Aprendizaje automático

Trabajo práctico 1

Spam Filter

Integrante	LU	Correo electrónico
Negri, Franco	893/13	franconegri2004@hotmail.com
Podavini Rey, Martín Gastón	483/12	marto.rey2006@gmail.com
Valdés Castro, Tobías	800/12	tobias.vc@hotmail.com

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Tabla de contenidos

1. Introducción

En este trabajo práctico nos propondremos clasificar mails en *spam* como en no *spam* (conocido también como ham}).

Para esto analizaremos los datos en busca de atributos útiles y luego experimentaremos con diversos clasificadores con la intención de encontrar los que mejor clasifiquen.

Además utilizaremos técnicas de reducción de dimensionalidad con la intención de mejorar aún más los modelos de los puntos anteriores.

Para tener una buena métrica de los resultados obtenidos, apartamos una porción del set de datos que nos fue entregado para ser utilizado al final de la experimentación y así tener una visión realista de qué tan buenos son los clasificadores elegidos. Con el resto de los datos, realizamos k-*folds* con k = 10 con la intención de minimizar en cierta medida el *overfitting* de datos.

2. Desarrollo

Como ya adelantamos en la sección anterior, el objetivo de este trabajo práctico será clasificar mails en *spam* y *ham*. Para ello, comenzaremos seleccionando atributos adecuados que consideramos que dividen bien el problema.

2.1 Extracción de atributos

Para la extracción de atributos observamos parte del *JSON* entregado en busca de palabras claves que pudieran distinguir entre *spam* y *ham*.

De este análisis se encontró que palabras tales como *viagra* o *nigeria* son muy frecuentes en los mensajes de *spam*, como también aquellas que hagan referencia a negocios o a dinero, por lo que incluímos atributos que cuenten las veces que son mencionadas estas palabras en el cuerpo y el encabezado del mensaje. Así también observamos que los mails de *spam* tienden a tener enlaces hacia sitios web o contenido HTML en el cuerpo del mensaje por lo que también contamos la cantidad de ocurrencias de palabras tales como *html* o *http* y símbolos especiales como la barra invertida, o el hashtag.

De esta manera reunimos alrededor de *100* atributos que utilizaremos a continuación para la experimentación.

Lista completa de atributos

Cantidad de apariciones de :

vicodin	viagra	free	insurance	hp	hpl	guarantee	nigerian
html	http	child	rate	george	lab	vacation	win
make	conference	lifetime	sample	labs	telnet	selected	hosting
[	all	(	wife	data	technology	natural	membership
3d	our	dear	friend	parts	pm	instant	bonus
over	remove	ad	click	direct	cs	promise	prince
internet	order	here	$	meeting	original	satisfaction	address
mail	receive	notspam	subscribe	project	re	!	marketing
will	people	junk	traffic	edu	table	.	testosterone
report	addresses	spam	trial	nigeria	buy	#	<espacio>
free	business	xanax	valium	income	money	;	opportunity
email	you	weight	hormone	extra	cash	inventory	disappointment
credit	your	\	growth	font			

También utilizamos una técnica para reducir la cantidad de atributos y quedarnos con los mejores (como métrica de mejor tomamos aquellos atributos que tengan una varianza mayor a 0.5). Estos fueron los siguientes:
html - make - all - our - over - will - people - report - free - business - email - you - credit - your - pm - table - cash - free - natural - (- [- ! - $ - . - \ - # - <espacio> - ;

Mas adelante explicaremos cómo esta reducción de atributos afectó a la performance.

2.2 Experimentación

Los clasificadores que utilizaremos para experimentar en este trabajo serán:

- Árboles de Decisión
- Multinomial Naive Bayes
- Vecinos más cercanos
- SVM (SVC)
- Random Forest

Para cada uno de ellos utilizaremos *GridSearch* para intentar encontrar los hiper parámetros que logren la mejor clasificación.

En cada una de las experimentaciones utilizamos *GridSearch* de sklearn con 10 *k-folds*

2.2.1 Árboles de decisión

Como vimos en clase, este clasificador intentará aplicar una serie de reglas sucesivas para determinar la clasificación. Por ejemplo, un posible árbol de decisión para clasificar *spam* podría ser: si el mensaje contiene la palabra *nigeria* mas de *4* veces, es *spam*, si no, podríamos ver cuántas veces se menciona *html* en el mensaje, si este resultado está entre *4* y *7* es *spam*, si no no... y así sucesivamente.

En particular, los hiper parámetros que queremos encontrar para este algoritmo son: la máxima profundidad del árbol, y la cantidad mínima de muestras necesarias para dividir un nodo.

Experimentamos variando los hiper parámetros entre los siguientes valores:

- *max_depth*: 1,3,5,10,15,50,100
- *min_samples_split*: 1,3,5,10,15

El mejor resultado obtenido fue de 0.9659 con un *max_depth* de *50* y un *min sample split* de *1*.

2.2.2 Naive Bayes

El objetivo del Naive Bayes será encontrar la clase más probable de cada instancia en base al cálculo de la probabilidad bayesiana.

Para este clasificador queremos encontrar el mejor suavizado laplaciano (*alpha*), para ello experimentamos con los siguientes valores:

- *alpha*: 0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.8,0.9,1

El resultado del *GridSearch* arroja que el mejor score resulto ser de 0.5877 con un *alpha* de 0.1.

Como podemos ver, el resultado es bastante pobre, básicamente estaríamos tirando una moneda por cada email y si sale cara votamos que es *spam*. Esto nos trajo ciertas dudas, ya que es uno de los principales clasificadores a la hora de hacer un filtro de *spam*. Entonces intentamos mejorando pensando verdaderamente qué es lo que está haciendo este clasificador.

Veamos un ejemplo: nos dimos cuenta que le estábamos asignando una probabilidad a que sea *spam* dado que encontró 24 ocurrencias de la palabra de *viagra*, luego otra probabilidad a que sea *spam* dado que encontró 12 ocurrencias de la palabra de *viagra*, nuevamente continuamos diciendo que es *spam* dado que en otro email encontró 7 ocurrencias de la

palabra viagra, etc. Por lo tanto, nos dimos cuenta que esto *no era lo que realmente queríamos hacer*, nosotros queríamos asignar una probabilidad **dado** que la palabra *viagra* fue encontrada. Es decir, qué probabilidad darle a un email para que sea *spam* dado que se encontró la palabra *viagra*, la palabra *nigeria*, y así sucesivamente con los restantes 99 atributos.

Cuando programamos esto con las correcciones mencionadas nos topamos con valores de *accuracy* mucho más altos (0.7693). En la sección *Testeo Final* se hace un análisis más profundo entre ambos Naïve Bayes (uno con probabilidades de ocurrencias, el otro con probabilidades de aparición de la palabra).

2.2.3 Vecinos más cercanos

La idea tras este clasificador es dada una nueva instancia a clasificar, comparar sus atributos contra todos los de la base de entrenamiento y quedarnos con *k* vecinos más *cercanos*. La *cercanía* de una instancia con otra puede calcularse como la norma 2 de los atributos al cuadrado o alguna otra función que nos permita definir una distancia relativa entre dos hiperplanos.

Para este clasificador queremos determinar la cantidad de vecinos óptima y la función de peso a utilizar en la predicción. Para esto último tendremos dos posibilidades, con pesos uniformes o por distancia. Para dar un ejemplo sobre cuál es la diferencia, supongamos que seteo como hiperparámetro que la cantidad de vecinos es igual a 3 y para un mensaje dado obtengo que el primer vecino es *spam* y los dos restantes son *ham*. Con una función uniforme, se determinará que el mail es *ham*, ya que *ham* es mayoría. Pero podría darse el caso en que el primer vecino se encuentre a una distancia ínfima del mail a clasificar y que los otros dos se encuentren a una distancia extremadamente grande. En ese caso sería más razonable asignarle mayor peso al primer vecino de manera tal que el mail sea clasificado como *spam*. Esta segunda solución será la de utilizar pesos variables que dependan de la distancia.

Para este clasificador probamos con 1,3,5,7,10 y 15 vecinos y el mejor resultado obtenido fue de 0.9083 con una cantidad de vecinos igual a 1 y pesos uniformes.

2.2.4 SVM (SVC)

Tuvimos problemas para correr este algoritmo por lo mucho que tardaba. Con una configuración única (valores por default, *kernel* lineal) pudimos obtener un valor de *accuracy* de 0.7976, y se completó en 4 horas aproximadamente. Usar *GridSearch* en este caso nos aumentaba mucho el tiempo y en 12 horas no había completado, por lo cual decidimos frenarlo y quedarnos con este resultado. Creemos que no tenía mucho sentido seguir con esto si además tardaba tanto, el *trade-off* no parecía lo suficientemente bueno teniendo en cuenta nuestras métricas y el tiempo que tomaba esto en correr.

2.2.5 Random Forest

Este clasificador consiste en crear una gran cantidad de árboles de decisión, y al momento de clasificar se elige el resultado que sea la moda de las clasificaciones, determinadas por los árboles individuales.

Para este estimador queremos definir la cantidad de árboles de decisión a utilizar, la profundidad máxima que se le permitirá tener a cada árbol, y la máxima cantidad de atributos a considerar cuando se esté realizando la división de un nodo.

Experimentamos con 2,5,10,15,40,100 árboles, 2,5,10,20 atributos a considerar al momento de la división de un nodo y una profundidad máxima de 3,5,20 y sin restricciones.

Para este estimador, el mejor resultado fue de 0.9790 con una cantidad de árboles igual a 100, una profundidad irrestricta y una cantidad de *features* a examinar igual a 10.

También lo testeamos con los atributos más relevantes (como ya habíamos contado en la extracción de atributos, aquellos que tengan mayor varianza) y obtuvimos un resultado de 0.9520. Por lo que podemos concluir que muchos de nuestros atributos aportaban poco a la hora de clasificar.

2.2.6 PCA

Para intentar mejorar la precisión de los resultados utilizamos PCA para obtener las componentes principales de nuestro set de atributos.

Además, utilizamos *GridSearch* para obtener cuál era la cantidad óptima de componentes principales para cada uno de los algoritmos. Utilizamos 2,5,10,40,70 componentes para la experimentación.

Lo que observamos fue que tanto en árboles de decisión como en Random Forests las respuestas resultaron ser levemente peores, pasando de un mejor resultado de 0.9659 a 0.9315 y de 0.9790 a 0.9679, respectivamente.

Para el algoritmo de Naive Bayes esta vez utilizamos un modelo gaussiano, pero los resultados siguieron sin ser muy buenos, obteniendo un score del 0.5963

Para vecinos más cercanos, el cambio de performance no fue notable, obteniendo una respuesta de 0.908.

Si bien entendemos que los resultados del *GridSearch* podrían haber sido mejores variando nuevamente los parámetros del apartado anterior junto con la cantidad de componentes (sobre todo en el caso de vecinos más cercanos), esto también resultaba muy caro computacionalmente y por cuestiones de tiempo *no* lo incluímos en el trabajo.

3. Testeo Final

Vamos a presentar dos tipos diferentes de testeo. En uno, vamos a entrenar con el 10% de los datos y testear con el 90% restante y viceversa. Los motivos de esto son que no siempre tenemos un dataset con muchas instancias para poder entrenar nuestros algoritmos.

3.1 Resultados con 10% del dataset para entrenamiento

3.1.1 Árboles de decisión

	Spam (predicho)	**Ham (predicho)**
Spam (real)	20405	20095
Ham (real)	1584	38916

	Precision	**Recall**	**F1-Score**
Ham	0.66	0.96	0.78
Spam	0.93	0.50	0.65

AUC: 0.7318

3.1.2 Naive Bayes

	Spam (predicho)	**Ham (predicho)**
Spam (real)	34269	6231
Ham (real)	12455	28045

	Precision	**Recall**	**F1-Score**
Ham	0.82	0.69	0.75
Spam	0.73	0.85	0.79

AUC: 0.8840

3.1.3 Naive Bayes Mejorado

	Spam (predicho)	**Ham (predicho)**
Spam (real)	34271	6229

Ham (real)	12477	28023

	Precision	**Recall**	**F1-Score**
Ham	0.82	0.69	0.75
Spam	0.73	0.85	0.79

AUC: 0.8531

3.1.4 Vecinos más cercanos

	Spam (predicho)	**Ham (predicho)**
Spam (real)	18480	22020
Ham (real)	8672	31828

	Precision	**Recall**	**F1-Score**
Ham	0.59	0.79	0.67
Spam	0.68	0.46	0.55

AUC: 0.6751

3.1.5 SVM (SVC)

	Spam (predicho)	**Ham (predicho)**
Spam (real)	34537	5963
Ham (real)	9196	31304

	Precision	**Recall**	**F1-Score**
Ham	0.84	0.77	0.81
Spam	0.79	0.85	0.82

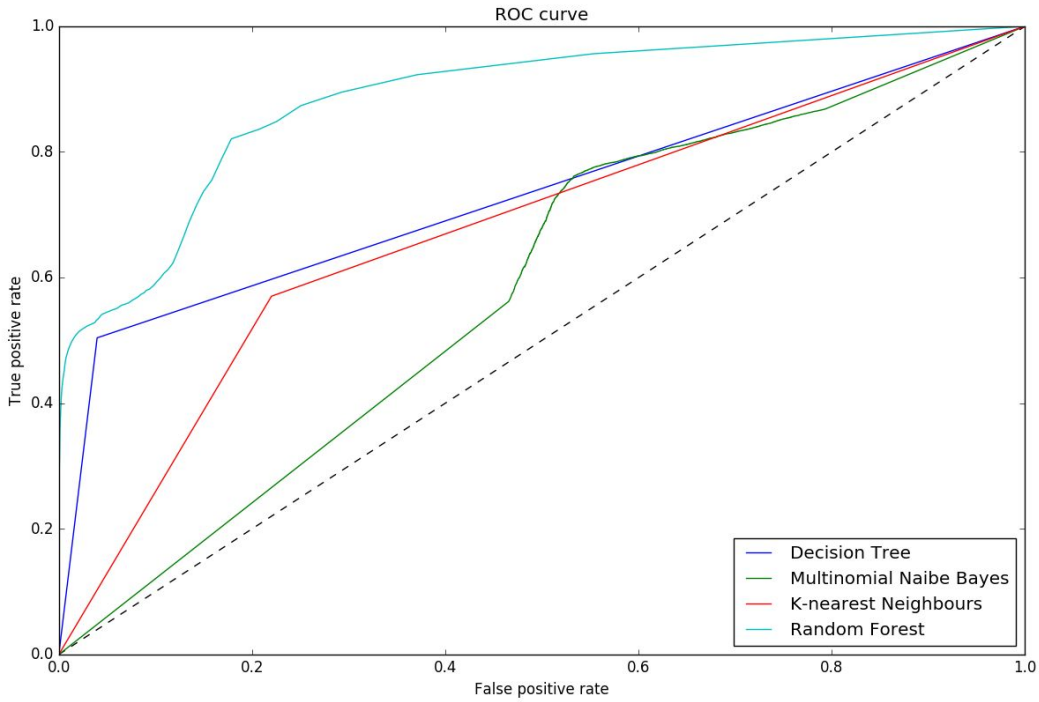
3.1.6 Random Forest

	Spam (predicho)	**Ham (predicho)**
Spam (real)	21352	19148
Ham (real)	1472	39028

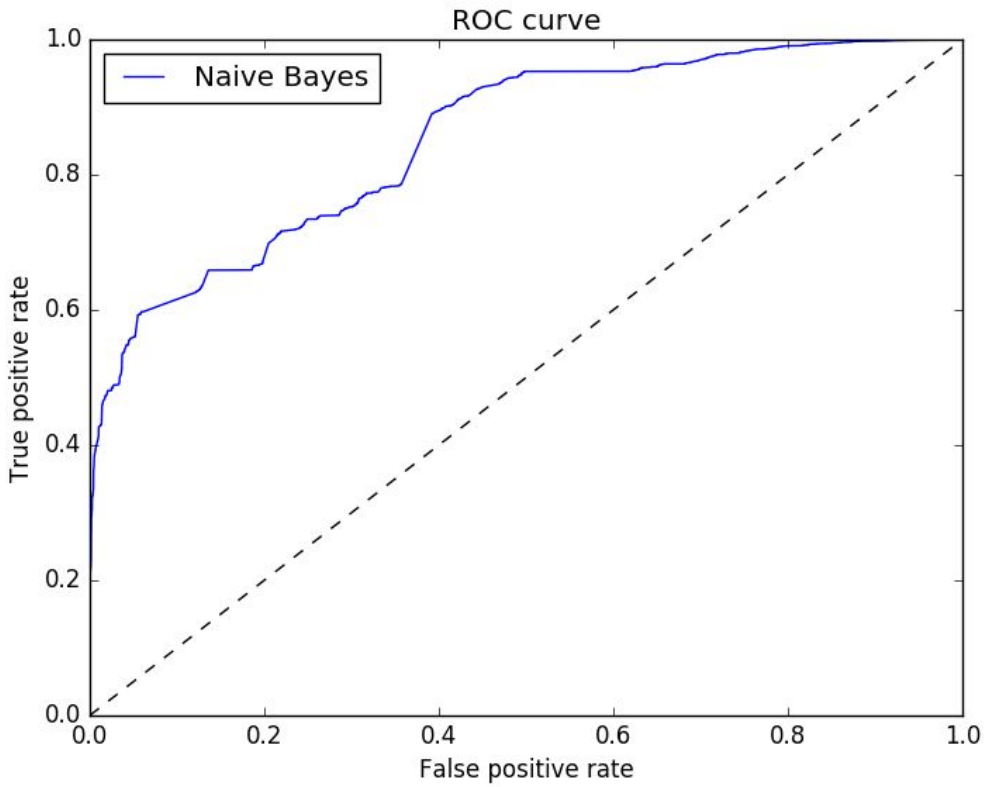
	Precision	Recall	F1-Score
Ham	0.67	0.96	0.79
Spam	0.94	0.53	0.67

AUC: 0.8747

3.1.7 Curvas ROC



Naive Bayes Mejorado:



3.2 Resultados con 90% del dataset para entrenamiento

3.2.1 Árboles de decisión

	Spam (predicho)	**Ham (predicho)**
Spam (real)	4303	197
Ham (real)	194	4306

	Precision	**Recall**	**F1-Score**
Ham	0.95	0.96	0.96
Spam	0.96	0.95	0.96

AUC: 0.9575

3.2.2 Naive Bayes

	Spam (predicho)	**Ham (predicho)**
Spam (real)	4450	50

Ham (real)	4120	380

	Precision	**Recall**	**F1-Score**
Ham	0.88	0.08	0.15
Spam	0.52	0.99	0.68

AUC: 0.7609

3.2.3 Naive Bayes mejorado

	Spam (predicho)	**Ham (predicho)**
Spam (real)	3152	1348
Ham (real)	523	3977

	Precision	**Recall**	**F1-Score**
Ham	0.75	0.88	0.81
Spam	0.86	0.70	0.77

AUC: 0.8871

3.2.4 Vecinos más cercanos

	Spam (predicho)	**Ham (predicho)**
Spam (real)	3519	981
Ham (real)	1161	3339

	Precision	**Recall**	**F1-Score**
Ham	0.77	9.74	0.76
Spam	0.75	0.78	0.77

AUC: 0.9148

3.2.5 SVM (SVC)

	Spam (predicho)	**Ham (predicho)**
Spam (real)	4131	369
Ham (real)	1107	3393

	Precision	Recall	F1-Score
Ham	0.90	0.75	0.82
Spam	0.79	0.92	0.85

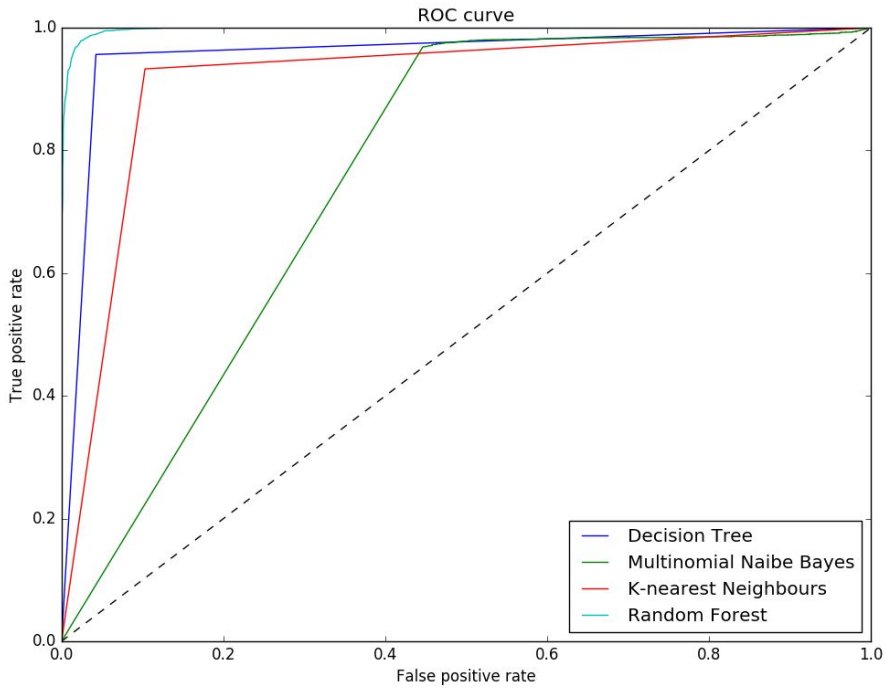
3.2.6 Random Forest

	Spam (predicho)	Ham (predicho)
Spam (real)	4438	62
Ham (real)	156	4344

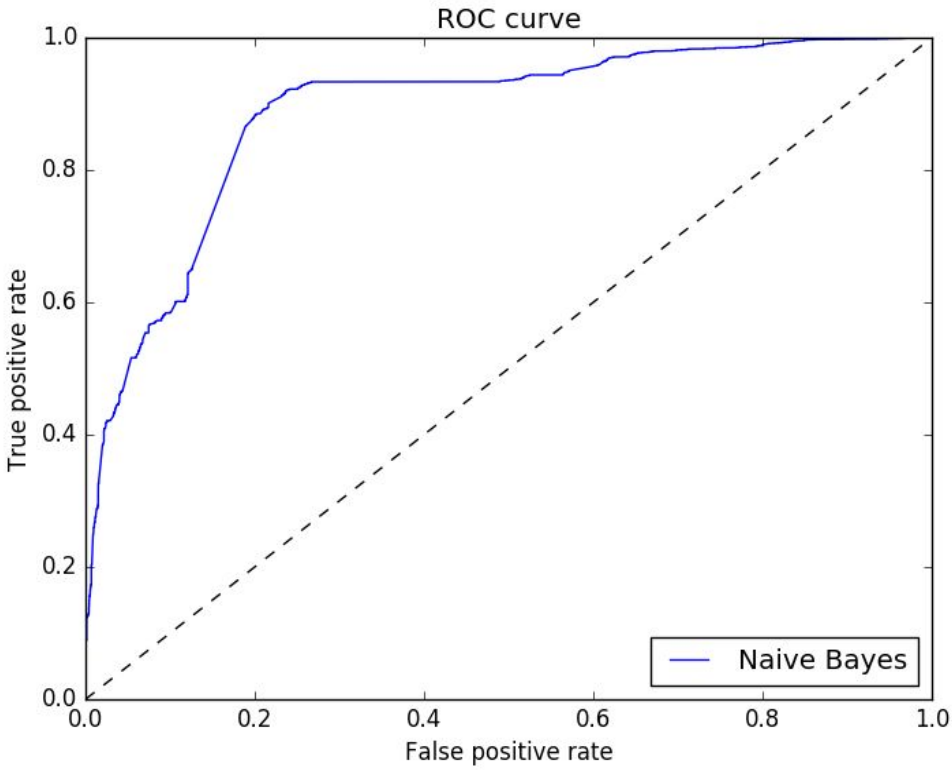
	Precision	Recall	F1-Score
Ham	0.99	0.97	0.98
Spam	0.97	0.99	0.98

AUC: 0.9977

3.2.7 Curvas ROC



Naive Bayes Mejorado:



4. Conclusiones

Como venimos diciendo, hay distintas métricas que nos permiten analizar distintos tipos de errores producidos los clasificadores que utilizamos. Nosotros nos vamos a centrar en el análisis de dos de ellas: *precision* y *F1-score*. Tomamos esta decisión ya que nos interesa minimizar los emails que no son *spam* y catalogamos efectivamente como *spam*, eliminandolos. La segunda métrica la utilizamos ya que nos permite hablar sobre *precision* y *recall* en un mismo número.

Para poder tomar una decisión sobre qué algoritmo utilizar, vamos a usar el siguiente criterio: vamos a ponderar ambas métricas y obtener un único valor. El valor final va a ser el 70% de la *precision* + el 30 % del *F1-score*, priorizando la *precision*. Para finalizar, vamos a compararlos con los valores de área bajo la curva obtenidos en las distintas curvas ROC.
Veamos los valores que obtuvimos para cada clasificador entrenando con un 10% del dataset:

	Árboles de decisión	**Naive Bayes mejorado**	**Vecinos más cercanos**	**SVM**	**Random Forest**
Precision(spam)	0.93	0.73	0.68	0.79	0.94
F1 (spam)	0.65	0.79	0.55	0.82	0.67
Ponderado	0.84	0.74	0.63	0.79	0.85
AUC	0.73	0.85	0.67		0.87

Ahora analicemos los valores obtenidos con un entrenamiento del 90% del dataset:

	Árboles de decisión	**Naive Bayes mejorado**	**Vecinos más cercanos**	**SVM**	**Random Forest**
Precision(spam)	0.96	0.86	0.75	0.79	0.97
F1 (spam)	0.96	0.77	0.77	0.85	0.98
Ponderado	0.96	0.83	0.75	0.80	0.97
AUC	0.95	0.88	0.91		0.99

Podemos destacar, que Random Forest predomina sobre los demás clasificadores tanto en la nueva métrica que introducimos así como en AUC.

Optaremos por aquel algoritmo que minimice la pérdida de performance entre ambos tipos de entrenamiento y obtenga el valor más alto. Es decir, aquel clasificador que sea más robusto en términos del tamaño del dataset. Para esto, calcularemos*:*

*0.5 * Ponderado 90% + 0.5 * Ponderado 10%*

Buscaremos aquel con el mayor valor.

	Árboles de decisión	**Naive Bayes mejorado**	**Vecinos más cercanos**	**SVM**	**Random Forest**
Ponderado	0.90	0.78	0.68	0.79	0.91
AUC	0.84	0.86	0.79		0.93

Concluimos que Random forest es el clasificador que obtiene mejores valores para las métricas propuestas, y es el más robusto en cuanto a tamaños de instancias para entrenarlo.

Algo importante a notar es que en nuestra métrica ponderada, Naïve Bayes obtuvo un valor de 0.78 mientras que Árboles de decisión obtuvo un 0.90, pero si nos ponemos a ver el valor del área bajo la curva, este lo supera.
Ademas encontramos un paper donde usan Random Forest para clasificación de mails tipo *spam* o *ham* [1] obteniendo buenos valores de *Accuracy, Precision* y *Recall*. También encontramos varias comparaciones entre Naïve Bayes y Random Forest. En el siguiente paper [2] hay una comparación entre que tipo de naive bayes mejoraba la clasificación de spam. Estos eran Bernoulli multivariada NB y Multinomial NB. Concluyen que el que mejor clasifica era Multinomial NB.

[1] http://www.irdindia.in/journal_ijacect/pdf/vol2_iss4/1.pdf
[2] http://www.aueb.gr/users/ion/docs/ceas2006_paper.pdf