Aprendizaje Automatico
Trabajo Práctico 1
Detección de Spam

15 de diciembre de 2016

Integrante	LU	Correo electrónico	Carrera
Martin Baigorria	575/14	martinbaigorria@gmail.com	Computación (Licenciatura)
Damián Furman	936/11	damian.a.furman@gmail.com	Computación (Licenciatura)
Germán Abrevaya	-	germanabrevaya@gmail.com	Física (Doctorado)

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

1. Introducción

El objetivo de este trabajo practico es experimentar con diferentes técnicas de aprendizaje automático para entrenar un clasificador de mensajes de correo electrónico en dos clases, 'spam' y 'ham'. Para ello, se utilizaran diferentes modelos y métodos para buscar los diferentes atributos que utilizaremos. Se experimentaran con diferentes técnicas de reducción de dimensionalidad. Además, se evaluaran los diferentes hiperparametros de cada modelo para intentar buscar una mejor performance de los algoritmos de aprendizaje. Dejamos el 10 % de los datos como testing set, utilizando el resto para experimentar con los diferentes modelos. Al elegir los hiperparametros sobre el training set, caemos en la posibilidad de estar haciendo overfitting. Siguiendo el Principio de Occam, una vez elegido el modelo que utilizaremos, también haremos feature selection para intentar encontrar el modelo mas simple posible con una performance que consideremos razonable.

2. Extracción de atributos

2.1. Bag of words

En una primera etapa, antes de intentar seleccionar atributos de forma manual, la idea fue analizar la frecuencia de cada palabra en cada mail, ignorando el orden de las palabras. Notar que los mails están en formato MIME, por lo que no se podía simplemente hacer un análisis de frecuencia sobre el MIME en plaintext, si no que había que utilizar un parser para luego poder analizar las diferentes partes de cada correo. Además, muchos de los correos no estaban en utf-8, lo que dificultaba el análisis. Por esa razón decidimos ignorarlos para el análisis de frecuencia.

Dado que el alto tiempo de computo para calcular la frecuencia de cada palabra en cada mail, decidimos simplemente tomar una variable binaria por mail que indica si la misma esta o no en el mismo. La idea fue calcular por clase cuantas veces aparecía cada palabra en el cuerpo de un correo, para luego ordenar todas las palabras por la diferencia absoluta en frecuencia relativa y de esa forma seleccionar los mejores atributos. Seguramente existe un orden mejor, pero lo que buscábamos era encontrar las palabras que mas distinguían entre las dos clases. Una idea que teníamos a priori además es que todas las conjunciones iban a aparecer con la misma frecuencia en ambas clases, por lo que al calcular su respectivo score se compensarían y no aparecerían en nuestros resultados finales. Además ignoramos todos los números, dado que creemos que no es un buen predictor.

En una primera etapa, para decidir que atributos se iban a tener en cuenta tomamos todas las palabras que mostraban una diferencia de frecuencia relativa entre clases de mails mayor o igual a 100. De esta forma terminamos seleccionando XXX atributos. Entre ellos se encuentran:

['please', 'original message', 'thanks', 'any', 'attached', 'questions', 'call', 'gas', 'date', 'corp', 'file', 'energy', 'need', 'meeting', 'group', 'power', 'following', 'there', 'final', 'should', 'more', 'schedule', 'review', 'think', 'week', 'some', 'deal', 'start', 'scheduling', 'contract', 'money', 'professional', 'been', 'last', 'work', 'schedules', 'issues', 'viagra', 'however', 'contact', 'thank', 'between', 'solicitation', 'comments', 'sex', 'messages', 'discuss', 'software', 'save', 'received', 'site', 'changes', 'txt', 'advertisement', 'parsing', 'prices', 'morning', 'click', 'sure', 'visit', 'stop', 'only', 'working', 'next', 'trading', 'plan', 'tomorrow', 'awarded', 'soft', 'detected', 'now', 'like', 'about', 'doc', 'who', 'windows', 'basis', 'online', 'product', 'conference', 'prescription', 'products', 'best', 'fyi', 'point', 'agreement', 'regarding', 'forward', 'north', 'family', 'world', 'team', 'process', 'help', 'cialis', 'adobe', 'down', 'results', 'thousand', 'first', 'issue', 'link', 'offers', 'note', 'scheduled', 'management', 'capacity', 'market', 'bill', 'employees', 'daily', 'dollars']

Por un lado unificamos las palabras 'original' y 'message', que en general muestran mensajes donde se hizo 'reply' o 'reply all'. Por otro lado, la palabra 'enron' aparecía bastante, lo que nos hizo pensar que el dataset que teníamos era el de Enron. Nos cuestionamos un poco que tan representativo es este dataset, dado que corresponde en general a mails corporativos. Notar que este mismo procedimiento se puede hacer para el título de los correos.

2.2. Selección manual

Luego de seleccionar los primeros atributos utilizando nuestra variacion de Bag of Words, decidimos elegir algunos otros atributos de forma manual. Estos fueron:

- has_closing_tags
- has_links
- cant_capital
- capital_in_a_row
- is_multipart

Con estos atributos buscamos captar distintas propiedades frecuentes en los mails de spam: la presencia de links hacia páginas web de contenido malicioso o de spam, el formato HTML que otorga formato a cierta publicidad, la presencia de muchas palabras en mayúscula o títulos o subtítulos con varias letras con mayúsculas seguidas.

A su vez, a las palabras determinadas por el Bag of Words le agregamos varias palabras que consideramos de posible aparición frecuente en mails de spam, a saber: offer, sex, viagra, Nigeria, discount entre varias otras.

3. Modelos

Luego de definir nuestros atributos, probamos distintos modelos con el objetivo de evaluar su performance. Decidimos utilizar para nuestra comparación Naive Bayes, Decision Tree Classifier, Random Forest Classifier, Support Vector Machine, K Nearest Neighbours y AdaBoost. Éste último es similar a la técnica de boosting vista en clase, en la cual la salida de otros algoritmos de machine learning ("weak learners") son combinadas en una suma pesada que representa la salida final del clasificador. La principal diferencia de un simple boosting respecto a AdaBoostes que este último es adaptativo en el sentido de que los weak learners subsiguientes son a su vez orientados para corregir las instancias mal clasificadas por los anteriores weak learners.

Antes de entrenar los modelos, se separó el 10 % de los datos totales para ser empleados como testing set y dar una medida más realista del desempeño de los modelos, entrenados con el 90 % restante de los datos.

4. Resultados

A contiunación se presentarán para distintos modelos el tiempo de cómputo para entrenar al modelo (sin cross-validation), *Train mean accuracy* (calculado sobre los datos de entrenamiento empleando un K-Folding Cross Validation con K = 10), cantidad de falsos y verdaderos casos positivos y negativos (calculados en base a la predicción de sobre el training set), *precision* y *recall*, definidos como:

$$\text{precision} = \frac{\text{verdaderos positivos}}{\text{verdaderos positivos} + \text{falsos positivos}} \quad \text{recall} = \frac{\text{verdaderos positivos}}{\text{verdaderos positivos} + \text{falsos negativos}}$$

4.1. Support Vector Machine

Este método resultó impráctico debido al descomunal tiempo de computo necesario para entrenar el modelo. Quitándole 90 atributos calculados mediante Bag of Words el tiempo de cómputo para entrenar al modelo (sin cross-validation) fue de 2:17 horas. El Test mean accuracy resultó en este caso de 0.7933, la precision fue de 0.8829 y el recall, 0.6739. SVM utiliza quadratic programming lo cual conlleva un gran tiempo de cálculo cuando se trabaja con muchas dimensiones. No hay un criterio marcado de localidad para buscar hiperplanos separadores. Para sacarle más provecho a este método haría falta un kernel especifico.

4.2. Naive Bayes

Naive Bayes tiene la interesante propiedad de dar resultados acpetables pagando un tiempo de cómputo mínimo (tarda 0.32 segundos en entrenarse sobre el corpus de texto completo compuesto por 9 mil mails). Sin embargo, sus resultados no son aceptables en términos de un clasificador real. Principalmente porque, como se puede ver en el Cuadro 1, si bien logra un *accuracy* de un 80 %, genera una gran cantidad de falsos positivos en relación a los falsos negativos (*recall* es de un 90 % mientras que *precision* es de un 75 %). Naive Bayes considera cada uno de los atributos como un factor independiente de los otros atributos que aporta 'probabilisticamente' a la decisión sobre la clasificación de un elemento. Por esto decidimos quitar varios de sus atributos e ir agregándoselos por partes para de esta manera comprobar cómo varían los resultados obtenidos.

	10 Atributos	**65 Atributos**	**130 Atributos**
Train mean accuracy	0.49	0.6162	0.8059
false positives	4465	3269	1321
false negatives	125	185	426
true positives	4266	4285	4100
true negatives	144	1261	3153
precision	0.4886	5672	0.7563
recall	0.9715	0.9586	0.9059

Cuadro 1: Resultados de Naive Bayes.

Lo primero que podemos observar es que el clasificador Naive Bayes es muy sensible a la cantidad de atributos: parecería mejorar invariablemente en la medida que se agregan más. Esto sucede debido a que como cada atributo se considera independiente del resto y aporta una 'porción' de la probabilidad, al agregar más atributos la probabilidad total debería estar más afinada. Por otro lado, Naive Bayes no distingue distintas importancias entre los atributos, por lo que la efectividad de éste método también está supeditada a que la elección de los atributos que se agregan sea correcta.

Otra observación que puede realizarse es que cuando el clasificador no tiene suficiente cantidad de atributos tiende a clasificar todo en la misma categoría. En el caso de nuestra experiencia puede verse como con 10 atributos casi todos los mails fueron clasificados como Spam, lo cual genera que la métrica de *precisión* sea bajísima mientras que el *recall* tiene valores muy altos.

4.3. Random Forests

El tiempo de cómputo de un modelo Random Forest con todos los atributos y con 100 árboles fue de 4.54 minutos. Se realizaron varias pruebas con el este clasificador variando uno de sus hiperparámetros más importantes: la cantidad de árboles a utilizar. En el Cuadro 2 se puede ve cómo este hiperparámetro afecta los resultados en términos de predicción.

	30 árboles	**50 árboles**	**100 árboles**	**150 árboles**	**200 árboles**
Test mean accuracy	0.9518	0.9521	0.9538	0.9566	0.9529
false positives	150	155	169	144	158
false negatives	284	276	247	247	266
true positives	4256	4296	4187	4256	4247
true negatives	4310	4123	4397	4353	4329
precision	0.9660	0.9652	0.9612	0.9673	0.9641
recall	0.9374	0.9396	0.9443	0.9451	0.9411

Cuadro 2: Resultados de Random Forest.

En primer lugar se puede observar que aumentar la cantidad de árboles en general tiende a mejorar la *accuracy* aunque su variación es relativamente pequeña (alrededor del 0,2 %). Sin embargo, con un exceso de arboles la *accuracy* comienza a disminuir, aunque también con una tasa de decrecimiento baja. En general la eficiencia de este algoritmo como clasificador para nuestro cuerpo de datos no parece ser muy sensible ante variaciones de la cantidad de árboles empleada, al menos en el rango de 30 a 200 árboles. Aún así, el mejor resultado, tanto en términos de accuracy como de recall y de precision, se obtuvo al correr Random Forest con 150 árboles.

4.4. kNN:

Para K Nearest Neigbours consideramos intentar mejorar los resultados a partir de variar los k vecinos considerados. Mientras que en una primera instancia utilizamos un k = 5, testeamos luego los resultados de aplicar un k = 10 y k = 20

	K = 5	**K = 10**	**K = 20**
Train mean accuracy	0.8714	0.8127	0.7949
false positives	762	686	758
false negatives	395	1000	1088
true positives	4131	3468	3444
true negatives	3712	3846	3710
precision	0.8443	0.8348	0.8196
recall	0.9127	0.7761	0.7599

Podemos observar que en la medida que aumenta el k, los resultados son peores. Esto se explica debido a que estamos permitiendo votar a más cantidad de 'vecinos' que se encuentran más lejos dentro del hiperplano con respecto a la coordenada en la que se evaluó el mail que queremos clasificar utilizando los parámetros dados. Si bien un k muy bajo corre el peligro de que el único criterio de clasificación provenga de un *outlyer*, un k muy alto difumina la idea del método kNN que es, a saber, clasificar nuestro corpus en base a la cercanía respecto al corpus que utilizamos como entrenamiento.

5. Reducción de dimensionalidad

Se probó el uso de PCA como técnica de reducción de dimensionaidad y algunos selectores de features, basados en test estadísticos univariados. Para ambos casos se vió que la eficiencia, en cuanto a accuracy, precision y recall, disminuía sin importar con cuantos features nos quedasemos. Sin embargo creemos que aplicando la reducción de dimensionalidad o el selector de features se disminuye el overfitting por usar features muy correlacionadas. Terminamos por elegir aplicar PCA antes de entrenar los modelos, ya que lo consideramos más robusto que el selector de features.

TODO: Elegir el mejor modelo. Hacer PCA vs Feature Selection, quizas probar luego con Grid Search.

6. Discusión

Los mejores resultados se obtuvieron con los métodos basádos en árboles, y en particular Random Forest, con el cual disminuyen especialmente la cantidad de falsos positivos (mejoran la métrica *precision*), propiedad consideramos que se debe priorizar al construir un clasificador de Spam, y es por esto que fue elegido para la competencia. Mientras que otros métodos que requieren mayor cómputo, como kNN o SVM, no alcanzaron a tener la efectividad de otros más 'sencillos'.

Son interesantes algunas de las estrategias que emplean los spammers, como por ejemplo *bayesian poisoning*, en la cual procuran emplear la información con la que se entrenan a los filtros de spam para disminuir su efectividad. Incluyen en sus mails spam palabras y otros features estadísticamente característicos de mails ham con el fin de aumentar las probabilidades de ser clasificados como ham. Por otro lado, si los filtros son entrenados con estos mails spam "envenenados", tenderá a aumentar la cantidad de falsos positivos.