Algoritmos de clasificación para detección de spam

José Roberto Salazar Espinoza

Profesor: Dr. Ramón Soto de la Cruz

UNISON
9 de diciembre de 2017

Contents

1 Introducción

En este reporte se abordarán los diferentes algoritmos de clasificación vistos en la clase de reconocimiento de patrones para resolver el problema de detección de spam de correos electrónicos. Primero se explicará la forma en que se limpiaron los textos, después veremos como convertir los textos a vectores de características numéricos, y por último se explicarán los distintos algoritmos de clasificación y como se aplican al problema de detección de spam. El código se puede encontrar en el siguiente repositorio de github https://github.com/robertosalazare/proyecto-patrones

1.1 Dataset utilizado

El dataset utilizado en este documento es el denominado CSDMC2010 el cual se puede conseguir en el siguiente link http://csmining.org/index.php/spam-email-datasets-.html

2 Limpieza de datos y vectores de características

En esta sección se explicará como se limpiaron los textos del dataset y despues, ya teniendo el texto en bruto se obtienen los vectores de características.

2.1 Limpieza de datos

En el conjunto de datos se pueden observar cuatro componentes principales, la carpeta TRAINING que contiene 4327 archivos de correo en formato .eml con si respectiva clase(0 para spam y 1 para no spam), después tenemos la carpeta TESTING que contiene 4292 archivos de correo también en formato .eml, después está el archivo SPAMTrain.label que contiene las etiquetas de los archivos en la carpeta TRAINING y por último tenemos el archivo ExtractContent.py el cual es un script de python que limpia un poco los archivos .eml originales y nos deja unicamente los titulos y cuerpos de los correos. Nosotros utilizaremos unicamente los datos de TRAINING, tomando 4000 datos para entrenar a nuestros diferentes clasificadores y los 327 restantes para hacer el test del clasificador.

El primer paso de la limpieza de datos sería correr el script ExtractContent.py sobre la carpeta TRAINING para generar la carpeta TRAINING_NEW con unos archivos nuevos .eml con unicamente el cuerpo y titulo de los correos en los cuales ya se puede empezar a trabajar. Para realizar todos los algoritmos posteriores se utilizará el lenguaje de programación python.

2.1.1 Texto a minusculas

La primer tarea que se realizará será convertir el texto a minusculas, ya que si no se realiza esto hay palabras que a pesar de ser la misma serían consideradas palabras diferentes, como por ejemplo palabras al principio de un párrafo

o después de un punto y seguido ya que estas empiezan con mayuscula. A continuación se muestra un ejemplo de como se convertiría un texto a minúsculas.

input: [SPAM] Give her 3 hour rodeoEnhance your desire, pleasure and performance! 100% GUARANTEED TO SEE AN INCREASE IN SIZE AND WIDTH http://pg.exqumloaf.com/

```
#se abre el archivo TRAIN_00003.eml
f=open("datos/TRAINING_NEW/TRAIN_00003.eml","r",encoding='charmap')
#se obtiene el texto
texto = f.read()
#se eliminan espacios dobles y saltos de linea inecesarios
texto = ' '.join(texto.split())
print(texto)
#se transforma el texto a minusculas
texto = texto.lower()
print(texto)
```

output: [spam] give her 3 hour rodeoenhance your desire, pleasure and performance! 100% guaranteed to see an increase in size and width http://pg.exqumloaf.com/

2.1.2 Limpieza con expresiones regulares

Ahora veremos como eliminar algunos elementos especiales con expresiones regulares. Primero que nada, después de revisar varios correos a vista me di cuenta de que había algunos correos que empezaban con "content-type: text/plain; charset=us-ascii content-disposition: inline content-transfer-encoding: quoted-pri" estos elementos no aportan nada para el análisis del texto por lo que vamos a eliminar, después se eliminarán los siguientes elementos:

- Los URL.
- Las direcciones de correo electrónico.

A continuación se muestra el código que realiza esto:

input: [SPAM] Give her 3 hour rodeoEnhance your desire, pleasure and performance! 100% GUARANTEED TO SEE AN INCREASE IN SIZE AND WIDTH http://pg.exqumloaf.com/

```
#se importa el modulo de python para expresiones regulares
import re

#se abre el archivo TRAIN_00003.eml
f=open("datos/TRAINING_NEW/TRAIN_00003.eml","r",encoding='charmap')
#se obtiene el texto
texto = f.read()
#se eliminan espacios dobles y saltos de linea inecesarios
texto = ' '.join(texto.split())
print(texto)
#se transforma el texto a minusculas
texto = texto.lower()
#expresion regular para eliminar los url, esta es muy larga por eso
     no la puse
texto = re.sub(espresion_regular,'',texto)
#expresiones regulares para elimiar las direcciones de correo
    electronico
texto = re.sub(r'[\w\.-]+@[\w\.-]+', " ", texto)
texto = re.sub(r'\w+:\/{2}[\d\w-]+(\.[\d\w-]+)*(?:(?:\/[^\s/]*))*',
     ' ',  texto)
#expresiones regulares especificas del dataset.
texto = re.sub(r'content-type: [\w;/\" -=]*\n','',texto)
texto = re.sub(r'content-disposition: [\w;/\" -=]*\n','',texto)
texto = re.sub(r'content-transfer-encoding: [\w;/\" -=]*\n','',
    texto)
print(texto)
```

output: [spam] give her 3 hour rodeoenhance your desire, pleasure and performance! 100% guaranteed to see an increase in size and width

2.1.3 Eliminación de etiquetas de marcaje

Hay ciertos correos que vienen en formato de código html, las etiquetas de dicho codigo no aportan nada de información útil para analizar el texto, por lo cual se deben eliminar. Para hacer esto usaré la librería beautifulsoup que conienene el método get_text que dada una cadena con código html te regresa el texto puro sin etiquetas. A continuación un ejemplo de esto:

```
#se importa el modulo del beautiful soup.
from bs4 import BeautifulSoup

#Texto de ejemplo, en este caso
#no se uso uno del dataset porque
#eran muy largos.
texto = """
<html>
        <body>
                <p>
                Hey, Want a simple way to double your sales?
                it's easy... just copy and paste these three emails
                    :
                </p>
                <a href="http://www.digitalmarketer.com/steal-these
                    -templates">
                http://www.digitalmarketer.com/steal-these-
                    templates
                </a>
                <p>
                This is the same 3-part followup series that we
                    send
                to our prospects...
                </p>
                <b>Enjoy!</b>
                <p>Ryan Deiss</p>
        </body>
</html>
"""
#se eliminan las etiquetas de marcaje.
texto = BeautifulSoup(texto,"html.parser").get_text()
#se eliminan espacios dobles y saltos de linea inecesarios
texto = ' '.join(texto.split())
print(texto)
```

output: Hey, Want a simple way to double your sales? it's easy... just copy and paste these three emails: http://www.digitalmarketer.com/steal-these-templates This is the same 3-part followup series that we send to our prospects... Enjoy! Ryan Deiss

2.1.4 Eliminar signos de puntuación y caracteres especiales

Es importante eliminar los signos de puntuación y los caracteres especiales, ya que no aportan mucho a los datos y pueden provocar ambiguedad en los datos, por ejemplo, la palabra "final" no es la misma que la palabra "final." y es por eso que se quetarán. Ahora se muestra un ejemplo de como se eliminan los caracteres.

input: [SPAM] Give her 3 hour rodeoEnhance your desire, pleasure and performance! 100% GUARANTEED TO SEE AN INCREASE IN SIZE AND WIDTH http://pg.exqumloaf.com/

```
#se lee el texto
f=open("datos/TRAINING_NEW/TRAIN_00003.eml","r",encoding='charmap')
texto = f.read()
```

```
#se eliminan espacios dobles y saltos de linea inecesarios
texto = ' '.join(texto.split())
#se eliminan signos de puntuacion.
eliminar = ['.', ';', ':', '!', '?', '/', '\\', ',', ')', '(', "\""
    ,"-",'*',"|","[","]","{","}","%"]
for e in eliminar:
    texto = texto.replace(e,'')
```

output: SPAM Give her 3 hour rodeoEnhance your desire pleasure and performance 100 GUARANTEED TO SEE AN INCREASE IN SIZE AND WIDTH httppgexqumloafcom

2.1.5 Expandir contracciones

En el lenguaje inglés existen las contracciones, las cuales pueden ser dos o mas palabras juntas en una sola, como por ejemplo i am y i'm, y es importante expandirlas para tratarlas como la cantidad de palabras que realmente son. Para hacer esto utilicé un diccionario de contracciones, el cual se puede ver en el siguiente link: Diccionario.
A continuación se muestra como se expanden las contracciones con el diccionario.

```
#se crea una expresion regular que contiene todas las contracciones
#cList es el diccionario de contracciones
c_re = re.compile('(%s)' % '|'.join(cList.keys()))

#metodo que dado un texto le cambia todas las contracciones que
    tenga
def expandContractions(text, c_re=c_re):
    def replace(match):
        return cList[match.group(0)]
    return c_re.sub(replace, text)

texto = "we're young 'cause we like the parties"
texto = expandContractions(texto)
```

output: we are young because we like the parties

2.2 Generando los vectores de características

En esta sección vamos a suponer que se tiene un módulo leer_datos el cual contiene los métidos readTexts el cual regresa una lista con todos los textos en un rango dado del en el intervalo (0,4327), y el método readClasses el cual regresa una lista con las clases del conjunto de entrenamiento en un intervalo dado. En la siguiente sección hablaremos del modelo bag of words.

2.2.1 Bag of words

El modelo bag of words es un método que se utiliza en el procesado de un texto para represantarlo ignorando el orden de sus palabras. En este modelo, cada documento parece una bolsa que contiene algunas palabras. Por lo tanto este método permite un modelado de las palabras basado en diccionarios, donde cada

bolsa contiene una cuantas palabras del diccionario.

En este modelo cada texto se representa de la siguiente manera:

- Se asigna un indice entero a cada palabra que pueda aparecer en cualquier documento del conjunto de entrenamiento (Hablando mas especificamente se construye un diccionario de palabras a indices enteros).
- Por cada documento i, se cuenta el número de veces que aparece cada palabra w en el diccionario.

Para generar los vectores de características utilizaremos la librería scikitlearn de python, y de esta usaremos un objeto de tipo CountVectorizer el cual nos venera los vectores de características con bag of words. Este utiliza un algoritmo de transformación para vectores dispersos, ya que la muchos de los elementos de un vector serán 0, esto únicamente para consumir menos memoria pero no afecta los resultados. A continuación se muestra el código para generár los vectores de componentes:

```
from sklearn.feature_extraction.text import CountVectorizer
import leer_datos

count_vect = CountVectorizer()
#se leen los datos de entrenamiendo.
training = leer_datos.readTexts(0,4000)
count_vect_train = count_vect.fit_transform(training)
print("Cantidad de palabras diferentes en los textos:", count_vect.
    vocabulary_.get(u'algorithm'))
```

output: Cantidad de palabras diferentes en los textos: 10623

2.2.2 tf-idf

Después de observar algunos correos me di cuenta de algo, algunos son mucho mas largos que otros, por lo que esto podría generar conflictos ya que los correos cortos se iban a separar mas de los largos aunque pertenezcan a la misma clase. Para evitar esto se utilizan dos conceptos principales, el tf (Term frequency) y el idf (Inverse document Frequency).

El tf se refiere a que tan frecuente un termino aparece en un documento. Como cada documento tiene diferente longitud, es mas probable que un termino aparezca más veces en un documento largo que en uno corto. Por lo tanto, el tf es usualmente dividido por la longitud del documento(El número total de terminos en el documento) como una forma de normalización. El tf se puede calcular como:

$$tf(t) = \frac{veces\_que\_aparece\_el\_termino\_t\_en\_el\_documento}{numero\_total\_de\_terminos\_en\_el\_documento}$$

El idf se refiere a que tan importante es un termino. Cuando se calcula el tf, todos los terminos son considerados igualmente importantes. Sin embargo se sabe que ciertos terminos, tales como "is", "of" y "that", van a aparecer muchas veces pero tienen muy poca importancia. Por lo tanto necesitamos bajar el peso de los terminos frecuentes mientras se aumenta el de los terminos raros, el idf se calcula con la siguiente formula:

$$tf(t) = ln(\frac{Numero\_total\_de\_documentos}{Numero\_de\_documentos\_con\_el\_termino\_t\_en\_el})$$

Ahora se puede definir el tf-idf de un termino como:

$$tf - idf = tf(t) * idf(t)$$

Ahora usaremos el sklearn para realizar la transformación tf-idf en todos los vectores del conjunto de entrenamiento, usaremos la clase TfidfTransformer que se encuentra en el módulo sklearn.feature_extraction.text como se muestra a continuación:

```
from sklearn.feature\_extraction.text import TfidfTransformer

tfidf_transformer = TfidfTransformer()
train_tfidf = tfidf_transformer.fit_transform(count_vect_train)
```

Ahora ya tenemos los datos listos para probar varios algoritmos de clasificación. Ese será el tema principal del siguiente capítulo.

3 Algoritmos de clasificación

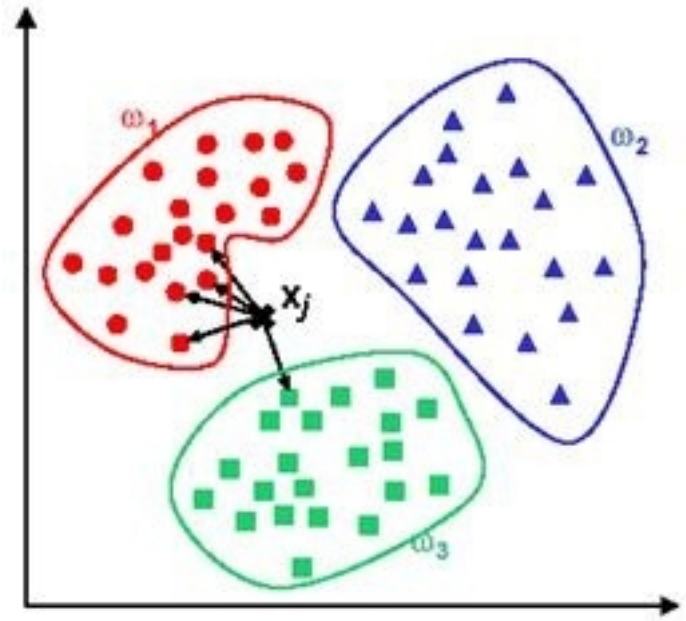
En este capítulo se explicarán varios algoritmos de clasificación diferentes y después se mostrarán ejemplos y resultados de lo obtenido con el sklearn con ese algoritmo.

3.1 K Vecinos proximos

El algoritmo de los k vecinos proximos es el algoritmo más simple de los algoritmos de clasificación, este se basa en la distancia entre dos puntos. Dado un conjunto de entrenamiento X con sus respectivas etiquetas y un vector x que se quiere clasificar lo que hace es lo siguiente:

- Calcular la distancia del vector x a todos los vectores de X.
- Obtener los k vecinos mas cercanos a x pertenecientes a X.
- Contar cuantos elementos de cada clase hay en los k vecinos mas cercanos.
- Regresar la clase que tenga mas vecinos cercanos.

En la siguiente imagen se puede observar como funciona el algoritmo de forma gráfica, con k=5, 4 de los vecinos mas próximos pertenecen a la clase roja y 1 a la clase verde, por lo tanto xj pertenece a la clase roja.



3.1.1 K vecinos proximos con sklearn

Ahora veremos como aplicar el algoritmo de k vecinos proximos con sklearn a los datos que transformamos previamente. Para esto utilizaremos la clase Pipeline, la cual recibe varios objetos de transformación de vectores y un clasificador a lo último, y todo se aplica en orden. El código para esto se muestra a continuación:

```
import leer_datos
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier
labels = leer_datos.readClasses()

#se leen los datos de entrenamiendo.
training = leer_datos.readTexts(0,4000)
trainingLabels =labels[0:4000]
print("leidos los datos de entrenamiendo.")

#se crea el pipeline con transformacion tf-idf y clasificador con k
    vecinos proximos.
text_clf = Pipeline([
    ('vect',CountVectorizer()), #Bag of words
    ('tfidf',TfidfTransformer()), #transformacion tf-idf
    ('clf',KNeighborsClassifier(n_neighbors=3)) #clasificador k
        vecino proximos
])

#se entrena el clasificador
text_clf.fit(training,trainingLabels)
print("Se termino de entrenar el clasificador.")

#se obtienen los datos de testing
testing = leer_datos.readTexts(4000)
testingLabels = labels[4000:]
print("Se terminaron de leer los datos de testing.")

#se clasifican los datos de testing
predicted = text_clf.predict(testing)
print("Datos de testing clasificados.")

#se calcula el porcentaje de bien clasificados
well_clasified = 0
for pred,value in zip(predicted,testingLabels):
    if pred == value:
        well_clasified += 1

print((well_clasified/327)*100)
```

3.1.2 Resultados

Los resultados obtenidos fueron los siguientes:

K	3	5	7	9	11
Aciertos	92%	90.5%	88.6%	87.1%	86.2%

3.2 Naive bayes

Los métodos de Naive bayes es un conjunto de algoritmos de aprendizaje supervisado basado en aplicar el teorema de bayes con la suposición "ingenua" de independencia entre cada par de características. Dada una variable de clase y un vector de característica dependiente x1 hasta xn, el teorema de bayes genera la siguiente relación:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots x_n \mid y)}{P(x_1, \ldots, x_n)}$$

Después utilizando la suposición ingenua de independencia se tiene lo siguiente:

$$P(x_i|y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|y),$$

Con lo cual la primera expresión se transforma en la siguiente:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \ldots, x_n)}$$

Pero como el denominador es constante se puede utilizar la siguiente regla de clasificación:

$$P(y \mid x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

$$\Downarrow$$

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y),$$

Ahora el priblema es encontrar el argunmento "y" que maximise la expresión para alguna distribución P.

3.2.1 Distribución gaussiana

Se puede utilizar naive bayes con una distribución P gaussiana normal con la siguiente formula:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

3.2.2 Distribución multinomial

Otra distribución que se puede utilizar es la multinomial, esta es una de las distribuciones más clásicas utilizadas con naive bayes para clasificación de textos. La distribución es parametrizada por los vectores $\theta_y = (\theta_{y1}, \ldots, \theta_{yn})$ para cada clase "y", donde n es el número da características y θ_{yi} es la probabilidad de P(xi | y).

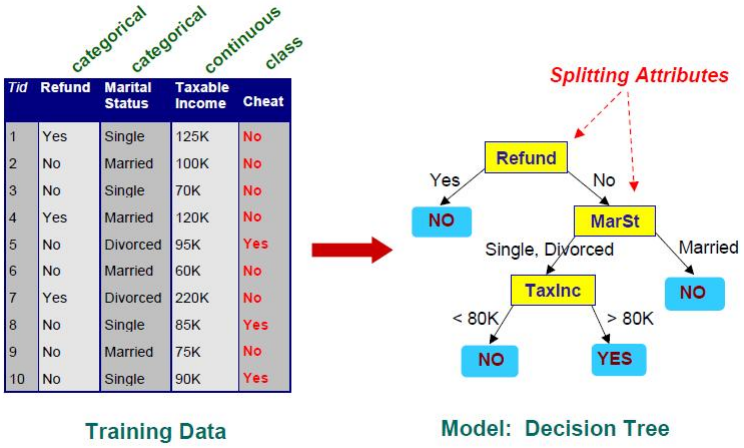
3.2.3 Resultados obtenidos

Utilicé un clasificador Naive bayes multinomial y uno con una distribución de bernoulli, y los resultados que se obtuvieron fueron los siguientes:

multinomial	83.1%
bernoulli	78.5%

3.3 Arboles de decisión

El clasificador con árbol de decision organiza una serie de preguntas y condiciones en una estructura de árbol. En la siguiente figura se muestra un ejemplo de árbol de decisión. En los árboles de decisión, la raíz y los nodos internos contienen atributos y condiciones para separar elementos con diferentes características, Todos los nodos terminales se les asigna una etiqueta con la clase a la que pertenece.



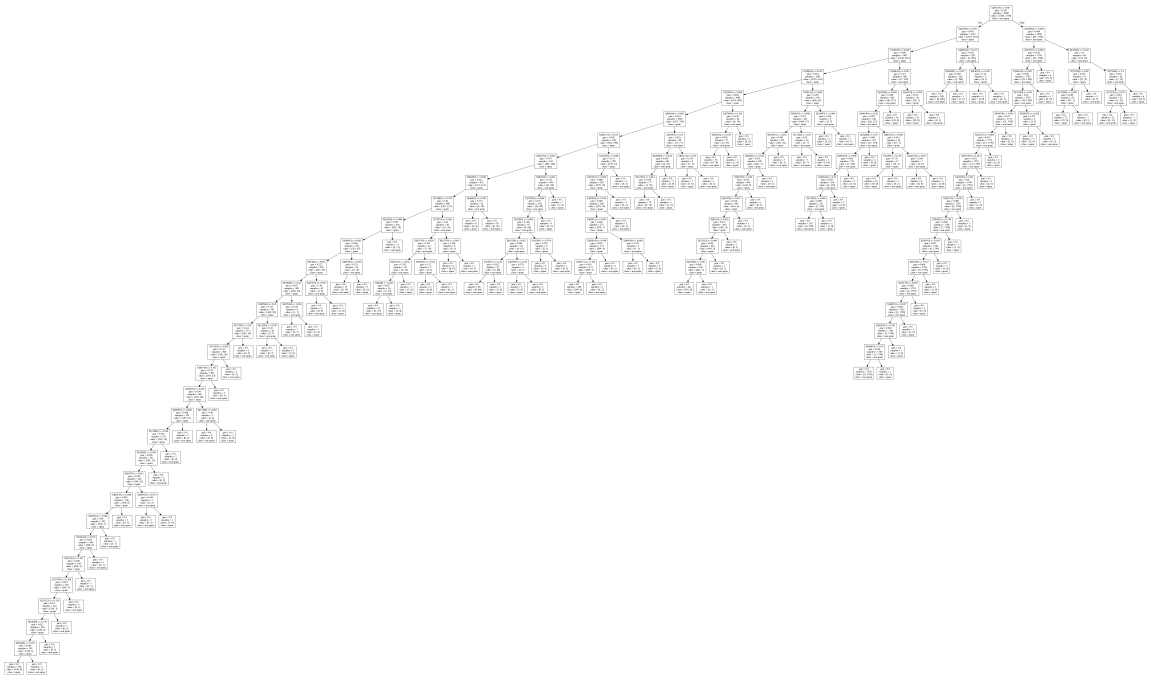
Y ahora a partir del árbol de decisión se puede clasificar nuevos datos.

3.3.1 Generando un árbol de decisión

A continuación se muestra el código con el que se genera un árbol de decisión a partir de un conjunto de datos y después se muestra el árbol.

```
types = ["spam","non_spam"]

labels = leer_datos.readClasses()

#se leen los datos de entrenamiendo.
training = leer_datos.readTexts(0,4000)
trainingLabels =labels[0:4000]
print("leidos_los_datos_de_entrenamiendo.")

count_vect = CountVectorizer()
count_vect_train = count_vect.fit_transform(training)

tfidf_transformer = TfidfTransformer()
train_tfidf = tfidf_transformer.fit_transform(count_vect_train)

clf = tree.DecisionTreeClassifier()
clf = clf.fit(train_tfidf, trainingLabels)
tree.export_graphviz(clf, out_file='arbol.dot',class_names = types)
```

Arbol de decisión generado.

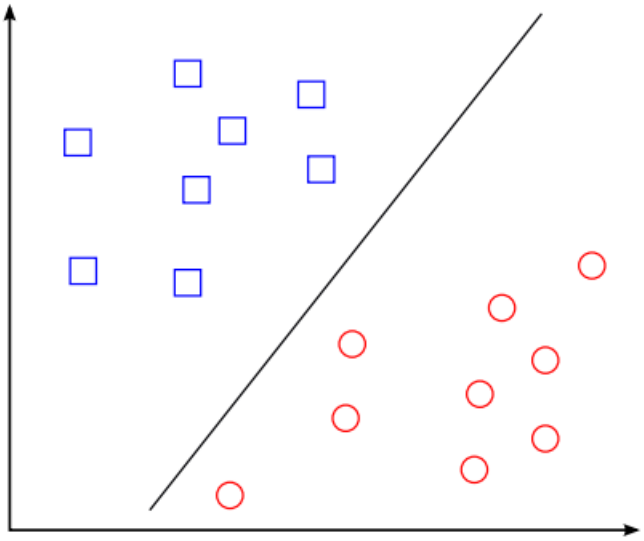


3.3.2 Resultados

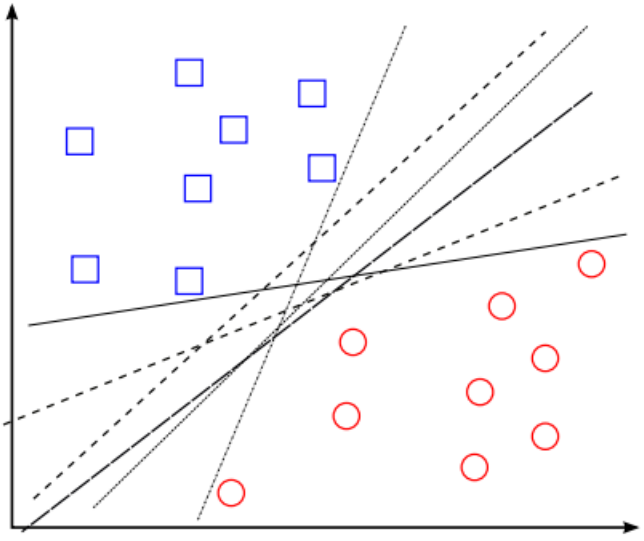
Con el árbol de decisión generado se obtuvo un 94.1% de elementos bien clasificados.

3.4 Maquinas de soporte de vectores

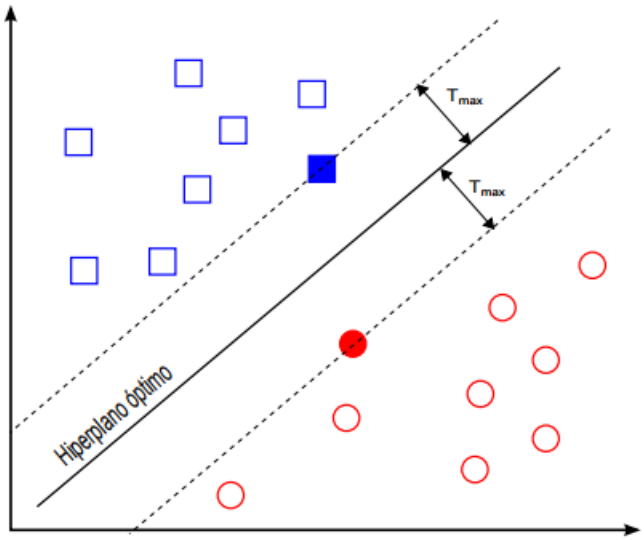
Las máquinas de soporte de vectores provienen de la idea de separar los datos a clasificar con un hiperplano, como se muestra en la siguiente imagen:



Pero podemos observar que existe una infinidad de rectas que pueden divir los datos como se ve a continuación:



Ahora la pregunta es ¿Cuál es la recta que mejor divide los datos? La respuesta es la que se encuentra a mayor e igual distancia de ambas clases, como se muestra en la siguiente imagen:



Y ahora el problema se reduce a encontrar el plano óptimo. Esto se hace con métodos de optimización en los cuales no entraré mucho a detalle.

3.4.1 Funciones kernel

En la vida real practicamente ningún conjunto de datos es linealmente separable, por lo que en las máquinas de soporte de vectores se utilizan como parámetro unas funciones denominada funciones kernel, las cuales transforman la separación lineal en una separación con alguna otra función. Entre las funciones kernel mas comunes se encuentran: el kernel polinomial, el sigmoide, el rbf(Radial basis function) y el lineal(El que se describió en la sección anterior)

3.4.2 Resultados obtenidos

Kernel	**Aciertos**
Lineal	98.7%
Sigmoide	65.1%
Polinomial(grado 3)	66.2%

3.5 Otros algoritmos utilizados

A continuación se muestran los resultados obtenidos con otros algoritmos de clasificación, tales como Random forest, bagging y adaboost.

Kernel	**Aciertos**
Ada boost	97.8%
bagging	90.2%
Random forest	96.9%

4 Conclusiones

En base a los resultados obtenidos a lo largo del documento, se puede concluir que los algoritmos de clasificación vistos son una muy buena opción, ya que en general casi todos los algoritmos dieron un porcentaje de bien clasificados arriba del 80%, a excepción de unos pocos. En los resultados que se obtuvieron en las máquinas de soporte de vectores se puede observar que el clasificador lineal es el que mejor clasificó los datos, dando 98% de datos bien clasificados y al rededor de 60% en los otros dos clasificadores, de esto se puede concluir que los datos son linealmente separables, lo cual es muy extraño que suceda. Esto nos lleva a pensar que la base de datos está modificada por usuarios para que sea linealmente separable. El siguiente paso para seguir con esto sería probar otras bases de datos para confirmar si realmente esto es cierto.

References

[1] `http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html`

[2] Alex Smola y S.V.N. Vishwanathan *Introduction to Machine Learning.* (2008) The Pitt Building, Trumpington Street, Cambridge, United Kingdom