Spam

March 1, 2017

1 Spam

A partir de la base de datos de spam con vectores de características de 2000 dimensiones contenidasen el archivo spam.csv:

- Entrena diferentes clasificadores despam usando clasificadores bayesianos ingenuos con distintas distribuciones
- Binariza los vectores de características y entrena los mismos clasificadores con esta representación
- Evalúa los clasificadores entrenados usando tanto los mismos datos de entrenamiento como datos distintos
- Reporta y explica el desempeño de los diferentes clasificadores

El archivo spam.csv contiene 2001 atributos por cada renglón, de los cuales los primeros 2000 corresponden al vector de características de un correo y el último corresponde a la clase, esto es, 1 si es spam y 0 si no lo es.

Primero vamos a generar un clasificador bayesiano ingenuo considerando una distribución del tipo multinomial ya que por la naturaleza de los datos consideramos que esta distribución es la que se mejor se adapta por otra vamos a considerar todo el conjunto de datos.

In [37]:
```
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.naive_bayes import MultinomialNB,BernoulliNB,GaussianNB
from pandas.tools.plotting import scatter_matrix
from sklearn import preprocessing


#Carga de los datos:
data_set = pd.read_csv("spam.csv",delim_whitespace=True,header=None)

X = data_set.ix[:,0:1999].as_matrix()
y = data_set.ix[:, 2000].as_matrix()

clf = MultinomialNB()
clf.fit(X, y)
```

```
        predicciones = clf.predict(X)

        predicciones = predicciones.astype(int)

        aciertos=0.0
        total_datos= float(data_set.shape[0])
        #print total_datos

        for ind in range(len(predicciones)):
            if int(predicciones[ind])==int(y[ind]):
                aciertos+=1


        #El porcentaje de aciertos es el siguiente:
        print "El porcentaje de aciertos es el siguiente:"
        print str((aciertos/total_datos)*100)+"%"
```

```
El porcentaje de aciertos es el siguiente:
95.3402938902%
```

Observamos que el rendimiento de nuestro clasificador es muy bueno. Lo que vamos hacer a continuación es volver a generar un clasificador con las mismas caracteristacas cona la diferencia de que utilizaremos un 80% de los datos para entrenar y un 20% para probar nuestro clasificador. Y veremos el rendimiento con esta cantidad de datos de entrenamiento.

```
In [20]: # Separar el conjunto de datos para training y para test.
         from sklearn.model_selection import train_test_split
         train, test = train_test_split(data_set, test_size = 0.2)

         X = train.ix[:,0:1999].as_matrix()
         y = train.ix[:, 2000].as_matrix()


         to_predict = test.ix[:,0:1999].as_matrix()
         to_predict_val = test.ix[:,2000].as_matrix()

         #print to_predict_val

         clf = MultinomialNB()
         clf.fit(X, y)
         predicciones = clf.predict(to_predict)
         predicciones = predicciones.astype(int)

         aciertos=0.0
         total_datos= float(to_predict_val.shape[0])
```

```
        for ind in range(len(predicciones)):
            if int(predicciones[ind])==int(y[ind]):
                aciertos+=1

        #El porcentaje de aciertos es el siguiente:
        print "El porcentaje de aciertos es el siguiente:"
        print str((aciertos/total_datos)*100)+"%"
```

```
El porcentaje de aciertos es el siguiente:
57.3913043478%
```

Como podemos observar el rendimiento de nuestro clasificador baja debido a que bajamos la cantidad de datos para entrenar a nuestro clasificador. Por otra parte lo que vamos hacer ahora es binarizar y aplicar una distribución del tipo Bernoulli y comparar el rendimiento.

```
In [35]: binarizer = preprocessing.Binarizer().fit(X)
         bina=binarizer.transform(X)

         X = bina
         y = data_set.ix[:, 2000].as_matrix()

         clf = BernoulliNB()
         clf.fit(X, y)

         predicciones = clf.predict(X)

         predicciones = predicciones.astype(int)

         aciertos=0.0
         total_datos= float(data_set.shape[0])
         print total_datos

         for ind in range(len(predicciones)):
             if int(predicciones[ind])==int(y[ind]):
                 aciertos+=1


         #El porcentaje de aciertos es el siguiente:
         print "El porcentaje de aciertos es el siguiente:"
         print str((aciertos/total_datos)*100)+"%"
```

```
El porcentaje de aciertos es el siguiente:
90.84213%
```

Como podemos observar el rendimiento de nuestro clasificador baja debido que al momento de binarizar estamos perdiendo información para nuestro clasificador.

2 Usando otra distribución:

Ahora lo que vamos hacer es usar otro tipo de distribución y ver el rendimiento que genera.

```
In [38]: #Carga de los datos:
         data_set = pd.read_csv("spam.csv",delim_whitespace=True,header=None)

         X = data_set.ix[:,0:1999].as_matrix()
         y = data_set.ix[:, 2000].as_matrix()

         clf = GaussianNB()
         clf.fit(X, y)

         predicciones = clf.predict(X)

         predicciones = predicciones.astype(int)

         aciertos=0.0
         total_datos= float(data_set.shape[0])
         #print total_datos

         for ind in range(len(predicciones)):
             if int(predicciones[ind])==int(y[ind]):
                 aciertos+=1


         #El porcentaje de aciertos es el siguiente:
         print "El porcentaje de aciertos es el siguiente:"
         print str((aciertos/total_datos)*100)+"%"
```

```
El porcentaje de aciertos es el siguiente:
92.807424594%
```

3 Conclusiones:

Por lo tanto vemos que el rendimiento tanto de binarizar como en el caso anterior de usar una distribución normal bajan el rendimiento de nuestro clasificador por una parte al momento de binarizar perdemos datos y por otra parte vemos que al forzar nuestro conjunto de datos a una distribución del tipo normal esta no se adecua de manera correcta provocando que el rendimiento de nuestro clasificador baje. En este caso el usar la distribución Multinomial es la que ha dado un mejor rendimiento.