se suelen realizar en 4 pasos distintos: [NA. (NA). Datamining (Minería de datos)...]

1. Determinar objetivos: ¿Que se quiere conseguir?
2. Preprocesamiento de los datos: Seleccionar, limpiar y filtrar los datos. Generalmente ocupa alrededor del 70% del tiempo total de un proyecto de data-mining [NA. (NA). Datamining (Minería de datos)...]
3. Determinación del modelo: Análisis estadístico y posterior visualización gráfica utilizando distintos algoritmos en función del caso [NA. (NA). Datamining (Minería de datos)...]
4. Análisis de los resultados: Verificar si los resultados obtenidos son coherentes [NA. (NA). Datamining (Minería de datos)...]

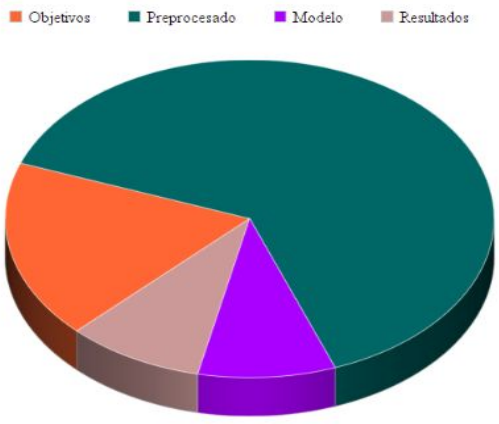


Figura 1: Gráfica con la carga aproximada de trabajo de las distintas fases

1.1- Algunos ejemplos sobre la minería de datos.

Para ayudar a comprender mejor que es la minería de datos vamos a poner unos pequeños ejemplos de usos de esta. Podemos encontrar algunos de estos en [Miriam Cecilia Moreno Luján. (NA). 5 ejemplos de aplicación] [Rodolfo Fernández González. (NA). Técnica de Inteligencia...]:

- Redes Neuronales.
- Generadores de reglas por inducción.
- Generadores de bases de casos a las que se accede mediante técnicas CBR.
- Sistemas clásicos de heurísticos.
- Sistemas GPS
- Procesos industriales

2- Pre-procesamiento de datos

2.1- ¿Qué es el preprocesado de datos?

El pre-procesar los datos es convertir la información recopilada en datos con un formato que facilite el uso de estos. [NA. (30/12/2016). Calidad de datos en minería de datos..]
En nuestro caso, vamos a transformar datos en el formato CSV, a el formato .ARFF, formato compatible con la libreria WEKA para el tratamiento de datos.
Además, frecuentemente los datos no están limpios, por tanto, es tarea del preprocesamiento de datos realizar las distintas operaciones hasta obtener unos datos lo más limpios posibles. [NA. (30/12/2016). Calidad de datos en minería de datos..]
Existen muchos métodos de preprocesado de datos, en función de los datos que se tengan, será mas conveniente usar unos u otros métodos. En nuestro caso usamos en Bag of Words (BoW) y el TF-IDF.

2.2- Diferencias entre los tipos Raw, BoW y TF-IDF

En el pre-procesamiento de datos podemos destacar los tipos de datos raw, BoW y TF-IDF:

- **RAW**: Son **datos brutos**, ”primarios” que aún no han sido procesados, estos en principio no tienen **ningún formato establecido,** aunque también llamamos RAW, a los datos en bruto con un formato válido para nuestra herramienta, pero que no han sido procesados ni tratados.

- **Bag of Words (BoW):** Es un **método muy simple** y **fácil** de implementar para tareas generales de clasificación de texto. Este se encarga de tomar las **palabras encontradas** en todo el texto, y calcular la **frecuencia de aparición** de cada palabra en los distintos textos, generando así un **diccionario** con la estructura (palabra, frecuencia), y representando en el conjunto de datos la frecuencia de cada palabra en cada instancia, siendo los atributos del conjunto cada palabra diferente.

Es una forma de extraer características del texto para poder utilizarlas con algoritmos de aprendizaje automático. [Jocelyn D'Souza. (03/04/2018). An Introduction to Bag-of-Words...]

Problema: La representación BoW tiende a emplear un número alto de atributos. La dimensión del vector viene establecida por el número de palabras presentes en el vocabulario del conjunto de entrenamiento. Además, no indica qué términos discriminan mejor la clase [Aitor Esteve Alvarado. (02/10/2013). Estudio y Piloto…]

- **TF-IDF** es una medida estadística para cuantificar la relevancia de un término dentro de un documento, Es una medida que combina las medidas TF e IDF. **TF** mide la frecuencia de una palabra en el texto e **IDF** es una medida para cuantificar la relevancia de un término en un conjunto de documentos (cuanto frecuente o rara es en todos los documentos). [Jocelyn D'Souza. (03/04/2018). An Introduction to Bag-of-Words...]

Es por ello un método muy útil, ya que no tiene únicamente en cuenta el texto en el que aparece, sino todos los demás textos del conjunto

Las fórmulas para el cálculo de TF-IDF son las siguientes:

$$TF(w_i, d_j) = \frac{f(w_i, d_j)}{\sum_{w_i \in V} f(w_i, d_j)}$$

donde:

- $f(w_i, d_j)$: representa el número de veces que aparece el término w_i en el documento $d_j \in \mathcal{D}$
- $\sum_{w_i \in d_j} f(w_i, d_j)$: representa el número total de términos que aparecen en el documento d_j (representa el número total de palabras del documento d_j, no el número de palabras distintas)
- V: el conjunto de términos o vocabulario de la aplicación, es decir, $V = \{w_1, \ldots, w_i, \ldots\}$ donde $w_i \neq w_i \forall i \neq j$ (conjunto de términos distintos en el conjunto de documentos $\mathcal{D}$)

$$DF(w_i) = \frac{\sum_{d_j \in \mathcal{D}} \delta(w_i, d_j)}{|\mathcal{D}|}$$

donde:

- $\delta(w_i, d_j)$: representa la delta de Kronecker sobre la pertenencia del término w_i en el documento d_j según indica la expresión (3).

$$\delta(w_i, d_j) = \begin{cases} 1 & w_i \text{ está presente en el documento } d_j \\ 0 & w_i \text{ no está presente en el documento } d_j \end{cases}$$

- $\sum_{d_j \in \mathcal{D}} \delta(w_i, d_j)$: representa el número de documentos que contienen el término w_i
- $|\mathcal{D}|$: representa el número total de documentos

$$TF \cdot IDF(w_i, d_j) = TF(w_i, d_j) \cdot \log\left[\frac{1}{DF(w_i)}\right]$$

Figura 2: Fórmulas de TF-IDF [Guión Parte 1 de eGela]

Es importante destacar, que tanto en BoW como TF-IDF, la representación de los datos se puede hacer como un vector **non-sparse**, en el cual los 0s se representan de forma explícita (hay un valor para cada atributo), por tanto puede haber bastantes 0s si el conjunto de dato es muy grande, o como un conjunto de datos **sparse**, en el cual os 0s no se representar de forma explícita, y únicamente se representan los valores distintos a 0 mediante pares de la forma (a d), siendo (a= número valores a su izquierda, d= valor) aunque hay algunos clasificadores que no aceptan esta última forma.

2.3- Marco Experimental

Para ver visualmente un ejemplo de los distintos tipos de datos, vamos a utilizar un pequeño fichero .arff creado por nosotros, el cual contendrá dos textos diferentes

```
@relation mineriadedatos

@attribute text string
@attribute class {DESC,ENTY}

@data
'El pez de Bilbao tiene una espada de plata',DESC
'El pez espada de Donosti',ENTY
```

Figura 3: .ARFF de los datos en RAW

Como podemos ver, este conjunto de datos solo tiene dos atributos, y para que el algoritmo pueda aprender y realizar las predicciones necesarias, necesitamos transformarlo al tipo de datos BoW o TF-IDF.
En el primer caso, el conjunto de datos BoW genera un diccionario con la frecuencia de aparición de cada palabra, y un conjunto de datos nuevo con un atributo por cada palabra diferente, en nuestro caso, podemos observar que tenemos 9 atributos diferentes (el,pez,de,Bilbao,tiene,una,espada,plata,Donosti). Además, para la primera palabra, podemos ver que la palabra de aparece dos veces, mientras que las demás aparecen una única vez, y en la segunda palabra, todas aparecen una única vez

Transformando el conjunto de datos a BoW tenemos:

```
@relation pruebabow

@attribute bilbao numeric
@attribute de numeric
@attribute el numeric
@attribute espada numeric
@attribute pez numeric
@attribute plata numeric
@attribute tiene numeric
@attribute una numeric
@attribute donosti numeric
@attribute class {DESC,ENTY}
@data
1,2,1,1,1,1,1,1,0,DESC
0,1,1,1,1,0,0,0,1,ENTY
```

Figura 4: .ARFF de los datos en BoW Non Sparse

Como podemos ver, cada palabra diferente en el conjunto de instancias aparece como un atributo, y en cada instancia, aparece la frecuencia de cada palabra (p.e. para la posición 2, referente a la palabra “de”, la instancia 1 tiene como valor 2, y la instancia 2 tiene como valor 1). Como podemos ver, esta representación es la del conjunto de datos non-sparse. Podemos pasarla al conjunto de datos sparse calculando los pares necesarios. Por ejemplo, para la segunda instancia, el primer 0 no se representa, y el primer valor a representar seria (1 1).

```
@relation bow_sparse

@attribute bilbao numeric
@attribute de numeric
@attribute el numeric
@attribute espada numeric
@attribute pez numeric
@attribute plata numeric
@attribute tiene numeric
@attribute una numeric
@attribute donosti numeric
@attribute class {DESC,ENTY}
@data
{1 1,2 2,3 1,4 1,5 1,6 1,7 1,8 1,9 DESC}
{1 1,2 1,3 1,4 1,8 1,9 ENTY}
```

Figura 5: .ARFF de los datos en BoW Sparse

Además, nos genera el siguiente diccionario con el número de apariciones en cada texto:

```
1  bilbao,1
2  de,2
3  el,2
4  espada,2
5  pez,2
6  plata,1
7  tiene,1
8  una,1
9  donosti,1
```

Figura 6: Diccionario obtenido en BoW

Como segunda opción tenemos utilizar TF-IDF. En este, cada instancia ofrece el valor TF-IDF para esa palabra:
Vamos a calcular como ejemplo, para la instancia 1, el valor TF-IDF de la palabra el:

$$TF(el,1) = \frac{f(el,1)}{\sum f(i,1)} = \frac{1}{9}$$

$$DF(el) = \frac{2}{2} = 1$$

$$TF - IDF(el,1) = TF(el,1) \cdot \log\left(\frac{1}{DF(el)}\right) = \frac{1}{9} \cdot 0 = 0$$

Figura 7: Cálculo TF-IDF

```
@relation pruebatdfidf

@attribute bilbao numeric
@attribute de numeric
@attribute el numeric
@attribute espada numeric
@attribute pez numeric
@attribute plata numeric
@attribute tiene numeric
@attribute una numeric
@attribute donosti numeric
@attribute class {DESC,ENTY}

@data
0.480453,0,0,0,0,0.480453,0.480453,0.480453,0,DESC
0,0,0,0,0,0,0,0,0.480453,ENTY
```

Figura 8: .ARFF de los datos en TF-IDF Non Sparse

Como podemos comprobar, en la primera instancia, el tercer valor que referencia la palabra 3 es 0. En este caso el conjunto de datos es muy pequeño, por tanto, no obtenemos datos muy relevantes, ya que la mayoría de las palabras aparecen en todos los documentos, y por tanto son 0, y el resto de palabras tan solo aparecen una vez en uno de lo documentos, por tanto tienen el mismo valor.
En este caso, la representación también es non-sparse, y al igual que con BoW, podríamos representarlo en el modo sparse.

2.4- Selección de atributos

La selección de atributos es una fase importante del pre-procesamiento de datos, ya que, el tener una cantidad grande de atributos ralentiza el proceso de clasificación, y en algunos casos, el tener atributos irrelevantes o inútiles en ciertos casos pueden confundir a los algoritmos de aprendizaje automático. [Witten, Ian H. & Frank, Eibe & Hall, Mark A. & Pal, Christopher J.. (2011). Data Mining: Practical Machine...]

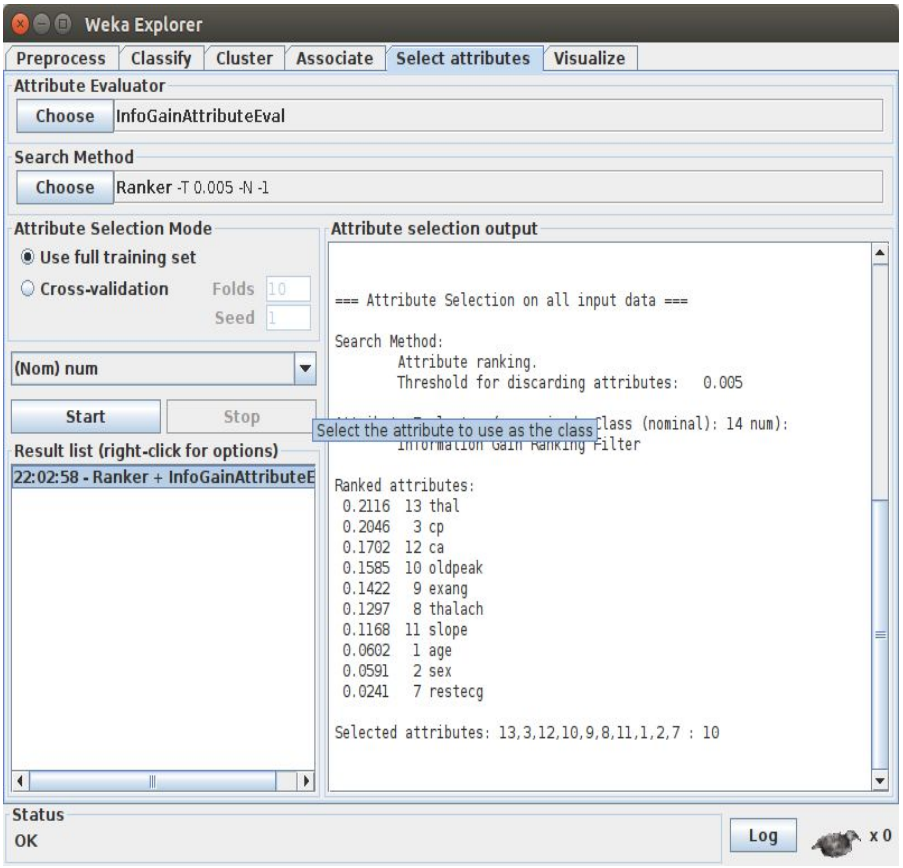
El objetivo de los algoritmos de selección de atributos es identificar aquellos atributos que tienen un mayor peso a la hora de clasificar si las instancias pertenecen a una clase u otra, es decir, obtener aquellos atributos que correlen adecuadamente con la clase y no correlen entre sí. [Witten, Ian H. & Frank, Eibe & Hall, Mark A. & Pal, Christopher J.. (2011). Data Mining: Practical Machine...]

2.5 Selección de atributos mediante técnicas de ganancia de información

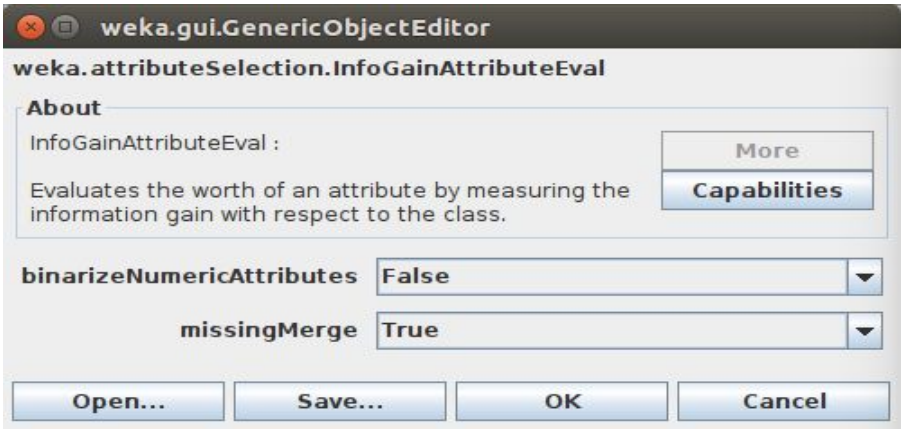
Marco Teórico

Es una técnica de selección que calcula la ganancia de información, también llamada entropía, para cada atributo para la variable de salida.
Valores de entrada: 0(sin información) y 1(información máxima), los atributos que aportan mayor información tendrán un mayor valor de ganancia de información y pueden seleccionarse y los que no agregan mucha información tendrán una puntuación más baja y con lo cual podrán ser eliminados.
Weka nos ofrece InfoGainAttributeEval el cual se debe utilizar con el metodo de busqueda de Ranker.
[Jason Browniee. (13 julio 2016). How to Perform Feature Selection With Machine]

En la GUI de weka:



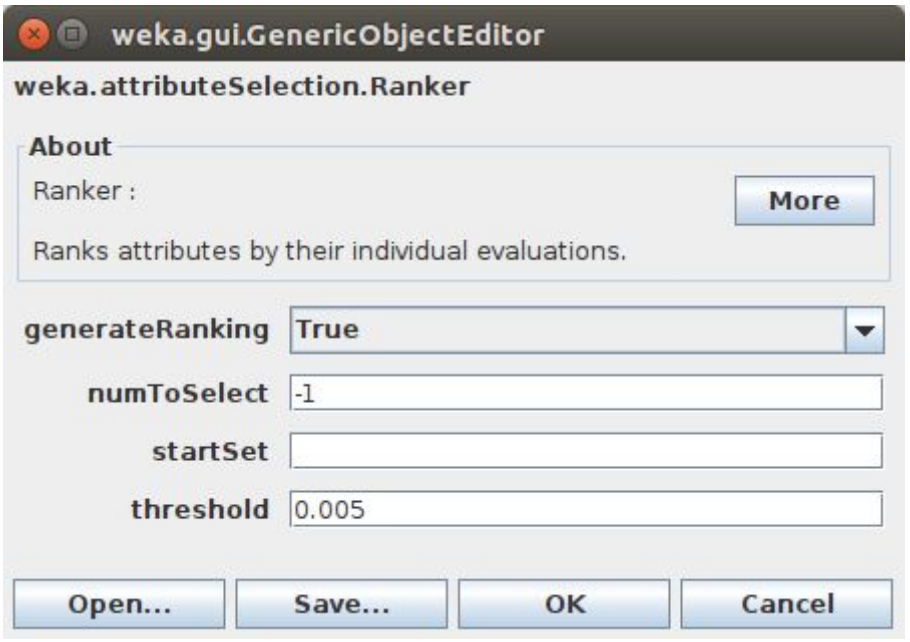
Parámetros:



- missingMerge: distribuye los recuentos de los valores faltantes. Los recuentos se distribuyen entre otros valores en proporción a su frecuencia. De lo contrario, la falta se trata como un valor separado.

- binarizeNumericAttributes: solo binarize los atributos numéricos en lugar de discretizarlos adecuadamente.

Ranker: Clasifica los atributos por sus evaluaciones individuales.



Parámetros:

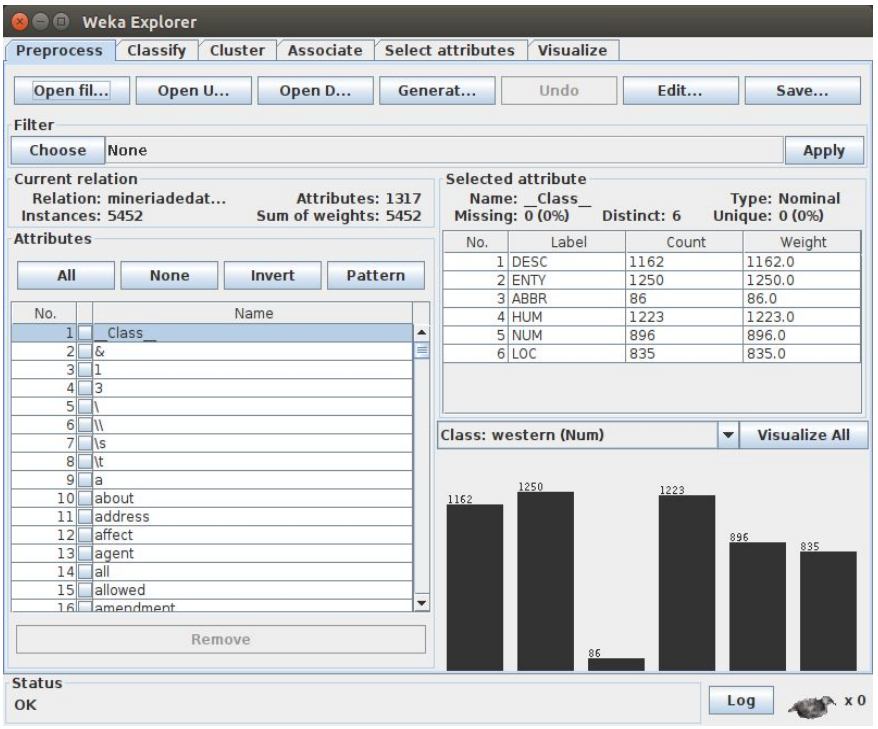
- generateRanking: una opción constante. Ranker solo es capaz de generar clasificaciones de atributos.

- numToSelect: especifique el número de atributos para retener. El valor predeterminado (-1) indica que se deben conservar todos los atributos. Use esta opción o un umbral para reducir el conjunto de atributos.

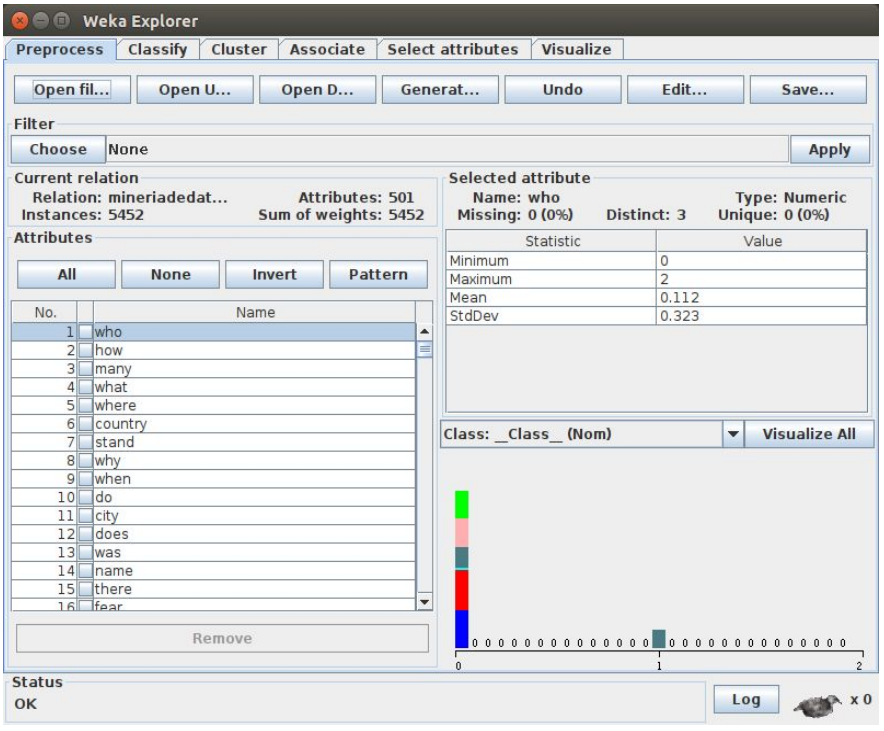
- threshold: establece el umbral mediante el cual se pueden descartar los atributos. El valor predeterminado da como resultado que no se descarten atributos. Use esta opción o numToSelect para reducir el conjunto de atributos.

- startSet: especifique un conjunto de atributos para ignorar. Al generar la clasificación, Ranker no evaluará los atributos en esta lista. Esto se especifica como una lista separada por comas de los índices de atributos que comienzan en 1. Puede incluir rangos. Por ej. 1,2,5-9.17.

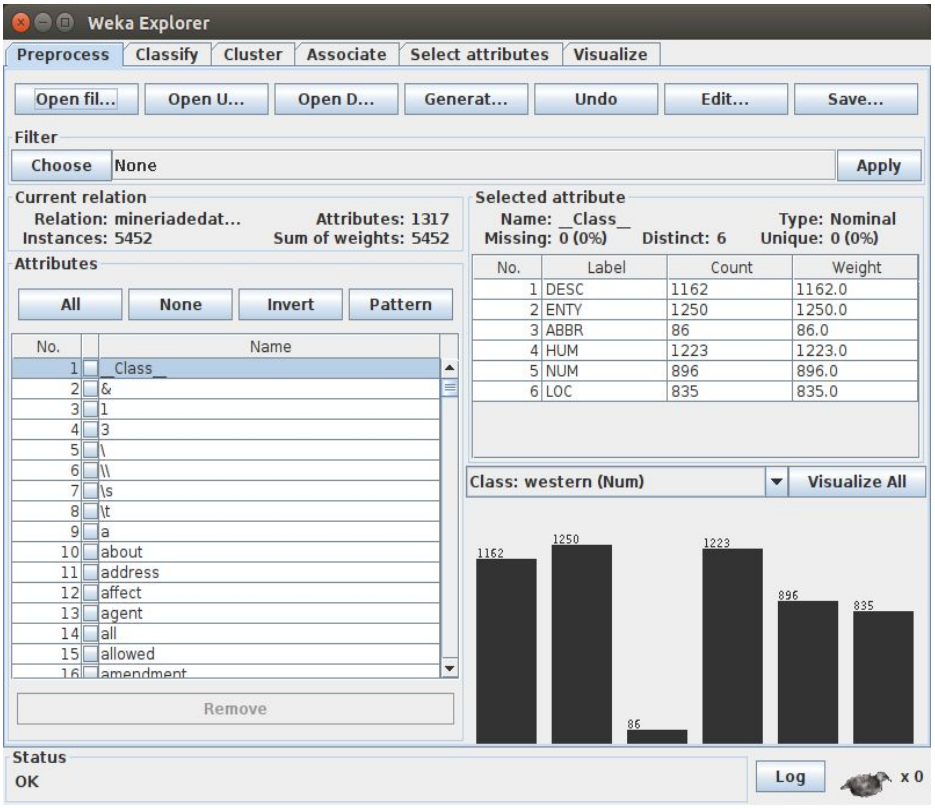
En la gui de weka podemos ver los resultados de nuestros experimentos con mas sencillez:



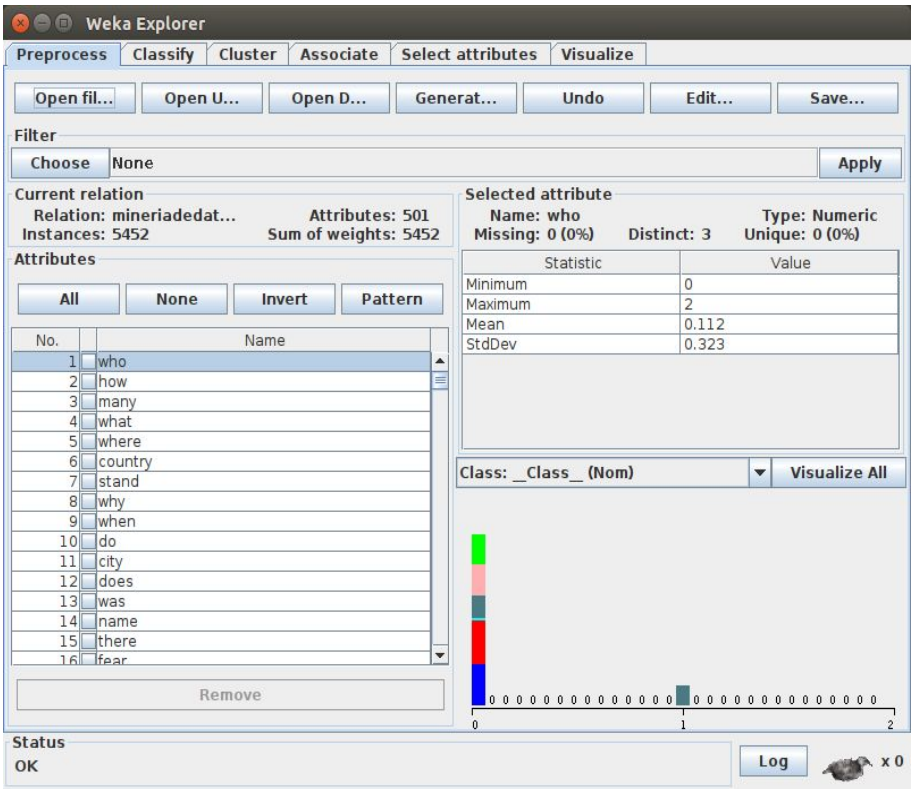
train_bow_nonsparse.arff



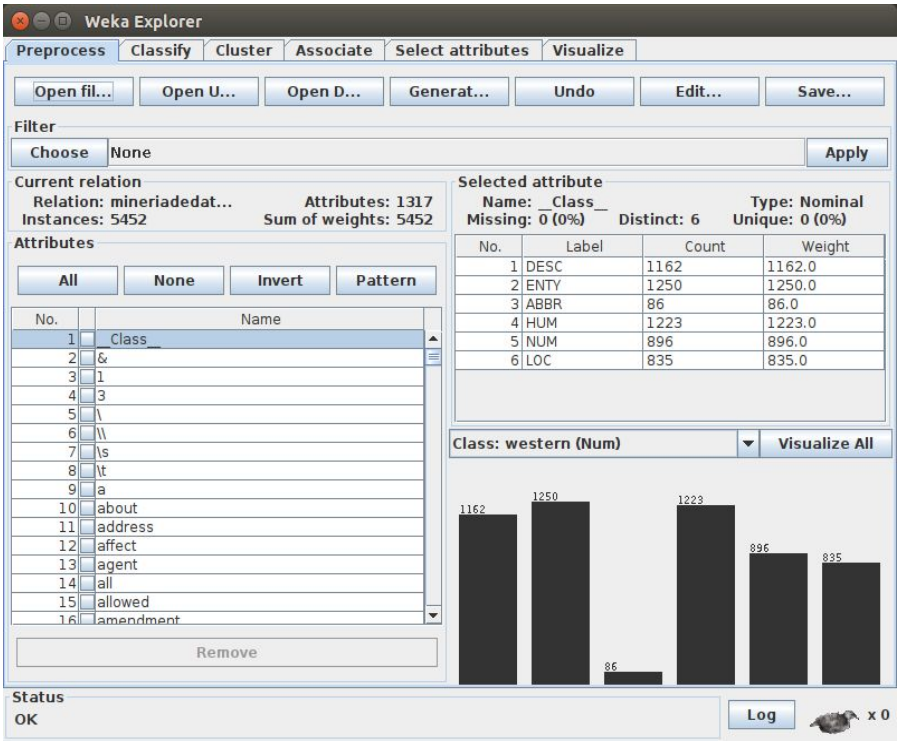
train_Bow_nonsparse_infoGain.arff



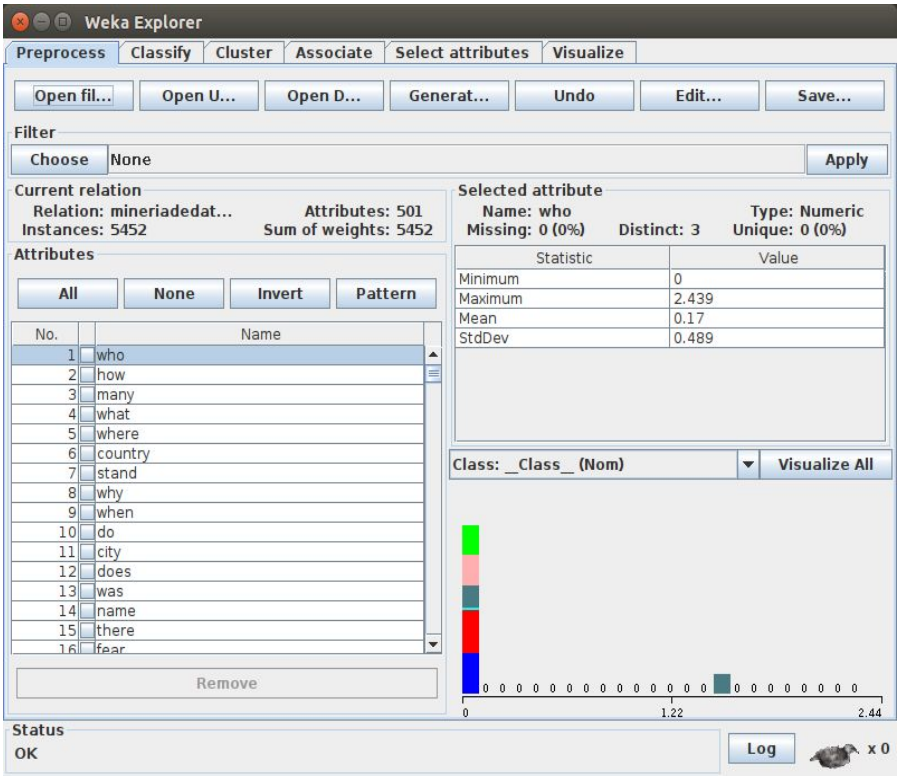
train_bow_sparse.arff



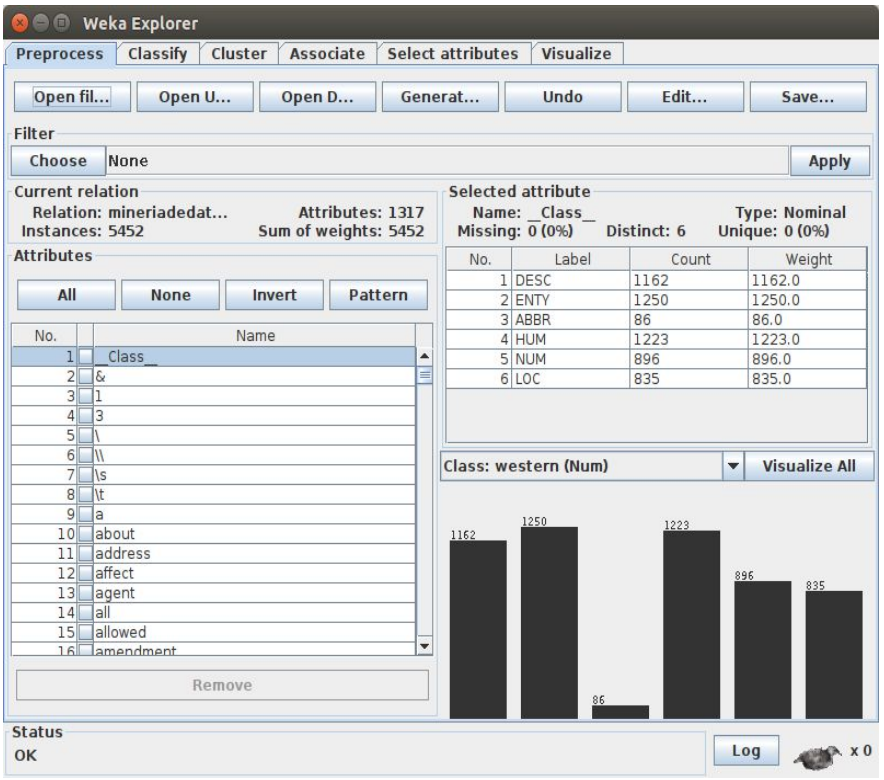
train_Bow_sparse_infoGain.arff



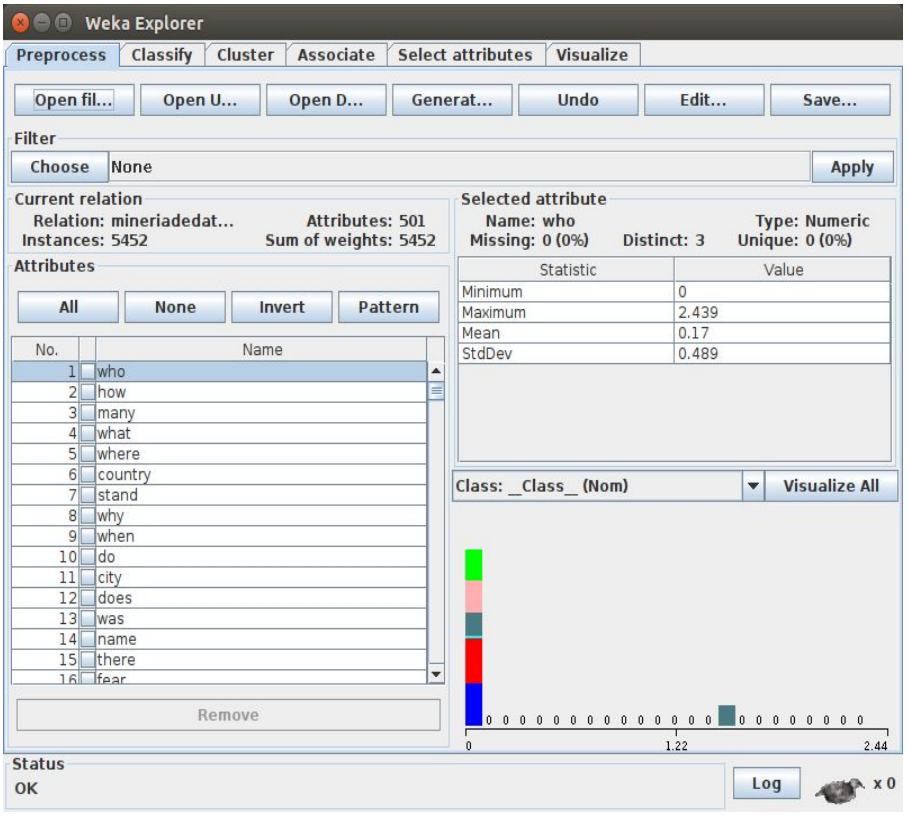
train_tf-idf_nonsparse.arff



train_tf-idf_nonsparse_infoGain.arff



train_tf-idf_sparse.arff



train_idf-tdf_sparse_ingoGain.arff

3- Inferencia del clasificador

3.1- Marco teórico

3.1.1-Baseline

El clasificador Naive Bayes se basa en un algoritmo probabilístico basado en el teorema de Bayes. La virtud que tiene este algoritmo es la de construir modelos de manera sencilla y con un buen comportamiento, gracias a su sencillez.
[Victor Roman. (Abril, 2019). Algoritmos Na…]

Como podemos observar en la fórmula de abajo, la cual hace referencia al teorema de Bayes, en el numerador tenemos la probabilidad condicionada y en el denominador, en cambio, la probabilidad total.

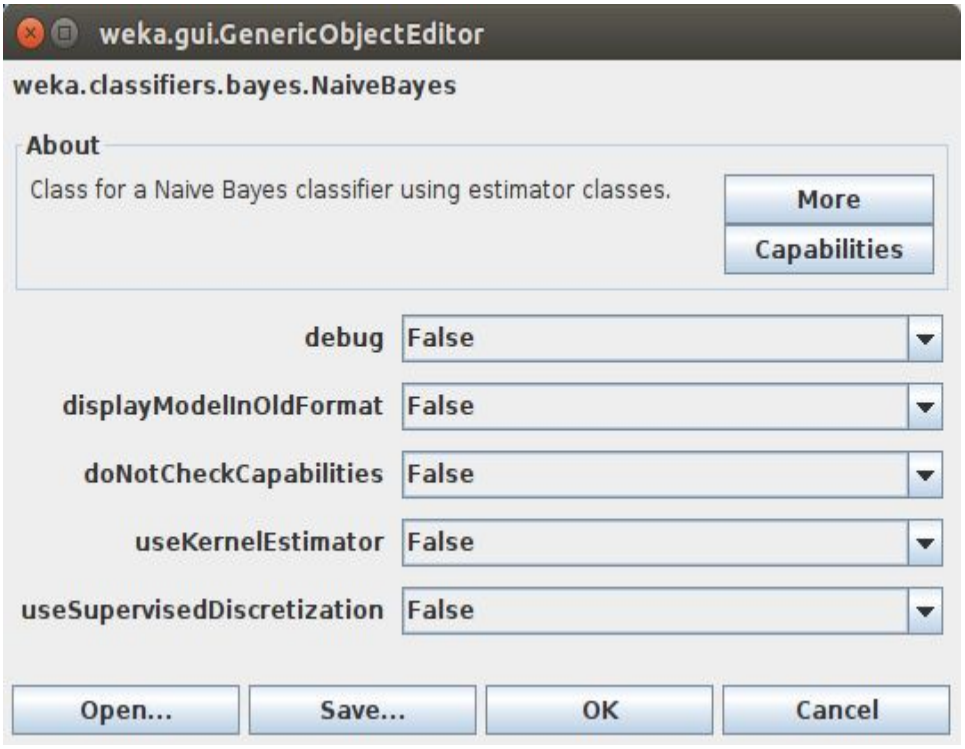
$$p(A_i|B)=\frac{p(B|A_i)*p(A_i)}{\sum_{i=1}^{n} p(B|A_i)*¿\,p(A_i)¿}$$

Donde (B) es el suceso del que tenemos información previa, y (Ai) son los distintos sucesos condicionados.
[José Francisco López. (N/A). Teorema de Ba...]

El proceso de aprendizaje mediante el teorema de Bayes se divide en dos etapas:

1. Aprendizaje estructural: Obtiene la estructura de la red Bayesiana.
2. Aprendizaje paramétrico: Obtiene las probabilidades a priori y condicionales que se necesiten teniendo en cuenta la estructura anterior.
[M.Ing. Enrique Fernández. (N/A). ANÁLISIS DE CL...]

En la GUI de weka:



3.1.2- Random Forest

Random Forest es una combinación de Árboles predictores.El algoritmo fue desarrollado por Leo Breiman y Adele Cutler y Random forest es su marca de fábrica.
[Wikipedia. (2013). Random forest……]

Construcción de un modelo Random Forest

- El número de casos en un conjunto de entrenamiento es N ,se toma una muestra de esos N casos aleatoriamente con reemplazo, y esta muestra será el conjunto de entrenamiento para construir un árbol(i).

- Existen M variables de entrada y m variables se seleccionan aleatoriamente de M.La mejor division de estos m atributos es usado para ramificar el árbol.El valor m se mantiene constante durante la generación de todo el bosque.

- Cada árbol crece hasta su máxima extensión posible y no hay proceso de poda.

- Nuevas instancias se predicen a partir de la agregación de las predicciones de los x árboles(Mayoría de votos para clasificación ,promedio para la regresión

El RandomForest hace una selección aleatoria de m predictores antes de evaluar cada división. De esta forma, un promedio de (p−m)/p divisiones no contemplarán el predictor influyente, permitiendo que otros predictores puedan ser seleccionados. Solo con añadir este paso extra se consigue *que no correlen entre si* los árboles, por lo que su agregación consigue una mayor reducción de la varianza.
La diferencia entre el random Forest y el Bagging :
En randomForest antes de cada división se selecciona aleatoriamente m predictores ,la diferencia en el resultado dependerá del valor m escogido. Si m=p los resultados de Random Forest y bagging son equivalentes.
Random forest evita este problema del bagging (la alta correlación entre árboles que apenas disminuye la varianza) .
[Joaquín Amat Rodrigo. (Febrero, 2017). Árboles de predicción: bagging, random forest, boosting y C5.0……...]

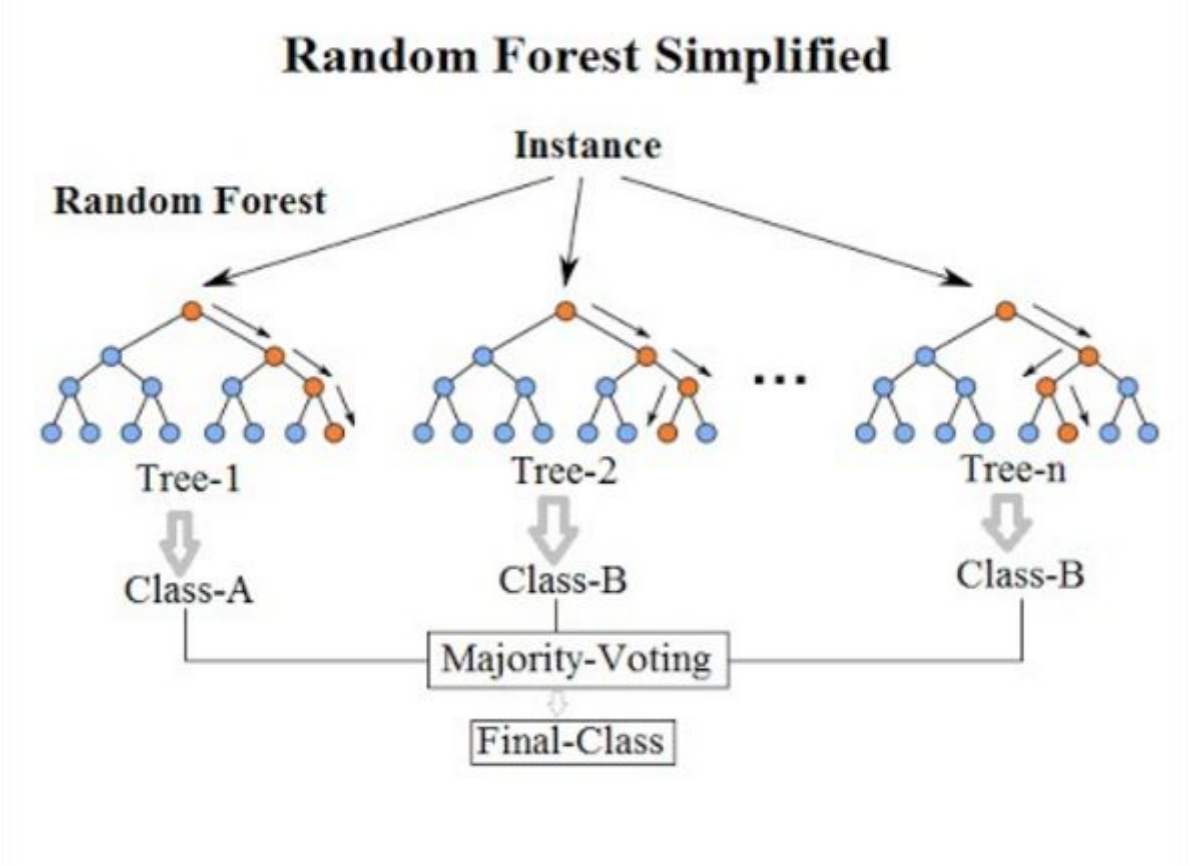
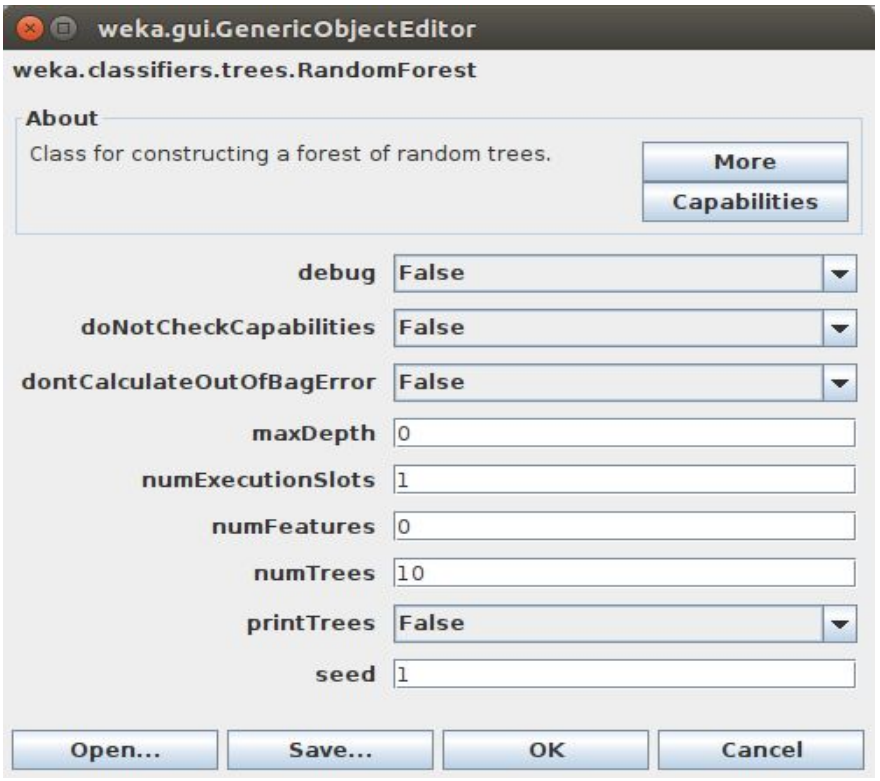


Imagen 1:[wjk68@case.edu.. (2017). Random Forest Simple Explanation]



Parámetros del Random Forest

debug: si está a true, el clasificador puede generar información adicional en la consola

● **doNotCheckCapabilities**: Si está a true, las capacidades del clasificador no se comprueban antes de que se construya el clasificador.

● **dontCalculateOutOfBagError:** Si es true, entonces el error no se calcula.

● **maxDepth:** La profundidad máxima de los árboles.

●**numDecimalPlaces**: Cuántos números decimales se van a utilizar para los números de salida en el modelo.

● **numExecutionSlots**: Número de espacios de ejecución (hilos) a usar para construir
el conjunto.

● **numFeatures**: El número de atributos que se van a utilizar en la selección aleatoria.

● **numTrees:** Número de árboles que se van a generar.

● **printTrees:** Dibuja los árboles individuales en la salida.

● **seed**: El número aleatorio de semilla que se va a utilizar.

3.2- Fase de inferencia

La fase de inferencia del modelo se de encarga obtener parámetros óptimos a través de un barrido de parámetros , sobre los dos parámetros más representativos del modelo .El resultado de esta fase es obtener parámetros que hacen que el modelo ofrezca mejores resultados.En este caso hemos elegido los parámetros que optimicen la f-mesure de la clase minoritaria con un esquema de evaluación hold-out , con el conjunto train.arff para el entrenamiento y el e conjunto dev.arff para la evaluación

3.3- Fase de aplicación del modelo para hacer predicciones

La fase de aplicación del modelo para hacer predicciones ,clasifica un conjunto de documentos en este caso el conjunto test ,dado un modelo predictor .
Vamos recorriendo la instancias del conjunto test y los estimamos para obtener la predicción y vamos comparando la clase predicha con la clase real.

3.4- Métodos ensembles (Métodos combinados de aprendizaje)

[Fernando Sancho Caparrini . (26 de Diciembre de 2018). Métodos combinados de aprendizaje……]
Los métodos combinados de aprendizaje ,utilizan múltiples algoritmos de aprendizaje para obtener un rendimiento predictivo mejor,que la predicción de cualquiera de los algoritmos individuales que lo constituyen.

Si nos centramos en los árboles uno de los problemas que presentan es el problema del equilibrio :cuánto se alejan en promedio las predicciones de un modelo respecto a los valores reales y cuanto varia el modelo dependiendo de la muestra empleada en el entrenamiento(varianza).

Los métodos de ensemble abarcan un conjunto de técnicas que combinan múltiples modelos predictivos para lograr un equilibro entre promedio y varianza. A continuación definimos dos de los métodos mas comunes.

3.4.1- Bagging

Construye varios modelos con el mismo algoritmo,cuando llega un dato de test ,la clase se decide por votación.
[N_A. (N_A). Introducción al Aprendizaje Automático y a la Minería de Datos……....]

3.4.1.1- Funcionamiento con árboles

En lugar de ajustar un único árbol, se ajustan muchos de ellos en paralelo formando un “bosque”. En cada nueva predicción, todos los árboles que forman el “bosque” participan aportando su predicción. Como valor final, se toma la media de todas las predicciones (variables continuas) o la clase más frecuente (variables cualitativas).Este método provoca una disminución de la varianza y evita el sobre ajuste.

Se ha probado que el bagging tiende a producir mejoras en los casos de modelos individuales inestables (como es el caso de las redes neuronales o los árboles de decisión), pero puede producir resultados mediocres o incluso empeorar los resultados con otros métodos, como el de los K vecinos más cercanos.
[Joaquín Amat Rodrigo. (Febrero, 2017). Árboles de predicción: bagging, random forest, boosting y C5.0……....]

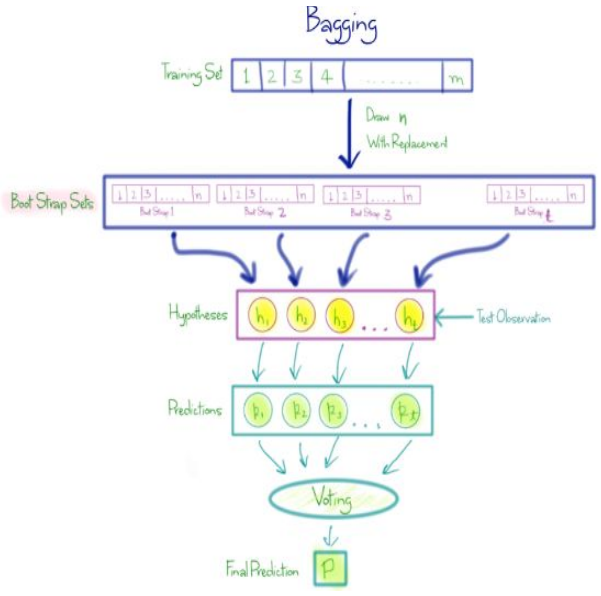
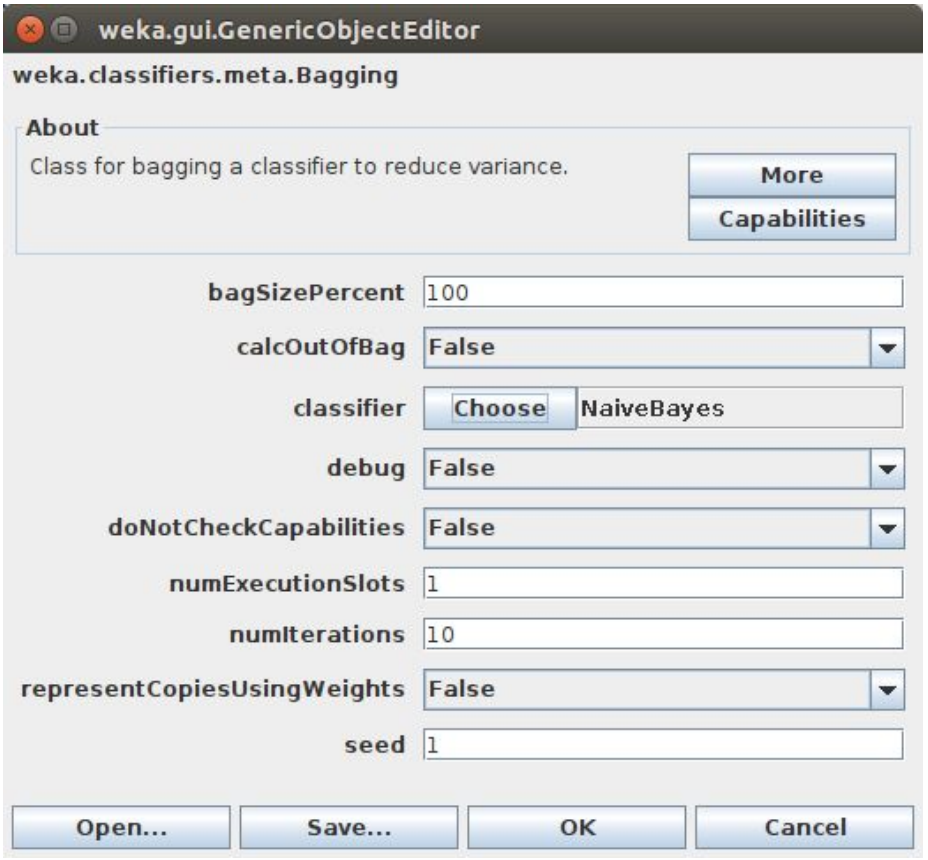


Imagen 2:[Fernando Sancho Caparrini . (26 de Diciembre de 2018). Métodos combinados de aprendizaje ……..]

En la GUI de weka:Para seleccionar **Bagging**, pulsamos **Choose** y en **Meta**, tendremos el método Bagging. Vamos a Configurarlo. En este caso las opciones más importantes son **numIterations** donde marcamos el número de modelos base que queremos crear, y Classifier, donde seleccionamos el método base con el cual deseamos crear los modelos base. Fijamos **numIterations=n ,Classifier=NaiveBayes.**
[José Hernández Orallo. (2006). Curso de Doctorado Extr…]



3.4.2- Boosting

Consiste en ajustar secuencialmente múltiples modelos sencillos, de forma que cada modelo aprende de los errores del anterior. Como valor final, al igual que en *bagging*, se toma la media de todas las predicciones (variables continuas) o la clase más frecuente (variables cualitativas). Tres de los métodos de *boosting* más empleados son *AdaBoost*, *Gradient Boosting* y * Stochastic Gradient Boosting*.

A diferencia del bagging, en el **boosting** no se crean versiones del conjunto de entrenamiento, sino que se trabaja siempre con el conjunto completo de entrada, y se manipulan los pesos de los datos para generar modelos distintos. La idea es que en cada iteración se incremente el peso de los objetos mal clasificados por el predictor en esa iteración, por lo que en la construcción del próximo predictor estos objetos serán más importantes y será más probable clasificarlos bien.

El método de **boosting** más famoso es el que se conoce como **AdaBoost** funcionamiento:
[Joaquín Amat Rodrigo. (Febrero, 2017). Árboles de predicción: bagging, random forest, boosting y C5.0……...]

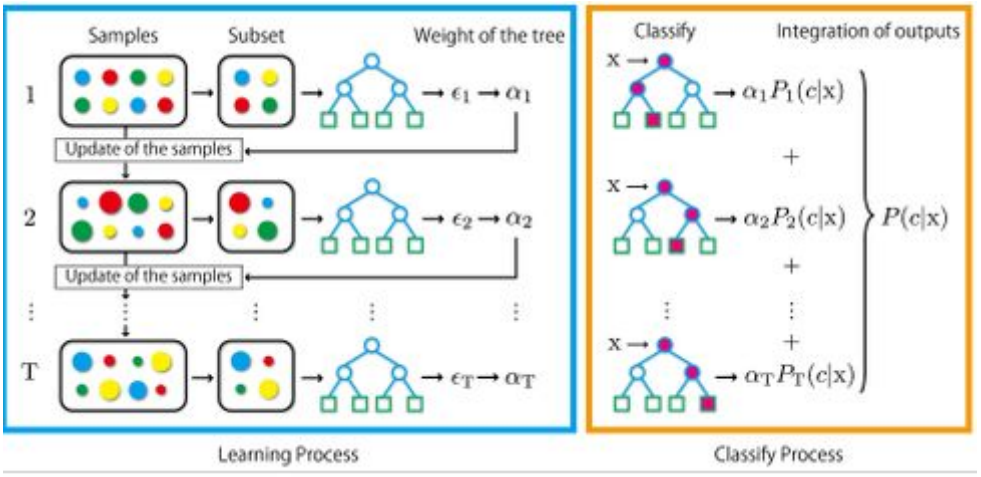
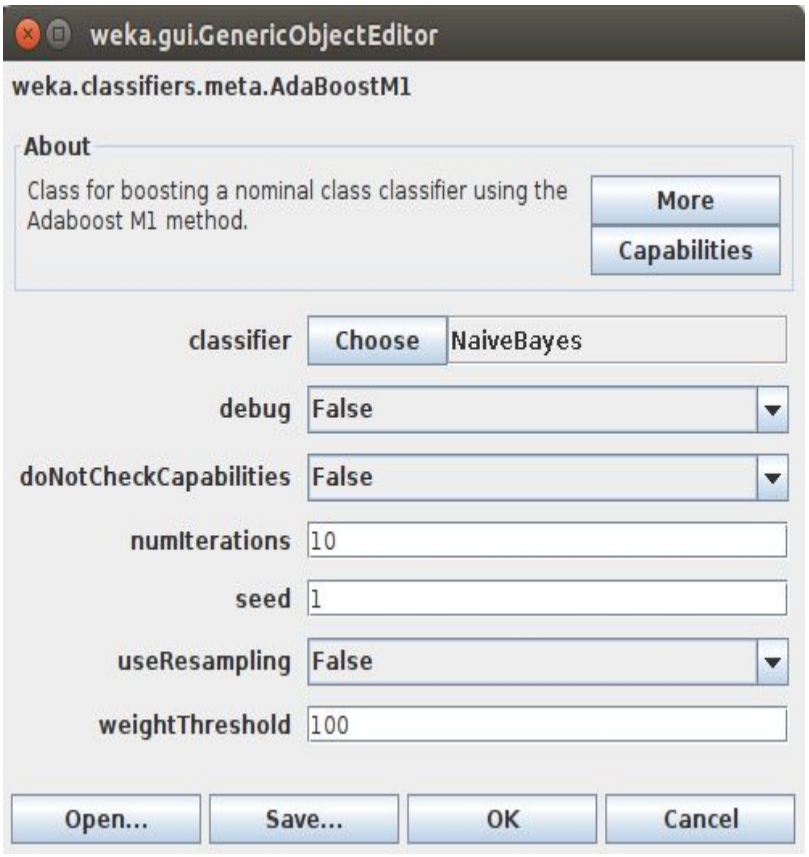


Imagen3[Fernando Sancho Caparrini . (26 de Diciembre de 2018). Métodos combinados de aprendizaje ……..]

En la GUI de weka: Seleccionamos **AdaBoostM1** en **Meta**. En este método las opciones más importantes son también **numIterations** donde marcamos el número de iteraciones máximas (es decir de modelos base), y **Classifier**, donde seleccionamos el método base con el cual deseamos crear los modelos base. Como en caso anterior, fijamos **numIterations=n** y **Classifier=NaiveBayes.**
[José Hernández Orallo. (2006). Curso de Doctorado Extr…]



3.5- Marco Experimental

3.5.1- Experimentación NaiveBayes con y sin bagging:

Como método de experimentación Baseline hemos decidido utilizar el NaiveBayes, un metodo basico pero efectivo para clasificar. Además, hemos decidido dar un paso más allá, y tratar de mejorar la cantidad de aciertos de las clasificaciones obtenidas con NaiveBayes, y para ello, hemos decidido utilizar métodos ensemble, más concretamente un método bagging de clasificadores NaiveBayes.

Para la experimentación, vamos a coger los distintos tipos de datos (Bow, TFIDF) con sus distintas representaciones (Sparse y NonSparse). Con estos datos, vamos a realizar distintas pruebas, tanto con el NaiveBayes, como el NaiveBayes con Bagging, y tanto empleando, como no, la selección de atributos, para lograr ver cuál es el método de clasificación más eficiente en nuestro caso:

En el caso de Bagging, tenemos que el parámetro más sensible es el número de iteraciones a ejecutar, obteniendo mejores o peores resultados en función de este, por ello, lo primero que hemos hecho es hacer un barrido de este parámetro para hallar el óptimo, pero debido al largo tiempo de ejecución, hemos tenido que realizar

un rango desde 5 iteraciones hasta 30, sumando de 5 en 5, y escogiendo como óptima aquella que tenga el mayor f-score para la clase minoritaria empleando 10-fold validation, para ello hemos hecho una prueba con ambos tipos de datos, tanto BoW como TFIDF, como con FSS y sin FSS, hemos obtenido los siguientes resultados:

	BoW NonSparse Sin FSS	TFIDF NonSparse Sin FSS	BoW NonSparse Con FSS	BoW NonSparse Con FSS
5 iteraciones	0.6054	0.5308	0.622222	0.554455
10 iteraciones	0.6010	0.5345	0.610169	0.587064
15 iteraciones	0.6	0.5339	0.625000	0.55999
20 iteraciones	0.6067	0.5327	0.617142	0.565656
25 iteraciones	0.62222	0.5327	0.61797	0.568527
30 iteraciones	0.614525	0.5277	0.62146	0.568526

Utilizando estos resultados obtenemos:

Tipo de datos	Sin Bagging		Con bagging		Con FSS y sin bagging		Con FSS y con bagging	
	% Aciertos (accuracy)	Minoritary f-score	% aciertos (accuracy)	Minoritary f-score	% aciertos (accuracy)	Minoritary f-score	% aciertos (accuracy)	Minoritary f-score
	10-fold validation							
Bow NonSparse	61,06%	0,5673	61,20%	0,612021	60,56%	0,58	60,82%	0,6145
Bow Sparse	61,06%	0.56730	61,20%	0,612021	60,56%	0,58	60,82%	0,6145
TFIDF NonSparse	60,58%	0,47148	61,83%	0,556603	60,25%	0,5529	61,18%	0,5870
TFIDF Sparse	60,58%	0,47148	61,83%	0,556603	60,25%	0,5529	61,18%	0,5870

	Hold out 10 veces (70% train- 30% test)							
Bow NonSparse	61,12%	0,54166	61,24%	0,52173	60,26%	0,5416	60,81%	0,5217
Bow Sparse	61,12%	0,54166	61,24%	0,52173	60,26%	0,5416	60,81%	0,5217
TFIDF NonSparse	60,88%	0,4905	61,12%	0,5098	60,33%	0,4727	60,88%	0,4814
TFIDF Sparse	60,88%	0,4905	61,12%	0,5098	60,33%	0,4727	60,88%	0,4814
	No honesta							
Bow nonSparse	62,45%	0,625	62,80%	0,64039	61,610%	0,62999	61,72%	0,6294
Bow Sparse	62,45%	0,625	62,80%	0,64039	61,610%	0,62999	61,72%	0,6294
TFIDF NonSparse	61,72%	0,4835	62,82%	0,55021	60,82%	0,56108	61,40%	0,59047
TFIDF Sparse	61,72%	0,4835	62,82%	0,55021	60,82%	0,56108	61,40%	0,59047

Tras realizar las pruebas, nos dimos cuenta de que en los datos Sparse y NonSparse obtenemos los mismos resultados, que obtenemos mejores resultados sin aplicar FSS, y aplicando bagging, y que los mejores resultados los obtenemos con los datos en BoW.

Ademas, en la mayoria de los casos, los datos obtenidos con el algoritmo baseline, en este caso NaiveBayes, son mejores a los obtenidos cuando aplicamos FSS y utilizamos tanto el algoritmo baseline como bagging.

Aun así, mientras realizamos la experimentación, nos dimos cuenta de que ciertas evaluaciones tardaban mucho más tiempo que otras, por tanto, antes de realizar unas conclusiones, decidimos volver a ejecutar todas las operaciones cronometrando el tiempo que tardan, y obtuvimos los siguientes resultados:

Tiempos con NaiveBayes:

	Bow NonSparse	BowSparse	TFIDF Sparse	TFIDF NonSparse
10 fold validation Sin FSS	13 segundos	15 segundos	15 segundos	13 segundos
10 fold validation Con FSS	4 segundos	6 segundos	6 segundos	4 segundos
Hold out 10 veces Sin FSS	30 segundos	31 segundos	31 segundos	30 segundos
Hold out 10 veces con FSS	11 segundos	12 segundos	12 segundos	11 segundos
No honesta Sin FSS	9 segundos	10 segundos	10 segundos	9 segundos
No honesta con FSS	3 segundos	4 segundos	4 segundos	3 segundos

Tiempos con bagging:

	Bow NonSparse	BowSparse	TFIDF Sparse	TFIDF NonSparse
10 fold validation Sin FSS	131 segundos	156 segundos	155 segundos	129 segundos
10 fold validation con FSS	47 segundos	61 segundos	61 segundos	46 segundos
Hold out 10 veces Sin FSS	300 segundos	324 segundos	319 segundos	296 segundos
Hold out 10 veces Con FSS	113 segundos	122 segundos	121 segundos	109 segundos
No honesta Sin FSS	95 segundos	99 segundos	96 segundos	92 segundos
No honesta con FSS	36 segundos	37 segundos	37 segundos	34 segundos

Tras realizar estas pruebas de tiempos, nos damos cuenta de que el FSS empleado es muy ineficiente, ya que obtenemos mejores resultados en menor tiempo utilizando el algoritmo baseline sin FSS, que utilizando bagging con FSS, por tanto, consideramos que es mejor no utilizarlo. Aunque utilizando el algoritmo baseline y empleando FSS obtenemos los resultados en el menor tiempo, estos resultados son los peores de todos los métodos, y además al ser los tiempos del algoritmo baseline tan pequeños tanto utilizando FSS como sin utilizarlo es más conveniente no aplicar nunca FSS y obtener mejores resultados.

Además, hemos visto que aunque con bagging obtenemos mejores resultados que sin aplicarlo, la diferencia de tiempo entre utilizar bagging y no utilizarlo es muy grande, y la diferencia de aciertos entre estos no es muy grande, por tanto, es una diferencia muy grande, y creemos que utilizar baseline es mejor salvo en ciertas ocasiones en las que se necesite el mayor número de aciertos posibles sin importar el tiempo empleado.

Entre los métodos de representación Sparse y NonSparse, obtenemos los mismos resultados, pero en todos los casos, utilizar una representación Sparse supone un mayor tiempo, por tanto, lo mejor será emplear siempre una representación NonSparse, ya que obtenemos los mismos resultados en un menor tiempo.

La diferencia de tiempos entre BoW y TF-IDF tanto con bagging como baseline no son muy grandes, por tanto, para tener mejores resultados, lo mejor será utilizar datos en BoW

3.5.2- Experimentación Ramdon Forest

Estas experimentaciones están hechas bajo una cota realista, donde por cada tabla y por cada columna fila obtenemos un modelo con un **F-measure** y un t**iempo** de ejecución. Finalmente analizamos cada tabla y por cada una de ellas destacamos el **F-Measure** más alto ,en el caso de que de él F-mesure más alto aparezca en la misma tabla pero con diferentes parámetros columna fila ,cogeremos el **F-measure** con un menor tiempo de ejecución.

Se puede observar que el **F-measure** más alto de cada tabla depende de un mayor número de árboles y un número de atributos que corren mejor con la clase, no puedo asegurar que sea en todos los casos,pero en la mayoría de las tablas si de acuerdo a nuestras experimentaciones. Hemos intentado hacer una experimentación lo más exhaustiva posible pero el tiempo de ejecución en varios casos ha sido elevado si las iteraciones iban de 1 en 1 con un numero de árboles elevado, con experimentaciones con saltos de 10,50,35..ect en el número de árboles los tiempo de ejecuciones han ido disminuyen así hemos podido elevar el número de árbol considerablemente en nuestras pruebas, de la misma forma con los atributos.

Como punto positivo en los tiempos de ejecuciones cabe destacar una importancia de rapidez con los tipos de archivo en la representación **Sparse** con **selección de atributos**(BowSpasreInfogain,TfIdfSparseInfoGain) ya que este filtro nos proporciona un espacio más reducido de atributos por la eliminación de atributos redundantes, esto contribuye en una mejora de eficiencia de **tiempo** y **predicción** ya que nos deja con los atributos que aportan mayor información por lo tanto correrán mejor con la clase.

Hemos elegido estos parámetros para la optimización por qué son los más importantes y aportan una mayor información para la eficiencia de nuestro clasificador. Los parámetros elegidos han sido **núm Trees**: especifica el número de árboles y en **num Features**: Especifica el número de atributos. Observamos que con un mayor número de árboles obtenemos una mejor predicción, y con los atributos que corren mejor con la clase, con respecto al número de atributos en la iteración de cada parámetro obtenemos el número óptimo de atributos.

Iteraciones Sin FSS (train (1083) y dev(324))

Bow/NonSparse (Train/Dev)	5 Trees	25 Trees	50 Trees
6 Features	***MaxFMeasure:***0,6232701178882624 ***Tiempo*** :271,507 seg	***MaxFMeasure***:0,7594 ***Tiempo***:6919,341 seg	***MaxFMeasure:***0,777 ***Tiempo:*** 35038,062 seg
13 Features	***MaxFMeasure:*** 0,6381 ***Tiempo :*** 1176,374 seg	***MaxFMeasure:*** 0,7615 ***Tiempo:*** 58211,016 seg	*
21 Features	***MaxFMeasure:*** 0,653 ***Tiempo:*** 3239,942 seg	**MaxFMeasure:** 0,7652 **Tiempo:**91519,376 seg	*

(*):Tiempo de ejecución demasiado prolongado(>6 horas con iteraciones uno a 1 a 1)

Bow/NonSparse (Train/Dev)	550 trees
6 Features	***MaxFMeasure:*** 0,8 **MaxTree**:150 **MaxFeature:**6 ***Tiempo:***10464seg

Bow/Sparse (Train/Dev)	5 Trees	25 Trees	50 Trees
6 Features	***MaxFMeasure***:0,679 ***Tiempo:***120,622 seg	***MaxFMeasure***:0,754 ***Tiempo:***3572,941 seg	***MaxFMeasure:***0,7794 ***Tiempo:***12834,757 seg
13 Features	***MaxFMeasure:*** 0,6802 ***Tiempo:*** 456,694 seg	***MaxFMeasure:*** 0,775 ***Tiempo:*** 20264,966 seg	*
21 Features	***MaxFMeasure:*** 0,7165 ***Tiempo:*** 1447,646 seg	*	*

(*):Tiempo de ejecución demasiado prolongado(>6 horas con iteraciones 1 a 1)(con clase mayoritaria)

Bow/Sparse (Train/Dev)	350 trees
12 Features	***MaxFMeasure:***0,775 ***maxfeatures:***6 ***Maxtrees:***300 ***Tiempo:***9500,618seg

(**)Experimentación con clase minoritaria

Tf-Idf/NonSparse (Train/Dev)	5 Trees	25 Trees	50 Trees
5 Features	***MaxFMeasure:***0,6613924050632911 ***Tiempo:***527,17seg	***MaxFMeasure:***0,764 ***Tiempo:***9876,765 seg	*
12 Features	***MaxFMeasure***:0,6699 ***Tiempo:***2183,34 seg	*	*
20 Features	***MaxFMeasure:***0,6735 ***Tiempo:***5904,058 seg	*	*

(*):Tiempo de ejecución demasiado prolongado(>6 horas con iteraciones 1 a 1)

Tf-Idf/NonSparse (Train/Dev)	550 trees
6 Features	***MaxFMeasure:***0,8 **Maxtree**:158 **MaxFeatures**=6 ***Tiempo:***11721,161 seg

(**)Experimentación con clase minoritaria

Tf-Idf/Sparse (Train/Dev)	5 Trees	25 Trees	50 Trees
5Features	***MaxFMeasure:***0,701 ***Tiempo:***241,857seg	***MaxFMeasure:***0,7707838479809975 ***Tiempo:***1685.815 seg	
12 Features	***MaxFMeasure:***0,71388 ***Tiempo:***7006,00 seg	***MaxFMeasure:***0,774 ***Tiempo:***45397.612 seg	*
20 Features	***MaxFMeasure***:0,71388 ***Tiempo:***1487,632 seg	*	*

(*):Tiempo de ejecución demasiado prolongado(>6 horas con iteraciones 1 a1)(Experimentaciones con la clase mayoritaria)

Tf-Idf/Sparse (Train/Dev)	550 trees
12 Features	***MaxFMeasure***:0,783 **Maxfeatures**:11 **MaxTrees:**150 ***Tiempo:***16748,945 seg

(**)Experimentación con la clase minoritaria

Iteraciones con FSS

Bow/Sparse/infoGain (Train/Dev)	5 Trees	25 Trees	50 Trees
5 Features	***MaxFMeasure:***0,7215 ***Tiempo:***71,758seg	***MaxFMeasure***:0,7559 ***Tiempo:***1488,692seg	***MaxFMeasure***:0.7638804148871262 ***Tiempo:***221.697 seg
12 Features	**MaxFMeasure:**0,72157 **Tiempo:**353,175 seg	**MaxFMeasure:**0.7613365155131266 **Tiempo:**7462.317 seg	***MaxFMeasure:*** 0.7615571776155717 ***Tiempo:***463.21 seg
20 Features	**MaxFMeasure:**0,72250 **Tiempo:**989,123 seg	**MaxFMeasure:**0.7551266586248493 **Tiempo:**421.18 seg **MaxFeatures:**5	**MaxFMeasure:**0.7607142857142857 **Tiempo:**1085.624 seg

(*):Se han usado tanto iteraciones con salto 1 a1 como con saltos mayores en las iteraciones(Experimentaciones con l clase mayoritaria)

BowSparse//InfoGain	550 Trees
12 Features	***MaxFMeasure:****0,74* ***MaxTree:****350* ***Tiempo:***10673 seg **MaxFeature**:11

(**)(Experimentación con la clase minoritaria)

Tf-Idf/Sparse/infoGain	5 Trees	25 Trees	50 Trees

(Train/Dev)			
5 Features	***MaxFMeasure:***0,7072 ***Tiempo:***108,433seg	***MaxFMeasure:***0,7685 ***Tiempo:***1539,523 seg	*
12Features	***MaxFMeasure:***0,722 ***Tiempo:***335,981 seg	***MaxFMeasure:***0,7685 ***Tiempo:***12407,464 seg	*
20 Features	***MaxFMeasure:***0,728 ***Tiempo:***942,265 seg	*	*

(*):Tiempo de ejecución demasiado prolongado(>6 horas con iteraciones 1 a1)experimentaciones con la Clase mayoritaria

Tf-Idf/Sparse/infoGain (Train/Dev)	550 trees
6 Features	***MaxFMeasure:****0,7378* ***MaxTree:****150* **MaxFeature**:6 ***Tiempo:***2644,903 seg

(**)Experimentación con la clase minoritaria

Iteración con Fss (train(1001) y dev(300)) modelo usado para la mejor prediccion

BowNonSparse InfoGain	**550 Trees**
6 Features	**maxFeature:**6 **maxTree:**150 **maxFemeasure:**0,7639 **tiempo:**6468,733 seg

3.5.3- Experimentación calidad esperada

Para esta experimentación hemos cogido los modelos con un f- Measure más altos de cada tabla de los diferente espacios de represention (BowNonSpare,BowSparse, TfIdfNonSparse,TfIdfSparse,BowSparseInfoGain ,TfIdfSparseINfoGain) teniendo en

cuenta el número de árboles(numTrees) y de atributos(NumFeatures).Como podemos observar nuestros modelo randomforest espera una calidad muy buena para cada esquema de valuación.

No honesta:

Las mediciones de la no honesta nos permiten medir la cota superior de nuestro modelo predictor

Tipo	No- Honesta		
	Precisión(Minoritary)	Recall(Minoritary)	F-Measure(minoritary)
BowNonSparse	0,9855	0,7906	0,877
BowSparse	0,9855	0,7906	0,877
TfIdfNonSparse	0,936	0,686	0,791
TfIdfSparse	0,985	0,7906	0,8774
BowSparseInfoGain	0,970	0,7558	0,849
TfIdfSparseInfoGain	0,9696	0,744	0,842
BowNonSparseInfoGain	0,9565	0,8	0,871

10 Fold Cross validation:

Con k-fold podemos hacer unas estimaciones que se acercan mas a lo que nuestro modelo va a darnos como resultado despues.

Tipo	10-Fold Crossvalidation		
	Precisión(Minoritary)	**Recall(Minoritary)**	**F-Measure(minoritary)**
BowNonSparse con Fss	0,812	0,604	0,694
BowSparse	0,769	0,581	0,662
TfIdfNonSparse	0,75	0,593	0,662
TfIdfSparse	0,771	0,627	0,6923
BowSparseInfoGain	0,812	0,604	0,69934
TfIdfSparseInfoGain	0,854	0,616	0,7162
BownonSparseInfoGa	0,772	0,618	0,686

Hold out:

Tipo	Hold out		
	Precisión(Minoritary)	Recall(Minoritary)	F-Measure(minoritary)
BowNonSparse con Fss	0,978	0,8	0,880
BowSparse	0,978	0,8	0,880
TfIdfNonSparse	0,978	0,8	0,880
TfIdfSparse	0,978	0,8	0,880
BowNonSparseInfoGain	0,82612	0,7209	0,78448

3.6- Baggaging Baseline vs Random Forest

Esta Experimentación se realiza primero en base a una cota superior, pero las conclusiones las tomamos en base a una cota realista (10 fold validation y hold out)

Tipo	No-Honesta	
	F-Measure	Acurracy%
BowNonSparseInfoGain RandomForest	0,871	98.4436 %
TfIdfNonSparse RandomForest	0,792	84,6478%
BowNonSparse Bagging Naive Bayes	0,64	62,80%
TfIdfNonSparse Bagging naive Bayes	0,55021	62,82%

Tipo	10 Fold Cross Validation	
	F-Measure	Acurracy%
Bow NonSparseInfoGain RandomForest	0,687	72.846 %
TfIdfNonSparse RandomForest	0,662	75,4952%
BowNonSparse Bagging Naive Bayes	0,612021	61,20%
TfIdfNonSparse Bagging naive		

Bayes	0,556603	61,83%

Tipo	**Hold Out**	
	F-Measure	**Acurracy%**
BowNonSparseInfoGain RandomForest	0,785	89.8569 %
TfIdfNonSparse RandomForest	0,792	83,8521%
Bow NonSparse Bagging Naive Bayes	0,52173	61,24%
TfIdf NonSparse Bagging naive Bayes	0,5098	61,12%

3.6.1 Experimentación predicciones con el test

A la hora de predecir, en este caso, tenemos las respuestas del test, algo que no es usual, por lo que podemos hacer un cálculo real de la tasa de aciertos, lo que nos sirve para comparar con la tasa esperada que hemos obtenido en anteriores experimentaciones.

Al realizar las predicciones y comprobar los resultados, hemos obtenido unos resultados bastante inesperados, y es que, aunque en un principio nos parecía que nuestros clasificadores iban a obtener muy buenos resultados, nos hemos dado cuenta, de que contra el test, nuestros clasificadores funcionan bastante mal, como podemos ver en la siguiente tabla:

	Calidad esperada	Calidad real
Baseline BoW NonSparse	61,06%	16,6%
Bagging TFIDF NonSparse	61,83%	22,8%
RandomForest BoW NonSparse	72.846 %	25.6%

Aunque se cumplan ciertas experimentaciones, como que bagging tarda más que baseline y con ello se obtienen mejores resultados, el número de aciertos reales obtenidos no se acerca para nada a nuestras estimaciones, por tanto, creemos que nuestros estimadores no son buenos para este test, aunque si son buenos para las instancias del train (si dividimos el train en train y dev, y evaluamos esta segunda, obtenemos unos resultados muy buenos), y deberíamos buscar otros algoritmos mejores para esta tarea.

4- Detalles del diseño e implementación de software e implementación de software

De cara al **diseño**: Se ha planteado hacer un diseño modular con distintos métodos independientes del programa principal, para poder así aprovechar ciertos métodos comunes a las distintas funcionalidades como cargar un fichero .arff, guardar…

En la **implementación del software** se ha intentado utilizar al máximo métodos ya implementados en la librería de Weka ,investigando tanto en la wiki de weka como en la GUI de weka ,para los distintos parámetros ,filtros etc.., por tanto, prácticamente todo el código es utilizando la librería ya proporcionada por Weka, siendo únicamente implementado por nosotros, en la fase de preprocesamiento del archivo .csv, un método para eliminar caracteres extraños que nos daban bastantes problemas a la hora de convertirlo en .arff, debido a que estos caracteres Weka los interpretaba como palabras reservadas, realizándose incorrectamente la conversión.

4.1- Distribución de tareas:

	Nicolas	**Xabier**	**Andrea**
Primera fase	Investigación y Conocimiento previo de la tarea Implementación de conversión RAW y TransformRaw Documentación de la parte teórica de las tres partes del proyecto y actualización de las bibliografías. Demo de la parte 1 de la presentación del proyecto.	preprocesamiento Investigación e implementación de conversión RAW, MakeCompatible y documentación práctica de los distintos tipos de datos (raw, BoW,TF-IDF) Mejora de la documentación de la parte 1 con sus respectivas bibiografias junto al marco experimental.	Investigación y conocimiento previo de la tarea Documentación de la parte teórica de Preprocesamiento de datos e implementación de la transformación a Non-sparse.
Segunda parte		Inferencia del clasificador Investigación,conocimiento e implementación de los métodos Baseline ,Bagging Baseline con sus respectivas calidades y el métodos de predicciones	Inferencia del clasificador Investigación , conocimiento previo e implementación de los métodos FSS,holdout ,Random Forest y sus respectivas calidades junto al marco experimental. Documentación de la parte teórica de la inferencia del clasificador
Tercera parte		Calidad Esperada del Baseline y Bagging Baseline Demo de la parte 2 Clasificador Baseline y Bagging Baseline	Calidad Esperada del RandomForest Demo de la parte 2 clasificador randomForest

5- Conclusiones

Haciendo un razonamiento crítico a lo largo de las tres partes del proyecto, podemos decir con respecto a la **parte 1** que no hemos hecho añadido nada nuevo a lo que se ha pedido, ni lo hemos hecho de una forma especial ni muy eficiente, simplemente hemos tomado los datos en el formato inicial, hemos eliminado aquellos caracteres problemáticos y pasado al formato requerido por Weka, además, hemos alcanzado las representaciones y transformaciones requeridas, e implementada la forma de hacer compatibles varios conjuntos de instancias.

En cuanto a la **parte 2** se han hecho muchas pruebas experimentales de cada clasificador ,en el caso del randomforest al parecer se ha conseguido un modelo bastante bueno según las pruebas y calidad esperada del mismo ,sin embargo a la hora de probar con el test y hacer la correspondiente predicciones con la clase real y estimada ,llegamos a la conclusión de que ,nuestro modelo no aprende bien y tiene poco porcentaje de aciertos con lo cual no se corresponde con la calidad esperada del mismo ,podemos decir que esa es una **debilidad** en nuestro software y con ello deducimos que tiene mucho que ver con el preprocesado.

En cuanto al algoritmo Baseline utilizado, y su mejora utilizando el método ensemble bagging, nos hemos dado cuenta, de que los métodos ensemble pueden ser una forma sencilla de mejorar la eficiencia de los algoritmos, aun así, al igual que con el caso del randomforest, aunque al parecer conseguimos un modelo decente, nos hemos dado cuenta de que nuestro modelo no se comporta de forma correcta con el test, y tenemos un porcentaje de aciertos parecido al conseguido con él randomforest.

Con respecto al trabajo futuro intentaremos estudiar mejor el preprocesado, mejorar los tiempos de las experimentaciones ,con más pruebas eficientes con respecto al tiempo ayudándonos con las saltos en la iteraciones , con servidores virtuales etc ...con el fin de hacer un barrido más óptimo y de un porcentaje mejor de aciertos. Además, en el caso de los métodos ensemble, podríamos probar a realizarlo con otros métodos como el Boosting ya explicado en el marco teórico, y con otros algoritmos más avanzados en vez de con NaiveBayes, para así lograr una mayor eficiencia en las predicciones.

6- Bibliografía

6.1- Bibliografía General

Esther Ribas. (8 Enero 2018). ¿Qués es el Data Mining?. 17/03/2020 Marzo, de IEBES Sitio web: https://www.iebschool.com/blog/data-mining-mineria-datos-big-data/

NA. (NA). Datamining (Minería de datos). 17/03/2020, de Sinnexus Sitio web: https://www.sinnexus.com/business_intelligence/datamining.aspx

NA. (30/12/2016). Calidad de datos en minería de datos a través del preprocesamiento. 17/03/2020, de PowerData Sitio web: https://blog.powerdata.es/el-valor-de-la-gestion-de-datos/calidad-de-datos-en-mineria-de-datos-a-traves-del-preprocesamiento

Miriam Cecilia Moreno Luján. (NA). 5 ejemplos de aplicación de minería de datos. 18/03/2020, de monografias Sitio web: https://www.monografias.com/trabajos81/5-ejemplos-aplicacion-mineria-datos/5-ejemplos-aplicacion-mineria-datos.shtml

Jocelyn D'Souza. (03/04/2018). An Introduction to Bag-of-Words in NLP. 18/03/2020, de Medium Sitio web: https://medium.com/greyatom/an-introduction-to-bag-of-words-in-nlp-ac967d43b428

Rodolfo Fernández González. (NA). Técnica de Inteligencia Artificial en Minería de Datos. 19/03/2020, de la Universidad Complutense de Madrird (UCM) Sitio web: https://webs.ucm.es/info/pslogica/rodolfo2.pdf

Aitor Esteve Alvarado. (02/10/2013). Estudio y Piloto de soluciones de análisis de contenido sobre información no estructurada. 19/03/2020, de Universitat Politècnica de Catalunya Sitio web: https://upcommons.upc.edu/bitstream/handle/2099.1/19300/91652.pdf?sequence=1&isAllowed=y

Guión Parte 1 de eGela

Witten, Ian H. & Frank, Eibe & Hall, Mark A. & Pal, Christopher J.. (2011). Data Mining: Practical Machine Learning Tools and Techniques (Morgan Kaufmann Series in Data Management Systems).

6.2- Bibliografía Baseline

Victor Roman. (Abril, 2019). Algoritmos Naive Bayes: Fundamentos e Implementación. Marzo, 2020, de Medium Sitio web: https://medium.com/datos-y-ciencia/algoritmos-naive-bayes-fudamentos-e-implementaci%C3%B3n-4bcb24b307f

José Francisco López. (N/A). Teorema de Bayes. Marzo, 2020, de Economipedia Sitio web: https://economipedia.com/definiciones/teorema-de-bayes.html

M.Ing. Enrique Fernández. (N/A). ANÁLISIS DE CLASIFICADORES BAYESIANOS. Marzo, 2020, de LABORATORIO DE SISTEMAS INTELIGENTES Sitio web: http://materias.fi.uba.ar/7550/clasificadores-bayesianos.pdf

6.3- Bibliografía RandomForest -Bagging-Boosting

Wikipedia. (2013). Random forest. Marzo 2020, de Wikipedia Sitio web: https://es.wikipedia.org/wiki/Random_forest

Joaquín Amat Rodrigo. (Febrero, 2017). Árboles de predicción: bagging, random forest, boosting y C5.0. Marzo 2020, de N_A Sitio web: https://www.cienciadedatos.net/documentos/33_arboles_de_prediccion_bagging_random_forest_boosting#m%C3%A9todo _de_ Ensemble

N_A. (N_A). Introducción al Aprendizaje Automático y a la Minería de Datos con Weka HERRAMIENTAS DE LA INTELIGENCIA ARTIFICIAL INGENIERÍA INFORMÁTICA. Marzo 2020, de MDWEBHIA Sitio web: http://ocw.uc3m.es/ingenieria-informatica/herramientas-de-la-inteligencia-artificial/contenidos/transparencias/MDWEBHIA-clase.pdf

José Hernández Orallo. (2006). Curso de Doctorado Extracción Automática de Conocimiento en Bases de Datos e Ingeniería del Software. Marzo, 2020, de Universidad Pública Valencia Sitio web: http://personales.upv.es/ceferra/weka/CursDoctorat-weka.pdf

*Imagen1:wjk68@case.edu.. (2017). Random Forest Simple Explanation. Marzo 2020, de N_A Sitio web:https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d
*Imagen 2:Fernando Sancho Caparrini . (26 de Diciembre de 2018). Métodos combinados de aprendizaje . Abril 2020, de N_A Sitio web: http://www.cs.us.es/~fsancho/?e=106

*Imagen3:Fernando Sancho Caparrini . (26 de Diciembre de 2018). Métodos combinados de aprendizaje . Abril 2020, de N_A Sitio web: http://www.cs.us.es/~fsancho/?e=106

Fernando Sancho Caparrini . (26 de Diciembre de 2018). Métodos combinados de aprendizaje . Abril 2020, de N_A Sitio web: http://www.cs.us.es/~fsancho/?e=106

Jason Browniee. (13 julio 2016). How to Perform Feature Selection With Machine Learning Data in Weka. 1 Abril , de Machine Learning Mastery Sitio web: https://machinelearningmastery.com/perform-feature-selection-machine-learning-data-weka/