Udacity Machine Learning Engineer Nanodegree

Predictive Analysis of Mammographic Masses Data

Henrique Dal Mora Rosendo da Silva
March 21, 2019

Capstone Project

I. Definition

Project Overview

Breast cancer is the most common among women. Statistics appoint that about 1 in 8 U.S. women, representing about 12.4%, will develop invasive breast cancer over the course of her lifetime. Sadly, about 41760 women in the U.S are expected to die in 2019 from breast cancer [1].

Over the years Computer Aided Detection Systems were developed to help the analysis of mammograms. However, new technologies integrated into Machine Learning and Deep Learning fields are being more widely used to develop systems focused on the prediction of Mammographic Masses. This is happening because these applications can have a better performance in the identification of breast cancer or the classification of Masses as benign or malignant cases.

Studies and applications are being developed during the course of the last decade. An example of research using a Supervised Learning approach is applied for mass detection in digital mammograms base on Support Vector Machines Algorithm [2]. This capstone project will follow a similar path for mammographic mass detection, but with more focus in the classification of masses data as benign or malignant based on a series of evaluation of Supervised Learning algorithms and Artificial Neural Networks performances.

Problem Statement

One of the most difficult and important jobs of a doctor can be informing a cancer diagnosis to a specific patient. There's an instant emotional, mental and physical reaction from the patient that represents the abrupt changes that will occur in his life. The most important aspect of a tumor can be its level of destruction, whether it is benign or malignant. This classification can inflect directly in the future health and medical difficulties the individual will face.

Therefore, the resulting application of this project can be used to automatically classify a Mammogram Mass with more efficiency in terms of time and accuracy, based on its clinical data.

To understand the dataset labels and values, initial data exploration will be performed. This preprocessing can guarantee better model predictions for our final results (e.g. Identify Features and targets).

After these actions, the step of evaluating the algorithms chosen in the "Solution Statement" section will take place. This process consists of building the models, make the respective predictions and, finally, select the best model. For last, as a final conclusion, the chosen model hyperparameters and variables will be tuned to achieve a better result.

Metrics

The performance of our final model, and consequently the best supervised algorithm to fit the mammographic mass data classification problem, is evaluated through the following statistical measures:

1. **Accuracy**
   The accuracy of the model can be acquired by the following relation:

$$Acc = \frac{TP + TN}{TN + FN + TP + FP}$$

The above variables represent:

TP: True Positive

TN: True Negative

FN: False Negative
FP: False Positive

The accuracy value obtained represents the number of correctly classified cases in our model output.

2. **Precision**
   For the models precision, we have the equation:

$$Pcs = \frac{TP + TN}{TN + FN + TP + FP}$$

3. **Recall**
   Finally, the Recall of the model is given by:

$$Rec = \frac{TP + TN}{TN + FN + TP + FP}$$

II. Analysis

Data Exploration

The chosen public dataset is provided by the UCI Machine Learning Repository. This data contains 961 instances of masses detected in mammograms each one with six attributes [3]. The classification goal is to predict if a mammogram data is related to a benign or malignant case of cancer, based on the input variables only.

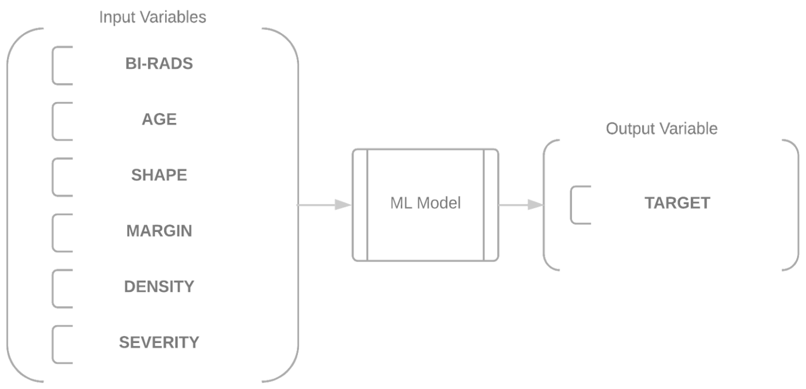


Figure 1 - Input and output variables relation

Input Variables

1. **BI-RADS**** assessment: 1 to 5 (ordinal)
2. **Age**: patient's age in years (integer)
3. **Shape**: mass shape: round=1 oval=2 lobular=3 irregular=4 (nominal)
4. **Margin**: mass margin: circumscribed=1 microlobulated=2 obscured=3 ill-defined=4 spiculated=5 (nominal)
5. **Density**: mass density high=1 iso=2 low=3 fat-containing=4 (ordinal)

Output Variable

1. **Target**: **Severity** - benign = ‘0’ or malignant = ‘1’ (binominal)

****BI-RADS** is an acronym for Breast Imaging-Reporting and Data System

Following are the files description that will be used in this project, and the respective URL where they can be found:

1. **mammographic_masses.data** - The dataset containing case values for all the attributes above.
2. **mammographic_masses.names** - Relevant informations about the dataset.

https://archive.ics.uci.edu/ml/machine-learning-databases/mammographic-masses/

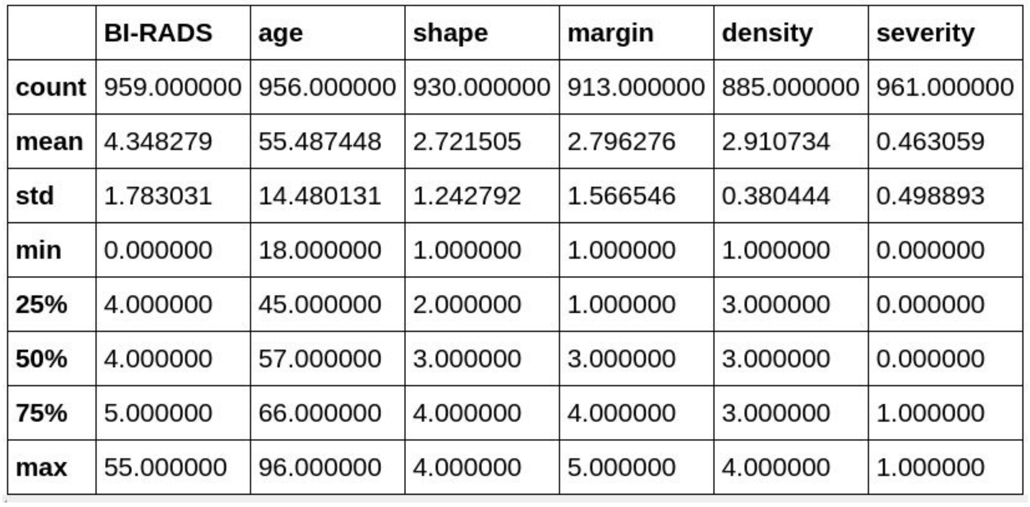
This dataset can be treated as a binary problem, where we only have two target outputs, 1 or 0 representing a malignant and benign case, respectively. We have a total of 961 input instances. The Class Distribution of our data and the other variables are described in the following table.

Mass classification	Output target	Class Distribution	Volume percentage	Total of instances
Malignant	1	445	46,3%	961
Benign	0	516	53,7%	

Table 1 - Dataset class distribution

According to the table above, and comparing the volume percentage that each target represents in the total instance, we can assure that this dataset is very well balance in terms of distribution of its classes. So, one relevant measure that can confirm the effectiveness of our predictions is the accuracy of the model. Since the data is balanced, the results won't be overfitted to a specific class.

```
masses_data.describe()
```



	BI-RADS	age	shape	margin	density	severity
count	959.000000	956.000000	930.000000	913.000000	885.000000	961.000000
mean	4.348279	55.487448	2.721505	2.796276	2.910734	0.463059
std	1.783031	14.480131	1.242792	1.566546	0.380444	0.498893
min	0.000000	18.000000	1.000000	1.000000	1.000000	0.000000
25%	4.000000	45.000000	2.000000	1.000000	3.000000	0.000000
50%	4.000000	57.000000	3.000000	3.000000	3.000000	0.000000
75%	5.000000	66.000000	4.000000	4.000000	3.000000	1.000000
max	55.000000	96.000000	4.000000	5.000000	4.000000	1.000000

Table 2 - Masses data description

The above table represents the descriptive statistics on the data. The feature ***'age'*** seems to have a lot more variance, so it could be a good predictor for this analysis.

Exploratory Visualization

Figures 2 and 3 below show us the **'age'** and **'density'** features distribution in our dataset. For the ***age*** feature, we can spot that the most occurrences in our data are from women with about 40 to 70 years. This can be related to the fact that most researches associated with breast cancer are made with people in this range of age. So, a first assumption is that the age feature has an important weight as a predictive feature in our model. For the ***density*** feature, we have basically four values of ***mass density*** to consider, there being:

Density feature values	**Value**	1	2	3	4
	Meaning	high	iso	low	fat-containing

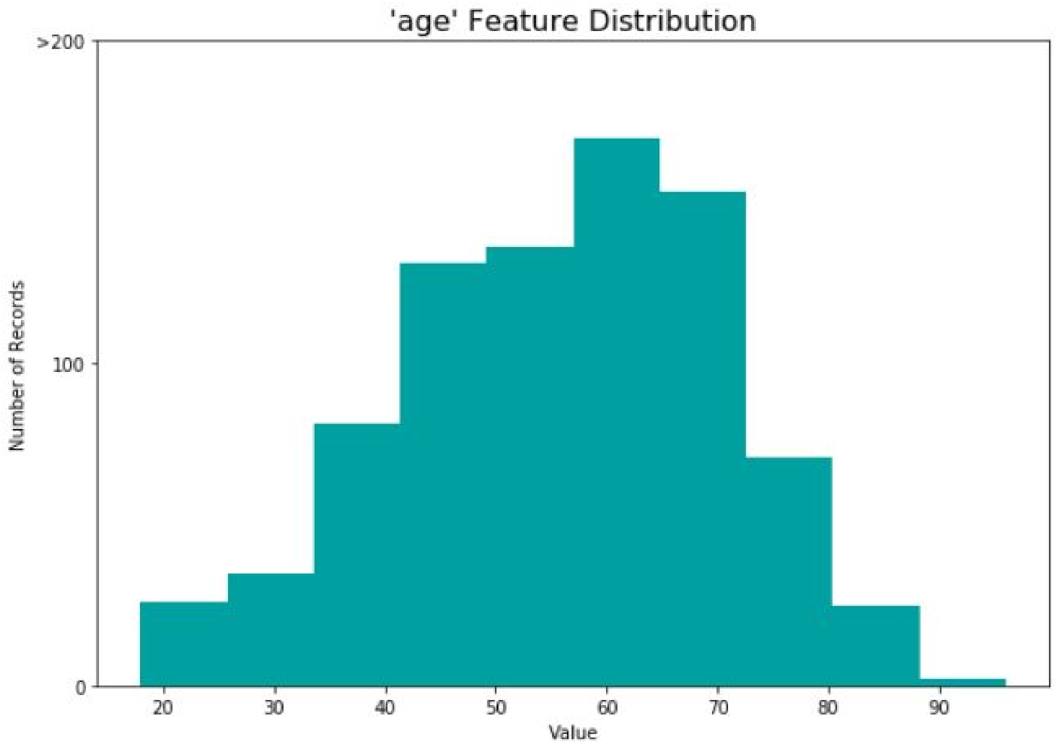


Figure 2 - Age feature distribution in masses data

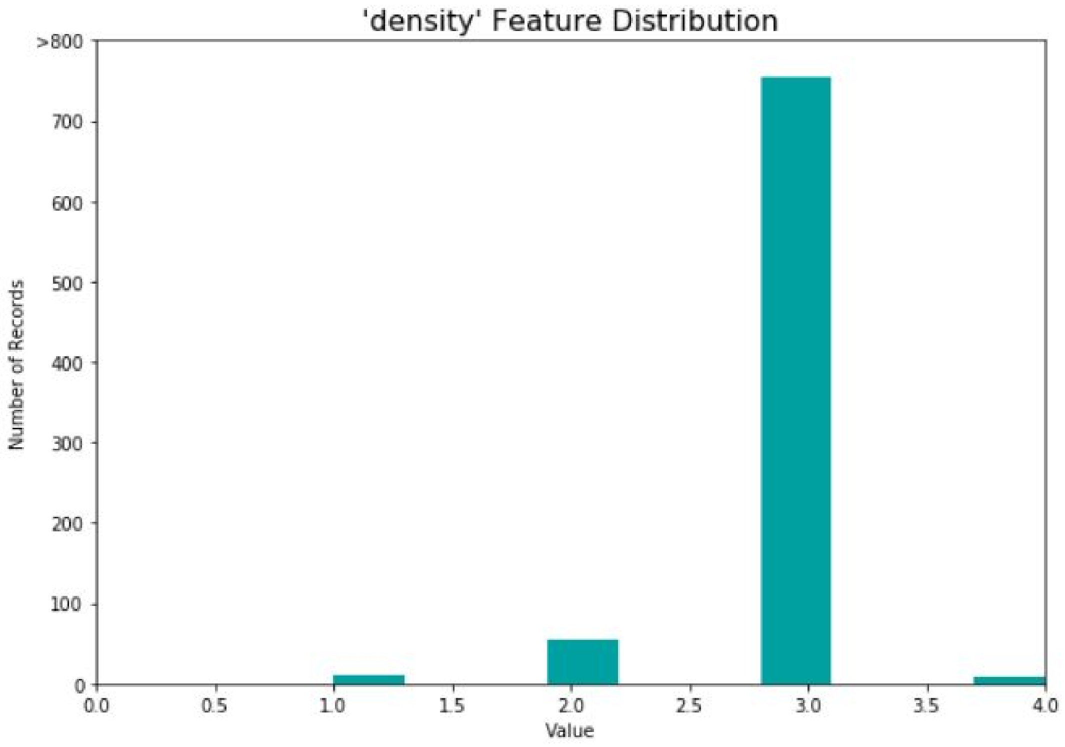


Figure 3 - Density feature distribution in masses data

Analyzing the figure above, the density feature is more concentrated in the value 3, meaning that most cases of mammographic data are from a low-density mass. This feature can also be a good predictor for our model since most cases of breast cancer in an advanced state can be correlated with values 1, 2 and 4. Also, a low-density class can be diagnosed as an early formed mass, with a higher probability to be treated if the cancer is predicted in this stage.

In Fig. 4 we applied an scatterplot matrix on the dataset to visualize inherent relationships between the features in our data.

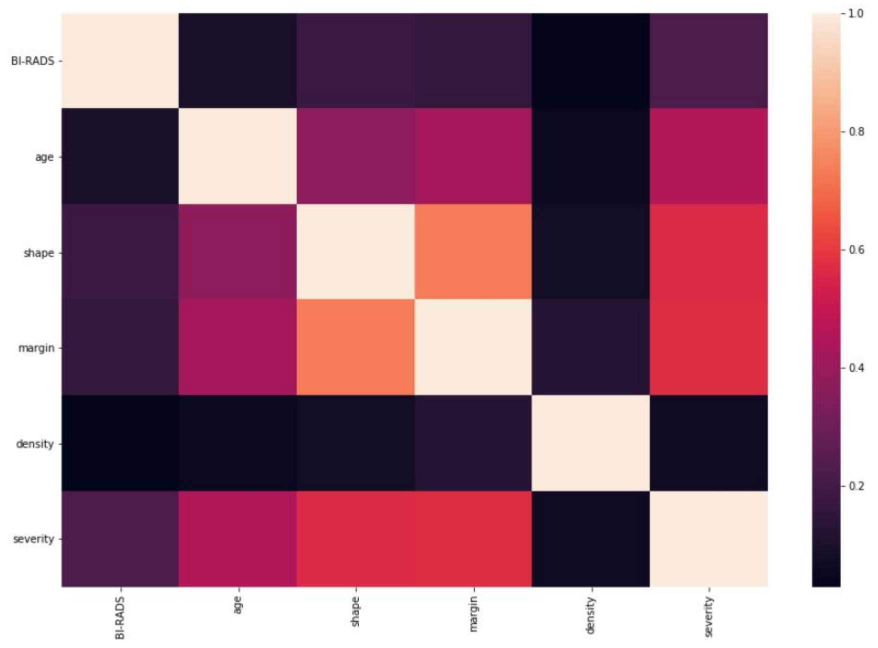


Figure 4 - Dataset scatterplot matrix

From the scatter matrix above we can see that there is not much of a strong relationship between our features.

An important task is to determine definitely which features in our dataset provides the most predictive power. In the previous discussions in this section, we conclude that the age and density features should represent good predictors. Through an ***ExtraTreesClassifier***, that has a ***feature_importances_*** attribute, we could determine visually and quantitatively the weight importance of our features. This relation is shown in the following figure.

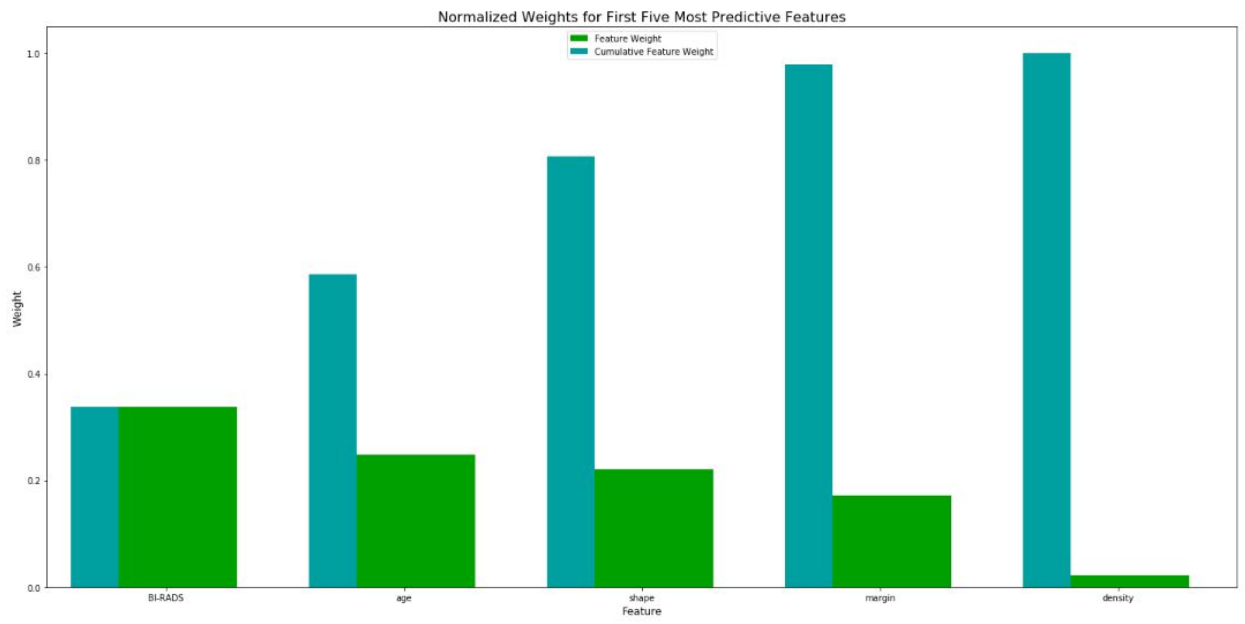


Figure 5 - Dataset scatterplot matrix

We can see that this graph confirms half of our early considerations. Indeed age is an important predictor in our model, but it's the second most important feature in our data. Actually, ***'BI-RADS'*** is the feature that has the most importance for our predictions. However, '***Density'*** is the least important feature in our dataset, representing the smallest feature weight importance.

Algorithms and Techniques

.The best supervised algorithm that will have the best accuracy and F-score for this prediction and classification problem is, at first, unknown. With this in mind, i decided to pick the following algorithms to evaluate

- Random Forest
- Support Vector Machines (SVM)
- Decision Trees
- Logistic Regression
- Naive Bayes
- K-Nearest-Neighbors (KNN)

These algorithms and techniques can be developed using TensorFlow and Keras Libraries. However, since the dataset is very simple and contains a low dimensionality (few number of features), this project will focus on the implementation of scikit-learn algorithms for classification and regression. The data will be splitted in a training and test sets, and the model will be trained based on these inputs.

A **Random Forest** is a supervised learning algorithm that can be used for both classification and regression problems. This is exactly the case of our model, where we have a target class that represents a binary system based on our features distributions. Basically, a Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction [4].

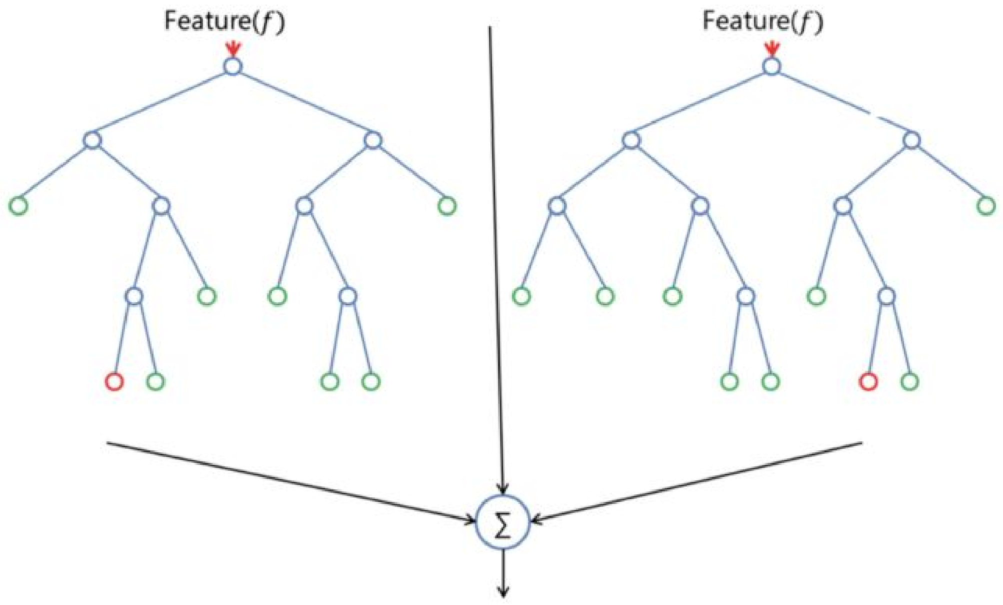


Figure 6 - Random Forest distribution

The above image represents a Random Forest with two classification trees. This method searches for the best feature among a subset of features to achieve more optimized predictions. **Decision Trees** have the same hyperparameters that need to be tuned in a Random forest classifier, but when the problem or the depth of the trees are too big, the Decision Tree tends to be overfitted. This is avoided exactly by the Random Forest, where the subset of features enables the structure of smaller trees that can be combined later. In other hands, this process can demand a lot more computational power. So, a regression approach by using the Random Forest regressor might also be a good modeling for this problem.

Supporting Vector Machines (SVM) are based by the theory of statistical learning. This theory state that a classifier with a very good generalization can be obtained by the capacity to predict correctly the class of new data inputs contained in the same domain where the learning occurred. The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points [5].

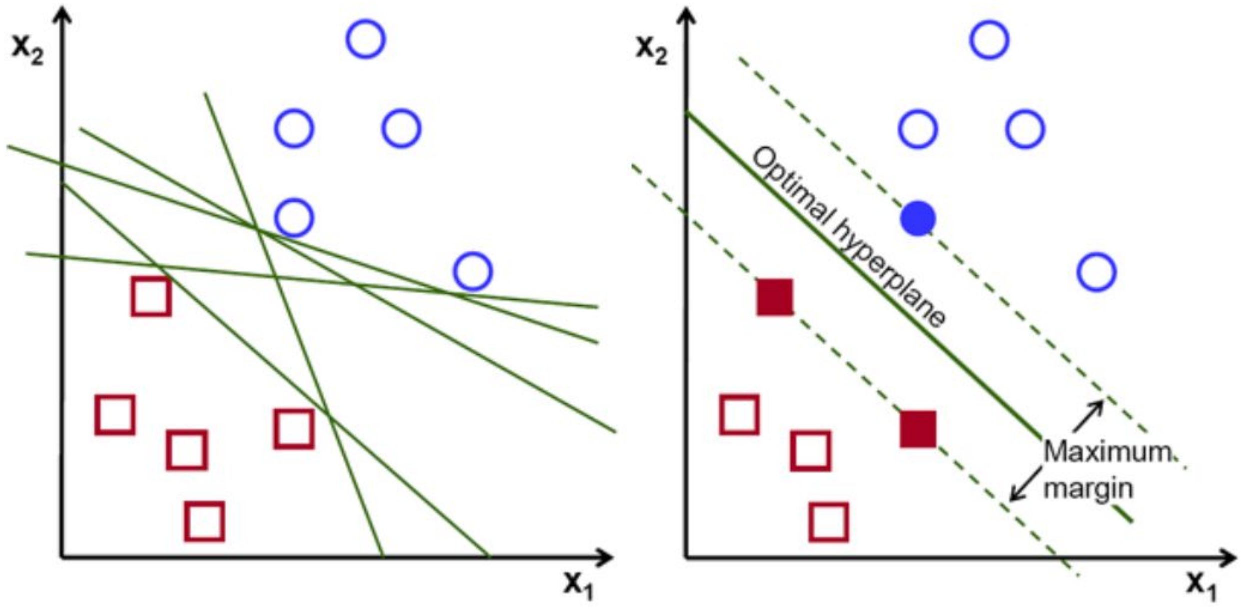


Figure 7 - possible Hyperplanes in SVM

Support vector machines are naturally resistant to overfitting and work very well when identifying boundary regions. A basic SVM with default hyperparameters settings will yield a very acceptable classification accuracy when the data target class is very well balanced.

The case of **Logistic Regression** is a perfect example to be applied in our model without the computational effort tradeoff to obtain a good score in the predictions. This method represents a predictive analysis. It is mainly used to describe data and to explain the relationship between one dependent binary variable and one or more binomial independent variables

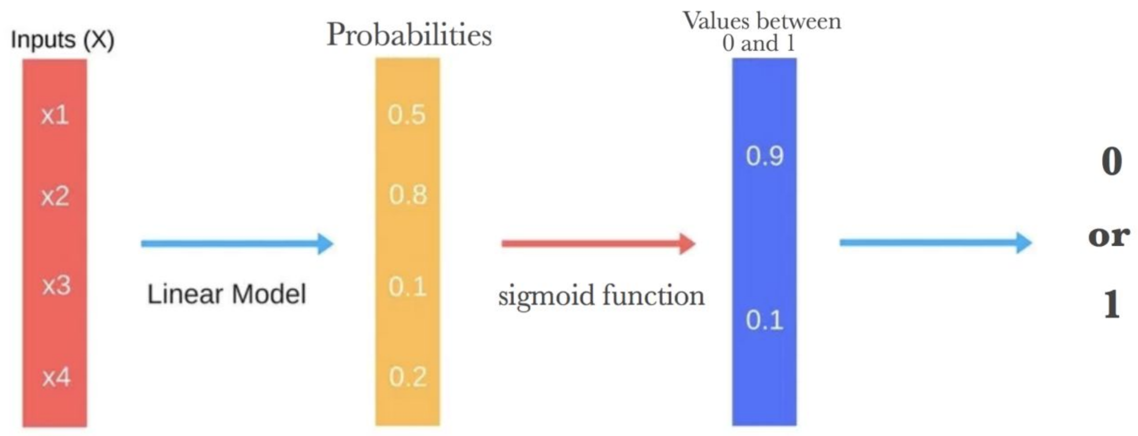


Figure 7 - Logistic Regression Model

This algorithm can be easily used to model our data because our output target class represents only two values, malignant or benignant (1 or 0). So, following the flux represented in the image above, from our feature inputs X, we model our probabilities based on regression and apply a sigmoid function, responsible for maintaining our value range between 0 and 1. Basically, this function states that If our probability density goes to infinity, the predicted value Y will become 1 and if it goes to negative infinity, Y (predicted) will become 0 [6]. This can be seen in the following graph:

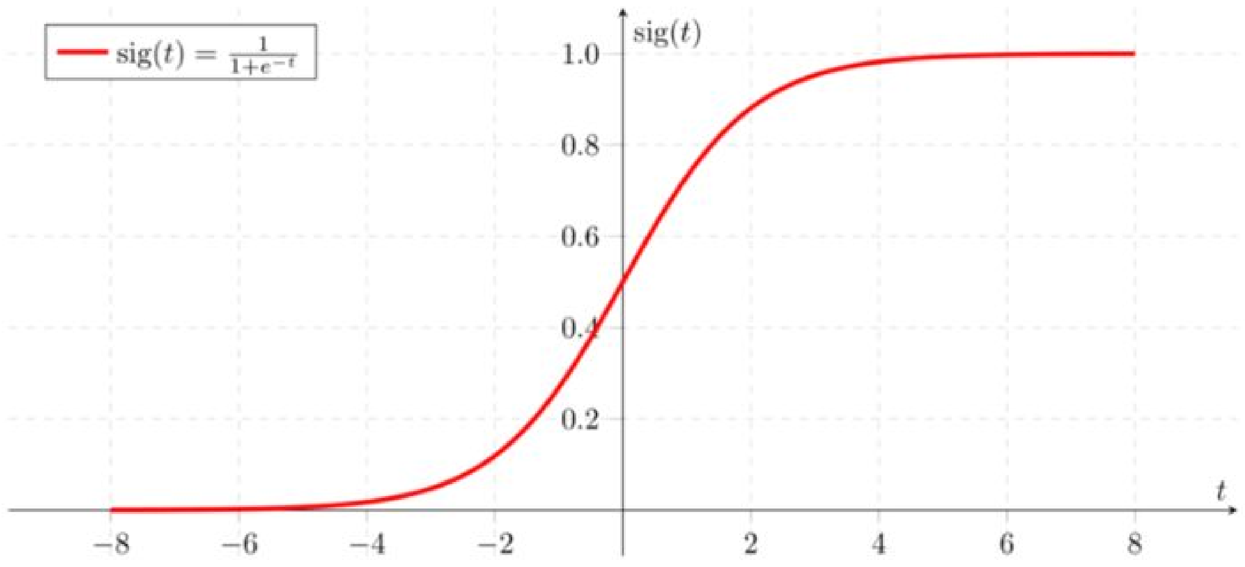


Figure 8 - Sigmoid function

These algorithms and supervised learning methods are the main focus of this projects, and further discussions and implementations will be made on top of them to achieve a good model to predict the mammographic masses based on our clinical data as inputs.

Benchmark

For comparisons, the following research will be used as benchmark model:

- **“Predicting the Severity of Breast Masses with Data Mining Methods ”** https://arxiv.org/ftp/arxiv/papers/1305/1305.7057.pdf

The results in this paper show us that a accuracy of about 81.25% where achieved in a test partition of the data when applying an SVM model. The dataset source for this research is the same from UCI Machine Learning Repository. My attempt with this project is to achieve an score of, at least, 75%.

III. Methodology

Data Preprocessing

We will begin the data preprocessing by reading the mammographic masses data with pandas, specifying the respective columns (features) names. The dataset must be checked for missing data, and associate all these inputs as outliers containing ***NaN*** values.

```
import pandas as pd

# Read the mammographic masses data with pandas, specifying the
respective columns (features) names
# Map all the missing data in the dataset represented with '?' as NaN
values
masses_data = pd.read_csv('mammographic_masses.data.txt',
na_values=['?'], names = ['BI-RADS', 'age', 'shape', 'margin',
'density', 'severity'])
masses_data.head()
```

	BI-RADS	age	shape	margin	density	severity
0	5.0	67.0	3.0	5.0	3.0	1
1	4.0	43.0	1.0	1.0	NaN	1
2	5.0	58.0	4.0	5.0	3.0	1
3	4.0	28.0	1.0	1.0	3.0	0
4	5.0	74.0	1.0	5.0	NaN	1

Table 2 - Masses data features with missing values as NaN

To check if the removal of the inputs with missing data would interfere with our prediction, we must guarantee that their distribution doesn't have any sort of correlation. To visualize this, we can print all the inputs with ***NaN*** values:

```
masses_data.loc[(masses_data['BI-RADS'].isnull()) |
                (masses_data['age'].isnull())     |
                (masses_data['shape'].isnull())   |
                (masses_data['margin'].isnull())  |
                (masses_data['density'].isnull()) |
                (masses_data['severity'].isnull())]
```

This instruction returns an table with 130 rows × 6 columns dimension. The ***NaN*** data seems randomly distributed. Therefore, we can consider dropping all the rows containing *null* values, as follows:

```
masses_data.dropna(inplace=True)
```

Next, the Pandas dataframes must be converted to numpy arrays, so we can use it directly into scikit-learn. For this to work, we can extract the features into an separate array from the classes (BI-RADS, age, shape, margin, density and severity).

```
import numpy as np

features_data = masses_data[['BI-RADS',
                             'age',
                              'shape',
                              'margin',
                              'density']].values
# np.asarray(features_data)

outputs = masses_data['severity'].values

feature_names = ['BI-RADS', 'age', 'shape', 'margin', 'density']
```

After these instructions, we have tree arrays to work with:

1. **features_data** - All the input rows data corresponding to each feature
2. **outputs** - Target predictions we want to achieve
3. **feature_names** - Features names we want to analyse

The final preprocessing action is to normalize our data, reducing the error function that affect our predictions. This can be achieved by the fitting of an **scaler** in our ***'features_data'*** array.

```
from sklearn import preprocessing

scaler = preprocessing.StandardScaler()
all_features_scaled = scaler.fit_transform(features_data)
```

Implementation

The Project Design workflow implementation is structured in the bellow flowchart. First, we performed an exploratory analysis of the dataset, determining the distribution of the target values based on the input features, as seen in ***table 1***. The following step consists of the data cleaning performed in the previous section, where we dropped all the rows containing null values from our data array. This was performed with the intention of removing these outliers from our analysis, since all NaN values where randomly distributed. Also, we splitted our Pandas dataframe in tree numpy arrays to use directly in scikit-learn.

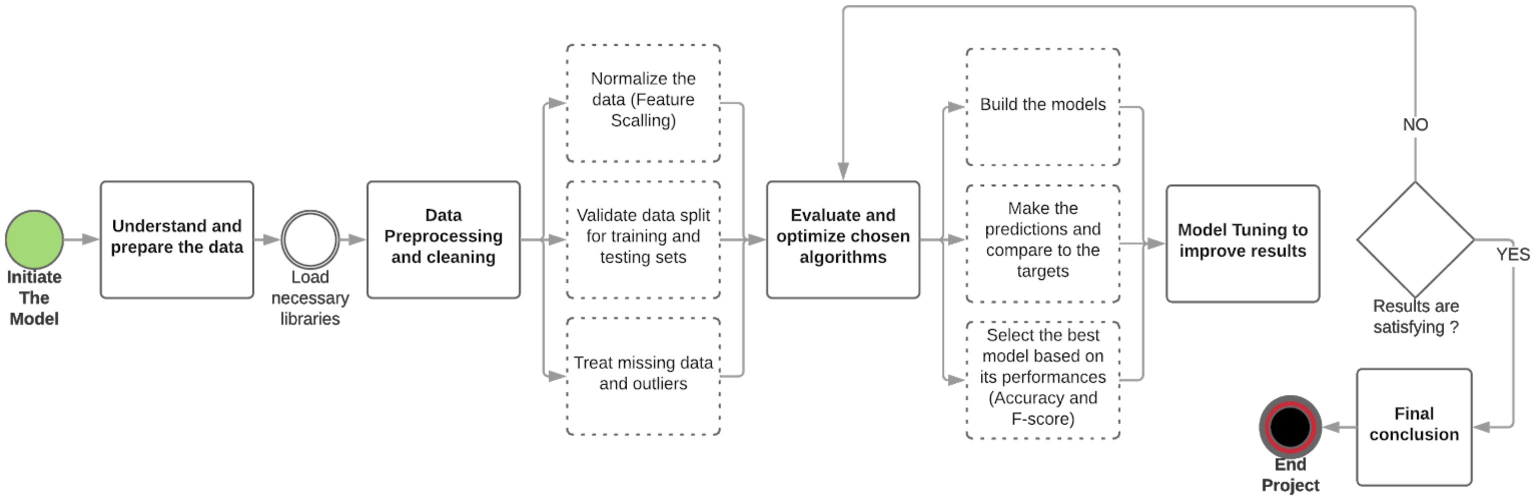

Initiate The Model
Understand and prepare the data
Load necessary libraries
Data Preprocessing and cleaning
Normalize the data (Feature Scalling)
Validate data split for training and testing sets
Treat missing data and outliers
Evaluate and optimize chosen algorithms
Build the models
Make the predictions and compare to the targets
Select the best model based on its performances (Accuracy and F-score)
Model Tuning to improve results
NO
YES
Results are satisfying ?
Final conclusion
End Project

Figure 6 - Project design workflow

Before any model definition, we need to split our data in training and testing sets. This process was performed through the ***train_test_split*** method from the ***sklearn**.**model_selection*** module:

```
from sklearn.model_selection import train_test_split

np.random.seed(1234)

(training_inputs,
 testing_inputs,
 training_classes,
 testing_classes) = train_test_split(all_features_scaled, outputs,
train_size=0.75, test_size=0.25, random_state=1)
```

The next step consists in the evaluation and optimization of the chosen algorithms to make our predictions. The following implementations of unsupervised learning algorithms obtained the best scores on the input data:

1. **Decision Trees**

```
from sklearn.tree import DecisionTreeClassifier

clf= DecisionTreeClassifier(random_state=1)
```

```
# Train the classifier on the training set
clf.fit(training_inputs, training_classes)

clf.score(testing_inputs, testing_classes)
```

2. **Random Forest**

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score


clf = RandomForestClassifier(n_estimators=10, random_state=1)
# instead of a single train/test split, use K-Fold cross validation
to get a better measure of your model's accuracy (K=10)
cv_scores = cross_val_score(clf, all_features_scaled, outputs, cv=10)

cv_scores.mean()
```

3. **Support Vector Machines (SVM)**

```
from sklearn import svm

C = 1.0
svc = svm.SVC(kernel='rbf', C=C, gamma='auto') # SVC with a rbf
kernel

cv_scores = cross_val_score(svc, all_features_scaled, outputs, cv=10)

cv_scores.mean()
```

4. **Logistic Regression**

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(solver='liblinear')
cv_scores = cross_val_score(clf, all_features_scaled, outputs, cv=10)
cv_scores.mean()
```

5. **KNN**

```
from sklearn import neighbors

clf = neighbors.KNeighborsClassifier(n_neighbors=10)
cv_scores = cross_val_score(clf, all_features_scaled, outputs, cv=10)

cv_scores.mean()
```

IV. Results

Model Evaluation and Validation

From the implementations above, the following scores were obtained for each model tuned:

Model	**Decision Trees**	**Random Forest**	**SVM**	**Logistic Regression**	**KNN (k = 43)**
Score	0.778	0.784	0.828	0.825	0.811

From the table above, the SVM model obtained the best score on the input data. Comparing this result with our benchmark, we have:

Model	**Benchmark Model (SVM)**	**Final Model (SVM)**
Accuracy Score	0.812	0.828

Both SVM and Logistic Regression are good approaches for our problem. Since it represents a binary system, the Logistic Regression is a very straightforward algorithm to implement, with much less effort in terms of computational work and model tuning to obtain a good score and predictions.

For a more depth evaluation of our **SVM model results**, we can apply a **grid search** using **GridSearchCV** with important parameters tuned with at least 3 different values. This was made in the project at the last cell of the jupyter notebook, and the following results were obtained:

Model	**Accuracy**	**F-Score**
Unoptimized	0.8576	0.7246
Optimized	0.8662	0.7420

Analyzing these results, we can see that the **Optimized model** achieved an Accuracy score bigger than the final model’s score obtained previously. Also, the F-score, considering the **precision** and **recall** of our model is very acceptable, resulted in **74.20%** of accuracy.

References

[1] https://www.breastcancer.org/symptoms/understand_bc/statistics
[2] https://iopscience.iop.org/article/10.1088/0031-9155/49/6/007/meta
[3] https://helda.helsinki.fi/bitstream/handle/10138/20368/mammogra.pdf?sequence=1
[4] https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd
[5]https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47
[6] https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc