Minería de datos
Práctica 5

Iñigo Sánchez

7 de diciembre de 2014

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

1. Introducción

El objetivo principal de esta práctica es obtener la capacidad de trabajar en equipo simulando un problema de minería de datos "real" .

Por otra parte, de manera individual,en este caso se trabajan las capacidades de implementar barridos de parámetros con el objetivo de hallar un óptimo para un conjunto de posibles clasificadores y continuar adquiriendo agilidad a la hora de hacer uso de librerías de minería de datos.

En esta ocasión, ha sido necesario ampliar la librería de Weka para poder utilizar el clasificador SVM con la librería libSVM. Además, se profundiza en la utilización de los distintos filtros, pudiendo ser necesaria su aplicación por diferentes motivos, los cuales serán debidamente desarrollados a lo largo de este documento.

Por otro lado, dado el carácter individual, decido abordar el problema con otra librería: *scikit-learn*, escrita en *Python* y con la librería *matplotlib* para visualizar los resultados y la evolución de los barridos.

2. Recursos

- Software de Weka
- Librerías de Weka.
- Manual de Weka.
- Librería libSVM
- Librería scikit-learn Python.
- Librerías matplotlib,numpy y scipy python.
- Librería LIAC-ARFF v2.0 Python.
- Repositorio proyecto java: `https://github.com/spolex/datamining-modules.git`
- Repositorio proyecto python: `https://github.com/spolex/pythonDataMining.git`
- Ficheros para los datos de la práctica: train.e2e.wOOV.obfuscated.arff, dev.e2e.wOOV.obfuscated.arff. test.e2e.wOOV.obfuscated.arff

3. Descripción y características de los datos

El conjunto de datos se presenta de-identificado, por lo que no se conoce la naturaleza real del problema al que nos enfrentamos, es decir sólo podemos analizar las características los datos sin la influencia del contexto del cual provienen. Esta es la primera "limitación" que nos encontramos. En las siguientes secciones se pasa a analizar de una forma más detallada el conjunto de los datos.

3.1. Class

El conjunto de datos representa un problema de clasificación con una clase que puede tomar dos valores: "V1" o "V2". Al analizar el conjunto de entrenamiento con Explorer de Weka nos encontramos con el primer problema, la clase está desbalanceada:

3.1.1. Representación de la clase positiva

En problemas donde la clase positiva no está suficientemente representada, no es posible entrenar un algoritmo para que obtenga unas medidas de desempeño muy altas. En este caso sin preproceso alguno y utilizando el clasificador J48 con los parámetros por defecto y utilizando como método de evaluación Hold-out, nos encontramos con una recall mayor de 0,9.

Aún así esta medida no es representativa para este tipo de problemas donde la clase que nos interesa no tiene representación. En el experimento anterior el resultado para la F1-score de la clase positiva es igual a 0.

Esto demuestra que el recall no es representativo para estos problemas. Por esto, la medida de desempeño que se utiliza para obtener y optimizar un clasificador, capaz de predecir con suficiente precisión instancias de las que no conocemos su clase es la F1-measure de la clase positiva.

3.2. Features

Cada instancia está representada por un espacio de 54 atributos y la clase. De los cuales sólo 7 son numéricos.Esto también es un problema ya que analizar que atributos correlacionan mejor con la clase, cuales tienen o no intercorrelación, cuales son más informativos...etcétera, llevaría un tiempo del cual no se dispone dado que el *deadline* se supone en tres semanas.

Otro problema derivado de los atributos, es la poca representación cuantitativa con respecto a la representación cualitativa. Es decir alrededor de sólo un 15 % de los atributos son numéricos y se desconoce por ahora si son o no relevantes. Por otro lado aunque los atributos categóricos nos pueden aportar información suficiente, suponen un coste y esfuerzo extra para poder procesarlos en el formato adecuado, antes de que un algoritmo de ML sea capaz de utilizarlos para ser entrenado.

3.2.1. Missing Values

Este problema no presenta perdida de información en forma de *Missing Values*.

3.3. Distinct Values

Esto es un gran problema, sobre todo a la hora de computar con los PC personales antes de preprocesar los datos, para obtener comparativas y resultados que justifiquen los procesos de selección de atributos aplicados. Además no siempre los atributos con más valores distintos son descartados por los preprocesos aplicados.

Este problema tiene una severidad importante además de encontrarse en muchos de los atributos que modelan el espacio de características.

4. Modelos

Para las distintas tareas a abordar individualmente, se han escogido las máquinas de soporte vectorial, escogiendo el ajuste suficiente en cada caso. En las siguientes secciones se explicará que algoritmos se han utilizado y para que tarea.

4.1. SVM

Aunque se decide aplicar preproceso a los datos para facilitar la elección y parametrizado de los modelos, mi elección es utilizar SVM por que es capaz de resolver problemas con la representación de la clase no balanceada, penalizando el error de clasificación. Además también es posible implementar un modelo *OneClass Classifier* capaz de detectar *outliers*. A continuación se justifica con una lista de fortalezas y debilidades que se han tenido en cuenta.

4.1.1. Fortalezas

- El entrenamiento es relativamente sencillo.Aunque aumenta el coste para entrenar este algoritmo.
- No hay óptimo local.
- Se escalan bien para datos en espacios dimensionales altos.
- El compromiso entre la complejidad del clasificador y el error puede ser controlado explícitamente.
- Versátil: pueden ser usados diferentes kernel para construir más de un modelo.Es posible implementar un kernel propio para especificar nuestra hipótesis.

4.1.2. Debilidades

- Si el número de muestras es mucho más pequeño que el número de muestras, el método podría perder eficiencia.
- SVM no proporciona directamente de las probabilidad de la estimación. Hay que utilizar otros métodos que aumentan el coste computacional del algoritmo.[4]

4.2. Optimización de hiperparámetros

4.3. *"Barrido"* ad-hoc

El barrido ad-hoc consiste en crear un programa específico para resolver un problema de optimización concreto. Este programa optimizará el clasificador C-SVM con distintos kernel para los problemas de naturaleza similar a la del actual, y que es detallada a lo largo del documento, siempre que éstos cumplan las condiciones necesarias para hacer uso de estos algoritmos. Con esta técnica se busca obtener dos o más modelos de clasificación, capaces de obtener una F1-score de la clase positiva igual o mayor a 0.5

En este caso, la *"score"* utilizada será la F1-score de la clase considerada positiva, que siempre será la que ocupa el primer lugar. Sin embargo, mediante unas pocas modificaciones en el código, es posible buscar óptimos en función a otras figuras de mérito.

4.4. GridSearch

En este caso utilizo la librería scikit-learn implementada en python que facilita las clases y métodos para esta tarea, me ha parecido suficiente, ya que permite, al igual que la API de *weka*, serializar el módelo y obtener las medidas de desempeño.

Sin embargo he tenido que implementar la función de *scoring* para que el barrido grid tenga en cuenta la misma medida de desempeño que la utilizada con el método ad-hoc implementado este último tanto en java como en python, haciendo uso de las librerías correspondientes.

Como ventaja me he encontrado la potencia gráfica de python, qué haciendo uso de la librería matplotlib me ha permitido visualizar la evolución de los métodos de optimización y los resultados, para un posterior análisis.

He de decir que de manera personal me inclino hacia el análisis gráfico, y matemático cuando los conocimientos me lo permiten, antes que el análisis frente a un software de referencia. Es por esto que he tratado de profundizar en la potencia de representación gráfica de python, que me ha resultado sumamente útil, para visualizar y comprender mejor el funcionamiento de las máquinas de soporte vectorial.

5. Diseño y algoritmos

5.1. Preproceso de los datos

Para preprocesar los datos se ha considerado adecuado el uso de los siguientes filtros:

5.1.1. Randomize

Tras un análisis de cómo funciona SVM a la hora de generar el clasificador, la conclusión es que el orden de las instancias no alterará el resultado, ya que esto no afectará a la disposición de éstas en el espacio muestral que se corresponda.

5.1.2. Normalize

El objetivo de normalizar los atributos de las instancias es evitar que los atributos en rangos numéricos superiores tengan mayor relevancia que aquellos en rangos menores[2, sección 2.2]. Este proceso, en nuestro modelo, debería escalar todos los atributos numéricos a un intervalo de [-1,1], dado que en el conjunto de datos de entrenamiento, los valores de algunos de los atributos son inferiores a cero.[3, sección 2.2,4]

No obstante la última implementación hecha con la API de weka no se ha normalizado, dando por muy bueno el último scoring obtenido.

En cambio y dado que en python hago uso de representaciones gráficas, en este caso si se normalizan las instancias. Para ello hago uso del método *StandarScaler()* que estandariza las características mediante la eliminación de la media y el cociente del resultado con la varianza.

5.1.3. Outliers y Extreme Values

Este filtro se aplica debido a que es posible que haya instancias de entrenamiento que, aún siendo de una clase determinada, tengan rasgos poco comunes a ésta, lo que pueden llevar a ser un clasificador menos preciso.

Esta circunstancia puede ser especialmente perjudicial para un SVM cuando nos encontremos con instancias que afecten de forma negativa a la generación de los vectores de soporte.

El filtro utilizado para lograr el filtrado es *Interquartile Range*, con el que podremos detectar instancias *Outliers* o con valores extremos.

Una vez realizado el filtro, se retiran las instancias resultantes del conjunto de datos.

Además y de forma experimental utilizo la implementación del *OneClass Classifier* como detector de outliers. Más adelante se profundizará en el uso y los resultados.

5.1.4. Balanceo de instancias

Se ha considerado la aplicación de un filtro de balanceo de instancias, con el que cabe la posibilidad de que si hay una clase dominante entre las instancias de entrenamiento, el clasificador cree una tendencia excesiva a clasificar instancias bajo dicha clase(evita overfit).

El filtro que he decidido utilizar es *Resample*.

En el caso de SVM la aplicación de este filtro no sería necesaria ya que emplea los vectores de soporte para decidir las clases, independientemente de cual sea la clase predominante en el conjunto de datos.

Aún así y dada la magnitud del problema abordado con respecto al tiempo, no se presentan resultados con un conjunto de datos no balanceado.

5.1.5. Optimización de parámetros escogida

El kernel RBF es, en general, una primera elección razonable. Este kernel puede mapear muestras en un espacio de multidimensional que, a diferencia del kernel lineal, es operativo incluso cuando la relación entre las clases y los atributos no es lineal. Además, el kernel lineal es un caso especial del RBF (Keerthi and Lin 2003), incluso el kernel sigmoidal se comporta como un RBF para ciertos parámetros (Lin and Liu 2003).

La segunda razón para la elección de RBF, es el número de hiperparámetros que influyen en la complejidad del modelo seleccionado. El kernel polinomial, por ejemplo, tiene más parámetros que el RBF.

Finalmente, RBF presenta menos dificultades numéricas. Un punto clave es que los valores de este kernel son $0<k_{ij}>1$, en contraste con los kernel polinomiales que podrían tomar valores infinitos ($\gamma x_i^T x_j + r>1$) o cero ($\gamma x_i^T x_j + r<1$) cuando el grado es grande.

Por otra parte, el kernel sigmoidal no es válido bajo algunos parámetros, por ejemplo, no es válido para el producto escalar de dos vectores (Vapnik, 1995).

También cabe comentar que existen algunas situaciones en las que RBF no es adecuado. En particular, cuando el número de características es demasiado grande –el caso que nos ocupa podría cumplir esta propiedad–, se puede usar el kernel polinomial y de hecho se utiliza y se presentarán resultados y análisis.[3, sección 3.1,4]

El kernel RBF hace uso de dos parámetros: C y γ (gamma). No son conocidos de antemano los valores para los parámetros que dará como resultado el mejor clasificador para un problema dado. Consecuentemente, algún tipo de selector del modelo (buscadores de parámetros) debe hacerlo. El objetivo es buscar los mejores valores para C y γ de manera que podamos obtener el clasificador óptimo.

Para terminar, se utilizará 10-fold cross validation, utilizando tanto API de Weka, haciendo uso de un método que he implementado que será explicado más adelante. La razón de su uso es que puede prevenir, o mitigar, el problema del *overfitting*.[3, sección 3.2,5]

5.1.6. Barrido de parámetros escogido

La manera de acotar estos dos parámetros la he realizado mediante análisis gráfico. Aunque se han realizado experimentos con otros conjuntos de datos, para seleccionar el barrido he utilizado el conjunto de datos de entrenamiento para entrenar el modelo y el conjunto dev para ajustar los parámetros. Aquí tengo que mencionar que algunos de los experimentos se han realizado con muestras representativas de ambos conjuntos.Esto es justificable por la carencia del tiempo y capacidad de computo necesaria par abordar el problema en toda su magnitud y con todas y cada una de las funcionalidades implementadas. .

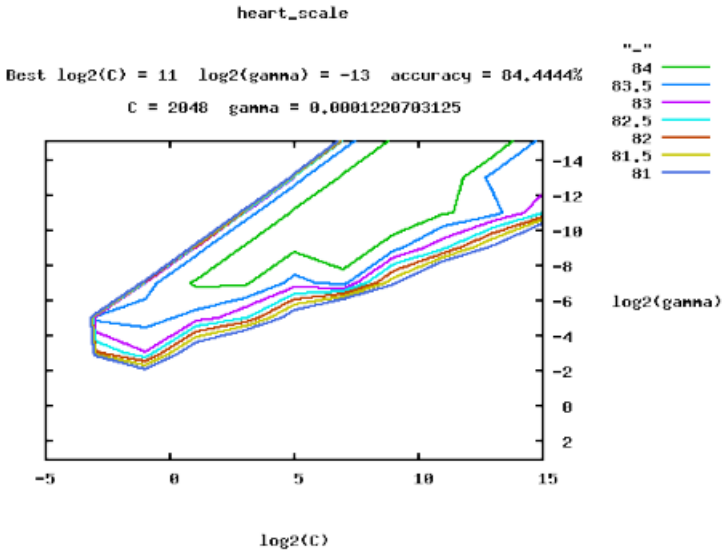

Figura 1: Ejemplo de selección de parámetros[1].

Brevemente, se explica por qué los valores de los parámetros de recorren en un rango de potencias de diez. Esta basado en la Figura [1], en lo que parece que la estructura del rango de la maya es logarítmica para el modelo de los parámetros.[3]

Se ha visualizado la evaluación y barrido determinando, que al menos para este problema suele ser suficiente recorrer el rango de 10^{-3} a 10^{3} tanto para gamma como para C, como puede observarse en la figura 2.

[1]Imagen obtenida de: [1, Section 9].

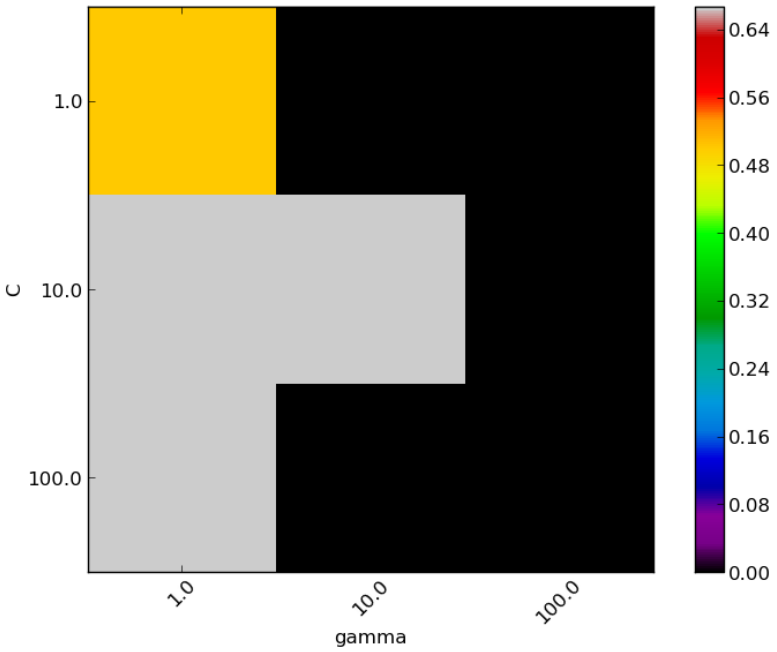

Figura 2: Scoring Matrix

He podido observar que aún no siendo proporcional el aumento de la f-measure de la clase positiva y no siendo evitable no conseguir un mayor desempeño a pesar de aumentar los parámetros, el mayor desempeño se encuentra en el rango antes mencionado.

Por lo tanto, basándonos en la figura anterior y en nuestras pruebas, C establezco dicho rango para ambos parámetros. En la sección de análisis, se podrá observar con mayor detalle este análisis

Una de los mayores beneficios que aporta el uso de la exploración logarítmica es el menor coste computacional frente a la exploración lineal. A pesar de este menor coste los resultados obtenidos son lo suficientemente buenos como para que sea computacionalmente conveniente el uso de este métodos de exploración tan extendido en los DSS.

Dicho todo esto se ha de recalcar que el mejor desempeño, tanto computacional como la capacidad predictiva lo he encontrado con el barrido Grid search de la librería *scikit-learn*, sobre el que se realiza un análisis gráfico.

5.2. Diseño

5.3. Algoritmo principal

El algoritmo principar consiste en optimizar los parámetros para el conjunto de instancias a clasificar. Para ello, se hace un barrido **ad-hoc** de los parametros a optimizar.Este barrido lo implemento en ambos lenguajes de programación, obteniendo un mejor resultado inicial de f-mesure con la API de weka, no siendo así tras algunos ajustes en las estructuras de datos.

La evaluación (*Hold-out*)para obtener el desempeño se realiza haciendo las predicciones con el conjunto development, obteniendose la f-measure de la clase positiva para reajustar los parámetros al mejorar esta medida.

5.3.1. Evaluación

Para evaluar se realizan tres test diferentes: No-honesto, Hold-out y 10-FCV. Para weka he tenido el tiempo y la capacidad de implementar un módulo de evaluación con los tres métodos. Como resultado se obtiene el modelo re-entrenado el modelo con las instancias de los conjuntos de entrenamiento y el usado para el ajuste de parámetros. De este método se obtiene el fichero *.eval* con el resultado de la evaluaciones y el fichero *.model* que contiene el modelo serializado para poder utilizarlo en el módulo *Ensemble*.

En python se realizan las mismas evaluaciones. No se obtiene el fichero con las evaluaciones, por el contrario se obtiene una matriz con la evolución de la medida de desempeño utilizada con respecto a los parámetros establecidos.

5.3.2. Problemas encontrados y soluciones adoptadas

La mayor barrera la he encontrado con la implementación en python, que dado a la inexperiencia me ha llevado bastantes horas formarme. Además los datos cargados a partir de un *.arff* tiene que ser transformados a un formato adecuado para poder entrenar el modelo, para ello he recurrido a las librerías *numpy* y *scipy* de python que proporcionan las herramientas necesarias para dicha tarea.

Otro problema que me ha consumido mucho tiempo de formación y experimentación es la extracción de los atributos categóricos en un formato con el que los modelos puedan ser entrenados, ya que no aportan la capacidad de trabajar con este tipo de características.

La solución adoptada para esta fase del preproceso de datos, pasa por formatear el contenedor de los atributos en un *dic* que servirá como parámetro de entrada a la clase DictVectorizer.Esta clase implementa el método *one-hot* para codificar atributos categóricos, es decir aquellos que están restringidos a una lista discreta de valores.También es usado como preproceso en modelos de procesamiento de lenguaje natural que normalmente incluyen "ventanas" alrededor de una palabra de interés.[5]

5.4. Métodos de evaluación

5.4.1. Método NO-honesto

La evaluación mediante el método no honesto, como su propio nombre nos sugiere, nos da una medida optimista del buen desempeño del clasificador.

Para realizar esta medida, se emplea para el test el mismo conjunto de datos que para el entrenamiento. Como es evidente, el rendimiento real del clasificador frente a datos distintos a los de entrenamiento es notablemente inferior al indicado por la evaluación no honesta, llegando ésta incluso a alcanzar un 100 % de accuracy con algunos de los conjuntos de entrenamiento.

Esta evaluación está accesible dentro del módulo de evaluación en el proyecto java, permitiéndonos obtener, de esta forma, la cota superior de calidad. La cual será incluida en el fichero *.eval*. Además es mostrada por consola en el proyecto python.

5.4.2. Método Hold-out

Esta evaluación está accesible dentro del módulo de evaluación, permitiéndonos obtener, de esta forma, una cota más realista de calidad. La cual será incluida en el fichero *.eval*.

Además de ser la evaluación utilizada para ajustar los parámetros en los barridos en ambos proyectos. Para ello se entrena el modelo con el conjunto de entrenamiento y se realiza el test con el conjunto *develop*.Una vez hecho esto se obtiene la f-measure de la clase positiva para decidir si reajustar o no el modelo.

En el proyecto python se obtiene en un contenedor la f-measure obtenida para cada vuelta, esto nos permite una visualización de la evolución del barrido en un gráfico como se muestra en la figura 2.

5.4.3. Método 10FCV

Esta evaluación está accesible dentro del módulo de evaluación en el proyecto java, permitiéndonos obtener, de esta forma, la cota más realista de calidad. La cual será incluida en el fichero *.eval*.

No es utilizada en los barridos de parámetros para el ajuste de estos dado su coste computacional. Para realizar esta evaluación se entrena de nuevo el modelo, pero con los dos conjuntos de instancias utilizadas en el método anterior (train+dev).Para el test se utiliza el conjunto de test del cual no conocemos la clase.

En el proyecto python el resultado se muestra por consola.

6. Resultados: Comparativa entre los distintos proyectos

Esta sección está dedicada al análisis de los resultados obtenidos, además del seguimiento gráfico que me ha ayudado a acotar los métodos de optimización, ahorrando en coste computacional y en tiempo. Siendo así y gracias a que se amplió el plazo de entrega, he podido profundizar más en las librerías de python mencionadas, obteniendo muy buenos resultados con un lenguaje de programación muy eficiente.

Dado que mi trabajo ha sido en mayor medida la optimización de los modelos y que para ello he usado dos librerías en dos lenguajes de programación diferentes, he centrada mi análisis en la evolución de los modelos y en la comparación de los resultados obtenidos con Weka y con Scikit-learn.

Al comenzar a implementar utilizando el API Scikit-learn no conseguía mejores resultados que con el API de weka, sin embargo al profundizar y definir estructuras de datos y extracción de atributos categóricos he conseguido mucho mejores resultados, además de disminuir el tiempo empleado en una tercera parte si no más, este dato no lo tengo recogido de manera explicita ya que he parado la mayoría de barridos de parámetros iniciados con Weka tras dos o tres días.

6.1. SVC RBF

6.1.1. Weka

La API de weka fue la primera con la que empecé a implementar. Inicialmente utilizo un SVM kernel rbf y sin ningún tipo de pre-procesado en los datos se obtiene el modelo óptimo, por lo que no era posible obtener un buen desempeño para la F1-score de la clase positiva como podemos ver en los resultados de aplicar el modulo Evaluación en un modelo entrenado con los datos mencionados:

```
Evaluaciones para el modelo LibSVM RBF 2014-11-23-13:41

C: 3.0517578125E-5
gamma: 0.125

Evaluación no-honesta
=======================================

Correctly Classified Instances       22861               99.6513 %
Incorrectly Classified Instances        80                0.3487 %
Mean absolute error                      0.0035
Root mean squared error                  0.0591
Relative absolute error                 49.8666 %
Root relative squared error            100.1748 %
Recall:  0.9965127936881566
Precision:  0.9930377479841745

=== Confusion Matrix ===

     a     b   <-- classified as
     0    80 |     a = V1
     0 22861 |     b = V2

=== Detailed Accuracy By Class ===

TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
0,000    0,000    0,000      0,000    0,000      0,000    0,500     0,003     V1
1,000    1,000    0,997      1,000    0,998      0,000    0,500     0,997     V2
0,997    0,997    0,993      0,997    0,995      0,000    0,500     0,993


Evaluación hold-out con dev
=======================================
```

```
Correctly Classified Instances       40192               99.6677 %
Incorrectly Classified Instances       134                0.3323 %
Mean absolute error                      0.0033
Root mean squared error                  0.0576
Relative absolute error                 48.6522 %
Root relative squared error            100.1659 %
Total Number of Instances            40326
Recall:  0.9966770817834648
Precision:  0.9933652053524034

=== Confusion Matrix ===

     a     b   <-- classified as
     0   134 |     a = V1
     0 40192 |     b = V2

=== Detailed Accuracy By Class ===

                TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                0,000    0,000    0,000      0,000    0,000      0,000    0,500     0,003     V1
                1,000    1,000    0,997      1,000    0,998      0,000    0,500     0,997     V2
Weighted Avg.   0,997    0,997    0,993      0,997    0,995      0,000    0,500     0,993
```

Como hemos mencionado con anterioridad, las medidas de desempeño *recall* y *precision*, en conjuntos de datos en los que la clase positiva está desbalanceada no son representativas. Podemos ver como en esta evaluación, en el método hold-out obtiene 0,9966 y 0,9933 respectivamente, mientras que la F-measure de la clase positiva es 0, igual que en la cota más optimista también lo es.

Una vez preprocesados los datos con Outliers, Resample y la primera versión para selección de atributos el modelo mejora mucho su desempeño del tras realizar una nueva búsqueda de parámetros con el barrido ad-hoc:

```
Evaluaciones para el modeloLibSVM SVC RBF 2014-12-01-15:18

Train set:
"data/Resample_2014112619/Outlier_2014112619/
train.e2e.wOOV.obfuscated.arffattSel_20141126-1919_OutlierClass_20141126-1920_ResampleClass.arff"
Develop set:
"data/dev.e2e.wOOV.obfuscated.arffattSel.arff"

Evaluación no-honesta
=======================================

Correctly Classified Instances       16556               72.1677 %
Incorrectly Classified Instances      6385               27.8323 %
Mean absolute error                      0.2783
Root mean squared error                  0.5276
Relative absolute error                 55.6665 %
Root relative squared error            105.5145 %
Total Number of Instances            22941
Recall:  0.7216773462359967
Precision:  0.7982294906581087

=== Confusion Matrix ===

     a     b   <-- classified as
  5448  6091 |     a = V1
   294 11108 |     b = V2
Parámetros óptimos
=======================================
|| C: 1.0054299011128027 ||
```

```
|| gamma: 2.0 ||
|| degree: 0 ||

=======================================
=== Detailed Accuracy By Class ===

TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
0,472    0,026    0,949      0,472    0,631      0,515    0,723     0,713     V1
0,974    0,528    0,646      0,974    0,777      0,515    0,723     0,642     V2
0,722    0,275    0,798      0,722    0,703      0,515    0,723     0,678


Evaluación hold-out con dev
=======================================

Correctly Classified Instances       33567               83.2391 %
Incorrectly Classified Instances      6759               16.7609 %
Mean absolute error                      0.1676
Root mean squared error                  0.4094
Relative absolute error                 33.4369 %
Root relative squared error             81.6717 %
Total Number of Instances            40326
Recall:  0.8323910132420771
Precision:  0.8426365824814233

=== Confusion Matrix ===

     a     b   <-- classified as
  5467  6126 |     a = V1
   633 28100 |     b = V2

=======================================
=== Detailed Accuracy By Class ===

TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
0,472    0,022    0,896      0,472    0,618      0,568    0,725     0,575     V1
0,978    0,528    0,821      0,978    0,893      0,568    0,725     0,819     V2
W0,832    0,383    0,843      0,832    0,814      0,568    0,725     0,748


Evaluación 10FCV con dev+train
=======================================

Correctly Classified Instances       67134               83.2391 %
Incorrectly Classified Instances     13518               16.7609 %
Mean absolute error                      0.1676
Root mean squared error                  0.4094
Relative absolute error                 36.7989 %
Root relative squared error             85.7288 %
Total Number of Instances            80652
Recall:  0.8323910132420771
Precision:  0.8426365824814233

=== Confusion Matrix ===

     a     b   <-- classified as
 10934 12252 |     a = V1
  1266 56200 |     b = V2

=======================================
=== Detailed Accuracy By Class ===
```

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0,472	0,022	0,896	0,472	0,618	0,568	0,725	0,575	V1
0,978	0,528	0,821	0,978	0,893	0,568	0,725	0,819	V2
0,832	0,383	0,843	0,832	0,814	0,568	0,725	0,748	

En la cota más realista (10FCV) se consiguen muy buenos resultados, sin apenas empeorar un 10 % las medidas menos representativas, el modelo consigue una F-measure para la clase positiva de 0.618, consiguiendo situar en buen lugar en el espacio ROC al clasificador con 0.725 de ROC AREA.

Para este modelo, con weka no he conseguido mejores resultados, si cabe mencionar que falta el experimento con la nueva versión del filtro de selección de atributos, que combinando los resultados de dos clasificadores, consigue disminuir la perdida de información ya que en lugar de entrenar el modelo con un único atributo conseguimos seleccionar hasta 8, que empíricamente queda demostrada la ganancia de información como podremos ver más adelante en el análisis.

6.1.2. Scikit-learn

Este kernel ha sido el que mejores resultados me ha ofrecido con un menor coste computacional ya que con un parámetro menos a barrer se obtienen los mismos o mejores resultados que con el kernel polinomial. La evolución del kernel a medida que aumentamos gamma y C la podemos ver en la figura3

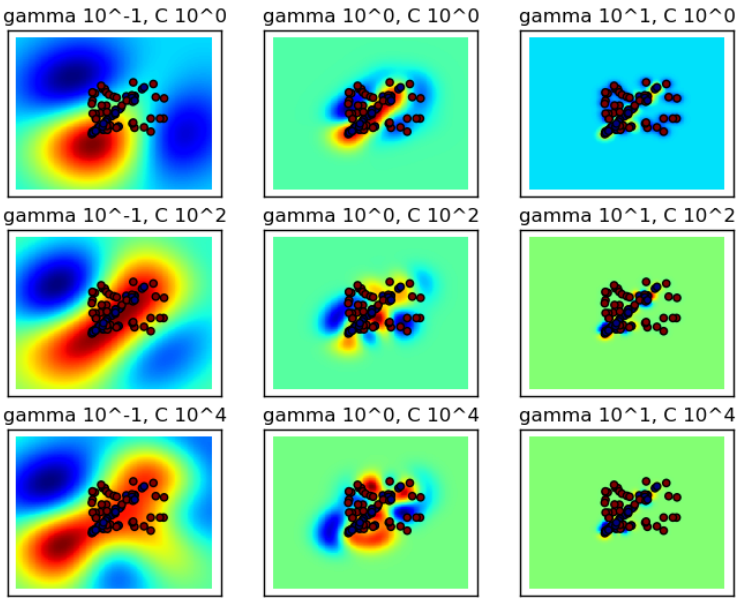

Figura 3: Kernel evolution

Como podemos ver a medida que aumentamos los parámetros el kernel disminuye y se ajusta mejor a los datos, también podemos observar como al aumentar demasiado los parámetros el kernel disminuye tanto que deja una frontera de decisión muy limitada para poder generalizar nuevas predicciones, por lo que el scoring de nuestra evaluación no mejora como se puede observar en la evolución de la f-measure presentada en la siguiente figura

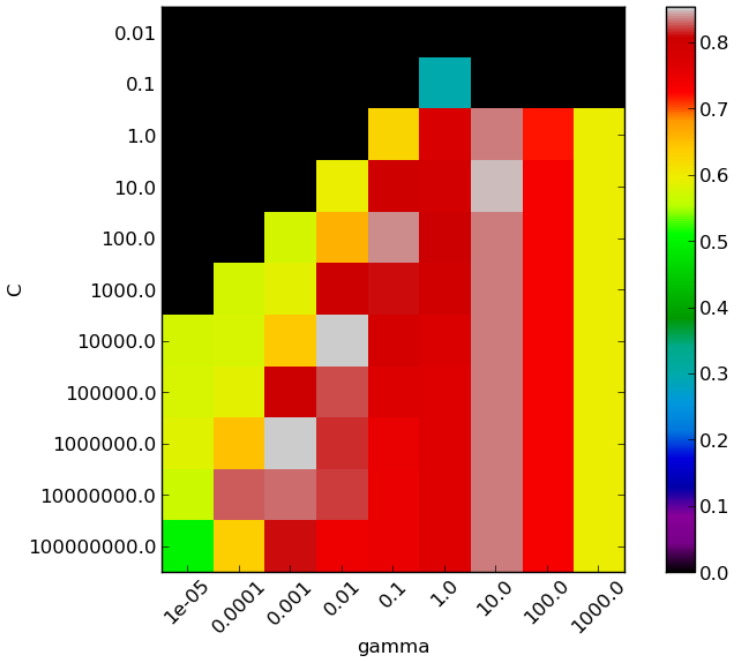

Figura 4: Scan:F1-score positive class evolution

Este método ha conseguido obtener una F-measure de la clase positiva mayor de 0.800 en el método hold-hout, aún así y como todos los métodos presenta un descenso de una o dos décimas al realizar la evaluación 10FCV, obteniendo una F-measure de la clase positiva 0.780. Observando estos resultados podemos acotar fácilmente el barrido de una manera práctica, sin necesidad de profundizar en conceptos teóricos más avanzados. Se puede observar como el *scoring* más alto se encuentra acotado en el rango $(10^{-3}$,$10^{2})$ para el parámetro gamma y $(10^{0}$, $10^{3})$ para el parámetro C.

6.2. SVC Polinomial

Este kernel no lo he analizado en profundidad dado que experimentalmente he encontrado que es más costoso ajustar el modelo y no se obtiene mejor desempeño, como veremos a continuación y en el análisis gráfico con la librería python.

6.2.1. Weka

```
Evaluaciones para el modeloLibSVM SVC POLY 2257

Train dataset:
data/Resample_2014112619/Outlier_2014112619/
train.e2e.wOOV.obfuscated.arffattSel_20141126-1919_OutlierClass_20141126-1920_ResampleClass.arff
Develop dataset
data/dev.e2e.wOOV.obfuscated.arffattSel.arff

Evaluación no-honesta
=======================================

Correctly Classified Instances       16376               71.3831 %
Incorrectly Classified Instances      6565               28.6169 %
Mean absolute error                      0.2862
Root mean squared error                  0.5349
Relative absolute error                 57.2358 %
Root relative squared error            106.9914 %
Total Number of Instances            22941
Recall:  0.713831132034349
```

```
Precision:  0.7818623261146133

=== Confusion Matrix ===

     a     b   <-- classified as
  5448  6091 |     a = V1
   474 10928 |     b = V2
Parámetros óptimos=======================================

|| C: 4.0 ||
|| gamma: 1.0006771306930664 ||
|| degree: 2 ||
=======================================
=== Detailed Accuracy By Class ===

TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
0,472    0,042    0,920      0,472    0,624      0,492    0,715     0,700     V1
0,958    0,528    0,642      0,958    0,769      0,492    0,715     0,636     V2
0,714    0,283    0,782      0,714    0,696      0,492    0,715     0,668


Evaluación hold-out con dev
=======================================

Correctly Classified Instances       33092               82.0612 %
Incorrectly Classified Instances      7234               17.9388 %
Mean absolute error                      0.1794
Root mean squared error                  0.4235
Relative absolute error                 35.7868 %
Root relative squared error             84.4928 %
Total Number of Instances            40326
Recall:  0.8206120121013738
Precision:  0.8222282624038749

=== Confusion Matrix ===

     a     b   <-- classified as
  5467  6126 |     a = V1
  1108 27625 |     b = V2
=======================================
=== Detailed Accuracy By Class ===

TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
0,472    0,039    0,831      0,472    0,602      0,531    0,717     0,544     V1
0,961    0,528    0,818      0,961    0,884      0,531    0,717     0,814     V2
0,821    0,388    0,822      0,821    0,803      0,531    0,717     0,737


Evaluación 10FCV con dev+train
=======================================

Correctly Classified Instances       65746               81.5181 %
Incorrectly Classified Instances     14906               18.4819 %
Mean absolute error                      0.1848
Root mean squared error                  0.4299
Relative absolute error                 40.5773 %
Root relative squared error             90.0225 %
Total Number of Instances            80652
Recall:  0.8151812726280812
Precision:  0.829363041121532
```

```
=== Confusion Matrix ===

     a     b   <-- classified as
  9398 13788 |     a = V1
  1118 56348 |     b = V2
========================================
=== Detailed Accuracy By Class ===

TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
0,405    0,019    0,894      0,405    0,558      0,519    0,693     0,533     V1
0,981    0,595    0,803      0,981    0,883      0,519    0,693     0,802     V2
0,815    0,429    0,829      0,815    0,790      0,519    0,693     0,724
```

Este modelo obtiene un desempeño muy similar al kernel rbf, se puede ver que es ligeramente peor analizando la medida de desempeño de referencia en el experimento con la cota más realista. Los parámetros aunque no implican un mayor coste computacional para un modelo ya entrenado no es así para optimizarlo, esto es mientras optimizar el kernel rbf supone un coste computacional $O(n * m)$ ya que sólo debemos ajustar dos parámetros, un kernel polinomial supone un aumento en coste de k veces ya que tenemos un tercer parámetro que computar $O(n * m * k)$.
Tras este análisis decido no continuar experimentando con en el kernel polinomial, aunque lo retomo de nuevo con el API Scikit-learn con la finalidad de justificar mis conclusiones de una manera más gráfica.

6.2.2. Scikit-learn

Como he mencionado en secciones anteriores me valgo de esta librería, y de la librería matplotlib,para no continuar modelando el problema con este kernel. La primera barrera que me encuentro es con el objetivo inicial de la decisión de implementar en python este kernel: la visualización. Dado que ahora se suma un nuevo componente dinámico que es el grado de la función, necesito implementar la visualización teniendo en cuenta este nuevo parámetro tanto para el propio gráfico, como para el número de éstos. La solución pasa por el producto entre las filas dispuestas y el número de valores que toma el parámetro.De esta forma aparecen de forma dinámica el número de gráficos como se puede ver en la figura.5

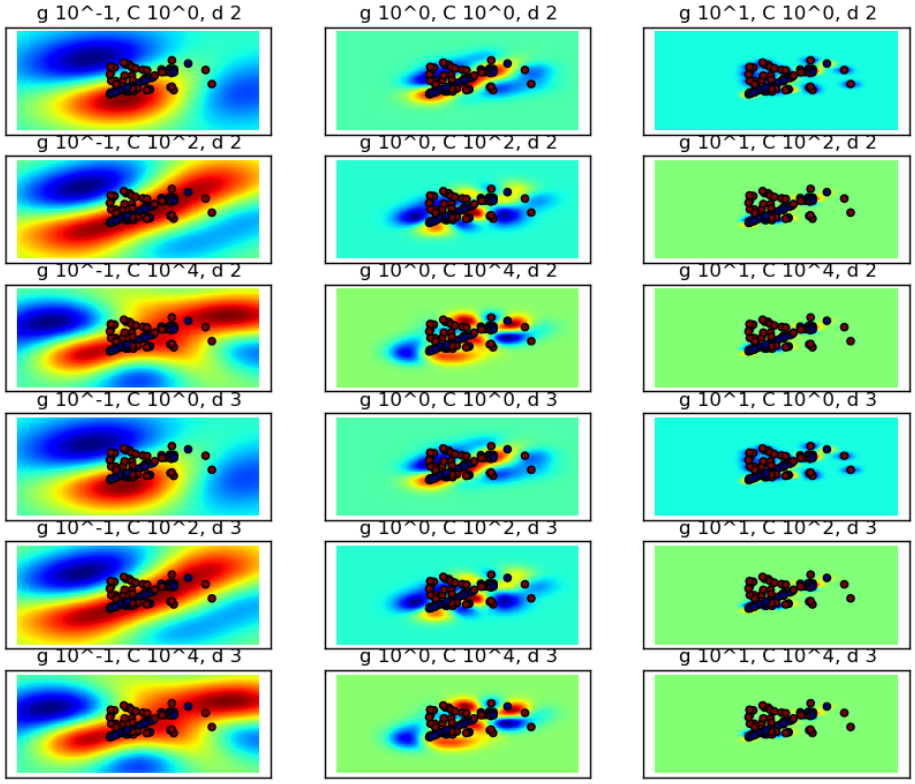

Figura 5: Polinomial Kernel evolution

En la figura podemos comparar como ambos grados ajustan de manera muy similar el kernel para los mismos valores del kernel. A continuación se muestra el resultado de la matriz de *scoring* para cada valor del parámetro *degree* en las figuras 6 y 7 que reflejan de manera más clara que los resultados de la figura de merito de referencia son muy similares, si no los mismos, para todos los valores del parámetro *degree*.

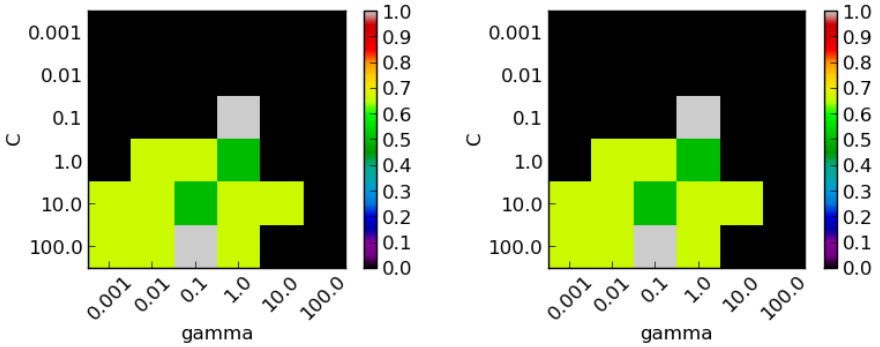

Figura 6: Polinomial Kernel F1-score evolution

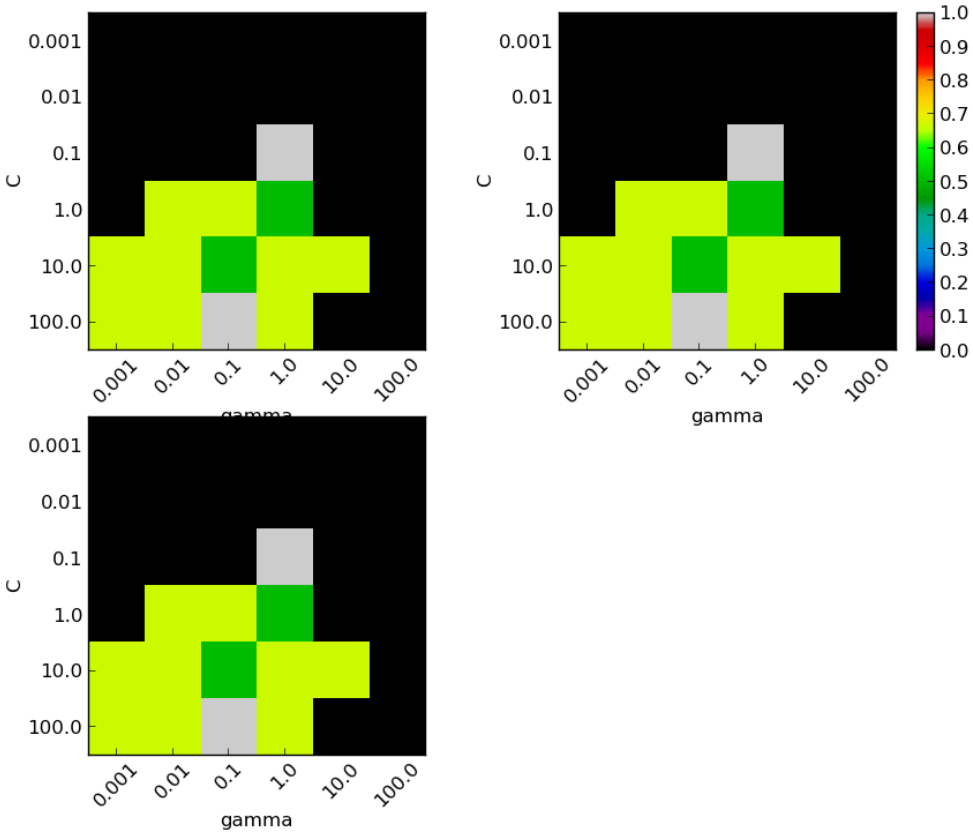

Figura 7: Polinomial Kernel F1-score evolution

6.3. SVC OneClass Outliers Novelty Detection

Ya que estaba inmerso, y continuaré, en las SVM, he querido implementar un OneClass Classifier como unas de mis mejoras con respecto a mi propuesta inicial.

Continuando con la librería Scikit-learn, aprovechando toda la información que tiene respecto a las SVM y su implementación, profundizo en el uso de este clasificador como detector de outliers.[6] Para obtener un modelo con estas características, inicialmente tenemos que conseguir un conjunto de entrenamiento con una "contaminación" ¡0.1. En esta tarea no profundizo en como conseguir este conjunto si no en como hacer uso de esta funcionalidad. Una vez tenemos el modelo entrenado con la información, éste es capaz de predecir, en el caso de el problema que nos ocupa, el valor de la clase con las etiquetas 0 y 1, si la instancia a predecir es un outlier, el modelo devolvera como predicción -1. La siguiente figura8 de la implementa ilustra el funcionamiento de la implementación.

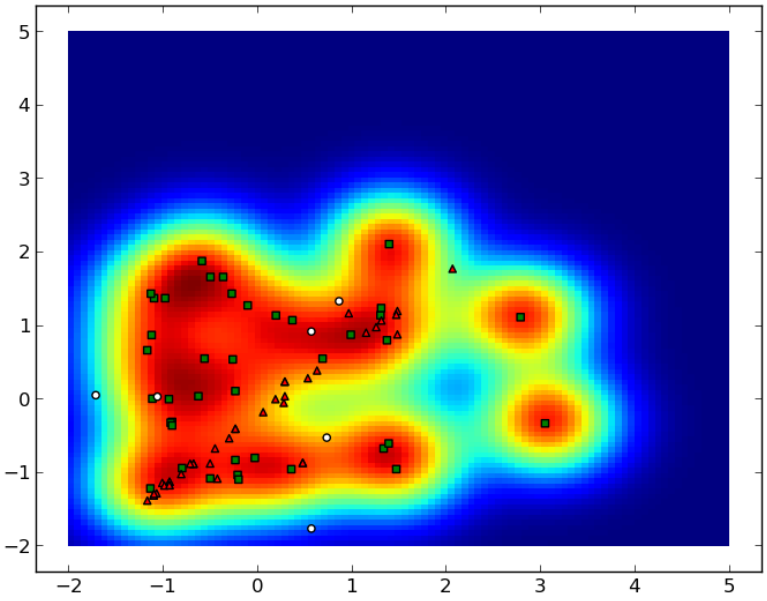

Figura 8: OneClass Classifier: outliers detection

En la figura se diferencian dentro del kernel, los cuadros verdes y los triángulos rojos(clase positiva). Las instancias a predecir las he representado con circulos blancos para indicar claramente que no tienen etiqueta. El modelo es capaz de predecir de que clase son las instancias que han caído dentro del kernel y las que quedan fuera son clasificadas como outliers. Este método es susceptible de uso en las etapas del preproceso como detector de outliers, aunque hay que tener en cuenta que la necesidad de disponer previamente de un conjunto de entrenamiento.

7. Conclusiones

La conclusión principal a tener en cuenta es la importancia del papel que desempeña en un proyecto de *datamining* el pre-proceso de los datos, siendo consciente de esta necesidad, tanto para obtener ganancia de información; como por ejemplo transformando los datos categóricos a un formato capaz de aportar información valiosa al modelo, como para facilitar el cómputo; seleccionando aquellas características que verdaderamente son informativas y no solo datos.

Con el análisis realizado en esta práctica las conclusiones podrían quedar reducidas a una función:

$$f(rubish) = rubish \tag{1}$$

Es decir, podemos tener el mejor modelo de clasificación implementado que sin la información adecuada, con sólo datos no nos servirá de mucho.

8. Valoración subjetiva

1. ¿Has alcanzado los objetivos que se plantean? ¿Te ha resultado de utilidad la tarea planteada?

   Entendiendo que los objetivos son los marcados en el documento inicial, considero que he superado mis propias expectativas con muy buen resultado, siendo consciente de la posibilidad de mejorar ciertos aspectos. Ha resultado de gran utilidad, gracias a la tarea se ha entendido el funcionamiento de una manera más profunda los modelos basados máquinas de soporte vectorial y la selección de parámetros. Además se han adquirido nuevas competencias, como la capacidad de ponerse objetivos inicialmente no marcados y de profundizar en el estudio y manejo de nuevas librerías de *datamining*, y encaminado a adquirir otras, como por ejemplo el análisis de modelos predictivos, representación gráfica con la librería *matplotlib*...etcétera.

2. ¿Qué dificultades has encontrado? Las dificultades que han ido surgiendo se han ido incluyendo en el informe. La mayor dificultad se me ha presentado en el uso de un nuevo lenguaje de programación, que aunque estoy contento con el resultado, se que lo tengo que trabajar para mejorar en un futuro cercano. De hecho cuento con profundizar más en este lenguaje por que me ha resultado mucho más eficiente y completo que java, sobre todo teniendo en cuenta las librerías de las que dispone.

3. ¿Cuánto tiempo has trabajado en esta tarea? Desglosado: Aunque no he recogido el tiempo de manera periódica, trataré de estimar el tiempo real invertido.

Coste temporal	
Diseño de software	2
Implementación de software	50
Tiempo trabajando con Weka	2
Búsqueda bibliográfica	2
Informe	3

4. Sugerencias para mejorar la tarea. Sugerencias para que se consiga despertar mayor interés y motivación en los alumnos.

   La tarea está bien propuesta para despertar el interés. Si bien es cierto que no está diseñada para todo tipo de perfiles, quiero decir que esta tarea requiere de una habilidad para ser capaz de identificar unos objetivos realistas, organizar la tarea como un proyecto y ser consciente del esfuerzo que supondrá llevar a termino la tarea proyectada. Opino que es una metodología que se debería trabajar más en la carrera para adquirir estas competencias tan necesarias en nuestra profesión fuera del ámbito académico.

5. Críticas(constructivas). Como siempre, ha sido muy positivo trabajar en equipo. A grandes rasgos estoy satisfecho con la propuesta de trabajo, me ha gustado tener la posibilidad de decidir que hacer sin limites ni siquiera en las herramientas. Si algo se puede criticar es el tiempo, por otro lado creo que también ayuda a aprender a poner limites en función a esta variable, siempre tan presente en cualquier contexto .Resumiendo, en lo que a mi respecta,la practica ha sido enriquecedora.

Referencias

[1] *LIBSVM: A Library for Support Vector Machines.* Department of Computer Science, National Taiwan University, Taipei, Taiwan, 2001.

[2] *A Practical Guide to Support Vector Classification.* Department of Computer Science National Taiwan University, Taipei 106, Taiwan, 2010.

[3] *A Practical Guide to Support Vector Classification.* Department of Computer Science, National Taiwan University, Taipei, Taiwan, 2010.

[4] http://scikit learn.org/stable/index.html. Support vector machines, 2014.

[5] http://scikit learn.org/stable/modules/feature$_e$*xtraction.html*. *Supportvectormachines*, 2014.

[6] http://scikit learn.org/stable/modules/outlier$_d$*etection.html*. *Supportvectormachines*, 2014.