VRIJE
UNIVERSITEIT
BRUSSEL

Techniques of Artificial Intelligence

Project report

Anass Hamdi

1MA Biomedical Engineering

Contents

1. Introduction

In this project machine-learning models will be used to try and detect breast cancer based on X-ray images. On these images tumours are difficult to see due to the anatomy of the breast, thus micro-calcifications (MCs or micros) are used to detect breast cancer. MCs are white calcium deposits that are easy to see on X-ray images, unlike tumours, and can be indicative of cancer as there is a correlation between MCs and tumours. Malignant MCs are MCs that are in the neighbourhood of tumours, while benign MCs are not.

The project consists of first training a model that can accurately predict whether a micro is malignant or benign. Afterwards, patients are classified whether they have cancer based on the classifications of the micros. For this project numerous methods are used from different open-source libraries. Although no methods are specifically mentioned: *NumPy*, *Joblib*, *IPython*, *graphviz*, and *Matplotlib* is also used for this project [1][2][3][4][5].

2. Data and methods

2.1 Data

The data used to classify the micros contains quantitative values representing the shape and texture of the MCs. In total, the data consists of 3562 MCs spread over 96 patients and each micro has 150 properties computed. Each row represents a micro, and each column represents one of the 150 properties. The data only has labels on whether a patient has cancer. To be able to classify the micros for this project the assumption is made that all the micros for a patient have the same label as the label on whether the patient has cancer. Before splitting the data, it is best to visualize the distribution of MCs spread over the patient IDs combined with the visualization of benign and malignant MCs per patient ID.

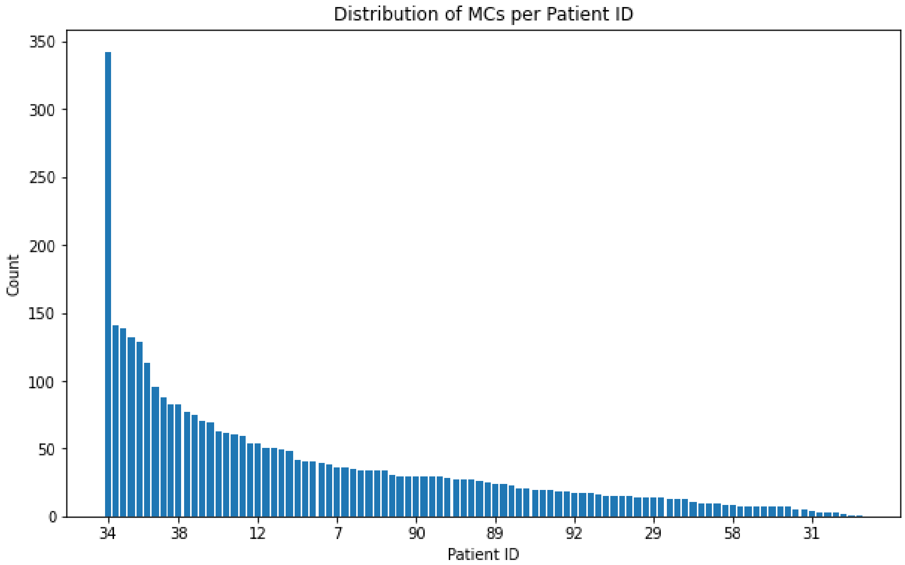


Figure 1: Distribution of MCs

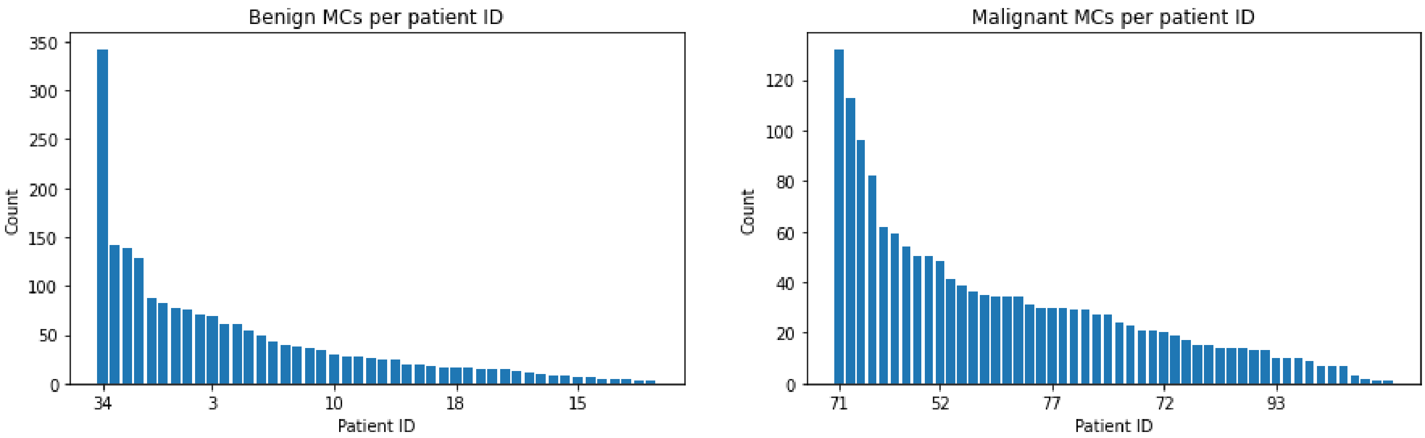


Figure 2: Benign and malignant MC count per patient ID

In Figure 1 and Figure 2 we see that the patient ID 34 has the highest number of MCs compared to all the other patients and causes the distribution of the benign MCs per patient ID to be different from the distribution of malignant MCs. This makes splitting the data a difficult task as there is a choice to be made. That choice is if one wants to have a balanced training set with a highly imbalanced test set composed of a limited amount of patients or the inverse. In this project the former is being used to have a varied and balanced training set.
The method used for splitting the data is *GroupShuffleSplit*[1].

The train and test split are a 79%/21% split when it comes to patient ID and 74%/26% split when it comes to the number of MCs/rows. The class fraction of the training set is very balanced and can be observed in Figure 3.

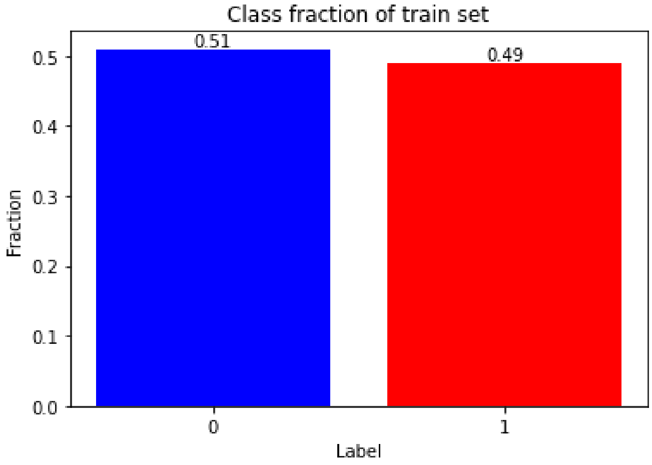


Figure 3: Class fraction of train set

[1] *https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GroupShuffleSplit.html*

2.2 Classifiers

The approach that is taken to solve the first part of the project, namely classifying the individual micros, consists of applying 5 different classifiers: *DecisionTreeClassifier*, *GaussianNB*, *SVC*[2]*, and MLP* from the *scikit-learn* library and lastly the *XGBClassifier*[3] from the *XGBoost* library [2][7]. The best two classifiers are then used in an ensemble method from *scikit-learn* called *VotingClassifier*[4] that will predict the class label with the method that has the highest sum of predicted probabilities.

The Gaussian Naïve Bayes (*GaussianNB*) classifier is a Naïve Bayes classifier that assumes the the features to have a Gaussian distribution, thus a Gaussian likelihood function is used for computing the Naïve Bayes.

The XGBClassifier was not seen in the lectures of TAI. This classifier is a form of ensemble learning where multiple decision trees are trained in sequence and every sequence the algorithm will improve the cases that were incorrectly predicted while trying to maintain the correct prediction of the other cases. Finally, the best decision trees are combined to vote on the final prediction.

For the second part of the project, the final model will be used to predict the MCs of the patients in the test set and then those predictions will be used in a thresholding approach. This approach involves calculating the mean of the labels, which gives the percentage of malignant MCs for a patient and finding a suitable threshold where the patients are classified as accurately as possible. This same thresholding approach was also used in the following paper [11]. The algorithm is a block of code that is not too complicated and can be found in Code 1. The predictions are grouped by patient ID to get the amount per label for each patient ID. The mean is calculated, which will give the percentage of malignant MCs. This is then used for thresholding that is defined as a lambda function where a percentage higher and equal to the threshold is classified as positive and negative otherwise.

This will give us the predictions for the patients and is then put in *accuracy_score* to compute the accuracy. The threshold values used are 5% to 59% in steps of 1%. The optimal threshold is the threshold that gives the predictions with the highest accuracy.

[2] *https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html*

[3] Learned about classifier from: *https://www.kaggle.com/code/alexisbcook/xgboost*
Source: *https://xgboost.readthedocs.io/en/stable/parameter.html*

[4] *https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html*

2.3 Hyperparameter tuning

Hyperparameter tuning and different pre-processing steps are used to improve each model and get better results.
The Decision Tree classifier is tuned according to two different parameters that both help the model avoid overfitting: *ccp_alpha*, and *max_depth* parameter. The *ccp_alpha* stands for cost complexity pruning alpha and will prune more nodes with increasing value for the *ccp_alpha* parameter. The *max_depth* parameter decides the maximum depth of the decision tree. Lastly, *GridSearchCV*[5] is also used for the DecisionTreeClassifier that fits multiple values for different hyperparameters and search for the best combination[2]. *GridSeachCV* also allows cross-validation when tuning the hyperparameters and is set to 3-fold cross-validation for the *DecisionTreeClassifier* and all other further uses.
The *max_depth* parameter is also used for the XGBClassifier to choose the maximum depth of the decision trees in the ensemble.

The last classifier for which a hyperparameter is tuned is the multi-layered perceptrons *(MLP)* with the *hidden_layer_sizes.* The *hidden_layer_sizes* parameter tunes the number of layers and perceptrons in each hidden layer. The size will be left to 1 hidden layer and the amount of neurons in this hidden layer will be varied from 1 to 150 neurons.

2.4 Preprocessing steps

The pre-processing steps that are applied for each classifier are the *MinMaxScaler*, *PCA*[6], *SelectKBest* [7]all from the *scikit-learn [1]* library and a self-defined class *PearsonFeatureReduction* that uses the *Pearson* correlation matrix method from *pandas* [9][10].
PCA is a method that will reduce the dimensions of the feature space by computing a number of principal component vectors that represent the directions in the features space where the variance is maximized when the data is projected on this principal component combined with minimizing the projection distances. This way the reduction of features is done in a way to minimize the error when removing a dimension in the feature space.
SelectKBest is a method that will select the best k number of features based on a scoring function of choice, which will be the mutual information between features defined in *scikit-learn* as *mutual_info_classif*.
The *PearsonFeatureReduction* is a class that will compute a number of lists with features that are all correlated to each other above a threshold of choice. From these lists, the feature with the lowest total correlation (sum of correlation with all other features) is chosen to keep and the other features removed as the feature kept will represent them.

[5] *https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html*
[6] *https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html*
[7] *https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html*

The above-described methods of feature reduction are combined with the *MinMaxScaler* as classifiers tend to perform better when all the data are scaled between the same relative range as opposed to having features ranging between large and small values.
The optimal threshold for *PearsonFeatureReduction* and number of features for *PCA* and *SelectKBest* is computed by *GridSearchCV* with 3-fold cross-validation.
All these pre-processing steps are passed in the instance of the *Pipeline*[8] class from *scikit-learn* that can sequentially apply in steps the feature reduction method, the *MinMaxScaler*, and fit the training dataset as final step. The steps before the estimator in *Pipeline* need to have an implementation of the *fit* and *transform* method and the final estimator will only implement *fit*.

2.5 Metrics

The metric used for tuning the hyperparameter and comparing models is the method *cross_val_score* [9]from *scikit-learn* that can be found in *get_mean_cv_rsme* where the mean of the 3-fold cross-validated root-mean-square errors are computed.

The performance of the model is visualized with the *plot_metrics* method that will plot the confusion matrix, with corresponding sensitivity, specificity, and the accuracy score with *accuracy_score*[10]. Furthermore, the ROC curve with AUC is also plotted with *roc_curve* [11]and *auc* [12]. The confusion matrix is plotted with *confusion_matrix* [13]and *ConfusionMatrixDisplay*[14].
The confusion matrix will plot a matrix where the columns represent the predicted label, and the rows represent the true labels. From this matrix we can get the true negatives (TN), false positives (FP), false negatives (FN), and the true positives (TP). The sensitivity shows us how accurate the model can predict the true positives and the specificity shows us how accurate the true negatives are predicted by the model.
The ROC curve and the AUC (Area Under (ROC) Curve) are a measure of how good the model can distinguish the different classes.

Lastly, the library *deepchecks* is used in the end to expand the evaluation of performance by having methods to evaluate the train-test split with *train_test_evaluation* [15]and the final model with *model_evaluation*[16][8].

[8] Found in https://www.kaggle.com/code/alexisbcook/pipelines
Source: *https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html*

[9] *https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html*

[10] *https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html*

[11] *https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html*

[12] *https://scikit-learn.org/stable/modules/generated/sklearn.metrics.auc.html*

[13] *https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html*

[14] *https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html#sklearn.metrics.ConfusionMatrixDisplay*

[15]*https://docs.deepchecks.com/stable/api/generated/deepchecks.tabular.suites.model_evaluation.html#deepchecks.tabular.suites.model_evaluation*

[16]*https://docs.deepchecks.com/stable/tabular/auto_tutorials/quickstarts/plot_quick_train_test_validation.htl*

3. Results

3.1 Classification of MCs

The best results per classifier are shown in Table 1 and the confusion matrix with roc curve for the classifiers in Table 1 can be found in the 7. Appendix under the following figures: Figure 6, Figure 7,Figure 8, Figure 9, and Figure 10.

Classifier	Metric	Values
Decision Tree Classifier with *ccp_alpha* = 0.01	Sensitivity	0.73
	Specificity	0.87
	Accuracy	0.84
	AUC	0.86
GaussianNB With *MinMax* scaling and *PCA: n_components* = 29	Sensitivity	0.68
	Specificity	0.82
	Accuracy	0.78
	AUC	0.84
SVC With *MinMax* scaling and *SelectKBest: k = 22*	Sensitivity	0.78
	Specificity	0.92
	Accuracy	0.89
	AUC	0.93
MLP with *MinMax* scaling and *hidden_layer_sizes* = (70,)	Sensitivity	0.84
	Specificity	0.89
	Accuracy	0.88
	AUC	0.92
XGBClassifier With *MinMax* scaling	Sensitivity	0.78
	Specificity	0.89
	Accuracy	0.86
	AUC	0.91

Table 1: Performance metrics of the best pipelines for the 5 classifiers

We can see from the table that according to the AUC and accuracy metric that *SVC* and *MLP* performed the best compared to the other classifiers. These classifiers are now used in the *VotingClassifier* to create the final model as an ensemble from the two best models. The performance results are in Table 2 with confusion matrix and ROC curve in Figure 4.

Classifier	Metric	Values
VotingClassifier with best pipelines for *MLP* and *SVC,* with weights 1 and 1.5 respectively	Sensitivity	0.84
	Specificity	0.91
	Accuracy	0.89
	AUC	0.94

Table 2: Performance metrics for ensemble with the 2 best classifiers (final model)

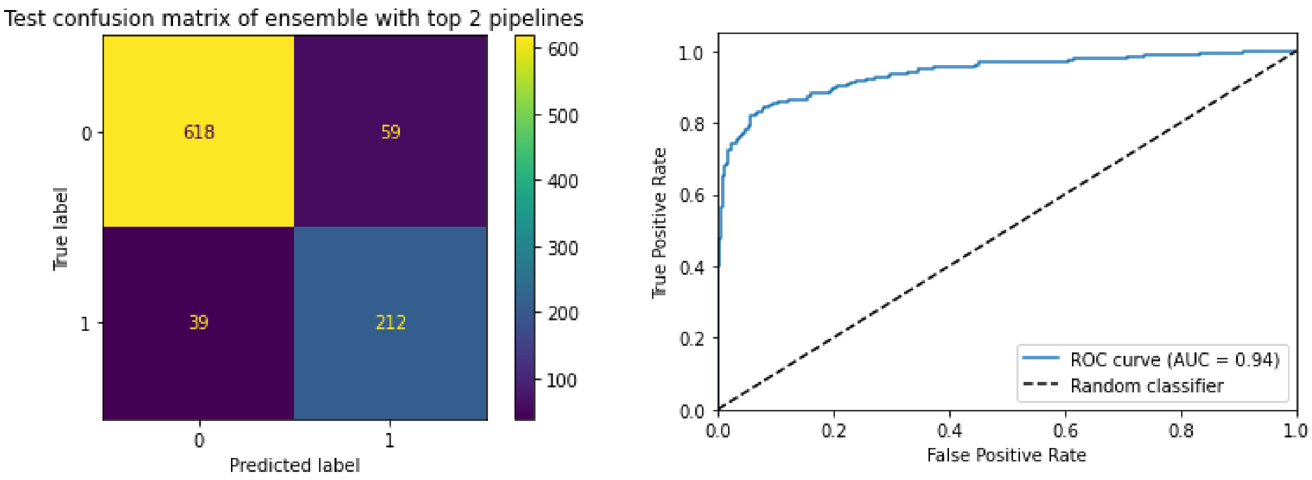


Figure 4: Confusion matrix and ROC curve of ensemble with the 2 best classifiers (final model)

The above performance metrics show very high results and give off the notion of a great model in predicting microcalcification. The specificity is at a very high value, which is probably attributed to the fact that the test set is very imbalanced and has more than twice as much as benign MCs compared to malignant MCs. The *model_evalution* from *deepchecks* passes all the different checks except for “weak segments Performance – Test Dataset”.

This evaluation reports that there exists a test segment in the data where the accuracy drops to 0.64 in comparison to the average score of 0.89 in Table 2. This low accuracy may be since the *model_evaluation* possibly does not consider the patient ID’s when splitting the data.

3.2 Classification of patients

The results from the thresholding algorithm show that the threshold of 14% is the lowest percentage that gives us the highest accuracy with the following performances in Table 3 and Figure 5.

Classifier	Metric	Values
Thresholding algorithm	Sensitivity	1.00
	Specificity	0.7
	Accuracy	0.85

Table 3: Thresholding algorithm classifying the patients

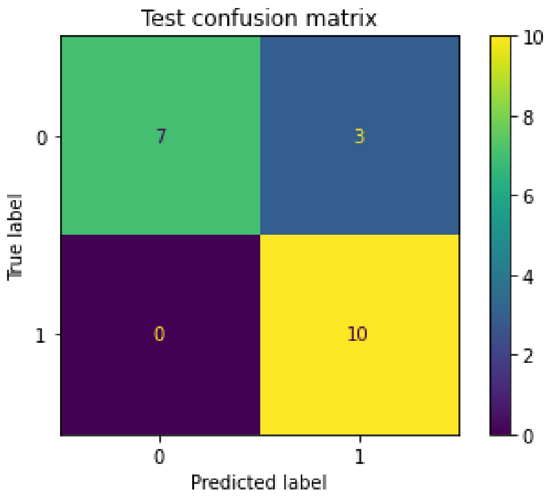


Figure 5:Confusion matrix of thresholding algorithm

The results show a sensitivity of 1.00, specificity 0.7, and accuracy 0.85. These are great values, but the number of patients is only 20 and chances are if there were more patients in the test set then the accuracy would probably decrease together with the sensitivity and specificity. On this test set it is possible to get a lower accuracy of 0.8 to get a higher specificity of 0.8 at the cost of the sensitivity decreasing to 0.8.

However, the model predicts whether a patient has cancer so we would want the sensitivity to be as high as possible such that the amount of false negatives is limited, while also keeping the specificity at an acceptable value as we would not want our model to only predict label 1.

4. Discussion

The hyperparameter tuning was rather simple in this project and limited to 1 or 2 hyperparameters. It may have been beneficial to have limited to 3 classifiers and gone more in depth for tuning the hyperparameters of the classifiers. When using the *cross_val_score* and the *GridSearchCV* the train set was passed in these methods, but it would have been better to use the complete dataset as these methods use cross-validation and will split the data in train and test folds.

The performance of the final model does extremely well on the test set, but when using *model_evaluation* method from *deepchecks* the performance results for the test set outperform the training set. A solution for this would be to run the model on different train-test splits or use *cross_val_score* with 5-fold cross-validation.

The results of the performance for the classifications of the patients feel dubious as the test set is very small with an amount of 20 patients. If there were more patients in the dataset or a better way to split the data, such that the test set has more patients. A possible solution was to have more patients in the test set and let the training set become more imbalanced. The imbalanced training set could be made more balanced by applying methods such as *SMOTE* that will create synthetic data to make the training set more balanced.

5. Conclusion

The classification of the MCs is done by the ensemble method *VotingClassifier* that uses the best classifiers, which are the *SVC* and *MLP* pipeline defined in Table 1. This model can accurately predict whether an MC is malignant or benign and shows a sensitivity of 0.84, specificity of 0.91, and accuracy of 0.89. Even though the test set is imbalanced, this final model can be a good measure for classifying the individual micros.

The classification of the patients uses a threshold algorithm that classifies a patient based on the percentage of malignant MCs over the total amount of MCs for the patient. The best threshold for the test set is 14% and shows a sensitivity of 1.00, specificity of 0.7, and accuracy of 0.8. These values are prone to change considerably as the test set only contained 20 patients, thus it is not possible to conclude that the threshold algorithm is a good measure to classify the patients.

6. References

[1] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020

[2] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[3] Fernando Pérez, Brian E. Granger, *IPython: A System for Interactive Scientific Computing*, Computing in Science and Engineering, vol. 9, no. 3, pp. 21-29, May/June 2007, doi:10.1109/MCSE.2007.53. URL: https://ipython.org

[4] Joblib Development Team. Joblib: running python functions as pipeline jobs, 2020.

[5] John Ellson, Emden Gansner, Yifan Hu, Stephen North, Don Caldwell, David Dobkin, Tim Dwyer, Eleftherios Koutsofios, Kiem-Phong Vo, Gordon Woodhull. https://gitlab.com/graphviz/graphviz/, August 2021

[6] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

[7] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’16, pages 785–794, New York, NY, USA, 2016. ACM.

[8] Shir Chorev, Philip Tannor, Dan Ben Israel, Noam Bressler, Itay Gabbay, Nir Hutnik, Jonatan Liberman, Matan Perlmutter, Yurii Romanyshyn, and Lior Rokach. Deepchecks: A library for testing and validating machine learning models and data. 23(265), 2022.

[9] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[10] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 –61, 2010.

[11] Brahimetaj, R., Willekens, I., Massart, A., Forsyth, R., Cornelis, J., Mey, J. D., & Jansen, B. (2022). Improved automated early detection of breast cancer based on high resolution 3D micro-CT microcalcification images. In BMC Cancer (Vol. 22, Issue 1). Springer Science and Business Media LLC. https://doi.org/10.1186/s12885-021-09133-4

7. Appendix

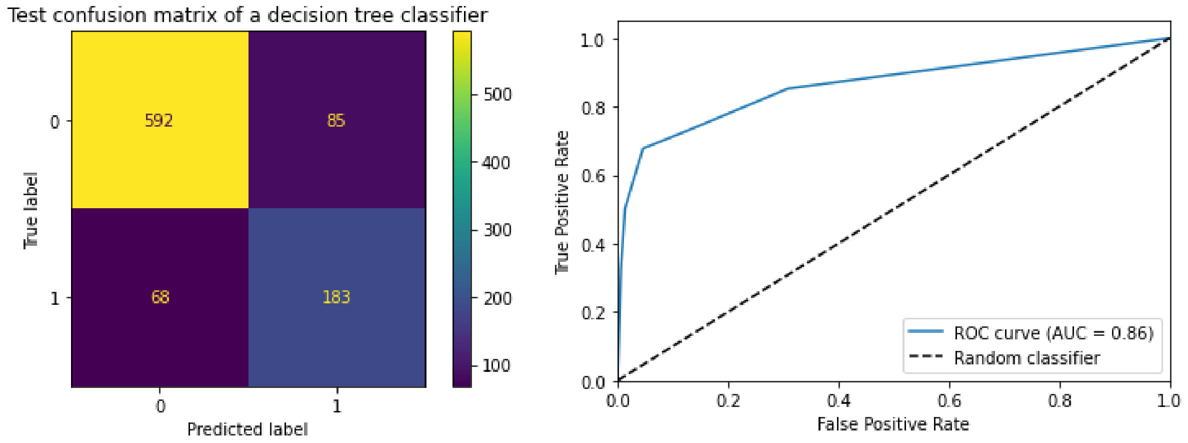


Figure 6: Confusion matrix and ROC curve for best Decision Tree Classifier pipeline

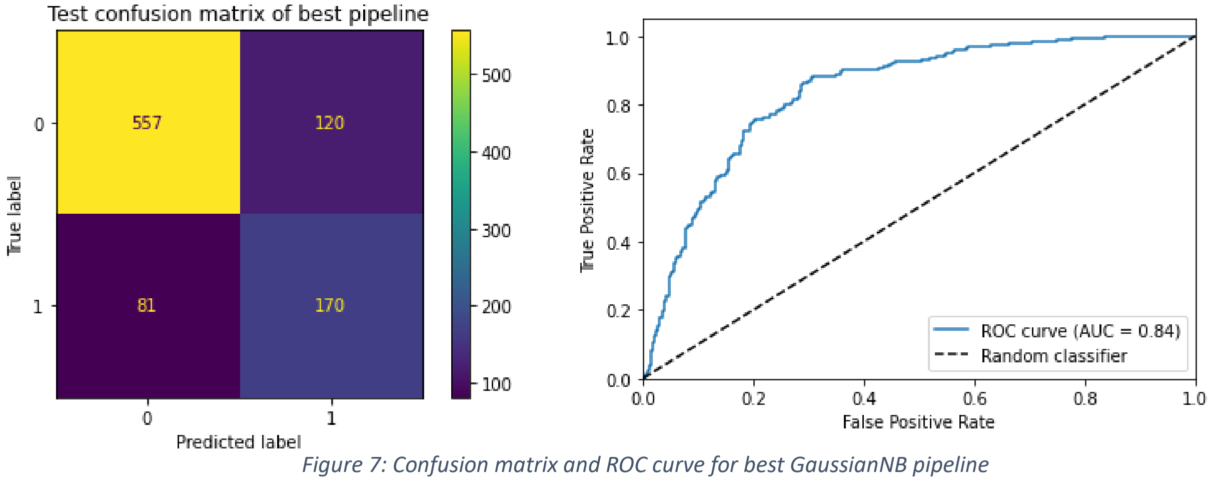


Figure 7: Confusion matrix and ROC curve for best GaussianNB pipeline

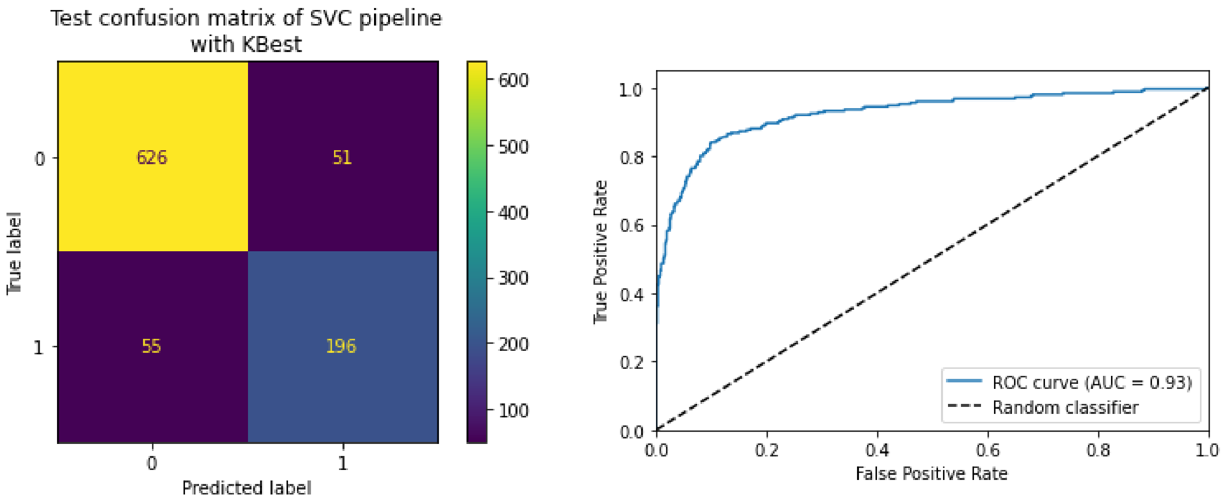


Figure 8: Confusion matrix and ROC curve for best SVC pipeline

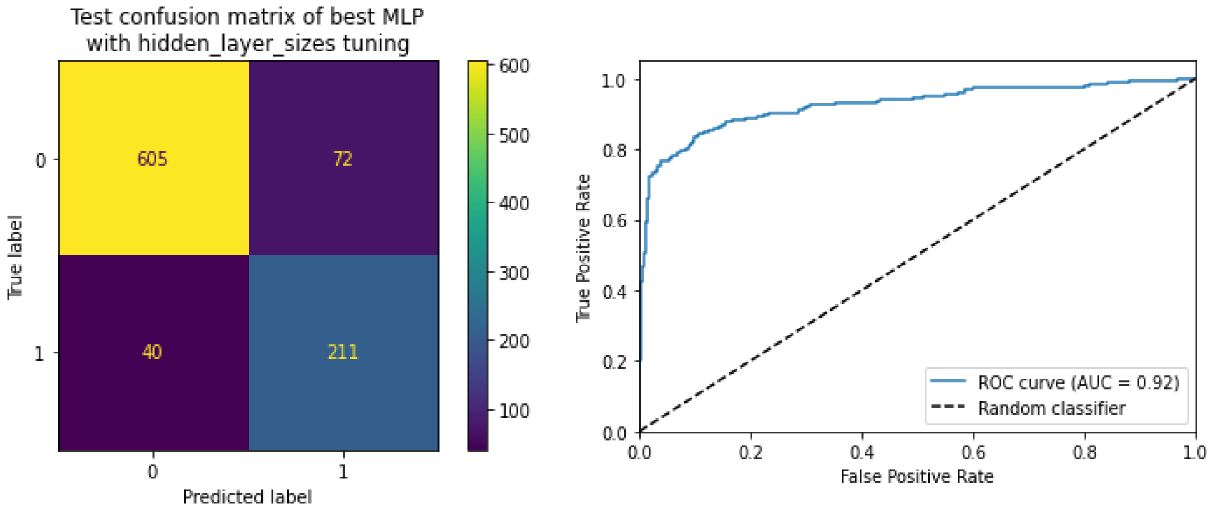


Figure 9: Confusion matrix and ROC curve for best MLP pipeline

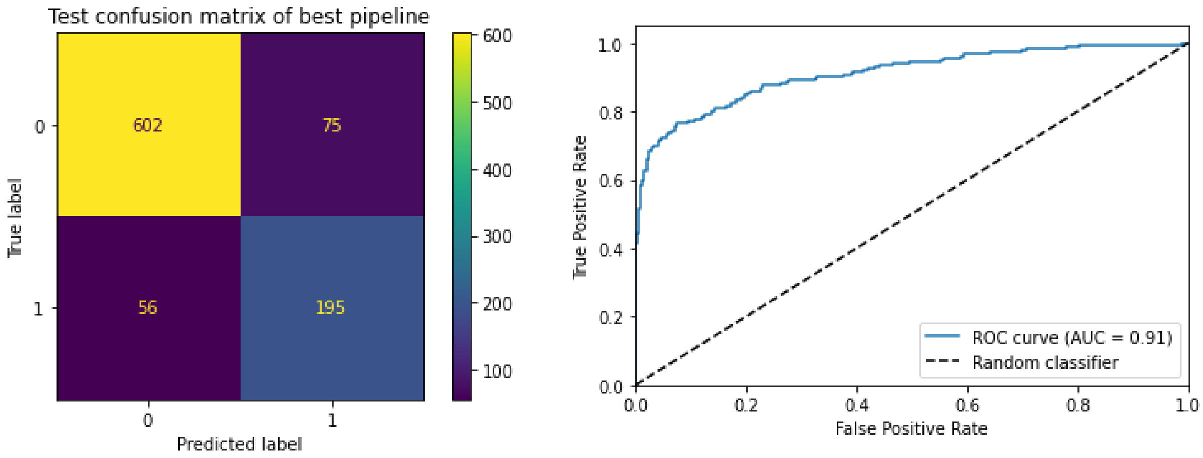
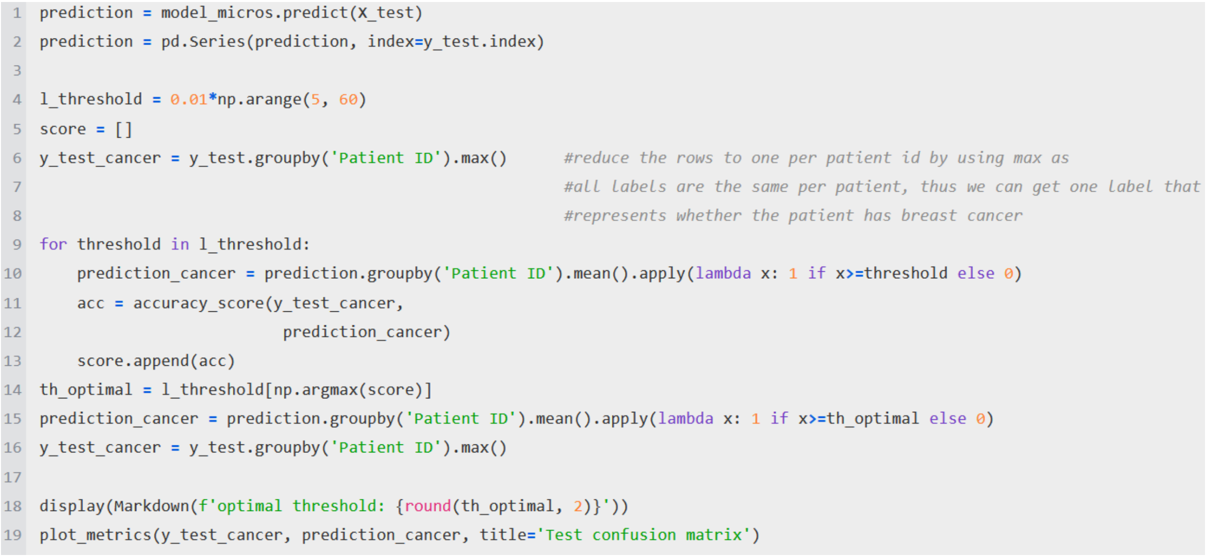


Figure 10: Confusion matrix and ROC curve for best XGBClassifier pipeline



```python
prediction = model_micros.predict(X_test)
prediction = pd.Series(prediction, index=y_test.index)

l_threshold = 0.01*np.arange(5, 60)
score = []
y_test_cancer = y_test.groupby('Patient ID').max()      #reduce the rows to one per patient id by using max as
                                                        #all labels are the same per patient, thus we can get one label that
                                                        #represents whether the patient has breast cancer
for threshold in l_threshold:
    prediction_cancer = prediction.groupby('Patient ID').mean().apply(lambda x: 1 if x>=threshold else 0)
    acc = accuracy_score(y_test_cancer,
                          prediction_cancer)
    score.append(acc)
th_optimal = l_threshold[np.argmax(score)]
prediction_cancer = prediction.groupby('Patient ID').mean().apply(lambda x: 1 if x>=th_optimal else 0)
y_test_cancer = y_test.groupby('Patient ID').max()

display(Markdown(f'optimal threshold: {round(th_optimal, 2)}'))
plot_metrics(y_test_cancer, prediction_cancer, title='Test confusion matrix')
```

Code 1: Thresholding for classification of patients