Manuel_Pasieka-Wisconsin_Breast_Cancer

February 20, 2019

1 Introducción

El enfoque de esta práctica crear un clasificador para el dataset 'Wisconsis Breast Canceer' Dataset utilizando un 'isolation forest'.

Los datos están disponibles en: https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

2 Resultados

El mejor clasificador consigue una f1 score de 0.68 par los casos benévolos y 0.81 para os casos malignas.

```
In [1]: import pandas as pd
        import numpy as np
        import seaborn as sns
        import sklearn
        from sklearn.ensemble import IsolationForest
```

3 Análisis descriptivo

El dataset contiene 569 ejemplos de pruebas histológicas con 357 ejemplos benévolo y 212 ejemplos malignas.

Hay 30 características numéricas, sin valores missing.

```
In [2]: T = pd.read_csv('data/data.csv')
        T.head()
```

```
Out[2]:          id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  smoothness
        0    842302         M        17.99         10.38          122.80     1001.0           0
        1    842517         M        20.57         17.77          132.90     1326.0           0
        2  84300903         M        19.69         21.25          130.00     1203.0           0
        3  84348301         M        11.42         20.38           77.58      386.1           0
        4  84358402         M        20.29         14.34          135.10     1297.0           0

           smoothness_worst  compactness_worst  concavity_worst  concave points_worst  symmetry
        0            0.1622             0.6656           0.7119                0.2654
        1            0.1238             0.1866           0.2416                0.1860
```

```
2          0.1444              0.4245           0.4504                0.2430
3          0.2098              0.8663           0.6869                0.2575
4          0.1374              0.2050           0.4000                0.1625
```

In [3]:
```
T.describe().T
```

Out[3]:

	count	mean	std	min	25%
id	569.0	3.037183e+07	1.250206e+08	8670.000000	869218.000000
radius_mean	569.0	1.412729e+01	3.524049e+00	6.981000	11.700000
texture_mean	569.0	1.928965e+01	4.301036e+00	9.710000	16.170000
perimeter_mean	569.0	9.196903e+01	2.429898e+01	43.790000	75.170000
area_mean	569.0	6.548891e+02	3.519141e+02	143.500000	420.300000
smoothness_mean	569.0	9.636028e-02	1.406413e-02	0.052630	0.086370
compactness_mean	569.0	1.043410e-01	5.281276e-02	0.019380	0.064920
concavity_mean	569.0	8.879932e-02	7.971981e-02	0.000000	0.029560
concave points_mean	569.0	4.891915e-02	3.880284e-02	0.000000	0.020310
symmetry_mean	569.0	1.811619e-01	2.741428e-02	0.106000	0.161900
fractal_dimension_mean	569.0	6.279761e-02	7.060363e-03	0.049960	0.057700
radius_se	569.0	4.051721e-01	2.773127e-01	0.111500	0.232400
texture_se	569.0	1.216853e+00	5.516484e-01	0.360200	0.833900
perimeter_se	569.0	2.866059e+00	2.021855e+00	0.757000	1.606000
area_se	569.0	4.033708e+01	4.549101e+01	6.802000	17.850000
smoothness_se	569.0	7.040979e-03	3.002518e-03	0.001713	0.005169
compactness_se	569.0	2.547814e-02	1.790818e-02	0.002252	0.013080
concavity_se	569.0	3.189372e-02	3.018606e-02	0.000000	0.015090
concave points_se	569.0	1.179614e-02	6.170285e-03	0.000000	0.007638
symmetry_se	569.0	2.054230e-02	8.266372e-03	0.007882	0.015160
fractal_dimension_se	569.0	3.794904e-03	2.646071e-03	0.000895	0.002248
radius_worst	569.0	1.626919e+01	4.833242e+00	7.930000	13.010000
texture_worst	569.0	2.567722e+01	6.146258e+00	12.020000	21.080000
perimeter_worst	569.0	1.072612e+02	3.360254e+01	50.410000	84.110000
area_worst	569.0	8.805831e+02	5.693570e+02	185.200000	515.300000
smoothness_worst	569.0	1.323686e-01	2.283243e-02	0.071170	0.116600
compactness_worst	569.0	2.542650e-01	1.573365e-01	0.027290	0.147200
concavity_worst	569.0	2.721885e-01	2.086243e-01	0.000000	0.114500
concave points_worst	569.0	1.146062e-01	6.573234e-02	0.000000	0.064930
symmetry_worst	569.0	2.900756e-01	6.186747e-02	0.156500	0.250400
fractal_dimension_worst	569.0	8.394582e-02	1.806127e-02	0.055040	0.071460

In [4]:
```
# How balanced are the target clases
T['diagnosis'].value_counts()
```

Out[4]:
```
B    357
M    212
Name: diagnosis, dtype: int64
```

In [5]:
```
# Are there any missing datapoints?
T.isna().sum()
```

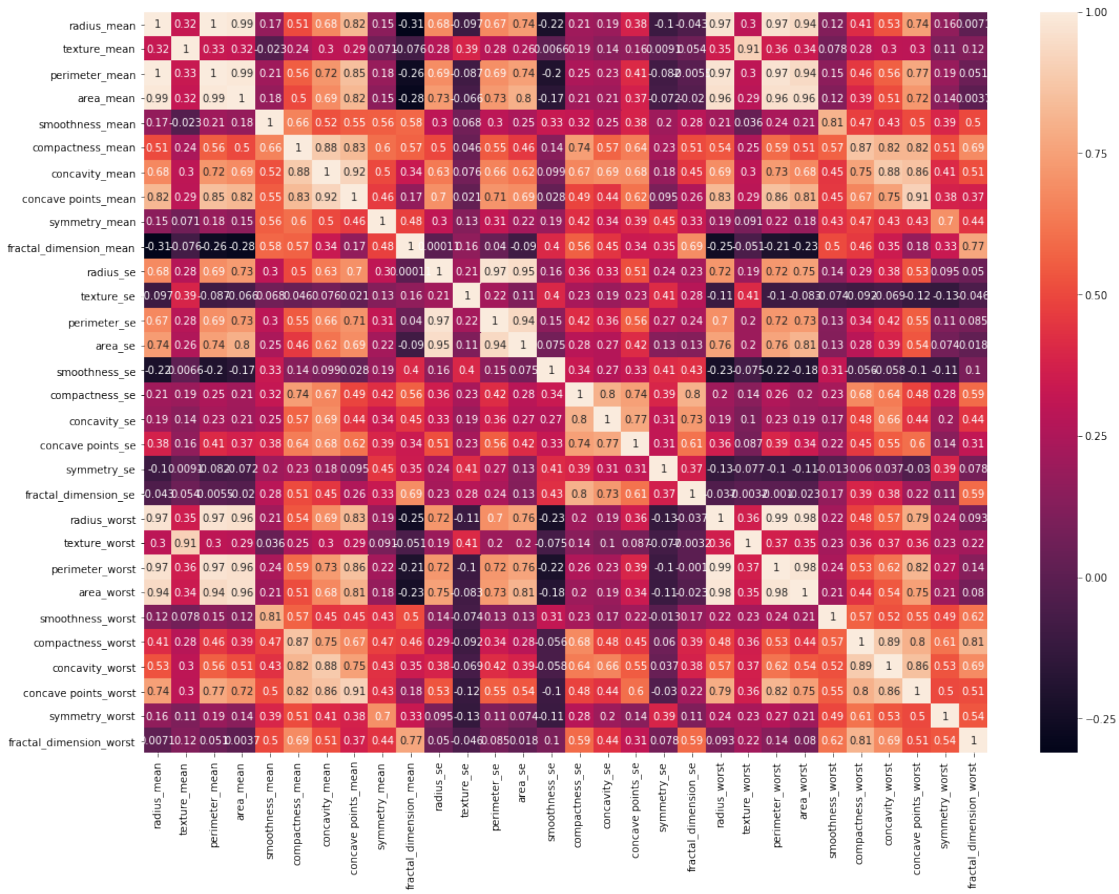
Out[5]:
```
id                          0
diagnosis                   0
radius_mean                 0
texture_mean                0
perimeter_mean              0
area_mean                   0
smoothness_mean             0
compactness_mean            0
concavity_mean              0
concave points_mean         0
symmetry_mean               0
fractal_dimension_mean      0
radius_se                   0
texture_se                  0
perimeter_se                0
area_se                     0
smoothness_se               0
compactness_se              0
concavity_se                0
concave points_se           0
symmetry_se                 0
fractal_dimension_se        0
radius_worst                0
texture_worst               0
perimeter_worst             0
area_worst                  0
smoothness_worst            0
compactness_worst           0
concavity_worst             0
concave points_worst        0
symmetry_worst              0
fractal_dimension_worst     0
dtype: int64
```

4 Preparación de datos

Cambiar el orden de filas al azar.
Quitar la columna 'id'
Primero calcular el valor standardizado para cada característica.

In [6]:
```
R = T.sample(frac=1)
R.drop('id', axis=1, inplace=True)
Y = R['diagnosis']
X = R.drop('diagnosis', axis=1)
```

In [7]:
```
plt.figure(figsize=(18,13))
sns.heatmap(X.corr(), annot=True)
plt.show()
```



4.1 Quitar cárácteristicas con una correlación alta

Quitar cárácteristicas con una correlación alta ha creato resultados peores, asi he decido <> hacerlo.

El código aqui esta para documentar el intento

`highly_correlated = ['id', 'radius_mean', 'perimeter_mean', 'radius_worst', 'texture_worst', 'area_worst', 'perimeter_worst', 'area_se', 'radius_se', 'compactness_worst', 'concavity_worst', 'concave points_worst', 'smoothness_worst', 'fractal_dimension_se', 'concavity_se', 'concave points_se', 'concavity_mean', 'concave points_mean'] X.drop(highly_correlated, axis=1, inplace=True) plt.figure(figsize=(18,13)) sns.heatmap(X.corr(), annot=True) plt.show()`

```
In [8]: nX = (X - X.mean())/X.std()
        nX.head()
```

Out[8]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactnes
40	-0.195029	0.532512	-0.238242	-0.261112	-1.048076	-0.8
212	3.967796	-0.190570	3.972634	5.240230	1.268455	0.8
287	-0.351099	-1.434456	-0.414792	-0.394952	-1.906288	-1.2

```
314    -1.569301     -0.160345       -1.558873  -1.232372         0.784956          -0.8
459    -1.240701      2.071676       -1.246515  -1.034312        -1.174640          -1.0

     compactness_worst  concavity_worst  concave points_worst  symmetry_worst  fractal_
40           -0.317568        -0.305278             -0.051820        0.150716
212          -0.652519         0.229655              0.682979       -2.024902
287          -0.887048        -0.736197             -0.927188       -0.956489
314          -1.122404        -1.304683             -1.743529        0.389937
459          -0.911200        -0.960044             -1.003254       -0.937093
```

5 Creat un modelo

```
In [9]: RANDOM_STATE = 42*42*42

        # Generate a -1, 1 version of Y
        y = Y.replace(['B', 'M'], [1, -1])

        # Calculate the ratio of B to M samples (aka contamination)
        c = Y.value_counts()
        ratio = c[1]/(c[0]+c[1])
```

```
In [10]: from sklearn.model_selection import GridSearchCV
         from sklearn.metrics import classification_report
```

```
In [11]: tuned_parameters = [{'n_estimators': [10, 100, 200, 300, 400, 500],
                              'max_samples': ['auto', 0.5, 0.8, 1.0],
                              'max_features': [0.1, 0.5, 0.8, 1.0]}]
         clf = GridSearchCV(IsolationForest(behaviour="new", random_state=RANDOM_STATE, contam
                            tuned_parameters, cv=5, scoring='f1', n_jobs=-1)
         clf.fit(nX, y)
```

```
/Users/manuel.pasieka/anaconda3/envs/py3/lib/python3.6/site-packages/sklearn/model_selection/_s
  DeprecationWarning)
```

```
Out[11]: GridSearchCV(cv=5, error_score='raise-deprecating',
                estimator=IsolationForest(behaviour='new', bootstrap=False,
                 contamination=0.37258347978910367, max_features=1.0,
                 max_samples='auto', n_estimators=100, n_jobs=None,
                 random_state=74088, verbose=0),
                fit_params=None, iid='warn', n_jobs=-1,
                param_grid=[{'n_estimators': [10, 100, 200, 300, 400, 500], 'max_samples': ['au
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring='f1', verbose=0)
```

```
In [12]: clf.best_estimator_
```

```
Out[12]: IsolationForest(behaviour='new', bootstrap=False,
                 contamination=0.37258347978910367, max_features=0.1,
```

```
                max_samples=1.0, n_estimators=100, n_jobs=None, random_state=74088,
                verbose=0)
```

In [13]:
```
best = clf.best_estimator_
p = best.predict(nX)
print(classification_report(y, p))
```

```
              precision    recall  f1-score   support

          -1       0.68      0.68      0.68       212
           1       0.81      0.81      0.81       357

   micro avg       0.76      0.76      0.76       569
   macro avg       0.74      0.74      0.74       569
weighted avg       0.76      0.76      0.76       569
```

In [14]:
```
# Confusion matrix
from sklearn.metrics import confusion_matrix
sns.heatmap(confusion_matrix(y, p), annot=True, fmt='2d',
            xticklabels=['B', 'M'], yticklabels=['B', 'M'])
plt.show()
```