Centrale
CASABLANCA

PROJECT

CODING WEEK REPORT

Detection Of Microcalcifications In Mammography

WORK BY

Abderahmane Salehi
Ramdani Nabil
Nahli Ghita
Benslimane Salma
Badreddine Saadioui
Selakh Jaafar

SUPERVISED BY

Mrs. Banouar Oumaima
Mrs. Zerhouni Kawtarabil

2022 / 2023

Acknowledgements

Our heartfelt thanks go first and foremost to Professor Banouar Oumaïma and Professor Zerhouni Kawtar, our supervisors, for their guidance, trust, and patience which were a considerable contribution in bringing this work to a successful conclusion.

We would also like to acknowledge the efforts of all those who have sacrificed a considerable amount of time to immerse us and get us started on this outline. Our thanks are rightly directed to all the people we consulted for advice, as well as to those who contributed directly or indirectly to giving structure to this work through their proposals and recommendations.

Abstract

This report provides an in-depth examination of the potential of machine learning tools for the detection of microcalcifications in mammograms, a crucial early sign of breast cancer. The study aims to investigate the effectiveness of different machine learning models in identifying and classifying microcalcifications, which can be difficult for human radiologists to detect. The report highlights the importance of developing accurate and efficient methods for detecting microcalcifications to improve the accuracy of breast cancer screening and ultimately save lives. The results of this study suggest that machine learning models have great potential in improving the accuracy of microcalcification detection in mammograms, thereby enhancing the early detection of breast cancer.

Keywords : microcalcifications, mammograms, machine learning, breast cancer screening, detection, classification, Imbalanced classification, accuracy, early detection, K-Nearest Neighbors, Support vector machine, Random forest...

Résumé

Ce rapport examine en profondeur le potentiel des outils d'apprentissage automatique pour la détection de microcalcifications dans les mammographies, un signe précoce crucial du cancer du sein. L'étude vise à évaluer l'efficacité de différents modèles d'apprentissage automatique dans l'identification et la classification des microcalcifications, qui peuvent être difficiles à détecter pour les radiologues humains. Le rapport met en évidence l'importance de développer des méthodes précises et efficaces pour la détection des microcalcifications afin d'améliorer la précision du dépistage du cancer du sein et de sauver des vies. Les résultats de cette étude suggèrent que les modèles d'apprentissage automatique ont un grand potentiel pour améliorer la précision de la détection des microcalcifications dans les mammographies, améliorant ainsi la détection précoce du cancer du sein.

Mots-clés : microcalcifications, mammographies, apprentissage automatique, dépistage du cancer du sein, détection, classification, classification déséquilibrée, précision, détection précoce, K-Nearest Neighbors, machine à vecteurs de support, Random Forest...

Contents

1. Introduction

Machine learning and deep learning are both used in detecting microcalcifications in mammography, but the choice of technique depends on the complexity of the problem and the availability of data. Machine learning algorithms such as support vector machines (SVM), random forests, and logistic regression have been successfully used in microcalcification detection tasks. These algorithms are effective in cases where the features of interest are easily detected. For example, radiologists can identify features such as size, shape, and texture of microcalcifications and use these features to train machine learning models. On the other hand, deep learning algorithms such as convolutional neural networks (CNNs) have shown great success in many image recognition tasks, including microcalcification detection. These algorithms can learn complex features directly from the images, without the need for hand-engineering. In our case, we'll be using the machine learning algorithms over the deep learning ones due to many reasons:

- Data Availability: machine learning does require less data. In cases where the amount of data available is limited, machine learning can still achieve good results, while deep learning requires large amounts of data for effective training.
- Interpretability: Machine learning algorithms are more interpretable than deep learning algorithms. In other words, it is easier to understand how a machine learning algorithm is making its decisions than it is for a deep learning algorithm. This can be important in medical applications where the ability to understand how a decision was made is critical.
- Computation: Deep learning models are generally more computationally intensive than machine learning models. This means that they require more processing power and time to train and deploy. For smaller clinics or hospitals with limited computing resources, machine learning may be a more practical option.
- Domain knowledge: In some cases, domain experts may have a good understanding of the features that are relevant for detecting microcalcifications. In these cases, a machine learning algorithm can be designed to incorporate this knowledge, while deep learning algorithms do not allow for the incorporation of domain knowledge.

2. Contextualization

Breast cancer remains a significant global public health concern, with approximately 2.3 million cases worldwide and 685,000 women death in 2020. Even more concerning is that every 14 seconds somewhere in the world, a woman is diagnosed with breast cancer. The causes of this disease are still unknown, making primary prevention seemingly impossible. Early detection is crucial to improving breast cancer prognosis, and mammography is a reliable method for this purpose. However, there are limitations to human observers, as it is difficult for radiologists to accurately and uniformly evaluate the enormous number of mammograms generated during widespread screening.

The detection of microcalcifications is an important and delicate task for the early diagnosisofthiscancer. The strong correlation between the appearance of microcalcification clusters and the presence of breast cancer makes computer aided diagnosis systems for automated detection/classification of microcalcification clusters a useful tool in breast cancer control.

This report discusses and compares the methods used in machine learning for detecting microcalcification clusters. Specifically, the enhancement and segmentation algorithms, mammographic features, classifiers, and their performances are studied and compared. The remaining challenges and future research directions in this area are also discussed.

3. Problem statement

Breast cancer is one of the leading causes of cancer deaths among women worldwide. Early detection and diagnosis of breast cancer are crucial in improving the survival rate and reducing mortality. Mammography is a widely used screening tool for breast cancer detection. However, the detection of microcalcifications in mammograms is a challenging task, as they are small, subtle, and can be easily missed by radiologists.

The aim of this project is to develop a machine learning model that can accurately detect microcalcifications using mammography's data. The proposed system will take mammogram data as input and use a based architecture model to classify them as either containing or not containing microcalcifications; which means our patient is either positive or negative to breast cancer. The model will be trained on a large dataset labeled with their corresponding ground-truth labels. The performance of the model will be evaluated using standard evaluation metrics such as accuracy, precision...

The proposed system has the potential to improve the accuracy and efficiency of breast cancer diagnosis by assisting radiologists in detecting microcalcifications, which can be indicative of early-stage breast cancer. The system can also help reduce the workload of radiologists and improve the overall quality of breast cancer screeningprograms.

4. Theoretical methods

There are various algorithms and techniques used for classification in machine learning, including decision trees, logistic regression, support vector machines, random forests, and neural networks. The choice of algorithm depends on the specific problem and the nature of the data being analyzed. Applications of classification in machine learning include image and speech recognition, fraud detection, spam filtering, and medical diagnosis. In our case, we'll be using three models to predict the detection of microcalcifications using mammography dataset to distinguish between negative and positive cases to breast cancer:

- K-Nearest Neighbors
- Random Forest
- Support Vector Machine

These approaches can be useful tools in aiding radiologists in the early detection of breast cancer, as micro calcifications can be an early indicator of the disease. However, it is important to note that the accuracy of the algorithms will depend on the quality and representativeness of the training data.

We applied a consistent set of steps across the three models, starting with:

- Visualization and analysis: this help to better understand the structure of the data and identify any patterns or relationship that might exist. As beginners in machine learning, we used various techniques to visualize the data (charts and graphs) and followed the same approach in all three algorithms: data exploration, feature selection, data visualization.
- Classification: once we have a good understanding of the data, we move on to the next step where we process the data to prepare it for the machine learning algorithm. This typically involves the following tasks:
  - Data cleaning: we remove any anomalies or outliers that we identified in the previous step.
  - Feature scaling: we scale the features so that they have a similar range of values. This can help improve the performance of our machine learning algorithms.
  - Data splitting: we split the data into training and testing sets so that we

can evaluate the performance of our machine learning algorithm.

- Evaluation of the performance: this is our final step where we typically use a variety of metrics to evaluate the performance of the machine learning models such as accuracy, precision, recall, F1 score.

Let's start with our first model:

4.1 K-Nearest Neighbors:

The K-nearest neighbors' algorithm, also known as KNN or k-NN, is a type of supervised learning classifier used to detect micro calcifications in mammography. To use the algorithm, a dataset of mammograms with known micro calcifications is first used to train the algorithm to recognize patterns and features associated with micro calcifications.

Once trained, the algorithm can be applied to new mammograms by comparing their features to those in the training set and identifying the K nearest neighbors based on feature similarity. The classification of the new mammogram is determined by the majority class among its K nearest neighbors. If the majority of its neighbors have micro calcifications, the algorithm will classify the new mammogram as a positive case and vice versa.

To predict the class of a given data point, the algorithm considers the classes of the 'K' nearest data points and chooses the class in which the majority of the 'K' nearest data points belong to as the predicted class. In our case, we have data points of Class A (-1) and B (0). We want to predict what the question mark (test data point: negative or positive case) is. If we consider a k value of 3 (3 nearest data points), we will obtain a prediction of Class B. Yet if we consider a k value of 6, we will obtain a prediction of either Class A or B. In brief, the value of K is important to consider.

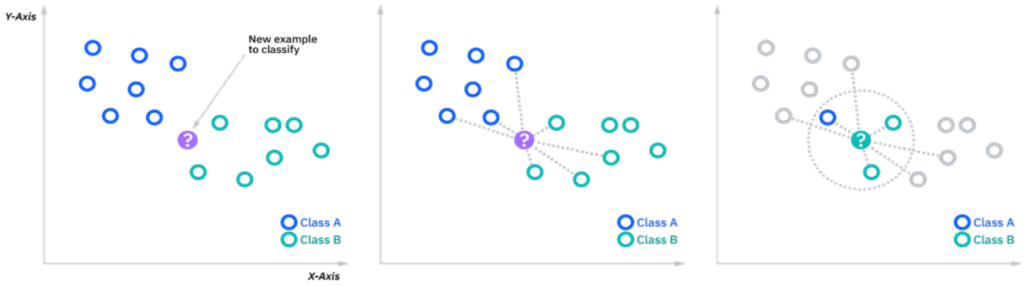
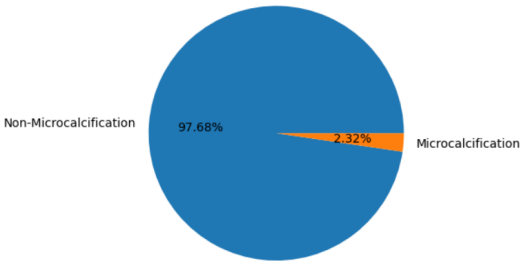


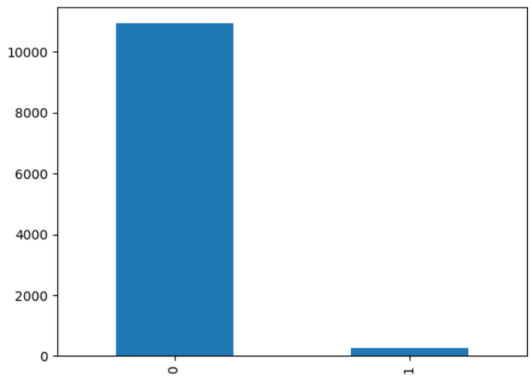
Figure 1: K-nearest neighbor approach demonstration.

We start by installing Scikit learn library, the most sufficient tool for machine learning modeling. Then we import the required libraries for our model (NumPy, Matplotlib, Pandas, Sklearn) and load the data using the pandas library. We were given six features: Area of object (in pixels), average gray level of the object, gradient strength of the object's perimeter pixel, root mean square noise fluctuation in the object, contrast, a low order moment based on shape descriptor. These features were considered the most relevant ones to pattern recognition.

	area of object	average gray level	gradient strengh	root mean square noise	contrast	low order	class
0	0.230020	5.072578	-0.276061	0.832444	-0.377866	0.480322	0
1	0.155491	-0.169390	0.670652	-0.859553	-0.377866	-0.945723	0
2	-0.784415	-0.443654	5.674705	-0.859553	-0.377866	-0.945723	0
3	0.546088	0.131415	-0.456387	-0.859553	-0.377866	-0.945723	0
4	-0.102987	-0.394994	-0.140816	0.979703	-0.377866	1.013566	0

Next, we move to data visualization and analysis using Matplotlib. Based on the information we have, we got 97.68% of non microcalcification cases and 2.3% of microcalcification ones. We noticed that our data is imbalanced. We'll be recovering this issue in the next part of our report.





Next step is setting features, normalizing, and standardizing data. It gives the data zero mean and unit variance to ensure that the information are in a suitable format for KNN. After that, we divide the dataset into a training test and a testing set. The training set is used to train the KNN model, while the testing set is used to evaluate the model's performance.

```
Train set: (8946, 6) (8946,)
Test set: (2237, 6) (2237,)
```

Then we implement our classifier using the sklearn library. At first, we train our algorithm with k=4. choose the value of k, the optimal one. We reserve a part of our data for testing the accuracy of the model then chose k=1. Then, we use the hyperparameter Grid search to optimize the value of K, the distance metric and the weight used. We start by supposing three distance metrics: Minkowski, Euclidean and Manhattan and two weights: Uniform and Distance.

```
grid_params = { 'n_neighbors' : [1,2,3,5,7,9,11,13,21],
                'weights' : ['uniform','distance'],
                'metric' : ['minkowski','euclidean','manhattan']}
gs = GridSearchCV(KNeighborsClassifier(), grid_params, verbose = 1, cv=3, n_jobs = -1)
g_res = gs.fit(X_train, y_train)
g_res.best_score_
g_res.best_params_
```

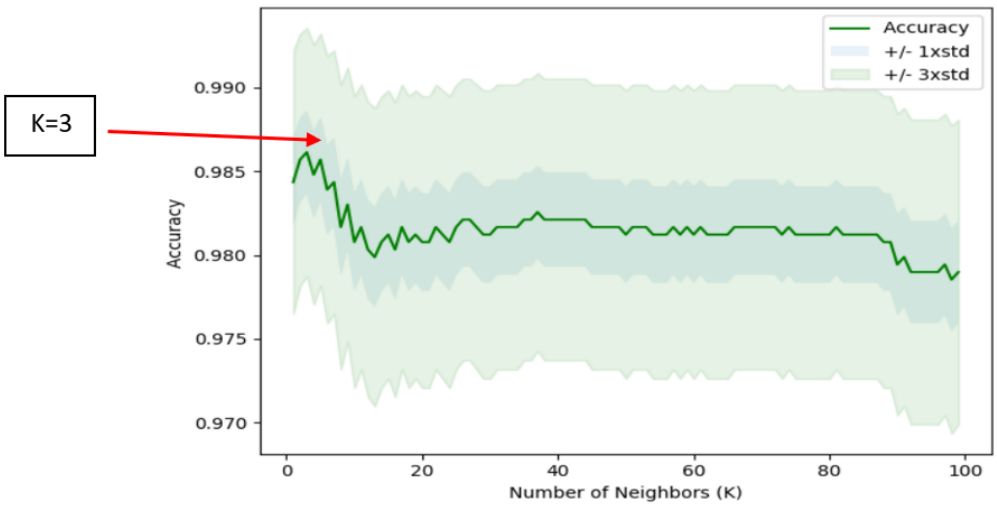
```
Fitting 3 folds for each of 54 candidates, totalling 162 fits
{'metric': 'manhattan', 'n_neighbors': 7, 'weights': 'distance'}
```

After running the grid search algorithm, we get the combination that produces the best evaluation metric.

```
[ ]
#Train Model and Predict
neigh = KNeighborsClassifier(metric= 'manhattan', n_neighbors= 7, weights= 'distance').fit(X_train,y_train)
neigh
```

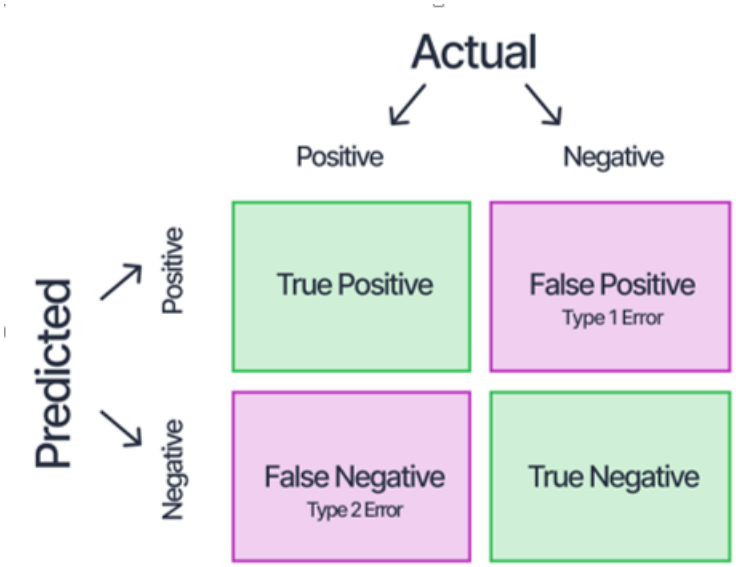
```
KNeighborsClassifier
KNeighborsClassifier(metric='manhattan', n_neighbors=7, weights='distance')
```

The next step consists of training the model and calculating the accuracy of prediction using all samples in our test set. By repeating this process and increasing the k, we can conclude that k=3 is the optimal value for our model



Now we'll be evaluating our model using different metrics such as: the F1 score, AUC, Recall, Precision, AP, the G-mean. Before defining each metric and its formula, let's talk about the confusion matrix. It is a table used to evaluate the performance of a classification model in machine learning. It shows the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) produced by a classification model.

A typical confusion matrix looks like this :



Now, let's define the metrics used in our model:

1. F1 score: a commonly used evaluation for binary classification that combines precision and recall into a single score. It's a single number that shows how well a binary classification model works. It considers both false positives and false negatives, and ranges from 0 to 1. A higher value means better performance. It is calculated as follows:

$$F - score\,(\%) = 2 \times \frac{\frac{TP}{TP+FP} \times \frac{TP}{TP+FN}}{\frac{TP}{TP+FP} + \frac{TP}{TP+FN}}$$

F1 score	Interpretation
> 0.9	Very good
0.8 - 0.9	Good
0.5 - 0.8	OK
< 0.5	Not good

2. AUC: The AUC, or "Area Under the Curve", is a metric that measures how well a binary classification model can distinguish between positive and negative classes. It represents the area under the receiver operating characteristic (ROC) curve, which shows the trade-off between sensitivity and specificity at different threshold levels. A higher AUC value indicates better model performance in distinguishing between the two classes.

AUC score	Interpretation
>0.8	Very good performance
0.7-0.8	Good performance
0.5-0.7	OK performance
0.5	As good as random choice

3. Recall: is the number of positive factors that were captured by the model. It shows how well the model is identifying positive cases, and a high recall score means that the model is detecting most of the positive cases. It is calculated as follows:

$$Recall(\%) = \frac{TP}{(TP + FN)}$$

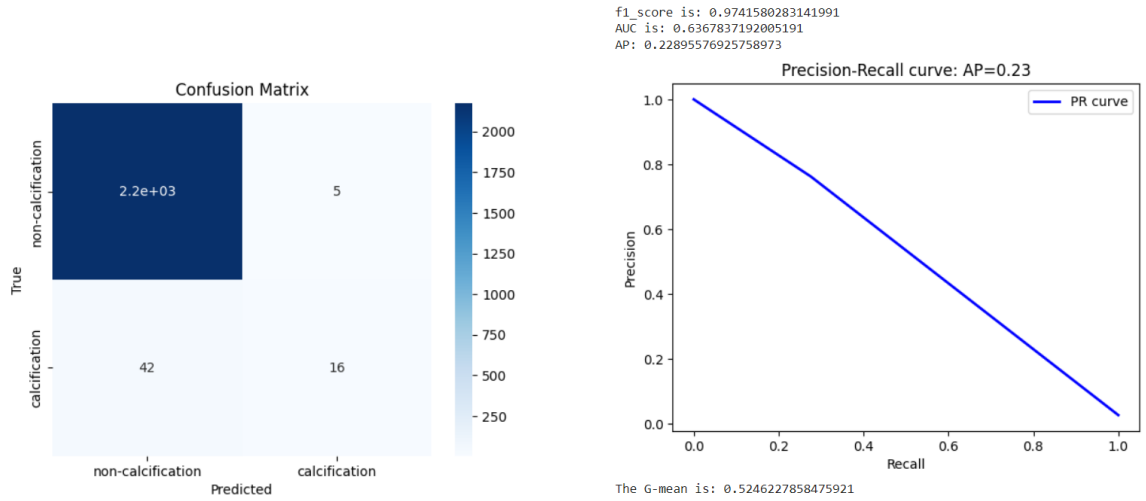
4. Precision: is the proportion of true positive cases out of all predicted positive cases. It shows how well the model is predicting positive cases, and a high precision score means that most of the predicted positive cases are actually true positive cases. It is calculated as follows:

$$Precision = \frac{TP}{TP + FP}$$

5. AP: is a number that measures the quality of a binary classifier by looking at both precision and recall. It considers how well the model identifies both positive and negative instances.

6. The G-mean: is a number that shows how well a binary classification model performs by considering both sensitivity and specificity. It is especially useful when there is an imbalance between positive and negative instances.

   In our K-NN model, we got the following confusion matrix and metrics values :



Conclusion

In our case, the model k-nearest neighbor has an accuracy score of 0.986142154671435 (very good performance), a F1 score of 0.9741580283141991 (very good performance) and an AUC score of 0.6367837192005191 (ok performance), which typically means that the model's overall performance is moderate. Our confusion matrix is biased and it's due to the highly imbalanced dataset. It means that our model has achieved a high accuracy by simply predicting the majority class all the time. To solve this problem, it is important to use metrics that focus on the performance of the minority class, and to consider techniques such as undersampling, oversampling and cost sensitive.

In fact, we cannot use cost sensitive with KNN. Cost-sensitive learning is not recommended for our model, the algorithm does not explicitly learn a model from the training data. Instead, it makes predictions based on the similarity between the input

instance and the training examples.

4.2 Random Forest:

Random Forest is a popular ensemble learning algorithm in machine learning that I used for classifications tasks. This model works by building multiple decision trees and combining them to make a final prediction. Each decision tree in the Random Forest is trained on a different subset of the data and with a random subset of features. This helps to reduce the variance in the model and prevent overfitting.

Random Forest has several advantages over other machine learning algorithms, including its ability to handle large datasets, high dimensionality, and noisy data. It is also a flexible algorithm that can be used for a wide range of applications and can handle both categorical and numerical data.

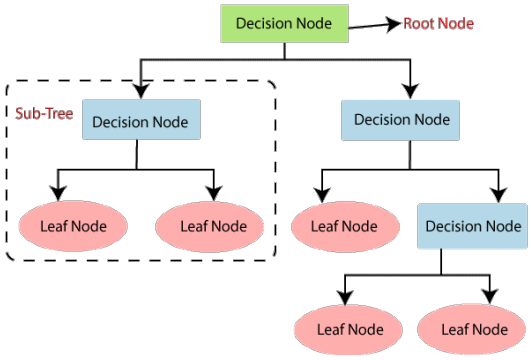


Figure 2: A decision tree

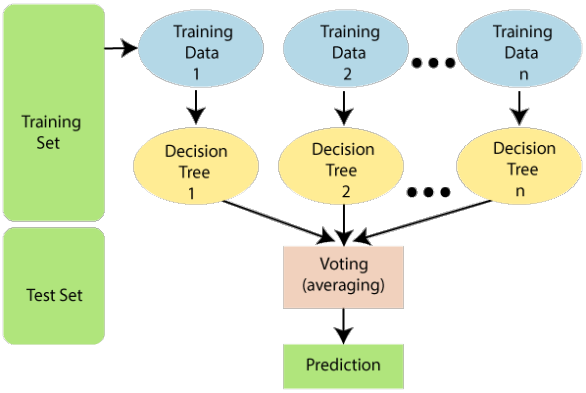
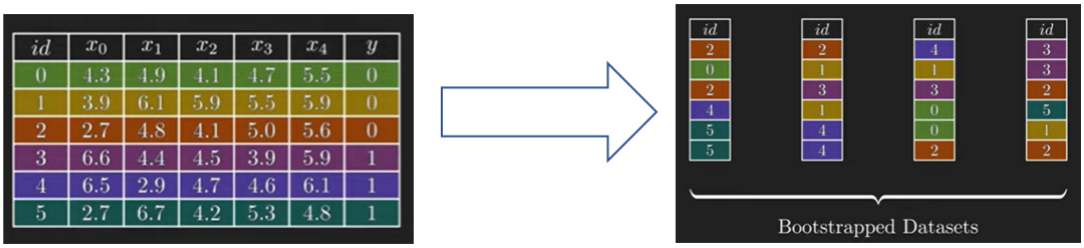


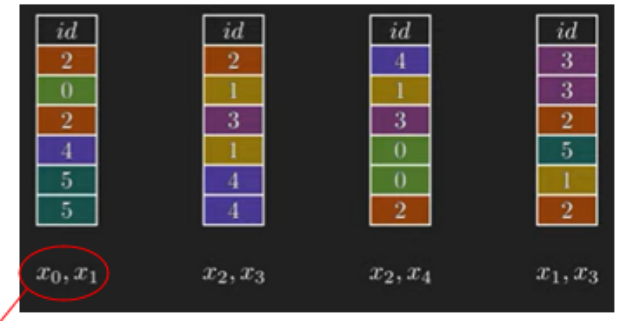
Figure 3: The Random Forest

Here's how the Random Forest algorithm works:

- **Choose a random subset of the training data.**



- **For each subset, grow a decision tree using the following steps:**
  a. Select randomly a subset of features from the total set of features.



Random selection of features for the first subset

b. Determine the best feature and split point using a criterion like information gain or Gini impurity. For example, if we are using information gain, we would like to choose the split that gives the maximum gain of information possible. The information gain is given by the formula:

$$IG = E(parent) - \sum_i w_i E(child_i)$$

E : is the entropy calculated by : H(x) = -$\sum P(x_i) \log_b P(x_i)$

With $P(x_i)$ is the probability of class I

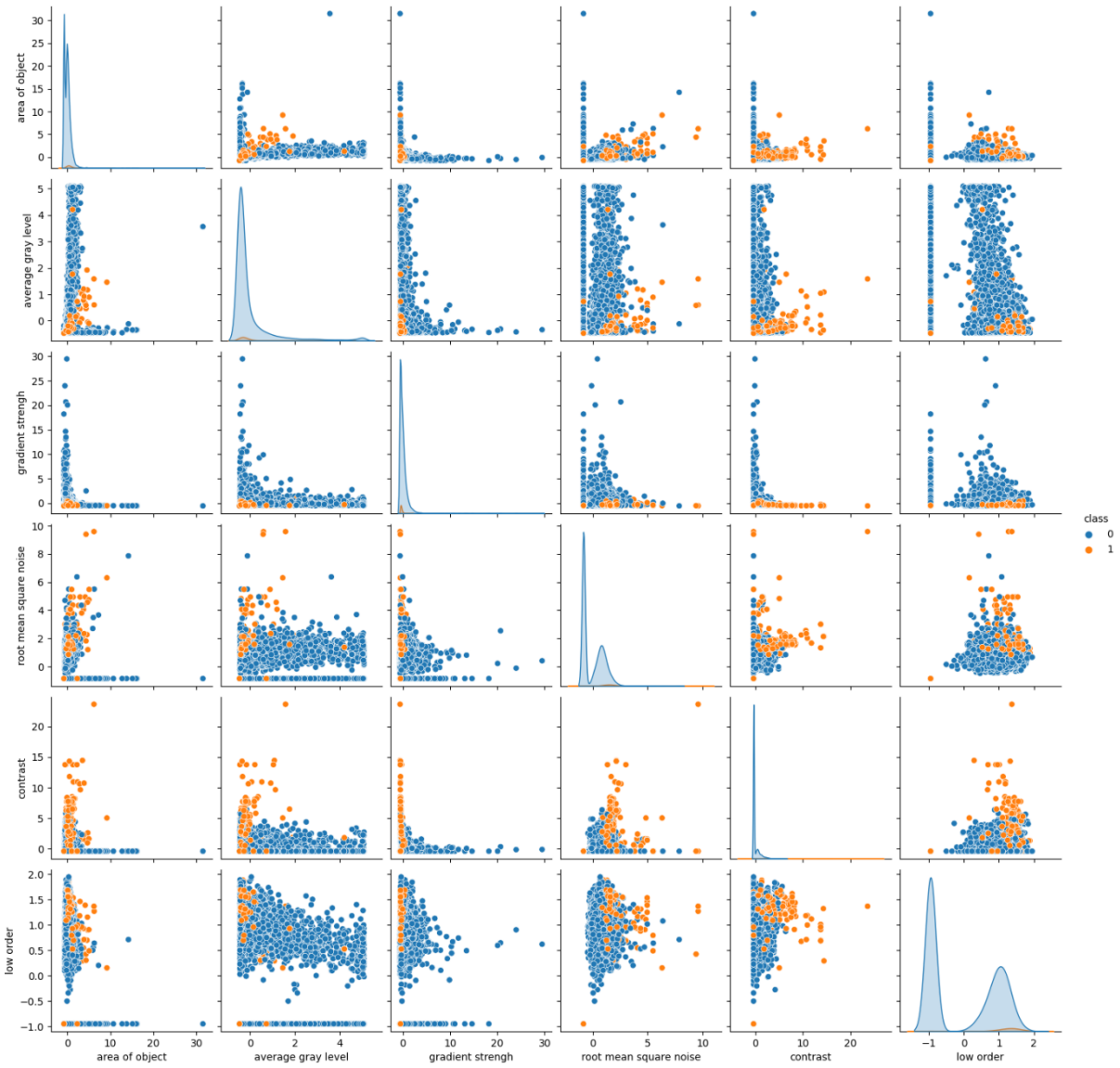
W_i: the weight of class I

c. Split recursively the data into smaller subsets based on the best feature and split point until a stopping criterion is reached (e.g., a maximum depth of the tree is reached, or there are no more features left to split on).

- **Repeat steps a and b for a pre-defined number of trees.**

For a new data point, make a prediction by aggregating the predictions of all the decision trees. For classification tasks, this can be done by majority voting. For regression tasks, this can be done by taking the average of the predictions.

We'll be using this algorithm for detecting the microcalcifications in mammography and let's start with our first step which is data visualization and analysis, this time we'll be visualizing the relationships between the multiple variables in the dataset, using the seaborn library as shown in the figure below:



Next, we move to data pre-processing by:

- **Separating dependent and independent variables (target and features):** X represents the features array, while Y is the target. The output is given by:

```
X= [[ 0.23001961  5.0725783  -0.27606055  0.83244412 -0.37786573  0.4803223 ]
 [ 0.15549112 -0.16939038  0.67065219 -0.85955255 -0.37786573 -0.94572324]
 [-0.78441482 -0.44365372  5.6747053  -0.85955255 -0.37786573 -0.94572324]
 ...
 [ 1.2049878   1.7637238  -0.50146835  1.5624078   6.4890725   0.93129397]
 [ 0.73664398 -0.22247361 -0.05065276  1.5096647   0.53926914  1.3152293 ]
 [ 0.17700275 -0.19150839 -0.50146835  1.5788636   7.750705    1.5559507 ]]
Y= [0 0 0 ... 1 1 1]
```

- **Splitting the dataset into training and test set:** by using the train_test_split function from sklearn. We choose the test size to be 20% of the data while the training data represents 80%. The shapes of the training and testing set are:

```
Train set: (8946, 6) (8946,)
Test set: (2237, 6) (2237,)
```

- **Scaling the data using StandardScaler ().**

Now we will fit the Random Forest algorithm to the training set. To do so, we will import the RandomForestClassifier class from the sklearn.ensemble library.

The classifier object takes below parameters:

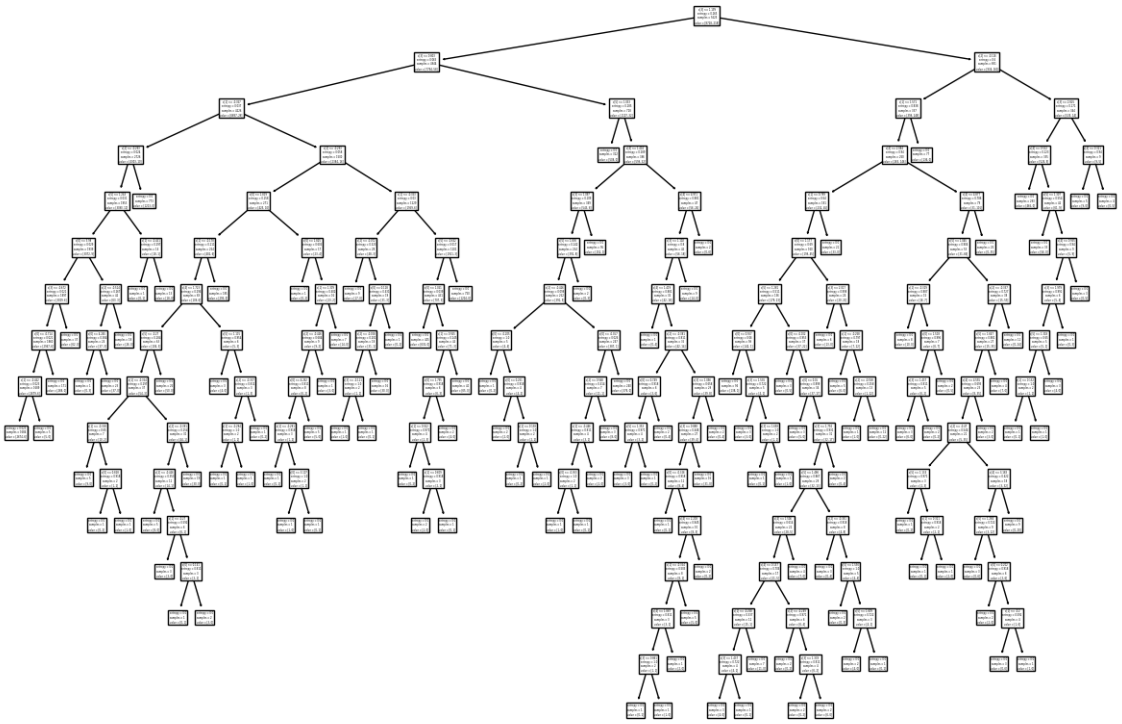
- n_estimators = The required number of trees in the Random Forest. The default value is 10. We can choose any number but need to take care of the overfitting issue.
- Criterion = It is a function used to analyze the accuracy of the split. Here we have taken "entropy" for the information gain.

Here is the code for it:

```
#Fitting Decision Tree classifier to the training set
from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(X_train, Y_train)
```

```
▾                    RandomForestClassifier
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

Although it is not required, we have the option to generate a visualization of one of the 10 decision trees (n_estimators=10) using the plot_tree function from matplotlib. Specifically, we will be visualizing the first decision tree. The resulting plot is displayed below:



Due to its size, the decision tree cannot be effectively visualized in a single figure. To enhance its readability, we constrain the depth of the tree by incorporating the max_depth parameter into our classifier. The following code snippet demonstrates this approach:

```
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy", max_depth=3)
classifier.fit(X_train, Y_train)
```

```
RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_depth=3, n_estimators=10)
```

Now we plot our tree, and we get :

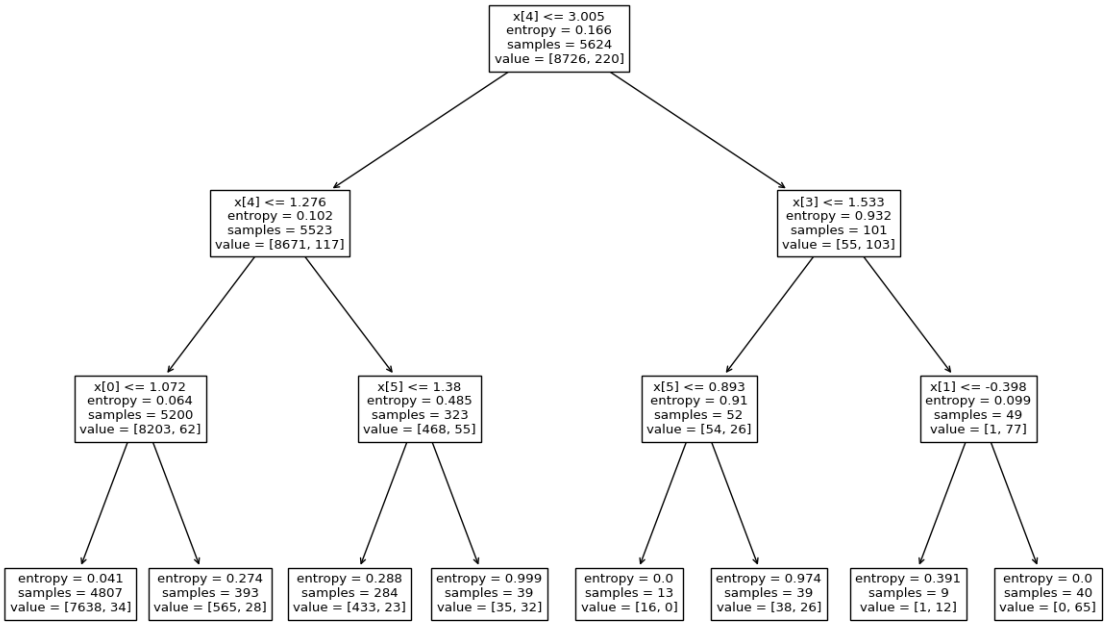


Figure 4: First decision tree from the Random Forest after limiting the depth

Since our model is fitted to the training set, we can move to the next step: predicting the test set result. For this step, we create a new prediction vector Y_pred, using classifier.predict.

Next step is creating the confusion matrix using the confusion_matrix function from sklearn.metrics to determine the correct and incorrect predictions. Then, we plot it:

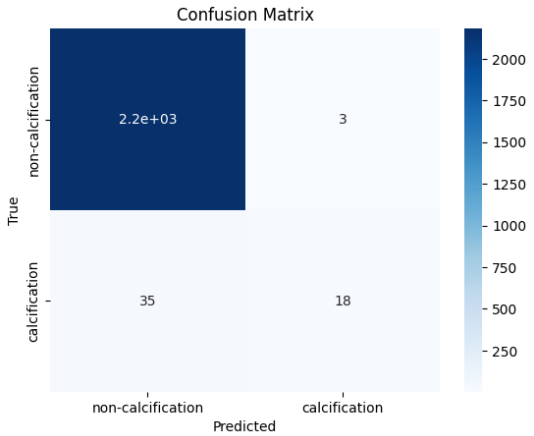


Figure 5: Confusion Matrix using the Random Forest model

The analysis reveals that the model made a total of 2.2e+03+18 correct

predictions and 38 incorrect predictions. Among the incorrect predictions, 3 correspond to non-calcification cases, while 35 correspond to calcification cases. Notably, the model appears to accurately predict the majority of non-calcification cases but struggles with calcification cases, with only 18 out of 53 being correctly classified. This outcome stems from the imbalanced nature of our dataset, where the minority class (Calcification) is not effectively captured by the model.

Then we move to evaluation, the performance of our model can be measured by mutliple metrics. We will use in our case the following ones: F1 score, AUC score, Average precision score, G-mean. These metrics are measured by the functions f1_score, roc_auc_score, average_precision_score, and we define a function for the Gmean metric based on its formula.

We get the following results:

1. F1 score: 0.9794018621376691
2. AUC: 0.6691245075679038
3. The G-mean: 0.5823711247065534
4. AP: 0.3067510756970191

We also plot the precision-recall curve:

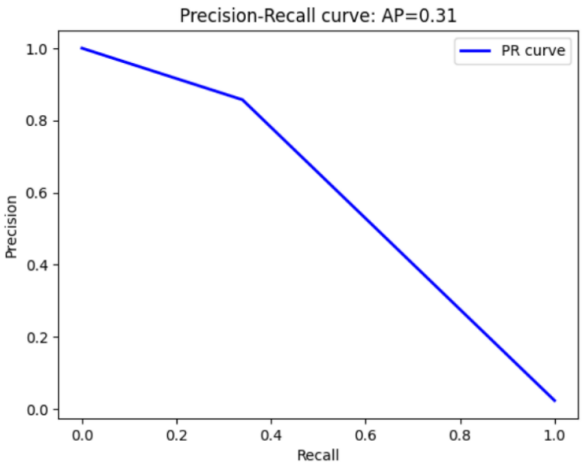


Figure 6: Precision-Recall curve using the Random Forest model

Conclusion

The performance metrics suggest that this classification model has achieved high precision and recall, as evidenced by the F1 score of 0.9794. This indicates that the Random Forest model excels in accurately identifying the positive class, which is a crucial aspect of the classification task at hand.

However, the AUC score of 0.6691 suggests that the model may not be performing well in terms of separating the positive and negative classes. This means that the model may have difficulty distinguishing between the two classes, which can lead to incorrect predictions.

The G-mean score of 0.5824 is also relatively low, which indicates that the model may not be effective at balancing sensitivity and specificity. This can lead to a high number of false positives or false negatives.

Finally, the average precision (AP) score of 0.3068 suggests that the model's performance is relatively low in terms of precision and recall trade-off, especially because the dataset is imbalanced.

In conclusion, while the model has high F1 score indicating good positive class identification, it needs to improve its AUC and G-mean scores to better differentiate between the two classes and balance sensitivity and specificity. Moreover, we know that the dataset is imbalanced, further investigation and optimization may be necessary to improve its performance.

4.3 Support Vector Machine:

SVM stands for Support Vector Machine, and it is a supervised learning algorithm used for classification and regression analysis.

The main idea behind SVM is to find a hyperplane (i.e., a linear decision boundary) that separates the different classes with the maximum margin. The margin is defined as the distance between the hyperplane and the closest data points from each class, also known as support vectors

Given a set of training examples (x_1 , y_1), (x_2, y_2), ..., (x_n, y_n), where x_i is the input vector and y_i is the corresponding output label (either +1 or -1), the objective of SVM is to find a hyperplane that separates the two classes with the maximum margin.

The hyperplane can be represented as $w.x + b = 0$, where w is the weight vector and b is the bias term. The distance between a point x and the hyperplane is given

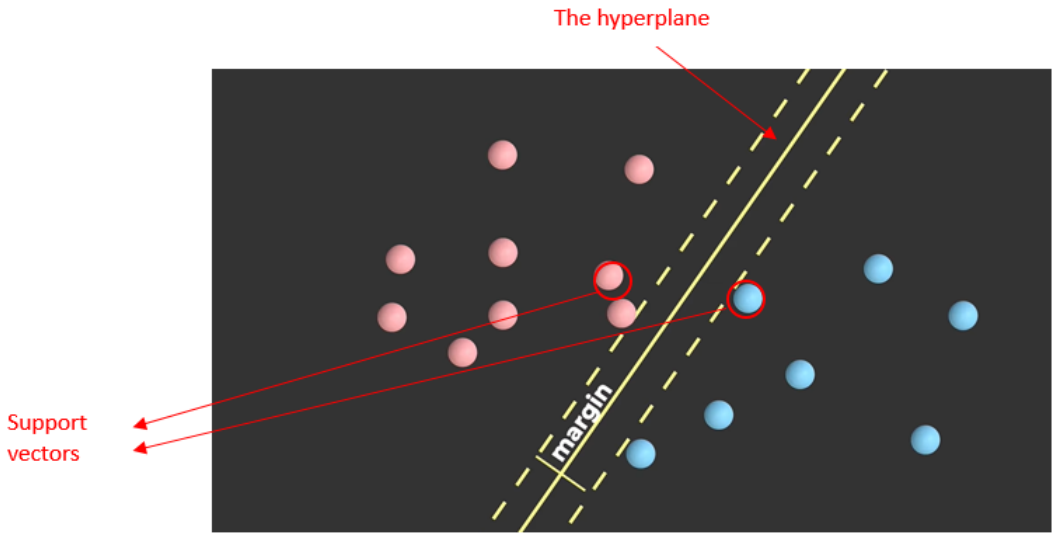


Figure 7: The SVM algorithm explained

by:

$$distance(x, w, b) = \frac{|w.x + b|}{||w||}$$

$||w||$ is the Euclidean norm of the weight vector.

The margin is given by the distance between the hyperplane and the closest points from each class:

$$margin(w, b) = \min_{xi \in \text{support vectors}} [distance(xi, w, b)]$$

The objective of SVM is to maximize the margin subject to the constraint that all data points are correctly classified:

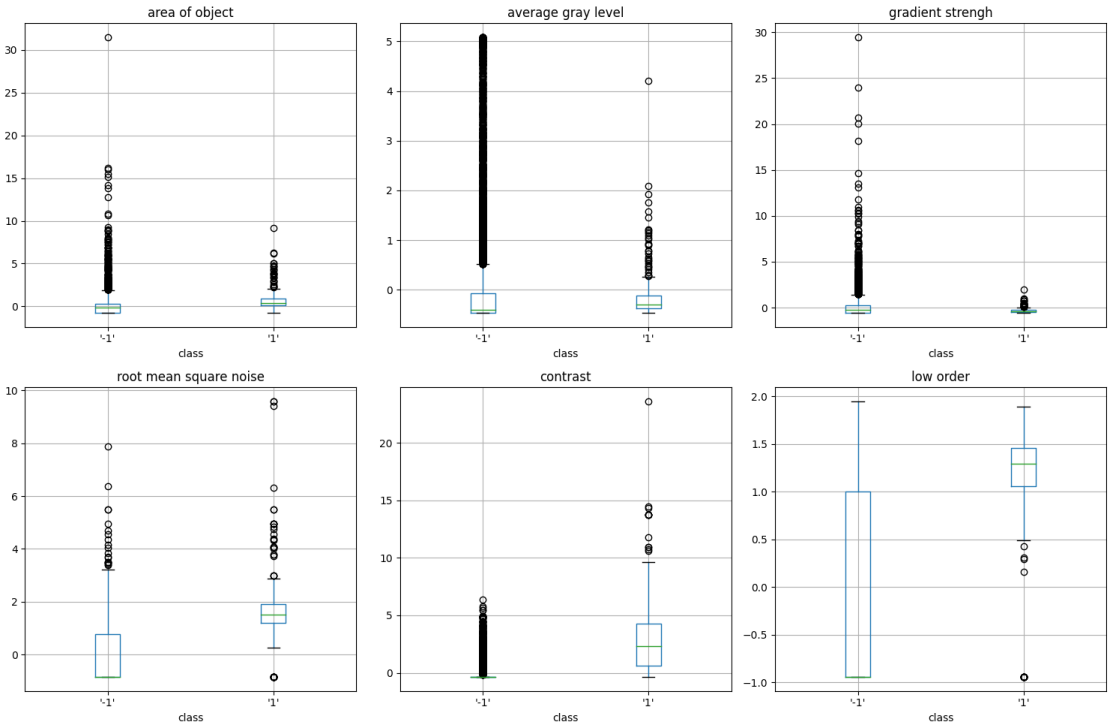
$$\text{maximize } margin(w, b) \text{ subject to } y_i(w.x_i + b) \geq 1 \text{ for all } i$$

This is a convex optimization problem, which can be solved using various methods such as gradient descent, quadratic programming, or the dual formulation.

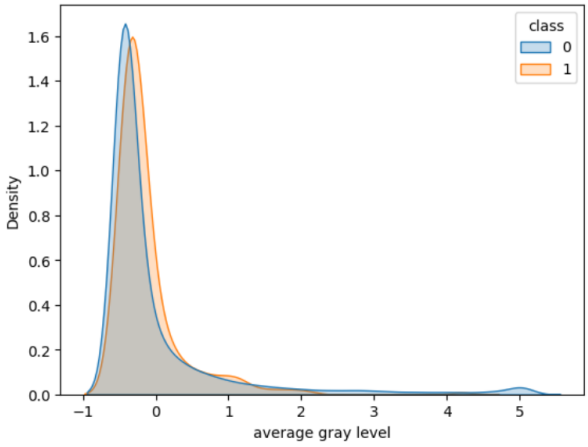
There are two types of SVM based on the type of decision boundary they use to separate the classes: linear SVM and non-linear SVM. The main difference between linear and non-linear SVM is the decision boundary they use to separate the classes. Linear SVM uses a straight line or a hyperplane, while non-linear SVM uses a curved line or a curved hyperplane. Linear SVM works well when the data is linearly separable, while non-linear SVM is used when the data is not linearly separable.

We'll be using this algorithm for detecting the microcalcifications in

mammography.Our first step is data visualization and analysis. To do so, we use a Kernel Density Estimation (KDE) Plot to visualize data. First, we explore boxplots of the distribution using each feature in order to determine which feature is best for the KDE Plot.



The 'average gray level' feature's boxplot presents a considerable number of outliers. It may be then interesting to explore the KDE plot using this feature:



We separate our dependent and independent variables and we split our dataset into train and test sets. Then we move to modeling, our second step.

The SVM algorithm offers a choice of kernel functions for performing its processing. Basically, mapping data into a higher dimensional space is called kernelling. The mathematical function used for the transformation is known as the kernel function, and can be of different types, such as:

- Linear

- Polynomial
- Radial basis function (RBF)
- Sigmoid

Each of these functions has its characteristics, its pros and cons, and its equation, but as there's no easy way of knowing which function performs best with any given dataset. We usually choose different functions in turn and compare the results. Let's just use the default, RBF (Radial Basis Function).

The code of our classifier is given below:

```
from sklearn.svm import SVC # "Support vector classifier"
classifier = SVC(kernel='rbf', random_state=0)
classifier.fit(X_train, Y_train)
```

```
SVC(random_state=0)
```

After being fitted, the model can then be used to predict new values, and we compare the result of Y_pred and Y_test to notice the difference. Next step is creating the confusion matrix:

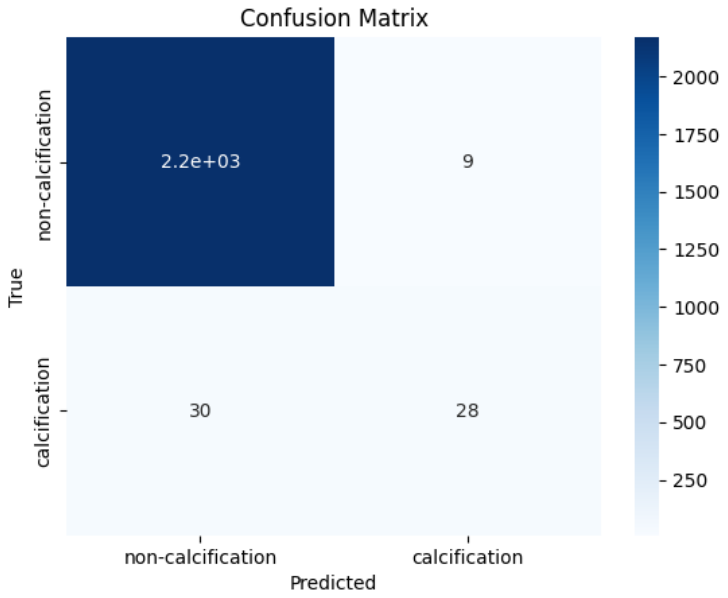


Figure 8: The confusion matrix using SVM

As we can see, there are 2.2e+03 + 28 correct predictions, and 30+9=39 incorrect predictions. We notice that almost every non-calcification case is well

predicted by our model, but the calcification cases aren't (just 28 out of 58 cases). This is due to the imbalanced dataset that we have. The minority class (Calcification) isn't well predicted.

The last step is evaluation. The performance of our model is measured by many metrics. We will use in our case the following ones: F1 score, AUC score, Average precision score, G-mean. We get the following results:

1. F1 score: 0.9806808172486536
2. AUC: 0.7393141428367964
3. The G-mean: 0.6933719534667628
4. AP: 0.38

We also plot the precision-recall curve:

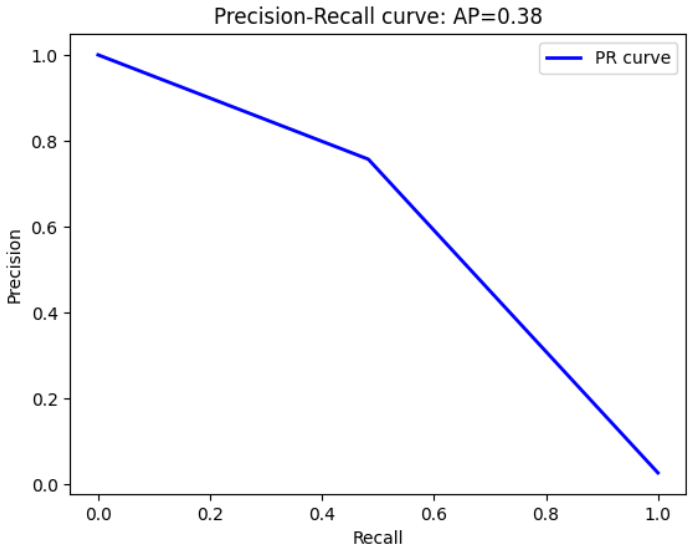


Figure 9: Precision-Recall curve using the SVM model

Conclusion

Based on the results provided, it appears that the SVM model has high accuracy, as indicated by the F1 score of 0.98. However, the AUC score of 0.74 suggests that the model may not be very effective at distinguishing between positive and negative instances. Additionally, the G-mean score of 0.69 indicates that the model may not perform well in detecting both positive and negative instances. The AP score of 0.38 suggests that the precision-recall tradeoff may not be well-balanced, and the model may have difficulty achieving high precision and recall simultaneously.

In conclusion, while the SVM model has high accuracy, its performance in terms

of AUC, G-mean, and AP scores indicates that it may not be the best choice for this task, and further analysis may be necessary to improve its performance.

Addressing the Challenges of Imbalanced Data: Strategies and techniques

Imbalanced data in machine learning refers to a situation where the distribution of classes in the training dataset is not uniform, meaning that one class has significantly more samples than the other classes. Imbalanced classification is a challenge in predictive modeling, it does lead to poor predictive performance for the minority class, which is typically more important. This problem is commonly found in cancer detection, such as detecting breast cancer from mammography scan.

When dealing with imbalanced classification tasks, relying solely on prediction accuracy as a performance metric can be inappropriate and even dangerous. This is because if one class makes up the majority of the data, achieving a high accuracy score can be easily done by always predicting that class, which provides no practical value. To avoid this, alternative metrics such as precision and recall scores should be used. These metrics focus on the performance of the model on the minority class, which is often more important in these types of problems. Furthermore, there are several techniques that can be used to efficiently address the problem of imbalanced data in machine learning, including:

- Resampling techniques: This involves either oversampling the minority class by replicating samples or undersampling the majority class by removing samples. Some commonly used techniques include Random Oversampling, SMOTE, Tomek links, and Random Under-sampling.
- Cost-sensitive learning: This approach assigns different costs to misclassifying samples from different classes, which can help the model prioritize correctly classifying the minority class.
- Ensemble methods: Ensemble methods combine the predictions of multiple models to improve overall performance. They can be particularly effective in addressing imbalanced data by giving more weight to the minority class.
- Synthetic data generation:This involves generating new data for the minority class using techniques such as Generative Adversarial Networks or Variational Autoencoders.

Choosing the appropriate technique depends on the specifics of the problem and the dataset. In this report, as beginners in machine learning, we thought it would be

a good idea to explore more deeply the domain of machine learning by trying multiple techniques and comparing their performance to select the best one, which are:

Oversampling: is a technique used to address the issue of imbalanced data in machine learning. It involves increasing the number of instances in the minority class to balance the data. One way to do this is by creating synthetic instances of the minority class. It is important to note that oversampling should be used carefully, as it can lead to overfitting and poor performance on new, unseen data. There are several methods to apply oversampling to a machine learning model, but one of the most commonly used is Synthetic Minority Over-sampling Technique (SMOTE). The SMOTE algorithm works as follows:

1. For each instance in the minority class, find its k nearest neighbors (k is a user-defined parameter).

2. Choose one of the k nearest neighbors randomly and create a synthetic instance at a random point between the two instances.
   (Repeat steps 1 and 2 until the desired number of synthetic instances is created.)

Undersampling: is a technique used in machine learning to balance imbalanced datasets. It involves reducing the number of instances in the majority class to match the number of instances in the minority class, thereby creating a more balanced dataset.

One common method of undersampling is random undersampling, which involves randomly selecting instances from the majority class until the desired balance is achieved. This method can be effective for small datasets, but it may not be as effective for larger datasets since it can result in the loss of important information.

Undersampling should be used with caution, as it can result in the loss of information and may not always improve the performance of classifiers. It is important to evaluate the performance of the classifier using cross-validation techniques to determine whether undersampling is appropriate for a given dataset.

Cost sensitive: is used when the costs of misclassification errors in a model are not equal for all classes. In traditional machine learning models, the cost of misclassifying each class is considered equal, but in real-world scenarios, the costs associated with misclassification errors can vary depending on the class being misclassified. Cost-sensitive learning aims to improve the performance of machine learning models by incorporating the costs of misclassification errors into the learning process. The goal is to reduce the overall cost of misclassification errors by assigning a higher weight to the minority class during the training process. This can be achieved

by modifying the loss function to consider the costs of misclassification errors, or by adjusting the class weights in the model to reflect the costs of misclassification errors.

One method to apply cost-sensitive learning is to use a technique called weighted learning, where each instance in the dataset is assigned, a weight based on its class label. The weight assigned to each instance is proportional to the inverse of the class frequency, such that instances from the minority class are given a higher weight than those from the majority class. This ensures that the model is trained on a more balanced dataset, with more emphasis on the minority class, which helps to reduce the cost of misclassification errors.

Now let's start with our first model:

4.4 KNN with Oversampling:

When using KNN with an imbalanced dataset, the algorithm tends to classify all observations as the majority class since there are more samples from that class. To address this issue, oversampling can be used to generate synthetic samples of the minority class so that KNN can make more accurate predictions on the minority class.

To implement it, we use the imblearn library, and its function RandomOverSampler. The code is given below:

```
from imblearn.over_sampling import RandomOverSampler
ros=RandomOverSampler(random_state=0)
X_train_resampled,Y_train_resampled=ros.fit_resample(X_train,y_train)
```

We visualize the class data in the pie diagram:

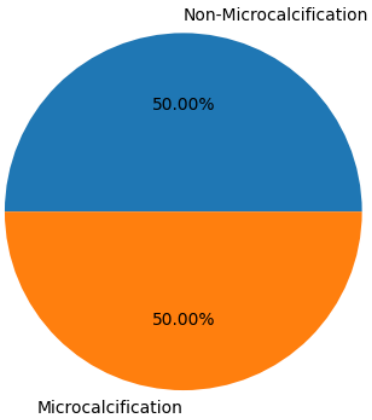


Figure 10: Pie diagram representing the 2 classes after oversampling

As we notice, the minority class contains more data points now. The two classes are equal. We draw our confusion matrix:

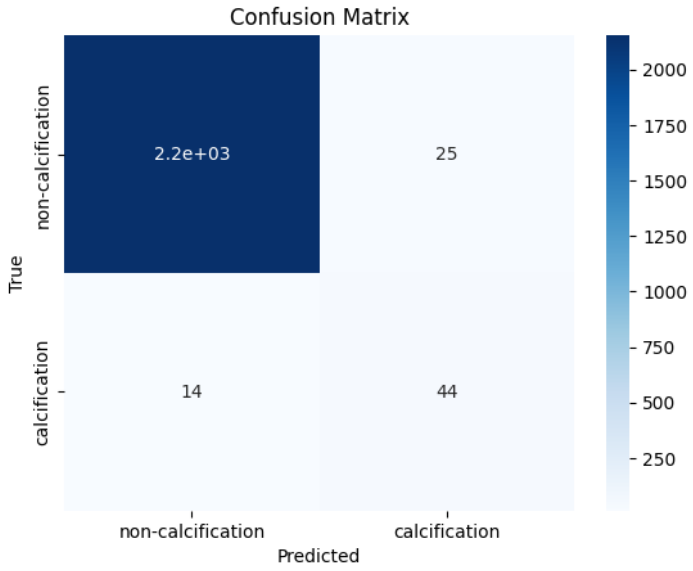


Figure 11: The confusion matrix using KNN with oversampling

As we notice from our confusion matrix, the number of correct predictions for our minority class has increased. We evaluate then our model using the previous metrics and we get:

i. F1 score: 0.9832988967378279

ii. AUC: 0.8735737684163883

iii. The G-mean: 0.8659774353576007

iv. AP: 0.49001650270081765

And we plot the precision-recall curve:

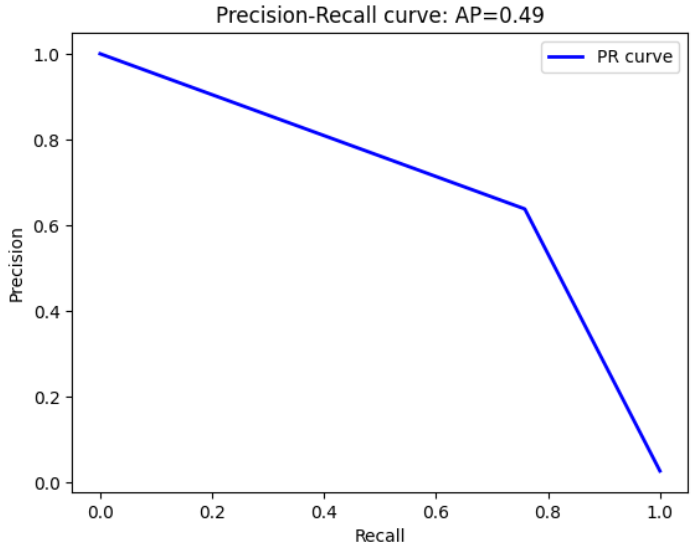


Figure 12: Precision-Recall curve using the KNN model with oversampling

Conclusion

Based on the results provided, it seems that the performance of the KNN model improved after applying the oversampling method. Specifically, the F1 score increased from 0.974 to 0.983, the AUC increased from 0.636 to 0.873, the G-mean increased from 0.524 to 0.860, and the AP increased from 0.230 to 0.490.

Comparing the performance of the KNN model without oversampling and the model with oversampling, it is clear that the latter performed better in terms of all the evaluation metrics

This indicates that the oversampling approach helped to improve the model's ability to accurately predict the minority class and rank instances in order of their likelihood of belonging to the minority class. This is a useful approach to our detection of microcalcifications.

4.5 Random Forest with Cost-Sensitive:

When using cost-sensitive learning in Random Forest, we want to minimize the overall cost of misclassification, which is the sum of the costs of misclassifying each class. We can achieve this by choosing the class weights that minimize a cost function, such as the weighted entropy.

By incorporating cost-sensitive learning into Random Forest, we can build models that are more effective in real-world scenarios where misclassifying certain classes is more costly than others. In our case, we use it to solve the problem of the imbalanced Data.

To code it, we add class_weight='balanced' to our classifier.

```
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy",class_weight="balanced")
```

Then, we plot the confusion matrix, and we get :

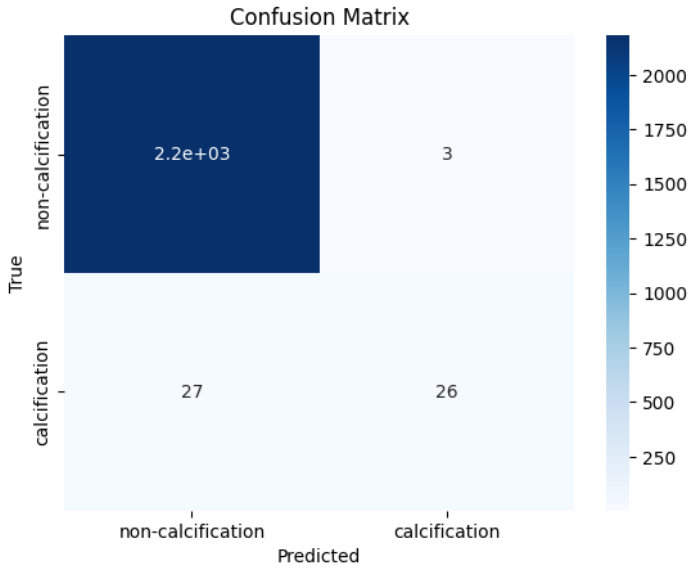


Figure 13: Confusion Matrix using the Random Forest model with cost-sensitive learning

As we notice from our confusion matrix, the number of correct predictions for our minority class has increased (26 out of 53), but we can judge that it didn't give the wanted results. Not even 50% of the minority class cases are well predicted. We evaluate our model using the previous metrics and we get:

i. F1 score: 0.9846632647790236

ii. AUC: 0.7445962056811113

iii. The G-mean: 0.6999229838263705

iv. AP: 0.4518875631895003

And we plot the precision-recall curve:

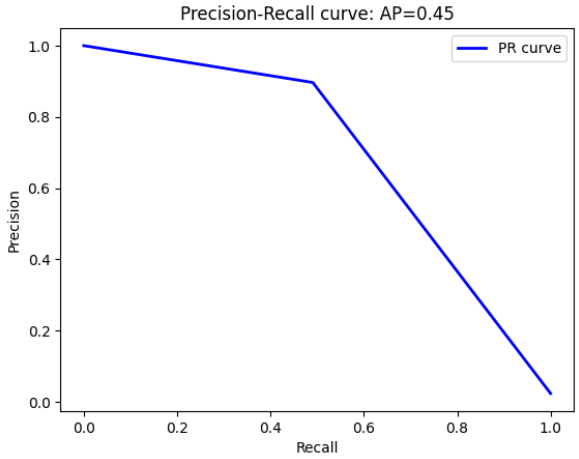


Figure 14: Precision-Recall curve using the Random Forest model with cost-sensitive learning

Conclusion

Based on the results provided, it seems that the performance of the random forest model improved after applying cost-sensitive learning. Specifically, the F1 score increased from 0.979 to 0.985, the AUC increased from 0.669 to 0.745, the G-mean increased from 0.582 to 0.700, and the AP increased from 0.307 to 0.452.

Comparing the performance of the random forest model without cost-sensitive learning and the model with cost-sensitive learning, the latter performed better in terms of all the evaluation metrics.

This indicates that the cost-sensitive learning approach helped to improve the model's ability to accurately predict the minority class and rank instances in order of their likelihood of belonging to the minority class. This technique may be a useful approach to our detection of microcalcifications.

4.6 SVM with Cost Sensitive:

By incorporating cost-sensitive learning into Random Forest, we can build models that are more effective in real-world scenarios where misclassifying certain classes is more costly than others. In this case, we use it to solve the problem of the imbalanced Data.

To code it, we add class_weight='balanced' to our classifier .

```
classifier = SVC(kernel='rbf', C=10, random_state=0, class_weight='balanced')
```

Then, we plot the confusion matrix, and we get :

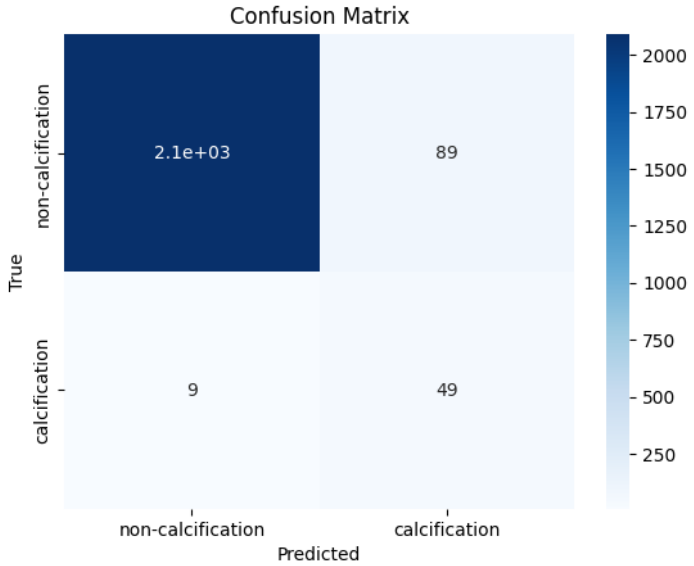


Figure 15: The confusion matrix using SVM

As we notice from our confusion matrix, the number of correct predictions for

our minority class has increased (49 out of 58), we can judge that it did give the wanted results. Most of the minority class cases are well predicted.

We evaluate our model using the previous metrics and we get the following results:

i. F1 score: 0.96472225478966

ii. AUC: 0.9019915810795841

iii. The G-mean: 0.9001783656746274

iv. AP: 0.30399825791172363

And we plot the precision-recall curve:

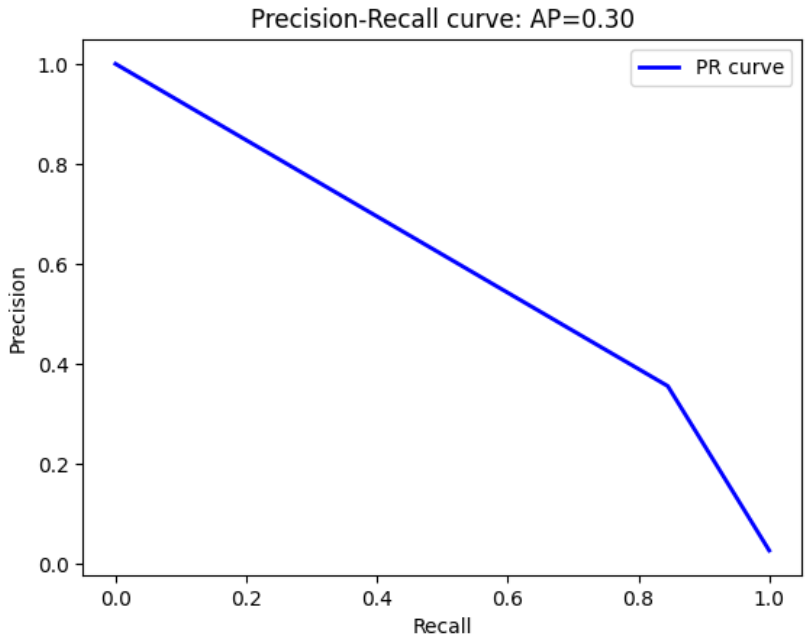


Figure 16: Precision-Recall curve using the SVM model with Cost Sensitive

Conclusion

In comparing the original SVM model and the cost-sensitive SVM model, it appears that the cost-sensitive model performed better in terms of AUC and G-mean, indicating better ability to distinguish between positive and negative instances and overall performance on both classes. However, the F1 score was slightly lower in the cost-sensitive model, indicating a potential trade-off between precision and recall.

Overall, the dataset is imbalanced and the cost of misclassifying the minority class is high, the cost sensitive SVM model may be a better choice.

However, it's important to consider the specific requirements and constraints of the problem at hand to determine the most appropriate approach. Additionally, other factors such as computational efficiency and interpretability may also be important

considerations in choosing the model.

5. Implemented solutions: Results and Discussions

When working with imbalanced datasets, we have concluded that the performance of a machine learning model can be negatively impacted. By balancing the dataset, we can improve the performance of the model. This was done through various techniques such as oversampling, undersampling or cost sensitive.

Comparing our three models that were trained on imbalanced data to the same models trained on balanced data using three different techniques, we did expect to see some significant differences in the results. Here are a few outcomes depending on the model used:

- Improved performance: Balancing the data can result in improved performance for the model. This is because the model is now trained on a more representative dataset, which can help it learn the patterns in the data more effectively.
- No change in performance: In some cases, balancing the data may not have a significant impact on the performance of the model. This could be because the model is already able to handle the imbalanced data or because the technique used to balance the data is not appropriate for the specific dataset.
- Decreased performance: In rare cases, balancing the data may actually decrease the performance of the model. This could be due to overfitting or other issues that arise when trying to balance the dataset.

In general, it is advisable to try out various methods of data balancing and evaluate their outcomes to determine the best approach for a specific dataset and model. This is precisely what we undertook to identify the most effective model for accurately detecting micro calcifications in mammography. The comparative chart below enables us to deduce which models are the most optimal: **KNN with oversampling** and **SVM with Cost Sensitive**

The ML Model / Metrics	F1 Score	AUC	The G-mean	AP	Correct Predictions	Incorrect predictions
KNN	0.9741580283141991	0.6367837192005191	0.5246227858475921	0.22895576925758973	2.2e+03 + 16	47
KNN with Oversampling	0.9832988967378279	0.8735737684163883	0.8659774353576007	0.49001650270081765	2.2e+03 + 44	39
Random Forest	0.9794018621376691	0.6691245075679038	0.5823711247065534	0.3067510756970191	2.2e+03 + 18	38
Random Forest with Cost-Sensitive	0.9846632647790236	0.7445962056811113	0.6999229838263705	0.4518875631895003	2.2e+03 +26	30
SVM	0.9806808172486536	0.7393141428367964	0.6933719534667628	0.38	2.2e+03 + 28	39
SVM with Cost Sensitive	0.96472225478966	0.9019915810795841	0.9001783656746274	0.30399825791172363	2.1e+03 +49	98

6. Conclusion and Perspectives

In conclusion, machine learning has shown great potential in detecting micro calcifications in mammography. Various machine learning models, including SVM, KNN, and Random Forest, have been utilized for this task, with promising results. However, the presence of imbalanced data in medical analysis requires special consideration, and appropriate adjustments should be made to address any bias towards the majority class.

Looking forward, there are several perspectives to consider for future research in this field. One perspective is to further improve the performance of machine learning models by utilizing more advanced techniques, such as deep learning, which has shown great success in medical image analysis. Another perspective is to explore the use of machine learning models in combination with other imaging modalities, such as ultrasound and magnetic resonance imaging to improve the accuracy and efficiency of micro calcification detection.

Moreover, the integration of machine learning models into clinical practice can have a significant impact on patient outcomes. The development of computer-aided detection systems, which utilize machine learning algorithms to assist radiologists in image analysis, has the potential to improve diagnostic accuracy, reduce false-positive rates, and ultimately improve patient outcomes.

Overall, the use of machine learning in detecting micro calcifications in mammography has shown great promise, and further research and development in this field can lead to significant advancements in medical analysis and clinical practice.