Cancer model

July 26, 2020

Hello, my name is Oriol-Boris Monjo Farré, I did a short code to create a classification model. I went trough different phases:

```
1-Importing & Cleaning Data
2-Feature Selection
3-Model Selection
4-Model Creation
```

I will be explaining every step I make, I hope it can be useful.

My main objective is to be able to predict with more than a 95% of accuracy whether a patient has cancer or not with the lowest number of variables possible.

The original dataset has 11 variables and I would like to know if someone is ill with just 4 or less features.

I start importing the libraries

[1]:
```
import pandas as pd
import numpy as np
```

1 Importing & Cleaning Data

I import and clean the data.

[2]:
```
df=pd.read_csv("breast-cancer-wisconsin.data.txt",header=None)
df.columns=["Sample code number","Clump Thickness", "Uniformity of Cell Size",␣
 ↪"Uniformity of Cell Shape", "Marginal Adhesion", "Single Epithelial Cell␣
 ↪Size", "Bare Nuclei", "Bland Chromatin", "Normal Nucleoli", "Mitoses",␣
 ↪"Class"]
```

I now take a look to the different features available and I see that there is a ID column Since this will just confuse the classification model, this will be the fist variable to be discarded.

[3]:
```
print(df.head())
df=df.drop(["Sample code number"],axis=1)

print("")
print(df.columns)
```

```
   Sample code number  Clump Thickness  Uniformity of Cell Size  \
0             1000025                5                        1
1             1002945                5                        4
2             1015425                3                        1
3             1016277                6                        8
4             1017023                4                        1

   Uniformity of Cell Shape  Marginal Adhesion  Single Epithelial Cell Size  \
0                         1                  1                            2
1                         4                  5                            7
2                         1                  1                            2
3                         8                  1                            3
4                         1                  3                            2

  Bare Nuclei  Bland Chromatin  Normal Nucleoli  Mitoses  Class
0           1                3                1        1      2
1          10                3                2        1      2
2           2                3                1        1      2
3           4                3                7        1      2
4           1                3                1        1      2

Index(['Clump Thickness', 'Uniformity of Cell Size',
       'Uniformity of Cell Shape', 'Marginal Adhesion',
       'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromatin',
       'Normal Nucleoli', 'Mitoses', 'Class'],
      dtype='object')
```

After a closer look, I realized there are interrogation signs in the column "Bare Nuclei", I assume that it means there is no data for these registers. There are several references with the interrogation sign, due to the high number of registersm, I preferer not to eliminate those lines.

Instead I will substitute it for -99999 number. I do this because in a classification model, such a negative number will have a non significant impact on the result, so, it won't be a problem.

[4]:
```
df[df["Bare Nuclei"] == "?"]
df=df.replace("?", -99999)
df["Bare Nuclei"] = df["Bare Nuclei"].astype(int)
```

Now I will divide the predictive features and the variable to predict. In the information document of this data frame it says that the "Class" variable (the one that I want to predict) has a 2 in the register where the person doesn't have cancer and 4 in the ones he/she has.

Just because of a personal preference I will change all the 2 for 0 and all the 4 for 1

[5]:
```
df["Class"]=df["Class"].replace(2,0)
df["Class"]=df["Class"].replace(4,1)
```

I proceed assigning the column "Class" to the Y variable and all the other features to the X.

[6]:
```
Y=df["Class"]
X=df.drop(["Class"],axis=1)
```

2 Features Selection

I will start applying some techniques to reduce the number of variables.

The fist one I will apply is the variance threshold analysis. This function will determine if there is any column that does not have a variance strong enough to be a good predictive variable.

3 — VARIANCE TRESHOLD —

[7]:
```
from sklearn.feature_selection import VarianceThreshold

var_th = VarianceThreshold(threshold = 0.6)
x_var = pd.DataFrame(var_th.fit_transform(X))
print("Number of variables", x_var.shape[1])
print("List of varibles", np.asarray(list(X))[var_th.get_support()])
x_var.columns=['Clump Thickness', 'Uniformity of Cell Size', 'Uniformity of␣
 ↪Cell Shape','Marginal Adhesion', 'Single Epithelial Cell Size','Bare␣
 ↪Nuclei','Bland Chromatin','Normal Nucleoli','Mitoses']

X=x_var
```

```
Number of variables 9
List of varibles ['Clump Thickness' 'Uniformity of Cell Size' 'Uniformity of
Cell Shape'
 'Marginal Adhesion' 'Single Epithelial Cell Size' 'Bare Nuclei'
 'Bland Chromatin' 'Normal Nucleoli' 'Mitoses']
```

It seems that all the variables have a great level of variance, in fact, it is possible to confirm this fact by looking at the different features.

Now I will proceed to apply the function SelectKBest with the function f_classif. These functions will apply to all the features the same test and select just 5, the ones that obtain the best results.

4 — SelectKBest —

[8]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif


var_sk = SelectKBest(f_classif, k = 5)
x_sk = pd.DataFrame(var_sk.fit_transform(X, Y))

print("Number of variables", x_sk.shape[1])
```

```
print("List of varibles", np.asarray(list(X))[var_sk.get_support()])

x_sk.columns=np.asarray(list(X))[var_sk.get_support()]
X=x_sk
```

```
Number of variables 5
List of varibles ['Clump Thickness' 'Uniformity of Cell Size' 'Uniformity of
Cell Shape'
 'Bland Chromatin' 'Normal Nucleoli']
```

The last technique that I will apply, is the VIF analysis. This will reveal how correlated the predictive variables are.

If a certain variable obtains a number greater than 5, it means that by creating a linear regression with the rest of variables it is possible to predict that feature. To create a model that contains repeated information can make it worst and because of that, if a feature obtains more than 5, I will delete the variable.

5 — VIF ANALYSIS —

[9]:
```
from sklearn.linear_model import LinearRegression

def calculateVIF(data):
    features = list(data.columns)
    num_features = len(features)

    model = LinearRegression()

    result = pd.DataFrame(index = ['VIF'], columns = features)
    result = result.fillna(0)

    for ite in range(num_features):
        x_features = features[:]
        y_featue = features[ite]
        x_features.remove(y_featue)

        x = data[x_features]
        y = data[y_featue]

        model.fit(data[x_features], data[y_featue])

        result[y_featue] = 1/(1 - model.score(data[x_features], data[y_featue]))

    return result


pd.set_option("display.max_columns",20)
print("\n" *3,">>> VIF first try <<<","\n")
```

```
print(calculateVIF(X))
```

```
 >>> VIF first try <<<

     Clump Thickness  Uniformity of Cell Size  Uniformity of Cell Shape  \
VIF         1.824833                 6.482377                  6.229411

     Bland Chromatin  Normal Nucleoli
VIF         2.572217         2.337489
```

It seems that the "Uniformity of Cell Size" shows a high level of correlation with the other data, I will eliminate this variable.

[10]:
```
X=X.drop(["Uniformity of Cell Size"],axis=1)
```

Now I will make this analysis again to see if the problem has ben solved.

[11]:
```
print("\n" *3,">>> VIF second try <<<","\n")
print(calculateVIF(X))
```

```
 >>> VIF second try <<<

     Clump Thickness  Uniformity of Cell Shape  Bland Chromatin  \
VIF         1.802849                  3.170464         2.416836

     Normal Nucleoli
VIF         2.279781
```

All the variables have obtained results under 5, so we can continue.

Before proceeding to find a good model, I want to clarify that at this point I could scale of normalize the data, however, considering that all the data seem to be in the same scale, I will continue without transforming it.

6 Model Selection

In order to find the right model I will use code that contain the following parts: 1-A dictionary with the different models and the parameters that I use the most for these models.

2-A for loop that contains the function "GridSearchCV" with the Cross-validation parameter set
That means that the data will be divided in 5 parts, using one part for testing and the oth
for training the model. This procedure will be repeated 5 times and each time one differen
will be the one used for testing.

Finally, the score for every model will be displayed with the parameters that the model had when it obtained the best results.

[12]:
```
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV



model_params = {
    'svm': {
        'model': svm.SVC(),
        'params' : {
            'C': [1,10,20],
            'kernel': ['rbf','linear'],
            'gamma' :['scale','auto']
        }
    },
    'random_forest': {
        'model': RandomForestClassifier(),
        'params' : {
            'n_estimators': [5,10,50,100,200],
            'criterion':['gini','entropy']
        }
    },
    'logistic_regression' : {
        'model': LogisticRegression(solver='liblinear',multi_class='auto'),
        'params': {
            'C': [1,5,10]
        }
    },
    'KNeighbors' : {
        'model': KNeighborsClassifier(),
        'params': {
            'n_neighbors': [5,10,20],
            'p': [1,2],
            'weights':['uniform','distance']
        }
    }
}
```

[13]:
```
scores = []
for model_name, mp in model_params.items():
    clf =  GridSearchCV(mp['model'], mp['params'], cv=5,␣
 ↪return_train_score=False)
```

```
    clf.fit(X, Y)
    scores.append({
        'model': model_name,
        'best_score': clf.best_score_,
        'best_params': clf.best_params_
    })

Modelos = pd.DataFrame(scores,columns=['model','best_score','best_params'])
pd.set_option("display.max_columns",20)
print(Modelos.sort_values(by="best_score",ascending=False))
```

```
                model  best_score  \
3           KNeighbors    0.961398
0                  svm    0.958530
1        random_forest    0.955673
2  logistic_regression    0.955673

                                         best_params
3  {'n_neighbors': 10, 'p': 2, 'weights': 'distan…
0        {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}
1          {'criterion': 'gini', 'n_estimators': 50}
2                                          {'C': 1}
```

Now that the code has shown the best possible models, I will proceed to create it. However, I should first highlight that if I apply the results that I obtained with the code without taking a closer look, the created model may be overfitted.

In order to find if this is the case, instead of using cross-validation to directly find the best model, I will divide the data using the "train_test_split" and different metrics to find if the different models have overfitting or not.

7 Model Creation

[21]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪confusion_matrix, precision_score, recall_score
```

I divide the data in training and test.

[22]:
```
X_train ,X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.3)
```

7.1 KNeighbors

KNeighbors The results obtained from the code suggest that I should use the following parameters: KNeighborsClassifier(n_neighbors= 10, p= 2, weights= "distance")

However, after taking a look on the metrics I realized that the model was overfitted In order to improve the situation, I have increased the number of n_neighbors and I changed the weights to "uniform".

[23]:
```
Mod_Knn=KNeighborsClassifier(n_neighbors= 15, p= 2, weights= "uniform")
Mod_Knn.fit(X_train, Y_train)

print("\n","------- TRAIN KNeighbors -------")
Pred_train_Knn=Mod_Knn.predict(X_train)
print(confusion_matrix(Y_train,Pred_train_Knn),"<-Confusion matrix")
print("Precision: ",accuracy_score(Y_train,Pred_train_Knn))
print("Exactitud: ", precision_score(Y_train,Pred_train_Knn))
print("Exhaustividad: ",recall_score(Y_train,Pred_train_Knn))
print(classification_report(Y_train,Pred_train_Knn))


print("\n","------- TEST KNeighbors -------")
Pred_test_Knn=Mod_Knn.predict(X_test)
print(confusion_matrix(Y_test,Pred_test_Knn),"<-Confusion matrix")
print("Precision: ",accuracy_score(Y_test,Pred_test_Knn))
print("Exactitud: ", precision_score(Y_test,Pred_test_Knn))
print("Exhaustividad: ",recall_score(Y_test,Pred_test_Knn))
print(classification_report(Y_test,Pred_test_Knn))
```

```
 ------- TRAIN KNeighbors -------
[[307  12]
 [ 10 160]] <-Confusion matrix
Precision:  0.9550102249488752
Exactitud:  0.9302325581395349
Exhaustividad:  0.9411764705882353
              precision    recall  f1-score   support

           0       0.97      0.96      0.97       319
           1       0.93      0.94      0.94       170

    accuracy                           0.96       489
   macro avg       0.95      0.95      0.95       489
weighted avg       0.96      0.96      0.96       489


 ------- TEST KNeighbors -------
[[136   3]
 [  5  66]] <-Confusion matrix
Precision:  0.9619047619047619
Exactitud:  0.9565217391304348
Exhaustividad:  0.9295774647887324
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       139
           1       0.96      0.93      0.94        71
```

```
    accuracy                           0.96       210
   macro avg       0.96      0.95      0.96       210
weighted avg       0.96      0.96      0.96       210
```

7.2 SVC

[24]:
```
Mod_SVC=svm.SVC(C= 1, gamma= 'scale', kernel= 'rbf')
Mod_SVC.fit(X_train, Y_train)


print("\n","------- TRAIN SVC -------")
Pred_train_SVC=Mod_SVC.predict(X_train)
print(confusion_matrix(Y_train,Pred_train_SVC),"<-Confusion matrix")
print("Precision: ",accuracy_score(Y_train,Pred_train_SVC))
print("Exactitud: ", precision_score(Y_train,Pred_train_SVC))
print("Exhaustividad: ",recall_score(Y_train,Pred_train_SVC))
print(classification_report(Y_train,Pred_train_SVC))


print("\n","------- TEST SVC -------")
Pred_test_SVC=Mod_SVC.predict(X_test)
print(confusion_matrix(Y_test,Pred_test_SVC),"<-Confusion matrix")
print("Precision: ",accuracy_score(Y_test,Pred_test_SVC))
print("Exactitud: ", precision_score(Y_test,Pred_test_SVC))
print("Exhaustividad: ",recall_score(Y_test,Pred_test_SVC))
print(classification_report(Y_test,Pred_test_SVC))
```

```
 ------- TRAIN SVC -------
[[305  14]
 [  9 161]] <-Confusion matrix
Precision:  0.9529652351738241
Exactitud:  0.92
Exhaustividad:  0.9470588235294117
              precision    recall  f1-score   support

           0       0.97      0.96      0.96       319
           1       0.92      0.95      0.93       170

    accuracy                           0.95       489
   macro avg       0.95      0.95      0.95       489
weighted avg       0.95      0.95      0.95       489


 ------- TEST SVC -------
[[136   3]
```

```
 [  5  66]] <-Confusion matrix
Precision:  0.9619047619047619
Exactitud:  0.9565217391304348
Exhaustividad:  0.9295774647887324
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       139
           1       0.96      0.93      0.94        71

    accuracy                           0.96       210
   macro avg       0.96      0.95      0.96       210
weighted avg       0.96      0.96      0.96       210
```

Random Forest The results obtained from the code suggest that I should use the following parameters: RandomForestClassifier(criterion= 'gini', n_estimators= 5)

But the metrics showed that the model was overfitted. I solved the problem by increasing the number of estimators, changing the max_features parameter to "log2" and, the most important thing limiting the extension of the model setting the min_samples_split to 4

[25]:
```
Mod_Ran=RandomForestClassifier(criterion= 'gini',␣
 ↪max_features="log2",n_estimators= 100, min_samples_split=40)
Mod_Ran.fit(X_train, Y_train)

print("\n","------- TRAIN Random Forest -------")
Pred_train_Ran=Mod_Ran.predict(X_train)
print(confusion_matrix(Y_train,Pred_train_Ran),"<-Confusion matrix")
print("Precision: ",accuracy_score(Y_train,Pred_train_Ran))
print("Exactitud: ", precision_score(Y_train,Pred_train_Ran))
print("Exhaustividad: ",recall_score(Y_train,Pred_train_Ran))
print(classification_report(Y_train,Pred_train_Ran))


print("\n","------- TEST Random Forest -------")
Pred_test_Ran=Mod_Ran.predict(X_test)
print(confusion_matrix(Y_test,Pred_test_Ran),"<-Confusion matrix")
print("Precision: ",accuracy_score(Y_test,Pred_test_Ran))
print("Exactitud: ", precision_score(Y_test,Pred_test_Ran))
print("Exhaustividad: ",recall_score(Y_test,Pred_test_Ran))
print(classification_report(Y_test,Pred_test_Ran))
```

```
 ------- TRAIN Random Forest -------
[[307  12]
 [  5 165]] <-Confusion matrix
Precision:  0.9652351738241309
Exactitud:  0.9322033898305084
Exhaustividad:  0.9705882352941176
```

```
              precision    recall  f1-score   support

           0       0.98      0.96      0.97       319
           1       0.93      0.97      0.95       170

    accuracy                           0.97       489
   macro avg       0.96      0.97      0.96       489
weighted avg       0.97      0.97      0.97       489


 ------- TEST Random Forest -------
[[136   3]
 [  3  68]] <-Confusion matrix
Precision:  0.9714285714285714
Exactitud:  0.9577464788732394
Exhaustividad:  0.9577464788732394
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       139
           1       0.96      0.96      0.96        71

    accuracy                           0.97       210
   macro avg       0.97      0.97      0.97       210
weighted avg       0.97      0.97      0.97       210
```

7.3 Logistic regression

[26]:
```
Mod_log=LogisticRegression(solver='liblinear',multi_class='auto', C=1)
Mod_log.fit(X_train, Y_train)


print("\n","------- TRAIN logistic regession -------")
Pred_train_log=Mod_log.predict(X_train)
print(confusion_matrix(Y_train,Pred_train_log),"<-Confusion matrix")
print("Precision: ",accuracy_score(Y_train,Pred_train_log))
print("Exactitud: ", precision_score(Y_train,Pred_train_log))
print("Exhaustividad: ",recall_score(Y_train,Pred_train_log))
print(classification_report(Y_train,Pred_train_log))


print("\n","------- TEST logistic regession -------")
Pred_test_log=Mod_log.predict(X_test)
print(confusion_matrix(Y_test,Pred_test_log),"<-Confusion matrix")
print("Precision: ",accuracy_score(Y_test,Pred_test_log))
print("Exactitud: ", precision_score(Y_test,Pred_test_log))
print("Exhaustividad: ",recall_score(Y_test,Pred_test_log))
```

```
print(classification_report(Y_test,Pred_test_log))
```

```
 ------- TRAIN logistic regession -------
[[308  11]
 [ 11 159]] <-Confusion matrix
Precision:  0.9550102249488752
Exactitud:  0.9352941176470588
Exhaustividad:  0.9352941176470588
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       319
           1       0.94      0.94      0.94       170

    accuracy                           0.96       489
   macro avg       0.95      0.95      0.95       489
weighted avg       0.96      0.96      0.96       489


 ------- TEST logistic regession -------
[[138   1]
 [  6  65]] <-Confusion matrix
Precision:  0.9666666666666667
Exactitud:  0.9848484848484849
Exhaustividad:  0.9154929577464789
              precision    recall  f1-score   support

           0       0.96      0.99      0.98       139
           1       0.98      0.92      0.95        71

    accuracy                           0.97       210
   macro avg       0.97      0.95      0.96       210
weighted avg       0.97      0.97      0.97       210
```

Now that I have solved the overfitting problems, that the KNeighbors and the Random Forest models showed, I will use the cross-validation function to definitely determine the score of the different models that I obtained.

[27]:
```
from sklearn.model_selection import cross_val_score

Mod_Knn=KNeighborsClassifier(n_neighbors= 15, p= 2, weights= "uniform")
Mod_SVC=svm.SVC(C= 1, gamma= 'scale', kernel= 'rbf')
Mod_Ran=RandomForestClassifier(criterion= 'gini',␣
 ↪max_features="log2",n_estimators= 10, min_samples_split=20)
Mod_log=LogisticRegression(solver='liblinear',multi_class='auto', C=1)

print("Cross Validation KNeighbors:",cross_val_score(Mod_Knn, X, Y,cv=5).mean())
```

```
print("Cross Validation SVC:",cross_val_score(Mod_SVC, X, Y,cv=5).mean())
print("Cross Validation Random Forest:",cross_val_score(Mod_Ran, X, Y,cv=5).
 ↪mean())
print("Cross Validation logistic regession:",cross_val_score(Mod_log, X,␣
 ↪Y,cv=5).mean())
```

```
Cross Validation KNeighbors: 0.9571017471736896
Cross Validation SVC: 0.958530318602261
Cross Validation Random Forest: 0.9599588900308325
Cross Validation logistic regession: 0.9556731757451182
```

Conclusion: -All the models that I have obtained show a score higher than 95% using just -4 features out of the 11 variables that we have started with. -The features that we can use to predict cancer are the following ones: -"Clump Thickness", "Uniformity of Cell Shape", "Bland Chromatin" and "Normal Nucleoli"