Proyecto Análisis de Datos

Parte 4: SVMs, Máquinas Vector Soporte

José Manuel Bustos Muñoz

Santiago Martínez De La Riva

1. Carga de datos y separación train-test.

Primero se realiza la carga de los datos, y podemos hacer un head() para ver los primeros datos del dataset cargado.

	id	target	ps_ind_01	ps_ind_02_cat	ps_ind_03	ps_ind_04_cat	ps_ind_05_cat	ps_ind_06_bin	ps_ind_07_bin	ps_ind_08_bin	...	ps_calc_11	ps_calc_12	ps_ca
0	7	0	2	2	5	1	0	0	1	0	...	9	1	
1	9	0	1	1	7	0	0	0	0	1	...	3	1	
2	13	0	5	4	9	1	0	0	0	1	...	4	2	
3	16	0	0	1	2	0	0	1	0	0	...	2	2	
4	17	0	0	2	0	1	0	1	0	0	...	3	1	

5 rows × 59 columns

```
(595212, 59)
```

Vamos a dividir los datos en dos conjuntos, el 70% para entrenar y el 30% restante para test, para ello usamos el método “train_test_split”.

```
X = dfP[lVarsTarg[2:]]
clases = dfP['target']
X_train, X_test, clases_train, clases_test = train_test_split(X, clases, test_size=0.3, random_state=10)
```

Número de ejemplos de cada clase en el conjunto de entrenamiento:

```
Número de ejemplos de entrenamiento = 416648
Número de ejemplos por clase:
0    401631
1     15017
Name: target, dtype: int64
Porcentaje de ejemplos por clase:
0    0.963958
1    0.036042
Name: target, dtype: float64
```

Número de ejemplos de cada clase en el conjunto de test:

```
Número de ejemplos de test = 178564
Número de ejemplos por clase:
0    171887
1      6677
Name: target, dtype: int64
Porcentaje de ejemplos por clase:
0    0.962607
1    0.037393
Name: target, dtype: float64
```

2. Preprocesamiento.

Realizamos un preprocesamiento de los datos antes de entrenar los clasificadores.
Primero visualizamos con info() los atributos de cada conjunto, el recuento de cada uno de ellos y el tipo de dicho atributo. Aquí ya se puede apreciar que no hay missing valúes en el dataset cargado.
También utilizamos describe() para ver los principales datos estadísticos de los atributos del conjunto. Podemos apreciar como la mayoría de atributos están en la misma escala con valores muy similares.

Vemos también con otros métodos que efectivamente no hay missing values, como puede ser con de la forma de la imagen:

```
print("Los datos de training tienen {} missing values.".format(X_train.isnull().sum().sum()))
print("Los datos de test tienen {} missing values.".format(X_test.isnull().sum().sum()))
```

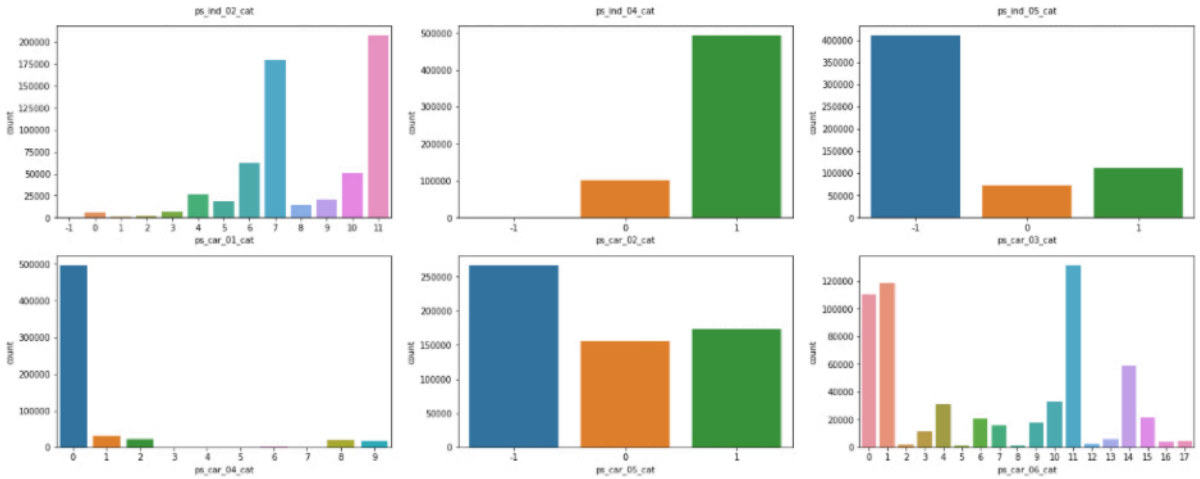
```
Los datos de training tienen 0 missing values.
Los datos de test tienen 0 missing values.
```

Para algunos análisis es necesario transformar los tipos de las columnas que no son continuas en categóricas. Lo realizamos de la siguiente forma:

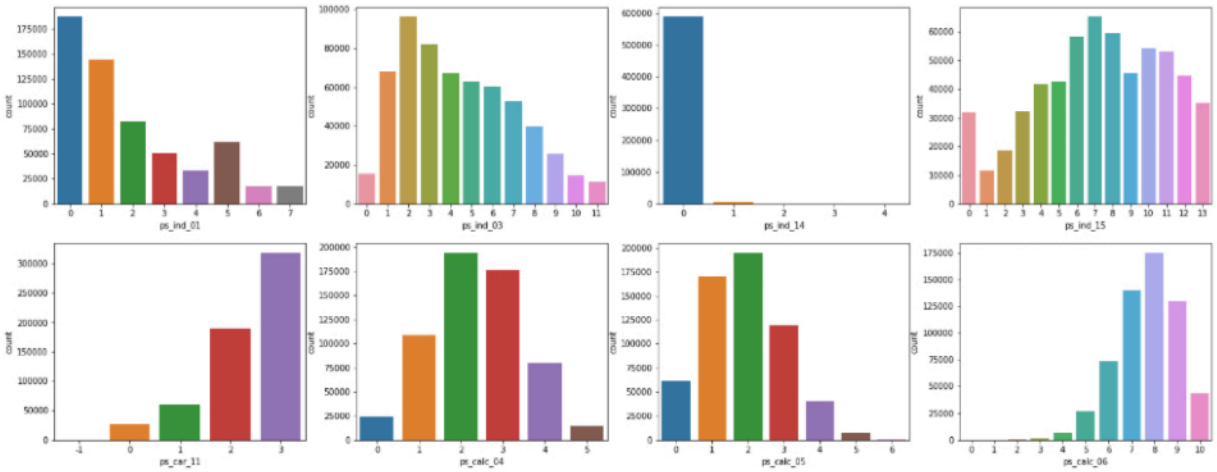
```
cat_cols = [col for col in X_train.columns if col.endswith('_bin') or col.endswith('_cat')]

X_train[cat_cols] = X_train[cat_cols].apply(pd.Categorical)
X_test[cat_cols] = X_test[cat_cols].apply(pd.Categorical)
```

Una vez realizada dicha transformación, podemos hacer otros análisis gráficos de los datos como serían los siguientes ejemplos de recuento de cada atributo:



Otro ejemplo de recuento de algunos atributos:



Los datos en el dataset vienen distribuidos en atributos de distinto tipo, además del formato, los agrupamos en los 4 grupos: inds, reg, car y calc:

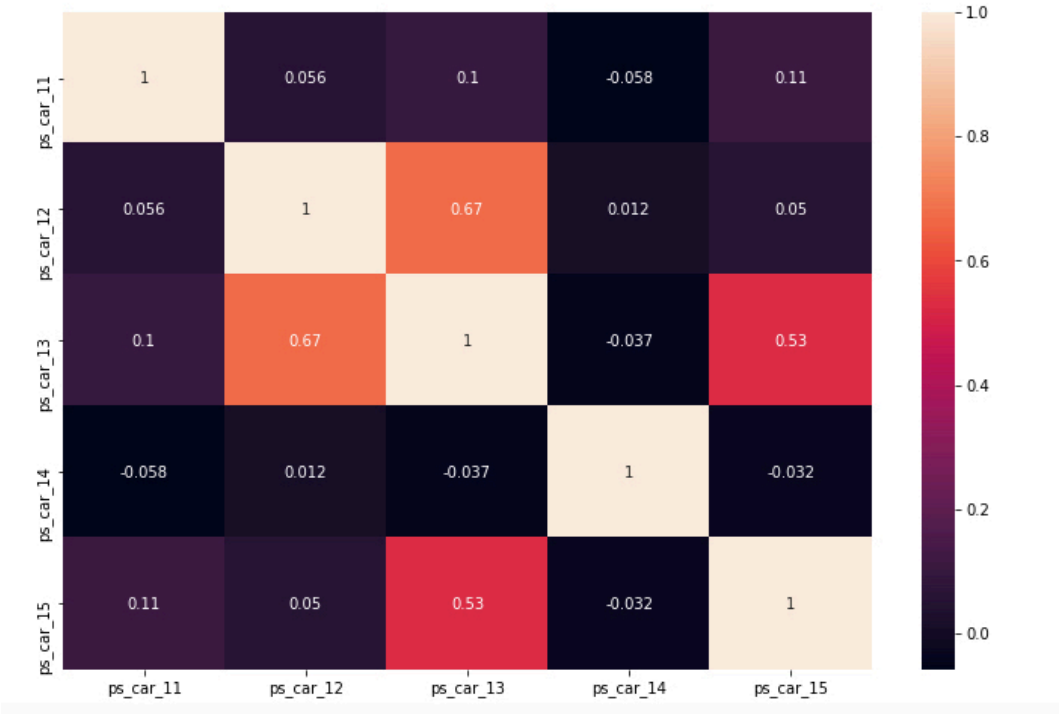
```
# Obtenemos los 4 grupos con los datos de entrenamiento
X_train_inds = X_train[inds]
X_train_reg = X_train[reg]
X_train_car = X_train[car]
X_train_calc = X_train[calc]

print(X_train_inds.shape)
print(X_train_reg.shape)
print(X_train_car.shape)
print(X_train_calc.shape)
```

```
(416648, 18)
(416648, 3)
(416648, 16)
(416648, 20)
```

Una vez tenemos agrupados los distintos tipos de atributos, podemos sacar de cada grupo la matriz de correlación, con mapa de color para ver mediante la intensidad de un color el grado de correlación entre cada par de variables continuas.

Ejemplo de gráfica de matriz de correlación:



Estudiando la correlación entre las variables continuas no se aprecian altas correlaciones entre variables, así que no nos sirve inicialmente para descartar variables muy relacionadas con lo que una podría estar contenida en la otra.

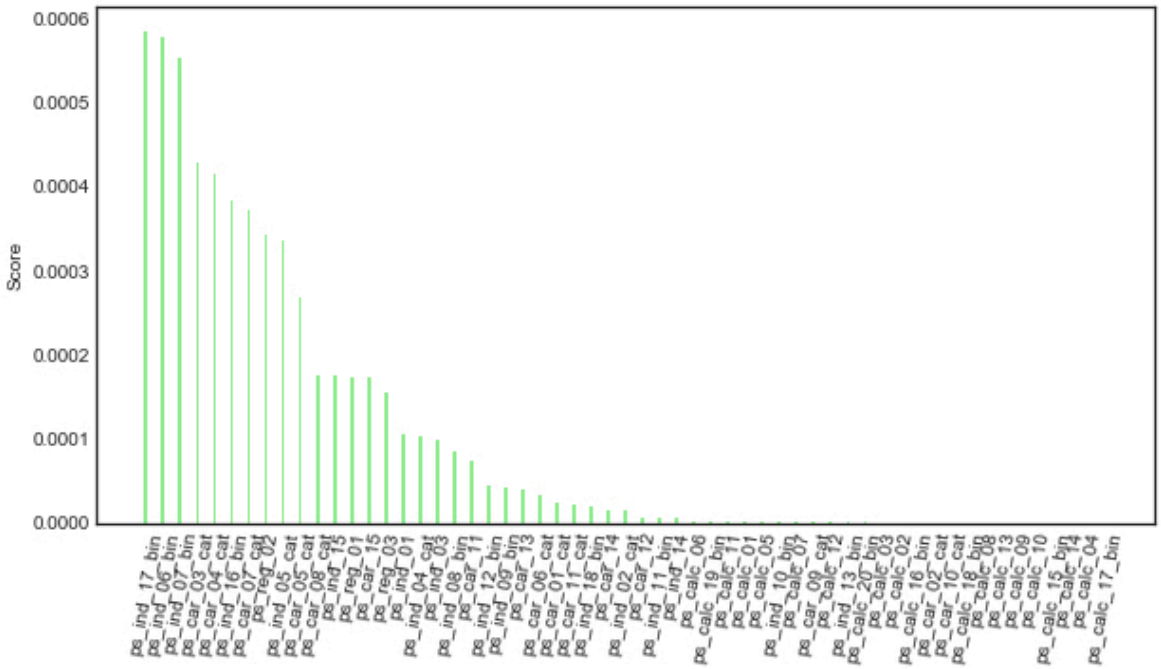
También se pueden estudiar los gráficos boxplot para ver la distribución de los datos en cada atributo así como los posibles valores outliers en cada uno.

Para eliminar los outliers podrían usarse funciones que para cada atributo miren aquellos datos que tienen un valor muy alejado del resto, utilizando para ello la media y distribución.

Para conseguir ver atributos o características a eliminar, ya que el dataset tiene un número muy elevado, podemos usar métodos de selección y filtrado. Estos métodos estudian individualmente cada atributo en relación al target, para ver aquellos que son más importantes para poder predecir el target; o bien, van estudiando en conjunto a todos los atributos contra el target, eliminando en cada iteración del proceso un atributo hasta quedarse con los atributos que maximizan el score para el target.

Usamos varios, y para todos ellos vemos como comparten la mayoría de atributos con alto score.

Ejemplo gráfico de lo obtenido con mutual information:



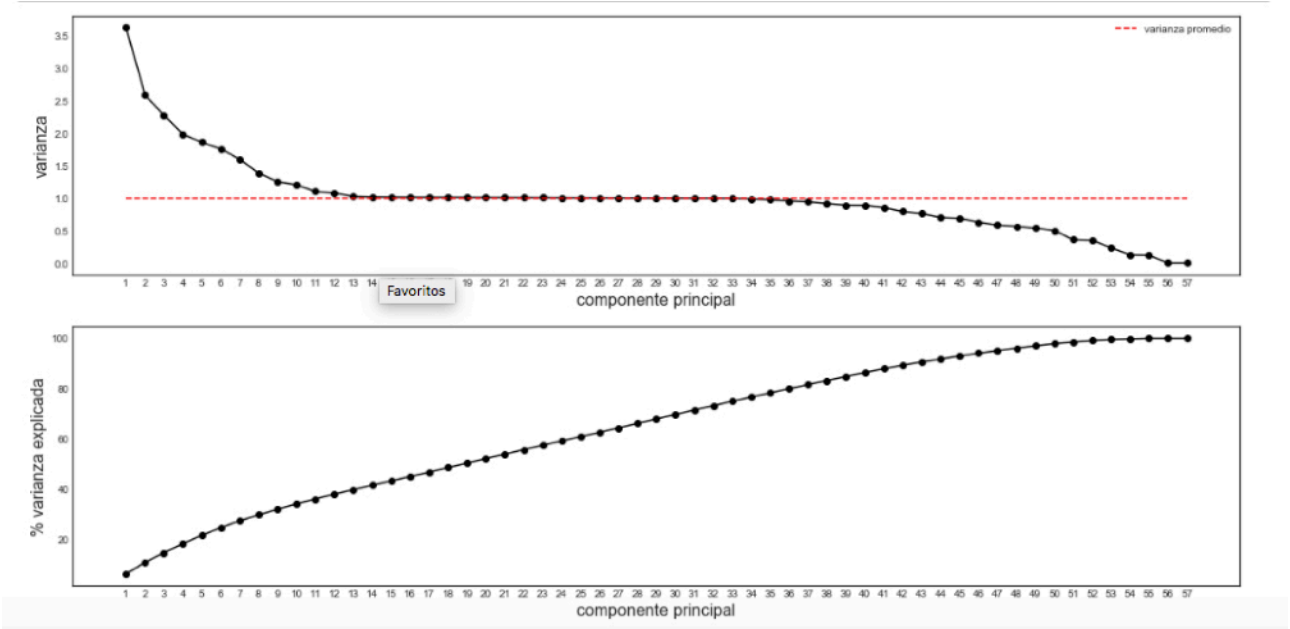
Con el método de wrapping que estudia en conjunto a los atributos, los seleccionados serían:

```
Elegidos:
['ps_ind_01' 'ps_ind_03' 'ps_ind_15' 'ps_reg_01' 'ps_reg_02' 'ps_reg_03'
 'ps_car_01_cat' 'ps_car_06_cat' 'ps_car_09_cat' 'ps_car_11_cat'
 'ps_car_11' 'ps_car_12' 'ps_car_13' 'ps_car_14' 'ps_car_15' 'ps_calc_01'
 'ps_calc_02' 'ps_calc_03' 'ps_calc_04' 'ps_calc_05' 'ps_calc_06'
 'ps_calc_07' 'ps_calc_08' 'ps_calc_09' 'ps_calc_10' 'ps_calc_11'
 'ps_calc_12' 'ps_calc_13' 'ps_calc_14']
```

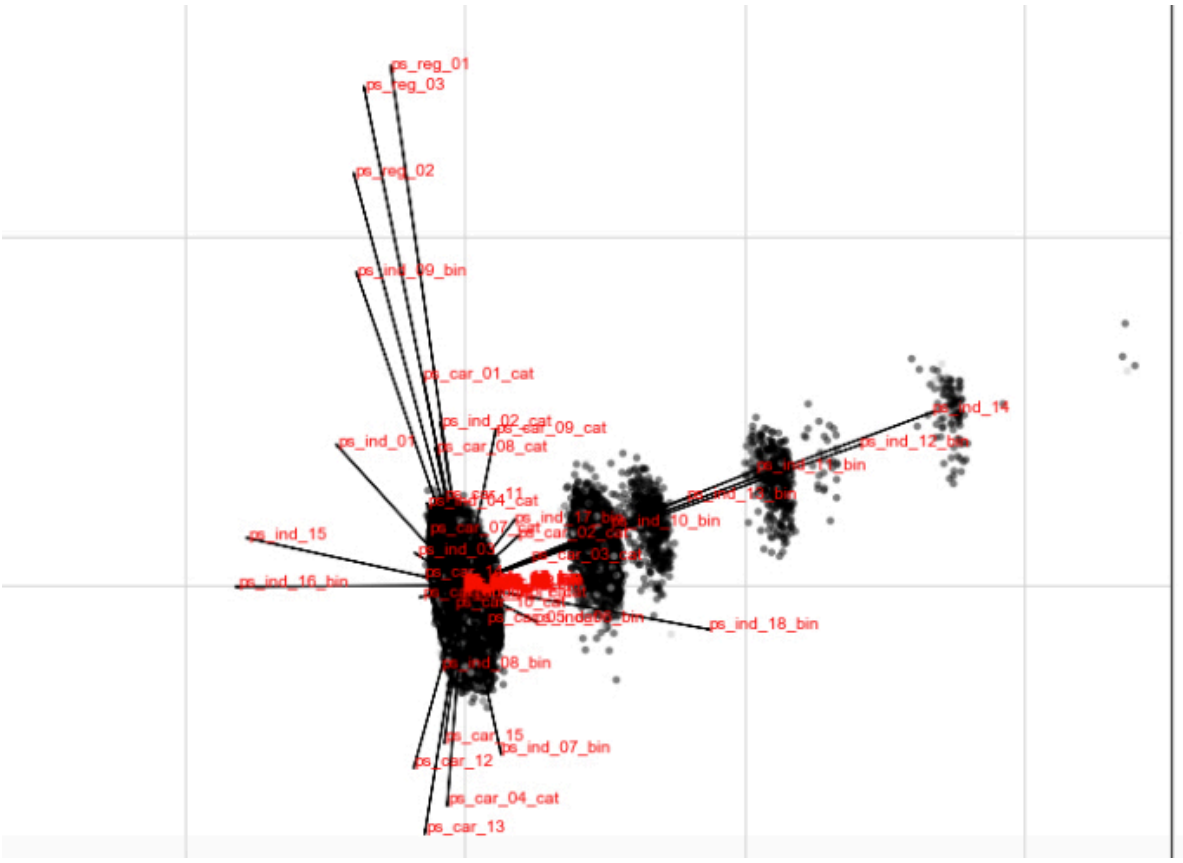
También se utiliza la técnica de PCA, de las componentes principales. En las siguientes gráficas se puede ver para cada componente principal el valor de cada una de varianza, y además la varianza acumulada a medida que vamos aumentando en componentes principales.

A la hora de elegir un número de elementos lo suyo es escoger un número que explique o contenga gran % de varianza (entre 75%-80%), o quedarse con las componentes principales que tienen varianza al menos 1%.

En este ejemplo nos quedaríamos aproximadamente con 30-35 atributos.



Luego visualizamos gráficamente algunos pares de componentes principales, además poniendo a cada atributo en el gráfico una flecha que con su dirección y tamaño indica la importancia que tiene en esa componente principal.



Una vez realizados todos estos procesos, decidimos eliminar según el método de mutual information, a los que menor puntuación tienen, y así nos quedamos aproximadamente con unos 30-35 atributos. Graficando las primeras componentes principales que son las que mayor varianza del conjunto de datos representan o contienen, vemos que los atributos que con esos métodos de filtrado tenían mejor score también son las que más valor dan a dichas componentes.

Nos quedamos con esos atributos y eliminamos el resto de los conjuntos de training y test:

```
print(X_train_final.shape)
print(X_test_final.shape)
```

```
(416648, 32)
(178564, 32)
```

3. Entrenamiento de un clasificador SVM y ajuste de parámetros.

Generamos samples de los conjuntos de training y test, ya que por problemas de capacidad no se pueden realizar ciertos análisis y ejecución de técnicas con todos los registros del dataset.

```
: # generamos samples de los datos de training ya que son muchos registros y da problemas de timeout
  sample_1 = X_train_final.sample(frac=0.01, replace=True)
  sample_2 = X_test_final.sample(frac=0.01, replace=True)
  sample_3 = clases_train.sample(frac=0.01, replace=True)
  sample_4 = clases_test.sample(frac=0.01, replace=True)
```

```
: print(sample_1.shape)
  print(sample_2.shape)
  print(sample_3.shape)
  print(sample_4.shape)
```

```
(4166, 32)
(1786, 32)
(4166,)
(1786,)
```

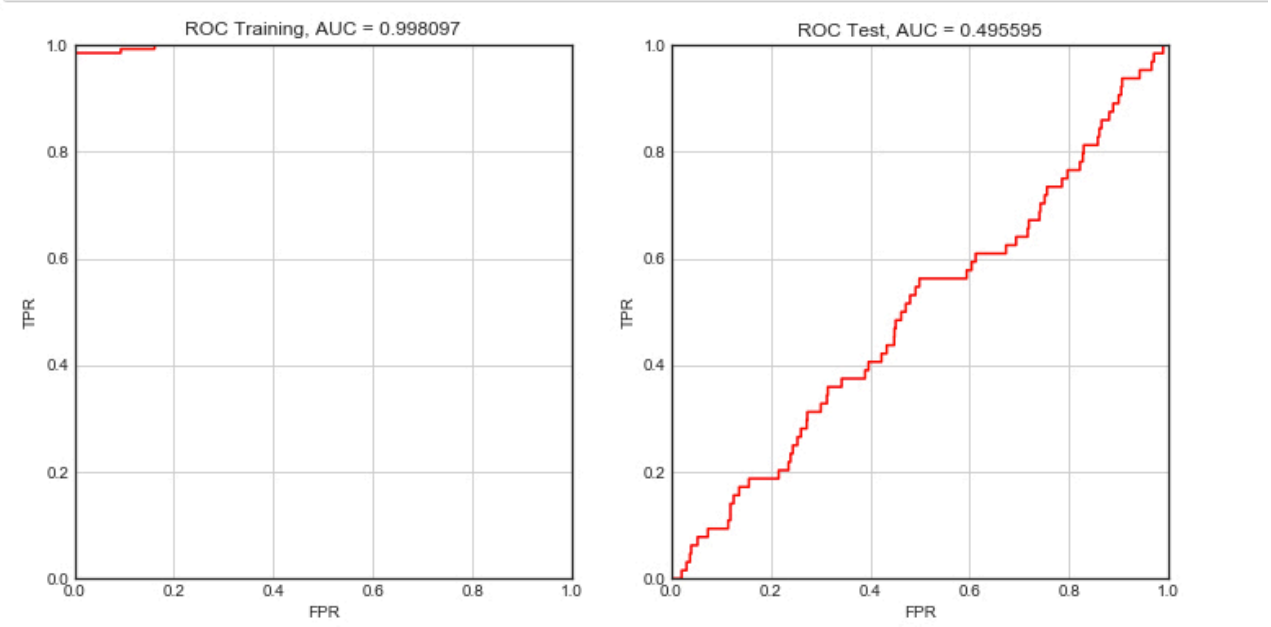
Una vez tenemos los samples ya se puede entrenar un modelo SVM y ver sus mejores parámetros.

```
Best score = 0.967835
Best model:
SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
  max_iter=-1, probability=True, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
Score in training set = 0.969755
Score in test set = 0.964166
```

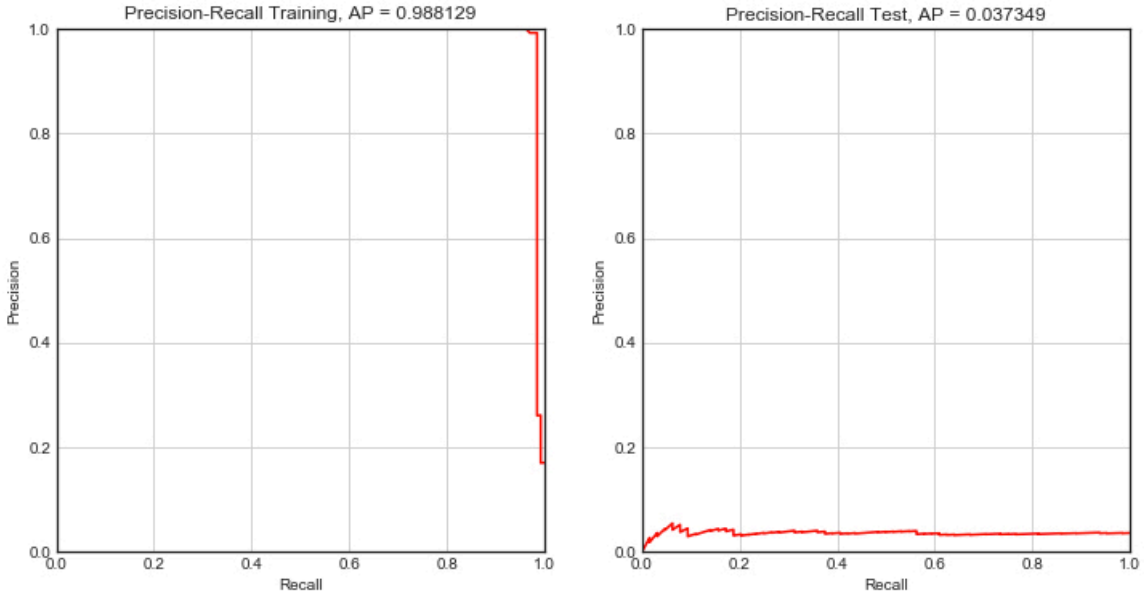
Matriz de confusión de SVM:

```
Training:
[[4032    0]
 [ 126    8]]
Test:
[[1722    0]
 [  64    0]]
```

Curvas ROC training y test con SVM:



Curvas Recall training y test con SVM:



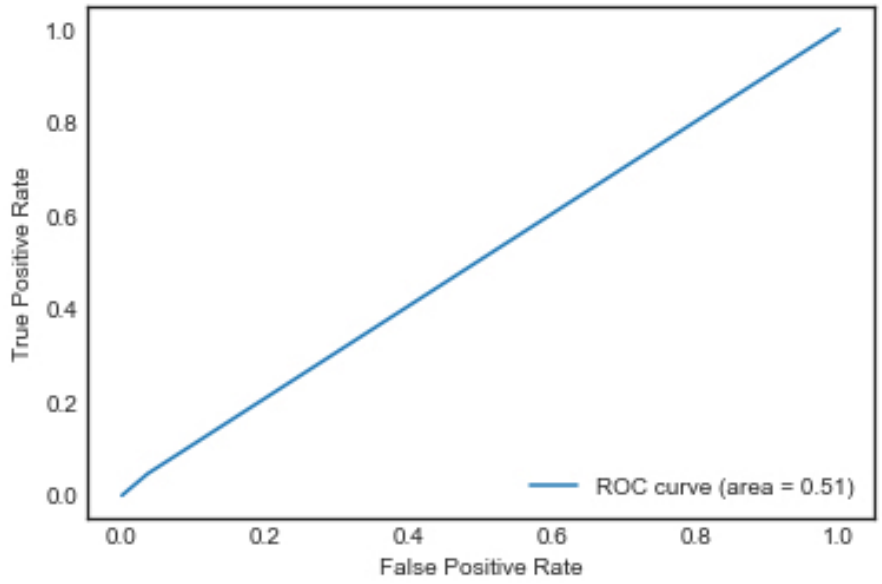
4. Otros clasificadores.

Se usan otros clasificadores como serían:

Arbol de decisión.

	precision	recall	f1-score	support
0	0.96	0.96	0.96	1722
1	0.05	0.05	0.05	64
avg / total	0.93	0.93	0.93	1786

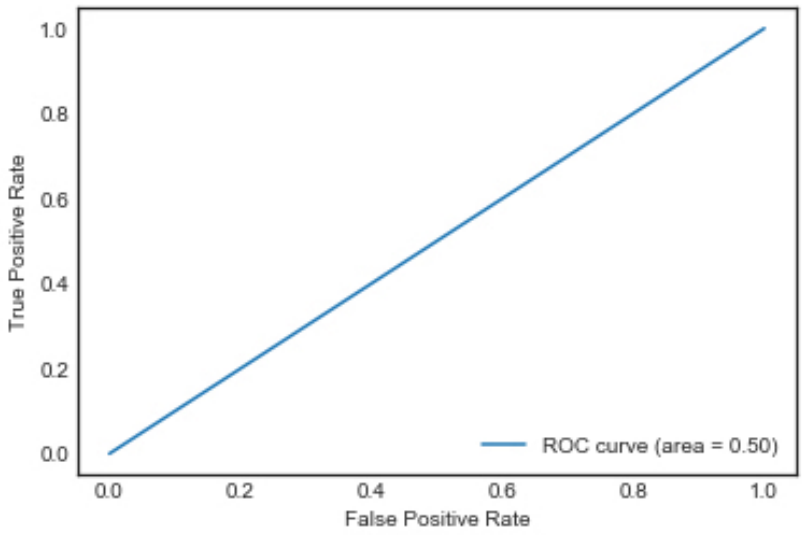
```
[[1660   62]
 [  61    3]]
```



Random Forest:

```
[[1722    0]
 [  64    0]]
```

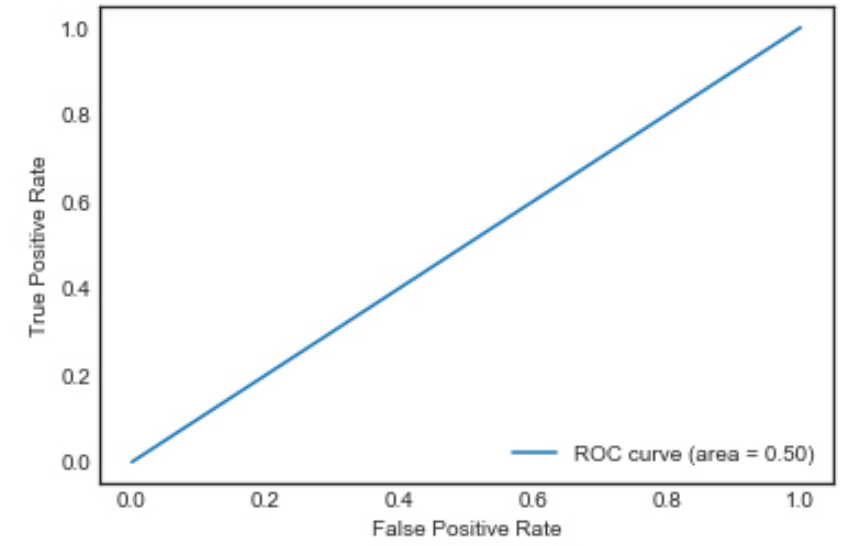
	precision	recall	f1-score	support
0	0.96	1.00	0.98	1722
1	0.00	0.00	0.00	64
avg / total	0.93	0.96	0.95	1786



Red Neuronal:

```
[[1722    0]
 [  64    0]]
```

	precision	recall	f1-score	support
0	0.96	1.00	0.98	1722
1	0.00	0.00	0.00	64
avg / total	0.93	0.96	0.95	1786



Viendo los resultados obtenidos en los dos puntos anteriores, de los distintos clasificadores vemos como los mejores resultados se obtienen con el uso de SVMs, así que usaremos esta técnica en el punto siguiente.

5. Validación con conjunto de test y conclusiones.

```
score_test = clf.score(X_test_std, sample_4.values)
print("Score in test set = %f" % (score_test))
```

```
Score in test set = 0.964166
```

```
preds_test = clf.predict(X_test_std)
acc_test = float(np.sum(preds_test == sample_4)) / num_test_sample
print ("Acc. test = %f" % acc_test)
```

```
Acc. test = 0.964166
```

Después de analizar los datos y aplicar distintas técnicas de filtrado, selección y extracción de características, decidimos quedarnos con un grupo de atributos, entre las 30 y 35 características, que pensamos viendo los resultados de las técnicas aplicadas que eran los que tenían mayor importancia a la hora de predecir el resultado esperado, ya que el dataset contiene un número bastante elevado de atributos y además de que algunos no parecen aportar gran valor al problema, mientras más atributos se tengan más suelen verse penalizados los modelos de predicción.

Una vez tenemos el conjunto de datos de entrenamiento y test sin los atributos que no queremos tener en cuenta, ya podemos aplicar distintos clasificadores y modelos y estudiar cual parece darnos mejores resultados para quedarnos con él. Hemos tenido el problema de que el número de registros es muy elevado y con los ordenadores personales que tenemos nos da problemas y no es capaz de obtener el resultado con el clasificador SVM. Así que hemos tenido que hacer unos samples con un número pequeño de registros para poder aplicar el resto de código, con lo que suponemos que puede ser un número insuficiente de datos y por tanto los resultados pueden verse afectados por ello.

El clasificador que mejor score parece obtener es el SVM, así que nos quedamos con él.