Techniques of Artificial Intelligence Project :

Project 3 : Classification of benignant/malignant calcification in the context of breast cancer

SKAF Joey
Student number : 0595931

Introduction

Breast cancer is unfortunatly a common cancer that impacts women. According to the World Health Organization, in 2020, there were 2.3 million women diagnosed with breast cancer and 685 000 deaths globally. Nonethless by an early diagnostic of the disease, breast cancer can be treated and cured. Prediciting it by analysing symptoms is thus crucial. One way to do that is by analysing micro-calcification in women's breast : their shape and texture properties of individual micros allow to predict malignancy of a tumor. Thanks to Machine Learning techniques, given a dataset with different features of the malcro-calcification, we can try to predict if a women has cancer or not.

The Dataset

The dataset we have is composed of 3561 micro-calcifications of 96 patients that has/had cancer or not (specified in the last column with 0 : no cancer and 1 : cancer). Each micro-calcification has as first feature, the patient on who we observed it, then 150 different features measurable, and last but not least, a label indicating if the patient has/had cancer or not.

Presentation of the approach

We will try to solve this task in two steps :

1. Because there is a strong correlation between micro-calcification and breast cancer, we can say that having cancer implies having malignant macro-calcification but because we assume all micros have the same label, having breast cancer implies that all the macros are malignant. And the same for the equivalence here : no-breast cancer <=> all micros are benign. So, we will do a supervised learning, saying that label "breast cancer" is equivalent, in our problem, to "malignant/benign macro-calcification".
2. Given the learning algorithm we acquired in the first step, we can classify the micros and give them a label (even if the label is wrong), we do not have anymore the hypothesis of all micros having the same label. So we will add as a feature of the calcification, the column "malignant/benign macro-calcification", and based on that, we will do a supervised learning, with as the supervised label, the "breast cancer" column.

Presentation of the technique

We will try here to test 4 different models to achieve the classification :

1. the Logistic Regression algorithm on a dimensionally reduced dataset (we use the Principal Component Analysis method to do so). PCA can be a good solution
2. the Support Vector Machine algorithm
3. a Neural Network
4. a Neural Network on a dimensionally reduced dataset (i.e applying PCA on the dataset)

Because the dataset is unbalanced (see bellow), we will use as an error metric the F1 score that is adapted to unbalanced dataset by considering the harmonic mean of precision and recall error meatrics. Nevertheless, we need to take into account the false negatives case to choose the model. It corresponds to the case where the patient has/had cancer, but the model predicts it as "no cancer". Thus, we also calculate the number of false negatives case.

The Algorithm

Importing some librairies

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import pandas as pd

np.random.seed(42)
```

Loading the data

In [ ]:
```
#!!! A EXECUTER UNE FOIS !!!
raw_data = pd.read_excel("data.xlsx").to_numpy()
```

In [ ]:
```
print("Number of feature:",len(raw_data[0]))
print("Number of micro-calcification:", len(raw_data[:,0]))
```

```
Number of feature: 152
Number of micro-calcification: 3562
```

Separating our data from the label needed for supervised learning. We also discard the feature "patient" that won't be usefull in our study

In [ ]:
```
X, y = raw_data[:,1:(len(raw_data[0])-1)], raw_data[:,len(raw_data[0])-1:len(raw_data[0])]
```

Step 1

First, we split the data in equal proportion between label 1 and 0 to not biais our training set

In [ ]:
```
X_label_0,y_label_0 = X[:2020],y[:2020]
X_label_1,y_label_1 = X[2020:],y[2020:]

print(len(y_label_0))
print(len(y_label_1))
```

```
2020
1542
```

We see that the labels are unbalanced between 0s and 1s. We notice this unbalance as a potential biais for our dataset so we choose as an error metric the f1 score

We split into training-validating set and test set our data for label 0s and 1s separately

In [ ]:
```
X_train_0, X_test_0, y_train_0, y_test_0 = train_test_split(X_label_0, y_label_0, test_size=0.025)
print(len(X_train_0))
print(len(X_test_0))
```

```
1969
51
```

In [ ]:
```
X_train_1, X_test_1, y_train_1, y_test_1 = train_test_split(X_label_1, y_label_1, test_size=0.025)
print(len(X_train_1))
print(len(X_test_1))
```

```
1503
39
```

We merge the labels to have same proportion of 0s and 1s in training set and test set

In [ ]:
```
X_test,y_test = np.concatenate([X_test_0,X_test_1]),np.concatenate([y_test_0,y_test_1])
X_train, y_train= np.concatenate([X_train_0,X_train_1]),np.concatenate([y_train_0,y_train_1])

print(len(X_train))
print(len(X_test))
```

```
3472
90
```

We reshuffle everything to have a good dataset for cross-validation

In [ ]:
```
train_indexes = np.arange(len(X_train))
np.random.shuffle(train_indexes)
X_train = X_train[train_indexes]
y_train = y_train[train_indexes]

test_indexes = np.arange(len(X_test))
np.random.shuffle(test_indexes)
X_test = X_test[test_indexes]
y_test = y_test[test_indexes]
```

We import the tools to center and reduce the data, plus the PCA for the first and the fourth techniques

In [ ]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

We center and reduce our data

In [ ]:
```
scaler = StandardScaler()
scaler.fit(X_train)
X_standardized = scaler.transform(X_train)
X_standardized
```

Out[ ]:
```
array([[-1.04504505e-01, -2.33066898e-01, -3.54485248e-01, ...,
         3.71720470e-01, -2.32587732e-01,  1.34990179e+00],
       [-3.81408105e-02, -7.54843247e-02, -2.85978183e-01, ...,
        -1.08423004e+00, -2.74558684e-01, -7.31260297e-01],
       [ 2.56634790e-01, -1.00437840e+00, -1.07254141e+00, ...,
        -6.98094048e-01, -7.20579346e-01, -5.74988353e-01],
       ...,
       [ 6.84225334e-01,  1.21647855e-01, -3.36565768e-04, ...,
         1.00898502e-01,  1.00384145e+00, -2.01344891e-01],
       [ 6.26355939e-01,  6.42151512e-01,  4.16013453e-01, ...,
         1.40181842e+00,  1.15428551e+00,  1.13150543e+00],
       [ 1.61380349e+00,  2.68972629e+00, -1.49471390e+00, ...,
        -1.07335416e+00,  3.14766809e-01, -5.64328362e-01]])
```

We do dimensional reducing to our dataset through the PCA technique

In [ ]:
```
pca = PCA(0.9)
pca.fit(X_standardized)
X_transformed = pca.transform(X_standardized)
X_transformed
```

Out[ ]:
```
array([[-12.35865328,  -2.38248564,   4.48287003, ...,   1.12712305,
          0.46651949,  -0.28492952],
       [  7.14758454,  -2.73631845,  -1.2845036 , ...,  -0.52552937,
          0.07613967,   0.19432478],
       [ -7.11700355,  -3.57085625,   1.6380319 , ...,  -0.02668062,
          0.96359428,  -0.6575074 ],
       ...,
       [  0.28911274,   4.43477764,  -3.22315701, ...,  -0.26068796,
         -0.47930749,   0.23702105],
       [ -7.60899869,   3.37293436,  -0.455115  , ...,  -0.72704821,
          0.39280078,  -0.29810586],
       [ 19.866319  ,   0.35883619,   9.64466459, ...,   0.70139385,
         -1.58196588,  -0.27437916]])
```

We see how much information each axe found by the PCA has to summarize the data

In [ ]:
```
print(pca.explained_variance_ratio_)
```

```
[0.44787719 0.11487575 0.0559034  0.04192724 0.02899031 0.02650838
 0.02479033 0.0195765  0.01524052 0.01292839 0.01262256 0.01137507
 0.00980236 0.00885644 0.00811045 0.00782032 0.0064192  0.00623578
 0.00614093 0.00553254 0.00535789 0.00526537 0.00488269 0.00455471
 0.0044519  0.00429625]
```

We choose to do a cross-validation on our training set, regarding the small number of data that we have

In [ ]:
```
from sklearn.model_selection import KFold
from sklearn.metrics import f1_score

kf = KFold(16)
```

First model : the Logistic Regression

In [ ]:
```
from sklearn.linear_model import LogisticRegression

models_scores_val_log_reg = []


for train_indexes, val_indexes in kf.split(X_transformed):
    model_temp = LogisticRegression()
    model_temp.fit(X_transformed[train_indexes], y_train[train_indexes].ravel())
    y_pred_temp = model_temp.predict(X_transformed[val_indexes])
    f1_score_temp = f1_score(y_train[val_indexes],y_pred_temp)
    y_folds = y_train[val_indexes]
    score_false_negativ = 0
    nb_positivs = 0
    for i in range(len(y_train[val_indexes])):
        if y_folds[i][0]:
            nb_positivs+=1
            if not(y_pred_temp[i]):
                score_false_negativ+=1
    models_scores_val_log_reg.append([model_temp,f1_score_temp,score_false_negativ/nb_positivs])
```

The different models given by the cross validation

In [ ]:
```
models_scores_val_log_reg
```

Out[ ]:
```
[[LogisticRegression(), 0.7251461988304092, 0.3333333333333333],
 [LogisticRegression(), 0.6818181818181819, 0.4],
 [LogisticRegression(), 0.6593406593406594, 0.4],
 [LogisticRegression(), 0.6330935251798562, 0.42857142857142855],
 [LogisticRegression(), 0.601156069364162, 0.4090909090909091],
 [LogisticRegression(), 0.5750000000000001, 0.43209876543209874],
 [LogisticRegression(), 0.6741573033707865, 0.38144329896907214],
 [LogisticRegression(), 0.7272727272727272, 0.2967032967032967],
 [LogisticRegression(), 0.711864406779661, 0.37],
 [LogisticRegression(), 0.7395833333333333, 0.297029702970297],
 [LogisticRegression(), 0.6144578313253013, 0.4069767441860465],
 [LogisticRegression(), 0.632183908045977, 0.47115384615384615],
 [LogisticRegression(), 0.5986394557823129, 0.4883720930232558],
 [LogisticRegression(), 0.6198830409356725, 0.47],
 [LogisticRegression(), 0.7282051282051282, 0.3238095238095238],
 [LogisticRegression(), 0.6666666666666666, 0.3829787234042553]]
```

We choose the model with the best f1 score given the lowest number of false negativ

In [ ]:
```
model_log_rec_selected = models_scores_val_log_reg[-7][0]
print("F1 score:",models_scores_val_log_reg[-7][1])
print("Number of false negatives:",models_scores_val_log_reg[-7][2])
```

```
F1 score: 0.7395833333333333
Number of false negatives: 0.297029702970297
```

Second model, SVM

In [ ]:
```
from sklearn.svm import SVC

models_scores_val_SVC = []


for train_indexes, val_indexes in kf.split(X_standardized):
    model_temp = SVC(gamma='auto')
    model_temp.fit(X_standardized[train_indexes], y_train[train_indexes].ravel())
    y_pred_temp = model_temp.predict(X_standardized[val_indexes])
    f1_score_temp = f1_score(y_train[val_indexes],y_pred_temp)
    y_folds = y_train[val_indexes]
    score_false_negativ = 0
    nb_positivs=0
    for i in range(len(y_train[val_indexes])):
        if y_folds[i][0]:
            nb_positivs+=1
            if not(y_pred_temp[i]):
                score_false_negativ+=1
    models_scores_val_SVC.append([model_temp,f1_score_temp,score_false_negativ/nb_positivs])
```

In [ ]:
```
models_scores_val_SVC
```

Out[ ]:
```
[[SVC(gamma='auto'), 0.7560975609756097, 0.3333333333333333],
 [SVC(gamma='auto'), 0.6857142857142857, 0.4],
 [SVC(gamma='auto'), 0.7934782608695652, 0.27],
 [SVC(gamma='auto'), 0.6917293233082706, 0.4025974025974026],
 [SVC(gamma='auto'), 0.6583850931677019, 0.3977272727272727],
 [SVC(gamma='auto'), 0.6527777777777778, 0.41975308641975306],
 [SVC(gamma='auto'), 0.6863905325443788, 0.4020618556701031],
 [SVC(gamma='auto'), 0.7904191616766467, 0.27472527472527475],
 [SVC(gamma='auto'), 0.7251461988304093, 0.38],
 [SVC(gamma='auto'), 0.7675675675675676, 0.297029702970297],
 [SVC(gamma='auto'), 0.6712328767123289, 0.43023255813953487],
 [SVC(gamma='auto'), 0.6227544910179641, 0.5],
 [SVC(gamma='auto'), 0.6619718309859154, 0.45348837209302323],
 [SVC(gamma='auto'), 0.650887573964497, 0.45],
 [SVC(gamma='auto'), 0.6927374301675978, 0.4095238095238095],
 [SVC(gamma='auto'), 0.6625, 0.43617021276595747]]
```

In [ ]:
```
model_svm_selected = models_scores_val_SVC[2][0]
print("F1 score:",models_scores_val_SVC[2][1])
print("Number of false negatives:",models_scores_val_SVC[2][2])
```

```
F1 score: 0.7934782608695652
Number of false negatives: 0.27
```

Third model, Neural Network (without PCA)

In [ ]:
```
from sklearn.neural_network import MLPClassifier

models_scores_val_NN = []


for train_indexes, val_indexes in kf.split(X_standardized):
    model_temp = MLPClassifier(hidden_layer_sizes=(6,10,6,4), activation="relu",max_iter=1000)
    model_temp.fit(X_standardized[train_indexes], y_train[train_indexes].ravel())
    y_pred_temp = model_temp.predict(X_standardized[val_indexes])
    f1_score_temp = f1_score(y_train[val_indexes],y_pred_temp)
    y_folds = y_train[val_indexes]
    score_false_negativ = 0
    nb_positivs = 0
    for i in range(len(y_train[val_indexes])):
        if y_folds[i][0]:
            nb_positivs+=1
            if not(y_pred_temp[i]):
                score_false_negativ+=1
    models_scores_val_NN.append([model_temp,f1_score_temp,score_false_negativ/nb_positivs])
```

In [ ]:
```
models_scores_val_NN
```

Out[ ]:
```
[[MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7474747474747475,
  0.20430107526881722],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6888888888888889,
  0.38],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.743455497382199,
  0.29],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6853146853146853,
  0.36363636363636365],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.625,
  0.375],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6405228758169934,
  0.3950617283950617],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7011494252873565,
  0.3711340206185567],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7630057803468209,
  0.27472527472527475],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6526315789473685,
  0.38],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7499999999999999,
  0.2871287128712871],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6225165562913907,
  0.45348837209302323],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.641304347826087,
  0.4326923076923077],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6543209876543209,
  0.38372093023255816],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7040816326530612,
  0.31],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7027027027027026,
  0.38095238095238093],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6844919786096257,
  0.3191489361702128]]
```

We could select the one doing the best in terms of f1 score but here we prefer selecting the one doing the less bad on numbers of false negatives

In [ ]:
```
model_NN_selected = models_scores_val_NN[0][0]
print("F1 score:",models_scores_val_NN[0][1])
print("Number of false negatives:",models_scores_val_NN[0][2])
```

```
F1 score: 0.7474747474747475
Number of false negatives: 0.20430107526881722
```

Fourth model, NN with PCA

In [ ]:
```
from sklearn.neural_network import MLPClassifier

models_scores_val_NN_PCA = []


for train_indexes, val_indexes in kf.split(X_transformed):
    model_temp = MLPClassifier(hidden_layer_sizes=(6,10,6,4), activation="relu",max_iter=1000)
    model_temp.fit(X_transformed[train_indexes], y_train[train_indexes].ravel())
    y_pred_temp = model_temp.predict(X_transformed[val_indexes])
    f1_score_temp = f1_score(y_train[val_indexes],y_pred_temp)
    y_folds = y_train[val_indexes]
    score_false_negativ = 0
    nb_positivs = 0
    for i in range(len(y_train[val_indexes])):
        if y_folds[i][0]:
            nb_positivs+=1
            if not(y_pred_temp[i]):
                score_false_negativ+=1
    models_scores_val_NN_PCA.append([model_temp,f1_score_temp,score_false_negativ/nb_positivs])
```

In [ ]:
```
models_scores_val_NN_PCA
```

Out[ ]:
```
[[MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7500000000000001,
  0.2903225806451613],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6547619047619048,
  0.45],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7708333333333333,
  0.26],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6808510638297872,
  0.37662337662337664],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6467065868263473,
  0.38636363636363635],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6103896103896104,
  0.41975308641975306],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6867469879518072,
  0.41237113402061853],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7701149425287357,
  0.26373626373626374],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6885245901639344,
  0.37],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7623762376237624,
  0.2376237623762376],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6335403726708074,
  0.4069767441860465],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6432748538011696,
  0.47115384615384615],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6490066225165563,
  0.43023255813953487],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6400000000000001,
  0.44],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6984126984126985,
  0.37142857142857144],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6626506024096385,
  0.4148936170212766]]
```

In [ ]:
```
model_NN_PCA_selected = models_scores_val_NN_PCA[2][0]
print("F1 score:",models_scores_val_NN_PCA[2][1])
print("Number of false negatives:",models_scores_val_NN_PCA[2][2])
```

```
F1 score: 0.7708333333333333
Number of false negatives: 0.26
```

As a final test, we test our models on the test data set, we center and reduce it with the mean and the standard deviation of the training data set and we project it for the models that needed it onto the subspace found thanks to PCA on the training dataset Then we evaluate the performance of our models on the test set thanks to the f1 score and the number of false negatives

In [ ]:
```
X_test_standardized = scaler.transform(X_test)
X_test_transformed = pca.transform(X_test_standardized)

y_pred_X_test_log_rec = model_log_rec_selected.predict(X_test_transformed)
y_pred_X_test_SVM = model_svm_selected.predict(X_test_standardized)
y_pred_X_test_NN_PCA = model_NN_PCA_selected.predict(X_test_transformed)
y_pred_X_test_NN = model_NN_selected.predict(X_test_standardized)

f1_score_log_rec = f1_score(y_test.ravel(),y_pred_X_test_log_rec)
score_false_negativ_log_rec = 0
for i in range(len(y_test)):
    if y_test[i] and not(y_pred_X_test_log_rec[i]):
        score_false_negativ_log_rec+=1

f1_score_SVM = f1_score(y_test.ravel(),y_pred_X_test_SVM)
score_false_negativ_SVM = 0
for i in range(len(y_test)):
    if y_test[i] and not(y_pred_X_test_SVM[i]):
        score_false_negativ_SVM+=1

f1_score_NN = f1_score(y_test.ravel(),y_pred_X_test_NN)
score_false_negativ_NN = 0
for i in range(len(y_test)):
    if y_test[i] and not(y_pred_X_test_NN[i]):
        score_false_negativ_NN+=1

f1_score_NN_PCA = f1_score(y_test.ravel(),y_pred_X_test_NN_PCA)
score_false_negativ_NN_PCA = 0
for i in range(len(y_test)):
    if y_test[i] and not(y_pred_X_test_NN_PCA[i]):
        score_false_negativ_NN_PCA+=1



print(f"f1_score for Logistic Regression with PCA transformation on X_test: {f1_score_log_rec} and the number of false negativ: {score_false_negativ_log_rec}")
print(f"f1_score for SVM on X_test: {f1_score_SVM} and the number of false negativ: {score_false_negativ_SVM}")
print(f"f1_score for NN on X_test: {f1_score_NN} and the number of false negativ: {score_false_negativ_NN}")
print(f"f1_score for NN with PCA transformation on X_test: {f1_score_NN_PCA} and the number of false negativ: {score_false_negativ_NN_PCA}")
```

```
f1_score for Logistic Regression with PCA transformation on X_test: 0.6176470588235294 and the number of false negativ: 18
f1_score for SVM on X_test: 0.6451612903225806 and the number of false negativ: 19
f1_score for NN on X_test: 0.6233766233766234 and the number of false negativ: 15
f1_score for NN with PCA transformation on X_test: 0.6285714285714286 and the number of false negativ: 17
```

NN with PCA seems here to be a good model to deal with our database, with the second highest f1 score and the second smallest number of false negatives, which is a good tradeoff. I choose to focus on this model.

Step 2

We go through the same process than above, using the 4 same models but trained on the new dataset made of the old X on which we added the column "benignant/malignant calcification".

In [ ]:
```
col_malignant = model_NN_PCA_selected.predict(pca.transform(scaler.transform(X)))
X_new = np.hstack([X,np.atleast_2d(col_malignant).T])
X_new_label_0 = X_new[:2020]
X_new_label_1 = X_new[2020:]
```

In [ ]:
```
X_train_new_0, X_test_new_0, y_train_new_0, y_test_new_0 = train_test_split(X_new_label_0, y_label_0, test_size=0.025)
X_train_new_1, X_test_new_1, y_train_new_1, y_test_new_1 = train_test_split(X_new_label_1, y_label_1, test_size=0.025)
X_test_new,y_test_new = np.concatenate([X_test_new_0,X_test_new_1]),np.concatenate([y_test_new_0,y_test_new_1])
X_train_new, y_train_new= np.concatenate([X_train_new_0,X_train_new_1]),np.concatenate([y_train_new_0,y_train_new_1])
train_indexes_new = np.arange(len(X_train_new))
np.random.shuffle(train_indexes_new)
X_train_new = X_train_new[train_indexes_new]
y_train_new = y_train_new[train_indexes_new]

test_indexes_new = np.arange(len(X_test_new))
np.random.shuffle(test_indexes_new)
X_test_new = X_test_new[test_indexes_new]
y_test_new = y_test_new[test_indexes_new]
```

In [ ]:
```
scaler_2 = StandardScaler()
scaler_2.fit(X_train_new)
X_new_standardized = scaler_2.transform(X_train_new)
pca_2 = PCA(0.9)
pca_2.fit(X_new_standardized)
X_new_transformed = pca_2.transform(X_new_standardized)
```

In [ ]:
```
models_scores_val_log_reg_new = []


for train_indexes, val_indexes in kf.split(X_new_transformed):
    model_temp = LogisticRegression()
    model_temp.fit(X_new_transformed[train_indexes], y_train_new[train_indexes].ravel())
    y_pred_temp = model_temp.predict(X_new_transformed[val_indexes])
    f1_score_temp = f1_score(y_train_new[val_indexes],y_pred_temp)
    y_folds = y_train_new[val_indexes]
    score_false_negativ = 0
    nb_positivs = 0
    for i in range(len(y_train[val_indexes])):
        if y_folds[i][0]:
            nb_positivs+=1
            if not(y_pred_temp[i]):
                score_false_negativ+=1
    models_scores_val_log_reg_new.append([model_temp,f1_score_temp,score_false_negativ/nb_positivs])
```

In [ ]:
```
models_scores_val_log_reg_new
```

Out[ ]:
```
[[LogisticRegression(), 0.776470588235294, 0.2826086956521739],
 [LogisticRegression(), 0.6628571428571428, 0.41414141414141414],
 [LogisticRegression(), 0.738095238095238, 0.30337078651685395],
 [LogisticRegression(), 0.6716417910447761, 0.4155844155844156],
 [LogisticRegression(), 0.75, 0.30303030303030304],
 [LogisticRegression(), 0.7675675675675675, 0.2828282828282828],
 [LogisticRegression(), 0.7362637362637363, 0.33],
 [LogisticRegression(), 0.7329192546583853, 0.2716049382716049],
 [LogisticRegression(), 0.7692307692307692, 0.3010752688172043],
 [LogisticRegression(), 0.728476821192053, 0.32926829268292684],
 [LogisticRegression(), 0.7560975609756098, 0.29545454545454547],
 [LogisticRegression(), 0.7513227513227513, 0.30392156862745096],
 [LogisticRegression(), 0.7676767676767676, 0.2962962962962963],
 [LogisticRegression(), 0.7484662576687118, 0.3146067415730337],
 [LogisticRegression(), 0.7771428571428571, 0.31313131313131315],
 [LogisticRegression(), 0.7772020725388601, 0.29245283018867924]]
```

In [ ]:
```
model_log_rec_selected_new = models_scores_val_log_reg_new[0][0]
print("F1 score:",models_scores_val_log_reg_new[0][1])
print("Number of false negatives:",models_scores_val_log_reg_new[0][2])
```

```
F1 score: 0.776470588235294
Number of false negatives: 0.2826086956521739
```

In [ ]:
```
models_scores_val_SVC_new = []


for train_indexes, val_indexes in kf.split(X_new_standardized):
    model_temp = SVC(gamma='auto')
    model_temp.fit(X_new_standardized[train_indexes], y_train_new[train_indexes].ravel())
    y_pred_temp = model_temp.predict(X_new_standardized[val_indexes])
    f1_score_temp = f1_score(y_train_new[val_indexes],y_pred_temp)
    y_folds = y_train_new[val_indexes]
    score_false_negativ = 0
    nb_positivs=0
    for i in range(len(y_train[val_indexes])):
        if y_folds[i][0]:
            nb_positivs+=1
            if not(y_pred_temp[i]):
                score_false_negativ+=1
    models_scores_val_SVC_new.append([model_temp,f1_score_temp,score_false_negativ/nb_positivs])
```

In [ ]:
```
models_scores_val_SVC_new
```

Out[ ]:
```
[[SVC(gamma='auto'), 0.788235294117647, 0.2717391304347826],
 [SVC(gamma='auto'), 0.6666666666666667, 0.41414141414141414],
 [SVC(gamma='auto'), 0.7710843373493976, 0.2808988764044944],
 [SVC(gamma='auto'), 0.6617647058823529, 0.4155844155844156],
 [SVC(gamma='auto'), 0.7734806629834253, 0.29292929292929293],
 [SVC(gamma='auto'), 0.7692307692307693, 0.29292929292929293],
 [SVC(gamma='auto'), 0.7717391304347825, 0.29],
 [SVC(gamma='auto'), 0.75, 0.25925925925925924],
 [SVC(gamma='auto'), 0.783132530120482, 0.3010752688172043],
 [SVC(gamma='auto'), 0.7534246575342465, 0.32926829268292684],
 [SVC(gamma='auto'), 0.7439024390243902, 0.3068181818181818],
 [SVC(gamma='auto'), 0.7473684210526315, 0.30392156862745096],
 [SVC(gamma='auto'), 0.781725888324873, 0.28703703703703703],
 [SVC(gamma='auto'), 0.8, 0.2808988764044944],
 [SVC(gamma='auto'), 0.7909604519774011, 0.29292929292929293],
 [SVC(gamma='auto'), 0.7748691099476439, 0.3018867924528302]]
```

In [ ]:
```
model_svm_selected_new = models_scores_val_SVC_new[0][0]
print("F1 score:",models_scores_val_SVC_new[0][1])
print("Number of false negatives:",models_scores_val_SVC_new[0][2])
```

```
F1 score: 0.788235294117647
Number of false negatives: 0.2717391304347826
```

In [ ]:
```
models_scores_val_NN_new = []


for train_indexes, val_indexes in kf.split(X_new_standardized):
    model_temp = MLPClassifier(hidden_layer_sizes=(6,10,6,4), activation="relu",max_iter=1000)
    model_temp.fit(X_new_standardized[train_indexes], y_train_new[train_indexes].ravel())
    y_pred_temp = model_temp.predict(X_new_standardized[val_indexes])
    f1_score_temp = f1_score(y_train_new[val_indexes],y_pred_temp)
    y_folds = y_train_new[val_indexes]
    score_false_negativ = 0
    nb_positivs=0
    for i in range(len(y_train[val_indexes])):
        if y_folds[i][0]:
            nb_positivs+=1
            if not(y_pred_temp[i]):
                score_false_negativ+=1
    models_scores_val_NN_new.append([model_temp,f1_score_temp,score_false_negativ/nb_positivs])
```

In [ ]:
```
models_scores_val_NN_new
```

Out[ ]:
```
[[MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.711111111111111,
  0.30434782608695654],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6519337016574586,
  0.40404040404040403],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7052023121387283,
  0.3146067415730337],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6715328467153285,
  0.4025974025974026],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7046632124352332,
  0.31313131313131315],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.770053475935829,
  0.2727272727272727],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7643979057591622,
  0.27],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7450980392156864,
  0.2962962962962963],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.689655172413793,
  0.3548387096774194],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7189542483660131,
  0.32926829268292684],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7388535031847134,
  0.3409090909090909],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6984126984126984,
  0.35294117647058826],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7282051282051282,
  0.3425925925925926],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7045454545454545,
  0.30337078651685395],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7513227513227514,
  0.2828282828282828],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7192118226600984,
  0.3113207547169811]]
```

In [ ]:
```
model_NN_selected_new = models_scores_val_NN_new[5][0]
print("F1 score:",models_scores_val_NN_new[5][1])
print("Number of false negatives:",models_scores_val_NN_new[5][2])
```

```
F1 score: 0.770053475935829
Number of false negatives: 0.2727272727272727
```

In [ ]:
```
models_scores_val_NN_PCA_new = []


for train_indexes, val_indexes in kf.split(X_new_transformed):
    model_temp = MLPClassifier(hidden_layer_sizes=(6,10,6,4), activation="relu",max_iter=1000)
    model_temp.fit(X_new_transformed[train_indexes], y_train_new[train_indexes].ravel())
    y_pred_temp = model_temp.predict(X_new_transformed[val_indexes])
    f1_score_temp = f1_score(y_train_new[val_indexes],y_pred_temp)
    y_folds = y_train_new[val_indexes]
    score_false_negativ = 0
    nb_positivs=0
    for i in range(len(y_train[val_indexes])):
        if y_folds[i][0]:
            nb_positivs+=1
            if not(y_pred_temp[i]):
                score_false_negativ+=1
    models_scores_val_NN_PCA_new.append([model_temp,f1_score_temp,score_false_negativ/nb_positivs])
```

In [ ]:
```
models_scores_val_NN_PCA_new
```

Out[ ]:
```
[[MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7305389221556886,
  0.33695652173913043],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6347305389221557,
  0.46464646464646464],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.691358024691358,
  0.3707865168539326],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6119402985074627,
  0.4675324675324675],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7085714285714285,
  0.37373737373737376],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7789473684210526,
  0.25252525252525254],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7407407407407408,
  0.3],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7044025157232704,
  0.30864197530864196],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7393939393939394,
  0.34408602150537637],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.6878980891719746,
  0.34146341463414637],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7388535031847134,
  0.3409090909090909],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.71875,
  0.3235294117647059],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.78,
  0.2777777777777778],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.703030303030303,
  0.34831460674157305],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7292817679558011,
  0.3333333333333333],
 [MLPClassifier(hidden_layer_sizes=(6, 10, 6, 4), max_iter=1000),
  0.7474747474747474,
  0.3018867924528302]]
```

In [ ]:
```
model_NN_PCA_selected_new = models_scores_val_NN_PCA_new[5][0]
print("F1 score:",models_scores_val_NN_PCA_new[5][1])
print("Number of false negatives:",models_scores_val_NN_PCA_new[5][2])
```

```
F1 score: 0.7789473684210526
Number of false negatives: 0.25252525252525254
```

In [ ]:
```
X_test_standardized_new = scaler_2.transform(X_test_new)
X_test_transformed_new = pca_2.transform(X_test_standardized_new)

y_pred_X_test_log_rec_new = model_log_rec_selected_new.predict(X_test_transformed_new)
y_pred_X_test_SVM_new = model_svm_selected_new.predict(X_test_standardized_new)
y_pred_X_test_NN_PCA_new = model_NN_PCA_selected_new.predict(X_test_transformed_new)
y_pred_X_test_NN_new = model_NN_selected_new.predict(X_test_standardized_new)

f1_score_log_rec_new = f1_score(y_test_new.ravel(),y_pred_X_test_log_rec_new)
score_false_negativ_log_rec_new = 0
for i in range(len(y_test_new)):
    if y_test_new[i] and not(y_pred_X_test_log_rec_new[i]):
        score_false_negativ_log_rec_new+=1

f1_score_SVM_new = f1_score(y_test_new.ravel(),y_pred_X_test_SVM_new)
score_false_negativ_SVM_new = 0
for i in range(len(y_test_new)):
    if y_test_new[i] and not(y_pred_X_test_SVM_new[i]):
        score_false_negativ_SVM_new+=1

f1_score_NN_new = f1_score(y_test_new.ravel(),y_pred_X_test_NN_new)
score_false_negativ_NN_new = 0
for i in range(len(y_test_new)):
    if y_test_new[i] and not(y_pred_X_test_NN_new[i]):
        score_false_negativ_NN_new+=1

f1_score_NN_PCA_new = f1_score(y_test_new.ravel(),y_pred_X_test_NN_PCA_new)
score_false_negativ_NN_PCA_new = 0
for i in range(len(y_test_new)):
    if y_test_new[i] and not(y_pred_X_test_NN_PCA_new[i]):
        score_false_negativ_NN_PCA_new+=1



print(f"f1_score for Logistic Regression with PCA transformation on X_test: {f1_score_log_rec_new} and the number of false negativ: {score_false_negativ_log_rec_new}")
print(f"f1_score for SVM on X_test: {f1_score_SVM_new} and the number of false negativ: {score_false_negativ_SVM_new}")
print(f"f1_score for NN on X_test: {f1_score_NN_new} and the number of false negativ: {score_false_negativ_NN_new}")
print(f"f1_score for NN with PCA transformation on X_test: {f1_score_NN_PCA_new} and the number of false negativ: {score_false_negativ_NN_PCA_new}")
```

```
f1_score for Logistic Regression with PCA transformation on X_test: 0.746268656716418 and the number of false negativ: 14
f1_score for SVM on X_test: 0.7605633802816902 and the number of false negativ: 12
f1_score for NN on X_test: 0.7142857142857142 and the number of false negativ: 14
f1_score for NN with PCA transformation on X_test: 0.7397260273972601 and the number of false negativ: 12
```

We clearly select the SVM model here to classify the micro-calcification as predicting cancer.

Discussion about the results

The results are nice but no satisfying and can be improved:

- In step 1, now that we know that NN with PCA seems to be a good choice to predict if a micro-calcification, we can play on hyperparameters and regularisation to improve the model. We could select them by choosing the hyperparameters minimizing the validation error.
- In step 2, SVM seems to do the job. We could imagine playing with the C argument that is a Regularization parameter to improve the model, in the training phase, by selecting the C value that minimize the validation error.
- In step 2, we could avoid a certain number of false negatives of the model, by taking "an average" when taking the decision of wheter a person has cancer : multiple micro-calcifications belong to one person, so if a majority of micro-calcifications predicts that the person has a cancer, we take the decision to state that the model predicts "cancer" for the patient. We could be even stricter by saying that if one micro-calcification of a person predicts cancer, we decide to state that the model predicts "cancer".

Conclusion

Machine learning techniques allow us here to predict really quickly whether a person has cancer or not. If the models above are improved, they could really get into the action to diagnose breast cancer. But we should keep in mind that the database was quite small and another way to improve the model could be a kind of continuous learning ("online learning") where the model is still trained with new or updated data. Last but not least, even if ML techniques offer a good solution to predict breast cancer, we should not let them do all the jobs, and a human with its critical point of view should stay behind to double check the results : when selecting the models, I (a human) was selecting the models by doing a tradeoff bewteen a good f1 score while keeping the ratio of false negatives low.

Reference

- https://towardsdatascience.com/the-5-classification-evaluation-metrics-you-must-know-aa97784ff226
- https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6