Pulmonary Nodules Data Analysis and Classification

*Shliakhtin, Evgenii*

*Ussenova, Assel*

*Zhang, Rongrong*

*Ojeda Azocar, Verena Maillen*

# Introduction

Lung cancer ranks as the most fatal form of cancer on a global scale [1]. A lung nodule, also known as a pulmonary nodule, is a small lump detected on an X-ray or CT scan of the chest. Doctors and specialist nurses assess the size of a nodule and its appearance to decide if it needs follow-up or further assessment [2].

Several studies showed that screening rounds of high-risks subjects using low-dose computed tomography reduces the mortality thus increasing the importance of tools that can automatically detect the nodules and categorize them properly to help experts in the early detection of lung cancer. Classical and deep learning techniques have been used to perform automatic classification of malignant pulmonary nodules with different results [3].

In this work, we investigated whether some classical models (i.e. decision trees, SVMs, K-Nearest Neighbor) can be used to assist in the automatic diagnosis of malignant pulmonary nodules. We train and evaluate the model on a subset of 1000 CT images from the LUNA16 dataset.



Image 1. Classification standard workflow.

For this work, the workflow presented in Image 2 was performed. First, the full dataset was read using SimpleITK. Then, features were extracted using the Pyradiomics library. After the feature extraction step a CSV file was generated, this data was splitted in train and test subsets with 70% for training and 30% for testing. The scikit-learn library was used for the training and classification, the selected models were trained using the training data subset and therefore tested using the test data subset. In the final step, accuracy, sensitivity and specificity metrics were calculated and used to evaluate and compare the different models.

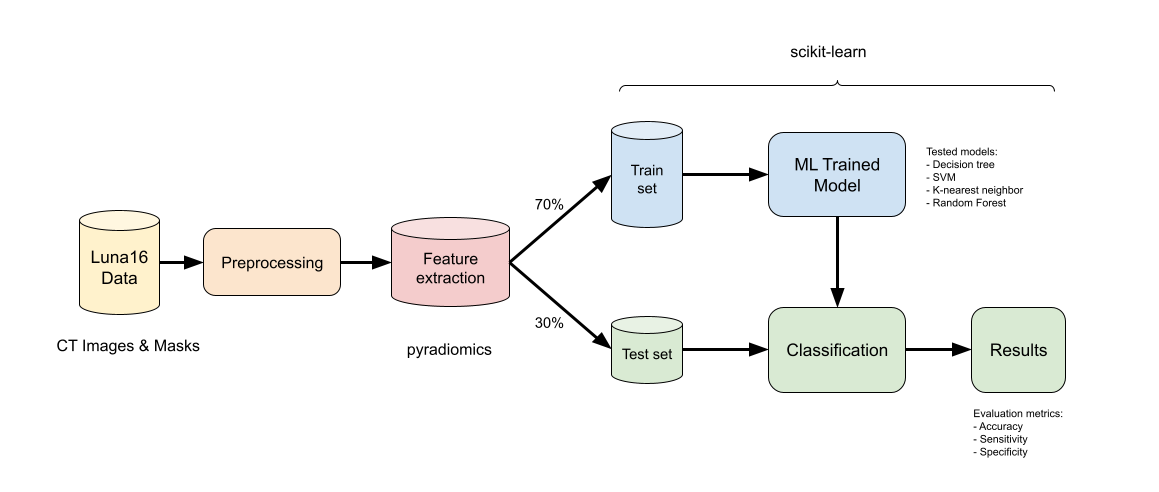


Image 2. This project classification workflow.

Methodology

In this section, the data set, the algorithms are described, the experiments performed are documented and the results are contextualized.

**Dataset**

A dataset of over 1000 computed tomography (CT) scans with pulmonary nodules was used to train and evaluate the classification models selected in this work [4]. The images of the dataset are a subset of the LUNA16 [5] dataset originally used for a pulmonary nodule detection challenge that is available online. Examples of the images can be seen in Image 3.

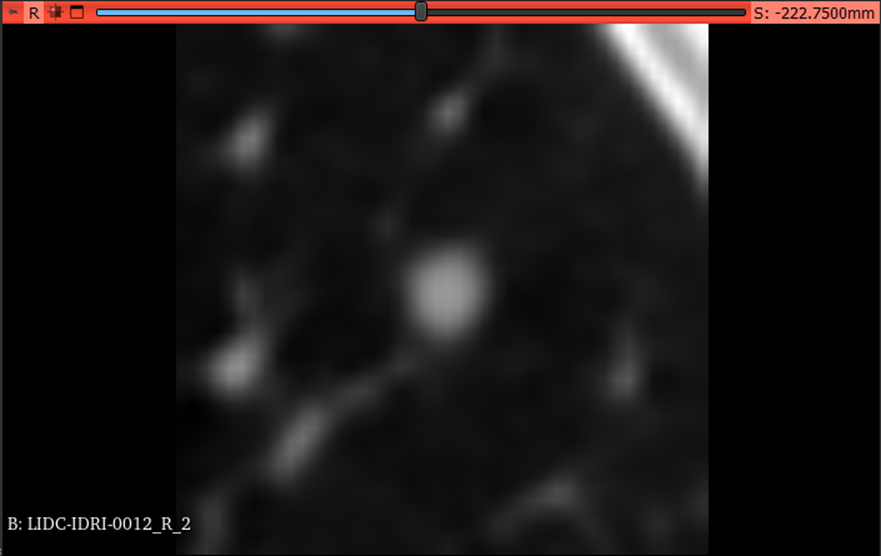
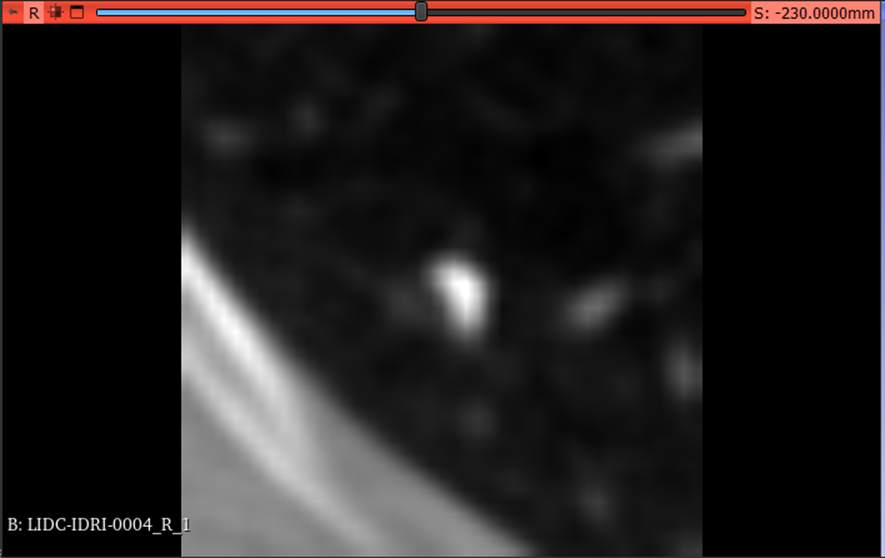
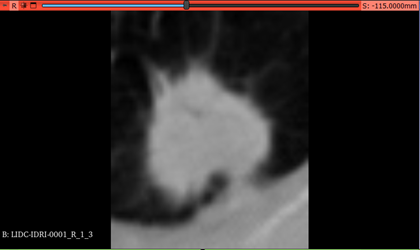


Image 3.

1. A Highly Unlikely malignancy nodule diagnosed as malignant.
2. A Highly Suspicious malignancy nodule diagnosed as malignant.
3. A Highly Unlikely malignancy nodule diagnosed as benign.
4. A Moderately Unlikely malignancy nodule diagnosed as benign.

For each image of the dataset the corresponding annotations and the manual segmentation from expert radiologists was also used. The annotations include the diagnosis that indicates if the nodule is **benign (0)** or **malignant (1)**, but they also indicate other aspects of the nodule such as calcification, internal structure, lobulation, etc. Additionally, the annotations also include the "malignancy" of the nodule that can take five different categories (assigned with value 1 to 5) and is used to determine the diagnosis. The malignancy types included in the dataset are:

* Highly Unlikely (1)
* Moderately Unlikely (2)
* Indeterminate (3)
* Moderately Suspicious (4)
* Highly Suspicious (5)

**In case the “malignancy” value is greater than 3, it is classified as malignant. Otherwise, it is considered benign.**

Examples of the different malignancy categories and their corresponding masks are shown in the following images.

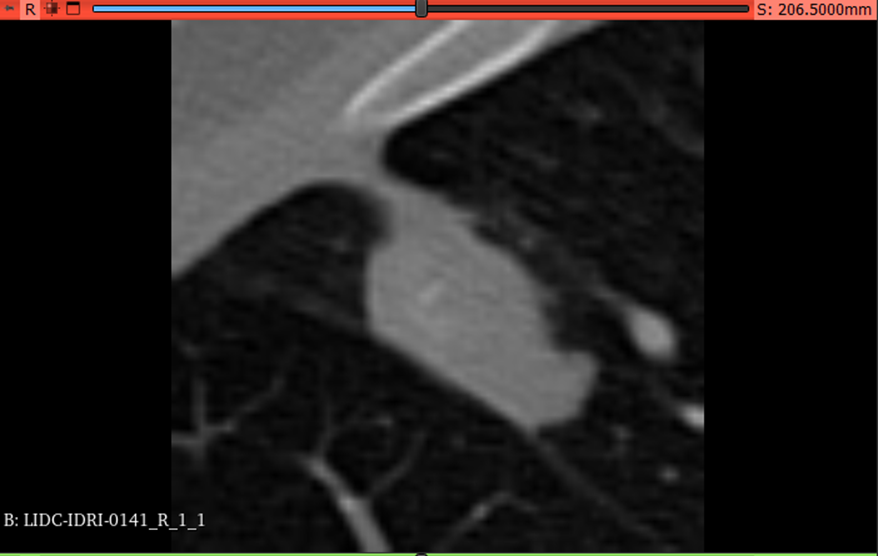


Image 4. Patient: 0141 - Nodule: 1. Malignancy: 1. Diagnosis: Malignant.

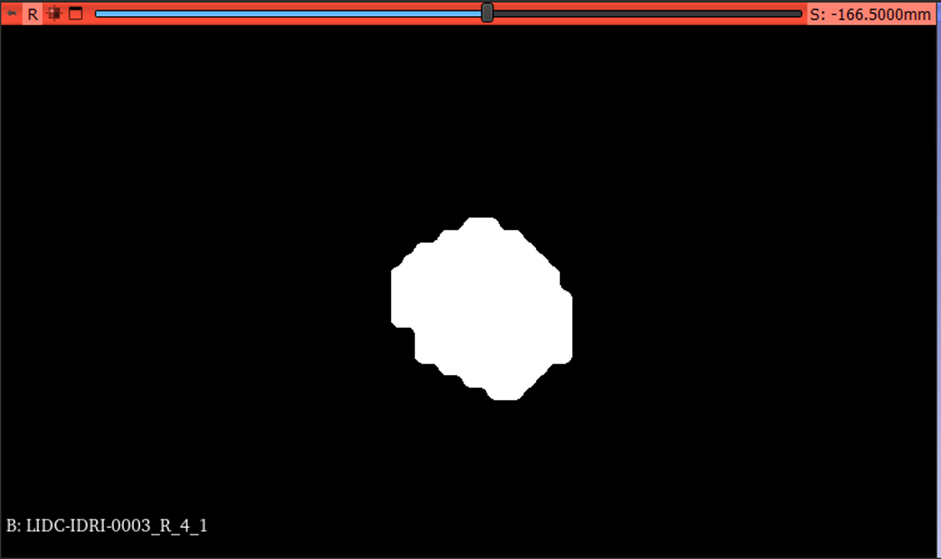
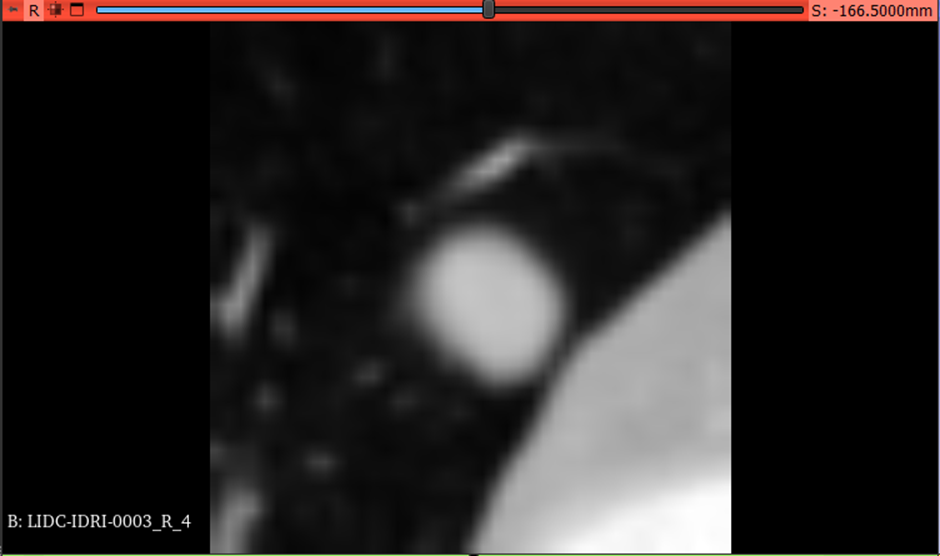


Image 5. Patient: 0003, Nodule: 4. Malignancy: 2. Diagnosis: Malignant.

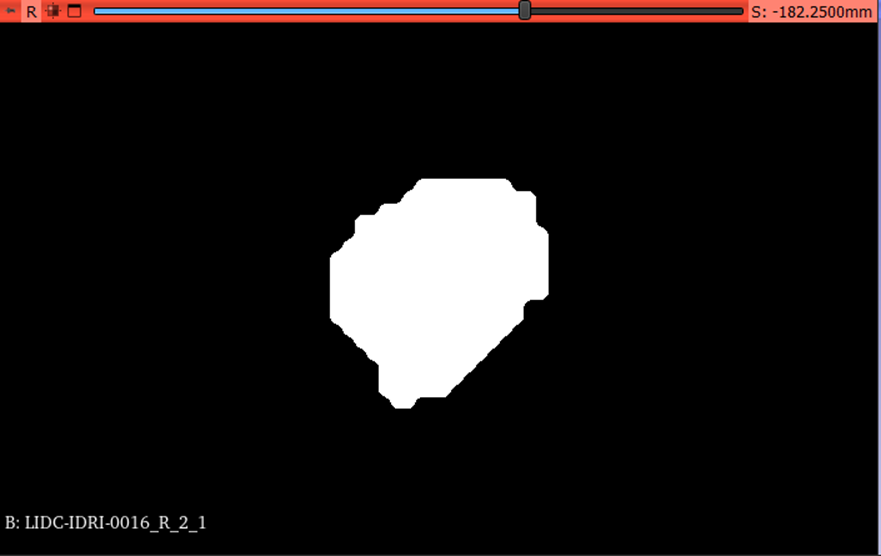


Image 6. Patient: 0016, Nodule: 2. Malignancy: 3. Diagnosis: Malignant.

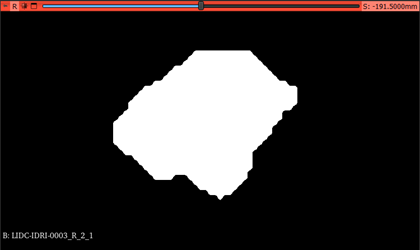
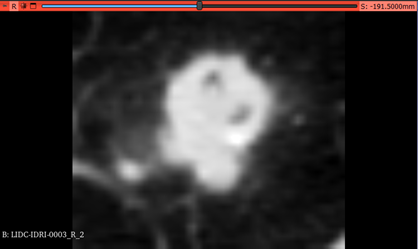


Image 7. Patient: 0003, Nodule: 2. Malignancy: 4. Diagnosis: Malignant.

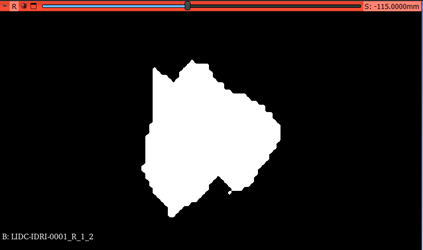
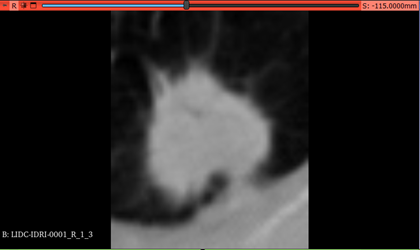


Image 8. Patient: 0001. Malignancy: 5. Diagnosis: Malignant.

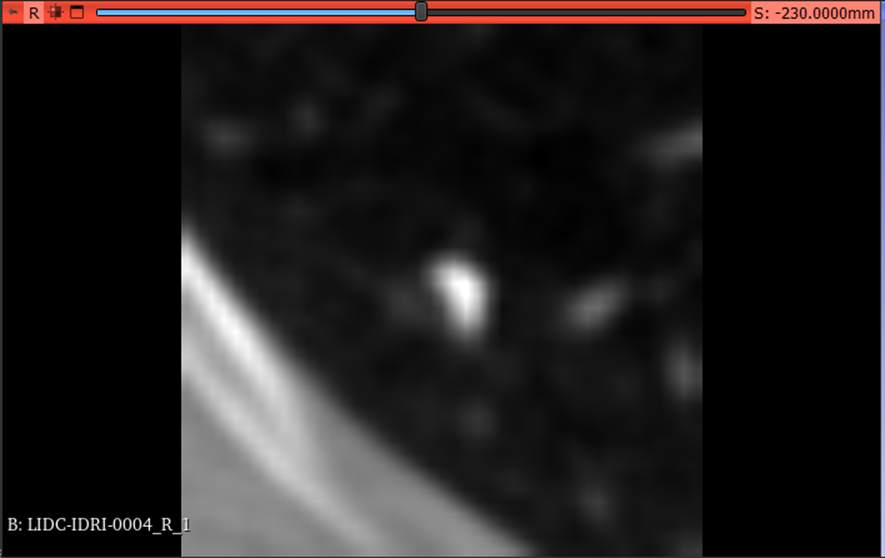


Image 9. Patient: 0004, Nodule: 1. Malignancy: 1. Diagnosis: Bening.

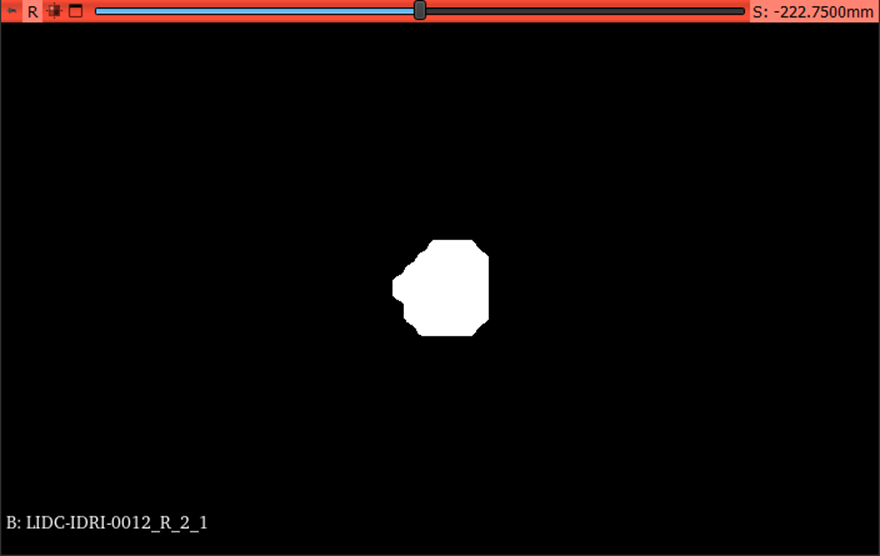
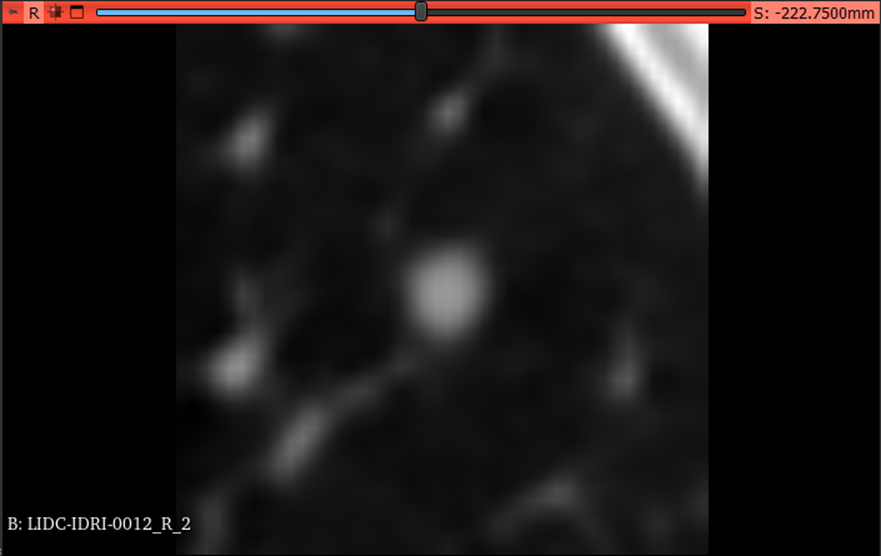


Image 10. Patient: 0012, Nodule: 1. Malignancy: 2. Diagnosis: Bening



Image 11. Patient: 0005, Nodule: 1. Malignancy: 3. Diagnosis: Bening

**Feature extraction**

For the classical methods, the first step was to extract the features from the images, for this Pyradiomics [6] library was used. Pyradiomics features are grouped in 5 different categories, the corresponding features for each category are shown in Table 1.

| **Group** | **# features** | **List of features** |
| --- | --- | --- |
| First order | 19 | Energy, Total Energy, Entropy, Minimum, 10th percentile, 90th percentile, Maximum, Mean, Median, Interquartile Range, Range, Mean Absolute Deviation, Robust Mean Absolute Deviation, Root Mean Squared, Standard Deviation, Skewness, Kurtosis, Variance, Uniformity.  Reference link: <https://pyradiomics.readthedocs.io/en/latest/features.html#radiomics.firstorder.RadiomicsFirstOrder> |
| Shape-based (3D) | 16 | Not used  Reference link: <https://pyradiomics.readthedocs.io/en/latest/features.html#radiomics.shape.RadiomicsShape> |
| Shape-based (2D) | 10 | Not used Reference link:<https://pyradiomics.readthedocs.io/en/latest/features.html#radiomics.shape2D.RadiomicsShape2D> |
| Gray level co-occurrence matrix | 24 | Autocorrelation, Joint Average, Cluster Prominence, Cluster Shade, Cluster Tendency, Contrast, Correlation, Difference Average, Difference Entropy, Difference Variance, Joint Energy, Joint Entropy, Informational Measure of Correlation (IMC) 1, Informational Measure of Correlation (IMC) 2, Inverse Difference Moment (IDM), Maximal Correlation Coefficient (MCC), Inverse Difference Moment Normalized (IDMN), Inverse Difference (ID), Inverse Difference Normalized (IDN), Inverse Variance, Maximum Probability, Sum Average, Sum Entropy, Sum of Squares.  Reference link: <https://pyradiomics.readthedocs.io/en/latest/features.html#radiomics.glcm.RadiomicsGLCM> |
| Gray level run length matrix | 16 | Short Run Emphasis (SRE), Long Run Emphasis (LRE), Gray Level Non-Uniformity (GLN), Gray Level Non-Uniformity Normalized (GLNN), Run Length Non-Uniformity (RLN), Run Length Non-Uniformity Normalized (RLNN), Run Percentage (RP), Gray Level Variance (GLV), Run Variance (RV), Run Entropy (RE), Low Gray Level Run Emphasis (LGLRE), High Gray Level Run Emphasis (HGLRE), Short Run Low Gray Level Emphasis (SRLGLE), Short Run High Gray Level Emphasis (SRHGLE), Long Run Low Gray Level Emphasis (LRLGLE), Long Run High Gray Level Emphasis (LRHGLE).  Reference link:  <https://pyradiomics.readthedocs.io/en/latest/features.html#radiomics.glrlm.RadiomicsGLRLM> |
| Gray level size zone matrix | 16 | Small Area Emphasis (SAE), Large Area Emphasis (LAE), Gray Level Non-Uniformity (GLN), Gray Level Non-Uniformity Normalized (GLNN), Size-Zone Non-Uniformity (SZN), Size-Zone Non-Uniformity Normalized (SZNN), Size-Zone Non-Uniformity Normalized (SZNN), Gray Level Variance (GLV), Zone Variance (ZV), Zone Entropy (ZE), Low Gray Level Zone Emphasis (LGLZE), High Gray Level Zone Emphasis (HGLZE), Small Area Low Gray Level Emphasis (SALGLE), Small Area High Gray Level Emphasis (SAHGLE), Large Area Low Gray Level Emphasis (LALGLE), Large Area High Gray Level Emphasis (LAHGLE).  Reference link: <https://pyradiomics.readthedocs.io/en/latest/features.html#radiomics.glszm.RadiomicsGLSZM> |
| Neighboring gray tone difference matrix | 5 | Not used  Reference link: <https://pyradiomics.readthedocs.io/en/latest/features.html#radiomics.ngtdm.RadiomicsNGTDM> |
| Gray level dependence matrix | 14 | Small Dependence Emphasis (SDE), Large Dependence Emphasis (LDE), Gray Level Non-Uniformity (GLN), Dependence Non-Uniformity (DN), Dependence Non-Uniformity Normalized (DNN), Gray Level Variance (GLV), Dependence Variance (DV), Dependence Entropy (DE), Low Gray Level Emphasis (LGLE), High Gray Level Emphasis (HGLE), Small Dependence Low Gray Level Emphasis (SDLGLE), Large Dependence Low Gray Level Emphasis (LDLGLE), Large Dependence High Gray Level Emphasis (LDHGLE).  Reference link: <https://pyradiomics.readthedocs.io/en/latest/features.html#radiomics.gldm.RadiomicsGLDM> |

Table 1. Pyradiomic summary of features.

For testing and understanding purposes we tried a round of training with only "First order" features, another with "Gray level dependence" features and another round with all of them. The results obtained are described in the "Results" section.

**Models**

As part of this work we trained, evaluated and compared three classical models: decision tree, support vector machine and nearest neighbor, and one embedded model: random forest. Additionally, a deep learning method was also trained to verify if we can get a better performance.

*Decision Tree*

Decision Trees (DTs) are a popular supervised learning method used for classification and regression tasks. They create models by learning simple decision rules from the input data to predict the target variable [7]. DTs are advantageous because they are easy to understand, require minimal data preparation, and can handle both numerical and categorical data. They also have a relatively low computational cost for making predictions. Additionally, DTs are interpretable, making it easier to explain the model's decisions. They can handle multi-output problems and can be validated using statistical tests [8].

Decision trees can be illustrated as a segmented space, where each segment corresponds to a leaf node, as shown in Figure 1. Decision tree analysis aims to determine the best model for dividing all records into different segments [9].

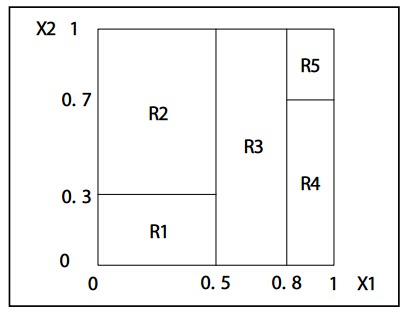


Figure 1. Decision tree illustrated using sample space view

Decision tree is a greedy algorithm. To predict the class of a given dataset, the algorithm starts from the root node of the tree as shown in Figure 2. The algorithm compares the values of the root attribute with the attribute of the actual dataset and based on the comparison, follows the branch and moves to the next node. For the next node, the algorithm again compares the attribute value with other subnodes and continues the process until it reaches the leaf node of the tree [10].

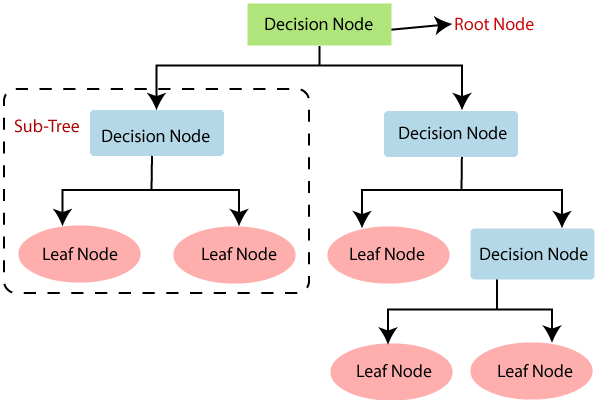


Figure 2. Decision Tree [9]

It selects the current best split that maximizes information gain [11]. “Equations 1 and 2”.

(1)

(2)

However, DTs have some drawbacks. It does not backtrack and change a previous split. So, all the following splits will depend on the current one. And this does not guarantee that we will get the most optimal set of splits. Greedy search makes our training a lot faster and it works really well despite its simplicity.

*Support Vector Machine (SVM)*

Support Vector Machine (SVM) is a machine learning algorithm used for classification and regression. The main idea of SVM is to construct a hyperplane that best separates objects of different classes in the feature space. SVM aims to find the hyperplane with the largest gap between classes to ensure the best generalization ability of the model [12] [13] [14].

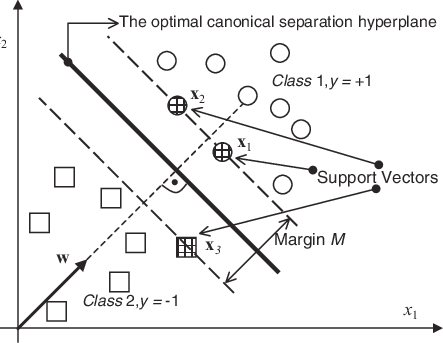


Figure 3. The optimal canonical separating hyperplane with the largest margin intersects halfway between the two classes. The points closest to it are support vectors. [14]

Advantages of support vector machine (SVM) [15]:

* Efficient in high-dimensional spaces, making it useful for analyzing data with a large number of features.
* Good generalization ability, which helps reduce the problem of overfitting and ensure reliable classification.
* Flexibility in choosing a kernel, allowing you to model complex nonlinear dependencies between features.

Disadvantages of support vector machine (SVM):

* Sensitivity to outliers, which can negatively impact model performance.
* Difficulty in choosing optimal parameters that require tuning.
* Computational complexity when training a model, especially for large data sets.

*Nearest Neighbor (KNeighbor)*

K-Nearest Neighbors or KNN is a simple classification and regression algorithm. When training a KNN model, the algorithm stores all available training data in memory so it can be used to classify new examples. To classify a new example, the KNN algorithm calculates the distance between the test sample and the specified training samples [16]. Distance can be measured using various metrics such as Euclidean distance (Equation 1) or Manhattan distance (Equation 2) [17].

Let be objects, and their attribute descriptions.

Euclidean distance:

(1)

Manhattan distance:

(2)

The algorithm then selects the K nearest neighbors with the smallest distance to the test sample, as shown in Figure 4. [18]

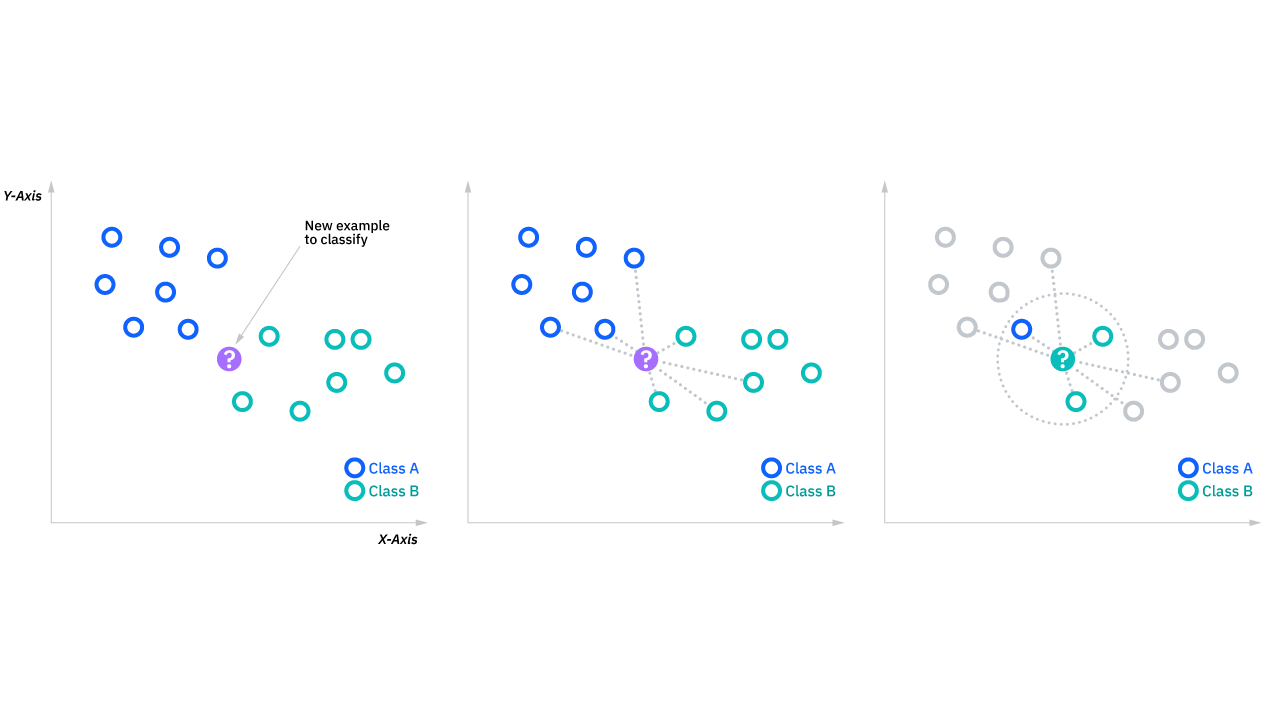


Figure 4. KNN diagram.

After selecting K nearest neighbors, the KNN algorithm uses their classes to determine the class of the test sample. The simplest approach is majority voting, where the class represented by the most neighbors is chosen as the predicted class for the test sample. The choice of the optimal value of K affects the performance of the model. A small value of K may lead to model underfitting and noisy predictions. While a high value for k may seem acceptable at first glance, it can cause problems with model performance and also increases the risk of overfitting, which can lead to loss of detail or blurred predictions [19].

In the case of using the method for regression, an object is assigned an average value over the k objects closest to it, the values of which are already known [16] [17] [18] [19].

*Random Forest Approach*

Random forests are an ensemble learning method used to solve problems of classification, regression, clustering, anomaly detection, feature selection, etc., which works by constructing multiple decision trees during training [20].

One of the key features of Random Forest is the use of random subsets of the training data chosen through a process called bootstrapping and prediction functions to train each tree. Trees are constructed recursively by selecting the best feature and splitting point. This method can increase the diversity and independence of trees and avoid overfitting and correlation. To make the final prediction, random forest aggregates the results of all decision trees by taking a majority vote or the average [21]. As shown in Figure 5 [22].

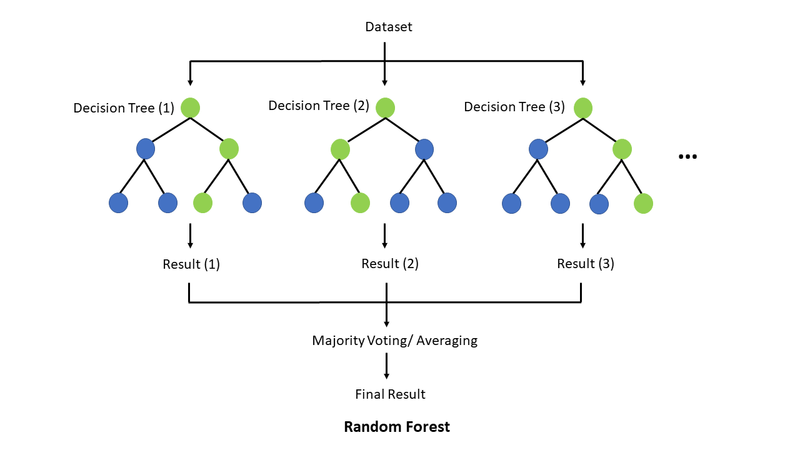


Figure 5. Random forest structure.

**One of the advantages of the random forest algorithm is its ability to evaluate model performance on new, unseen data without requiring the creation of a separate validation set, due to the fact that it provides a built-in out-of-bag (OOB) evaluation mechanism. On the other hand, implementing the random tree algorithm requires a significant amount of computing resources [23].**

*Deep learning*

Deep learning involves training a machine to automatically discover complex representations from raw data. The fundamental concept of deep learning is the composition of layers of features, where each layer refines and abstracts the representation learned from the previous layer. This allows the model to learn complex features and distinguish important features for tasks such as detection or classification. In image classification, the initial layers can capture edges, and then the layers detect motifs and assemble them into parts of recognizable objects [24]. Figure 6 shows how a deep learning system represents a complex concept, such as an image of a person, by combining simpler concepts such as angles and contours. The visible layer receives input data, represented as pixel values in the case of images. Each hidden layer extracts increasingly abstract elements from the image. The first layer easily identifies edges, the second layer detects corners and contours, and subsequent layers combine them into larger combinations that match parts of familiar objects. This layer-by-layer progression allows the model to understand the complex relationships in the observed data [25].

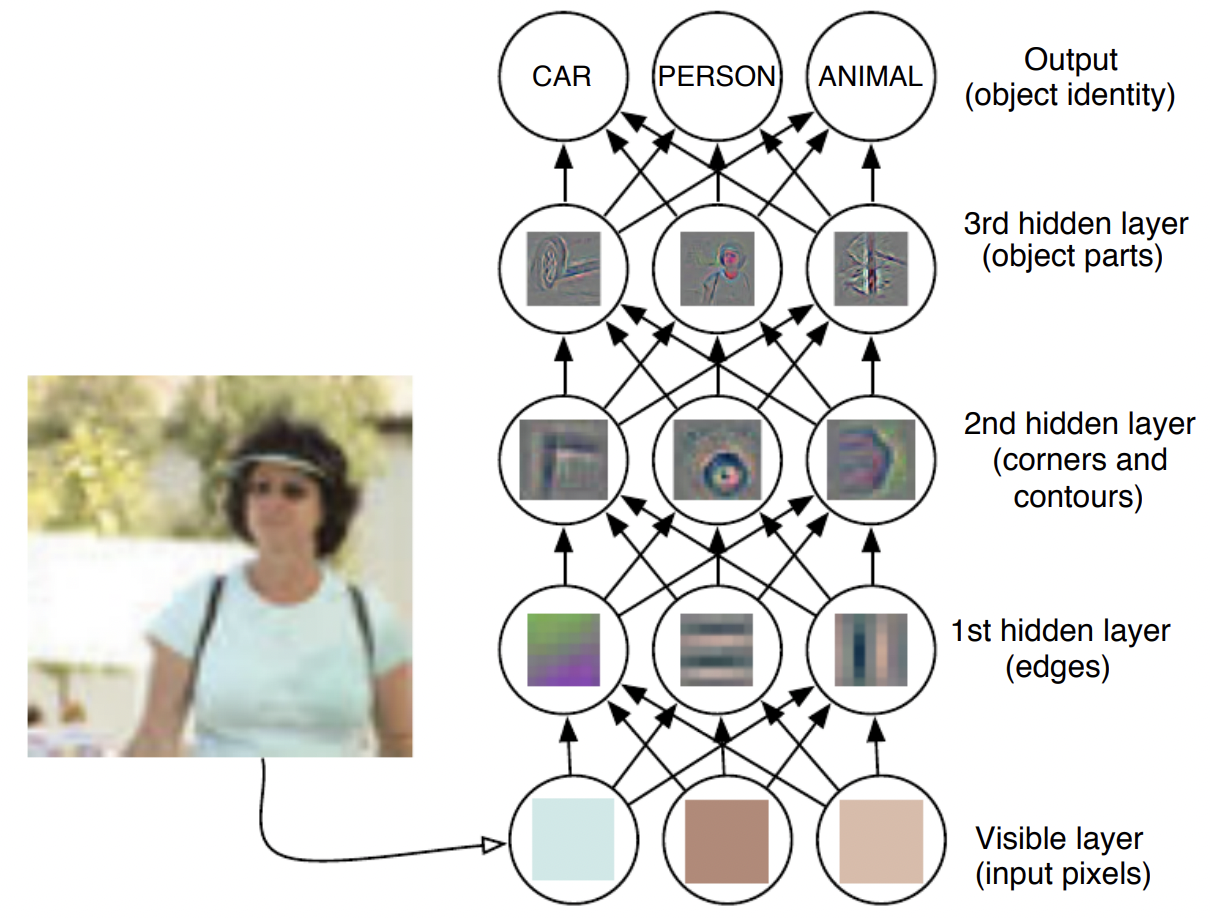


Figure 6. Illustration of a deep learning model [25].

Another way of looking at deep learning is that it allows a machine to learn a sequential computer program. Each level represents the state of the computer's memory after a set of instructions has been executed, allowing instructions to be executed sequentially. This sequential execution provides significant computational power, allowing the model to return to earlier results in subsequent processing steps.

To classify lung nodules, we implemented a neural network using PyTorch. This network is designed for binary classification based on features extracted from pulmonary nodules [32].

**Metrics**

In the following section we explain the metrics used in order to evaluate the different classification models.

*Confusion matrix*

The confusion matrix is a tool for assessing the accuracy of classification models. It is a table, where each row corresponds to the actual class of objects, and each column corresponds to the predicted class of the model, as shown in Figure 7 [26]. The cells of the matrix indicate the number of objects that belong to a certain combination of true and predicted classes.

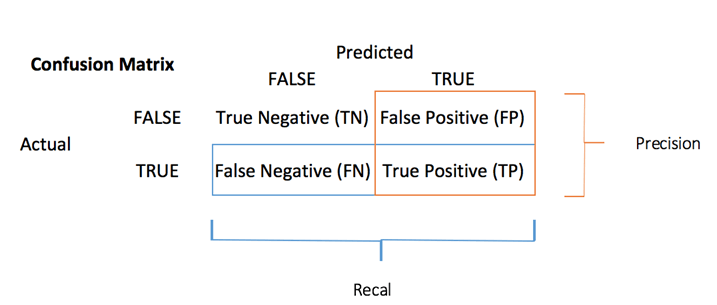
****

Figure 7. Confusion matrix.

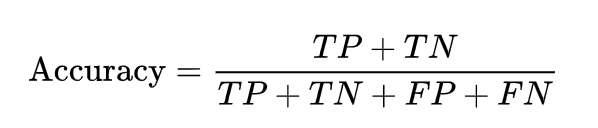
The confusion matrix allows us to visually evaluate how well the model correctly classifies objects. It contains four main elements [26]:

* True Negative (**TN**): The number of objects that were correctly classified as negative.
* False Positive (**FP**): The number of objects that were incorrectly classified as positive.
* False Negative (**FN**): The number of objects that were incorrectly classified as negative.
* True Positive (**TP**): The number of objects that were correctly classified as positive.

Using the values from the confusion matrix, we can calculate various metrics to evaluate the model's performance, such as accuracy, recall, precision, and F1-score. These metrics give an idea of how capable the model is of correctly identifying objects of different classes. The confusion matrix is a useful tool for analyzing and interpreting classification results. It helps you understand where and what errors a model makes and can be used to improve modeling and decision-making on classification problems.

*Accuracy*

In general, in order to evaluate and compare the models we use accuracy which is defined as:



Formula 1. Accuracy of a model

*Sensitivity and specificity*

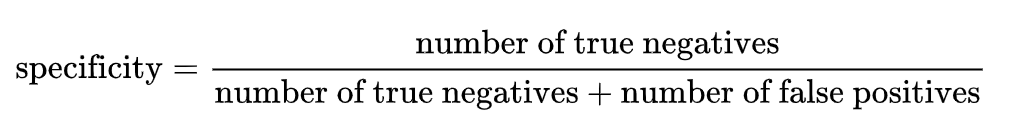
When evaluating a model it is also important to analyze some other metrics besides accuracy. Sensitivity and specificity are measures of a test's ability to correctly classify a person as having a disease or not having a disease.

**Sensitivity** refers to a test's ability to designate an individual with disease as positive. A highly sensitive test means that there are few false negative results, and thus fewer cases of disease are missed.



Formula 2. Sensitivity of a model (also known as recall or true positive rate)

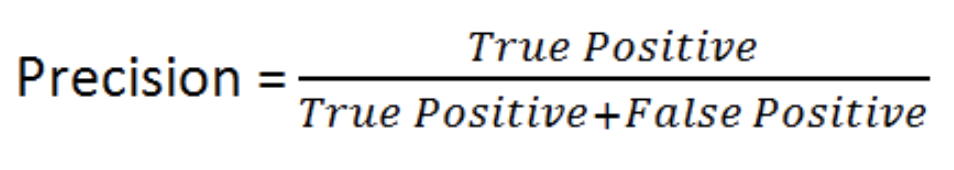
The **specificity** of a test is its ability to designate an individual who does not have a disease as negative. A highly specific test means that there are few false positive results. It may not be feasible to use a test with low specificity for screening, since many people without the disease will screen positive, and potentially receive unnecessary diagnostic procedures.



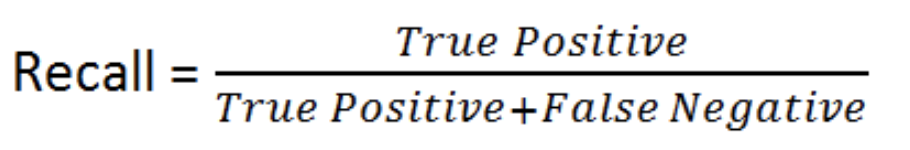
Formula 3. Specificity of a model (also known as selectivity or true negative rate)

*Precision, recall and f1-score*

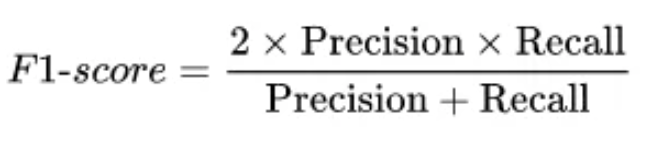
**While using the scikit-learn library we can get a classification report that includes these three metrics. The formulas of each one of them are presented below.**



Formula 4. Precision.



Formula 5. Recall (also known as sensitivity)



Formula 6. f1-score (also known as the harmonic mean of precision and sensitivity)

Results

In this section we explain each of our experiments with different models and their corresponding results.

For all the following experiments, we standardized the data before training and testing the dataset using scikit-learn StandardScaler method.

**Decision tree**

In order to train and test a Decision Tree model we used scikit-learn library [27]. An overview of the code is included in Image 12.

clf = tree.DecisionTreeClassifier(max\_depth=4)

clf = clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

### calculation results

tn, fp, fn, tp = confusion\_matrix(y\_test, y\_pred).ravel()

print('Confusion matrix results')

print(f'True positive (TP): {tp}')

print(f'False positive (FP): {fp}')

print(f'True negative (TN): {tn}')

print(f'Flase negative (FN): {fn}')

acc = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy of the model is: {acc}')

sensitivity = tp / (tp + fn)

specificity = tn / (tn + fp)

print(f'Sensitivity: {sensitivity}')

print(f'Specificity: {specificity}')

Image 12. Decision tree implementation

As shown in the Image 12, we use the function *confusion\_matrix().ravel()* from the *sklearn.metrics* libraryto compute the true negative, false positive, false negative and true positive values. Using Formula 1, we calculate the accuracy value. The application of Formula 2 and Formula 3 for sensitivity and specificity.

The first execution we get the following results and the model presented in Figure 8.

* Accuracy: 0.866
* Sensitivity: 0.859
* Specificity: 0.869

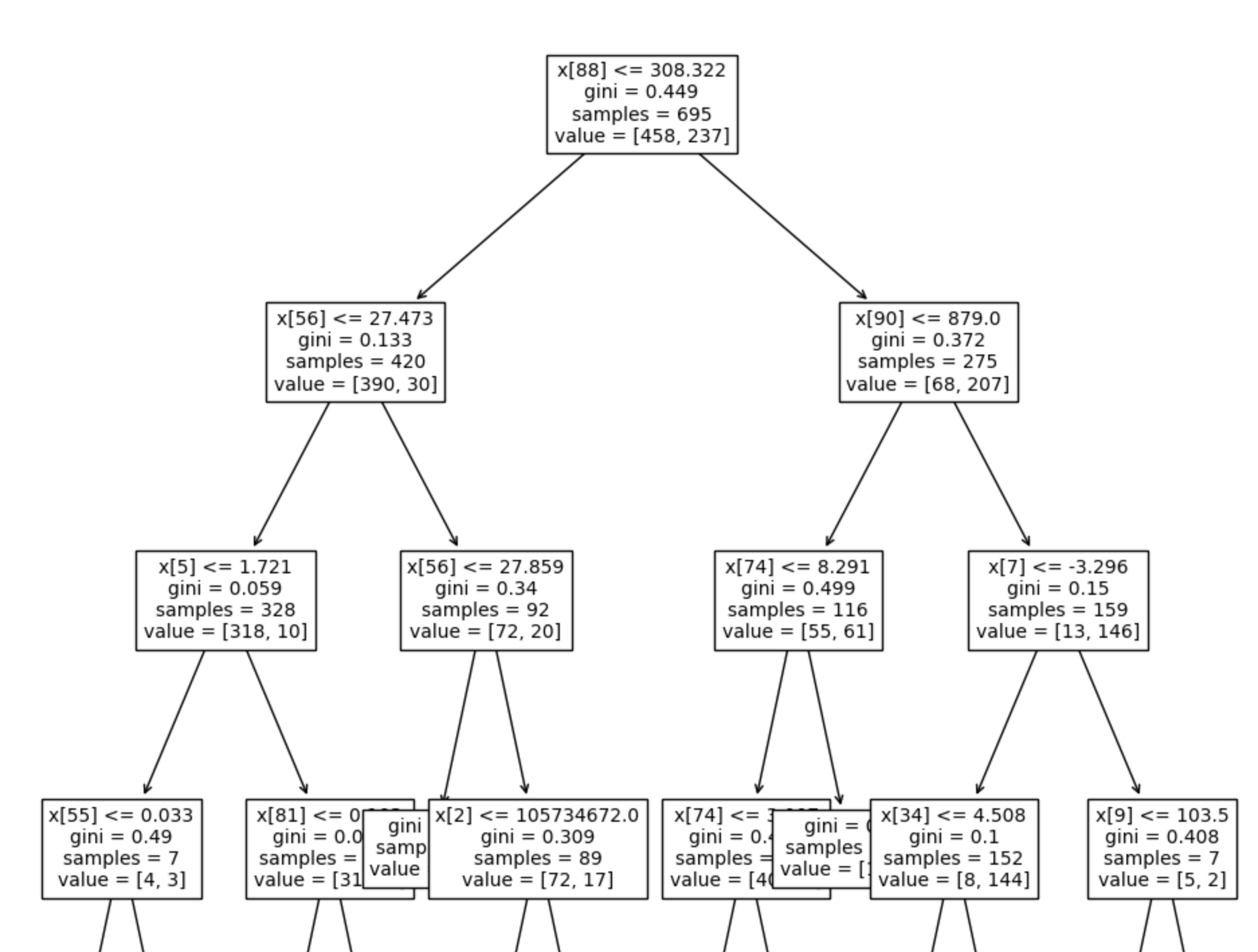


Figure 8. Overview of the generated DT model.

With the second execution we get the following results and the model presented in Figure 9.

* Accuracy: 0.876
* Sensitivity: 0.878
* Specificity: 0.875

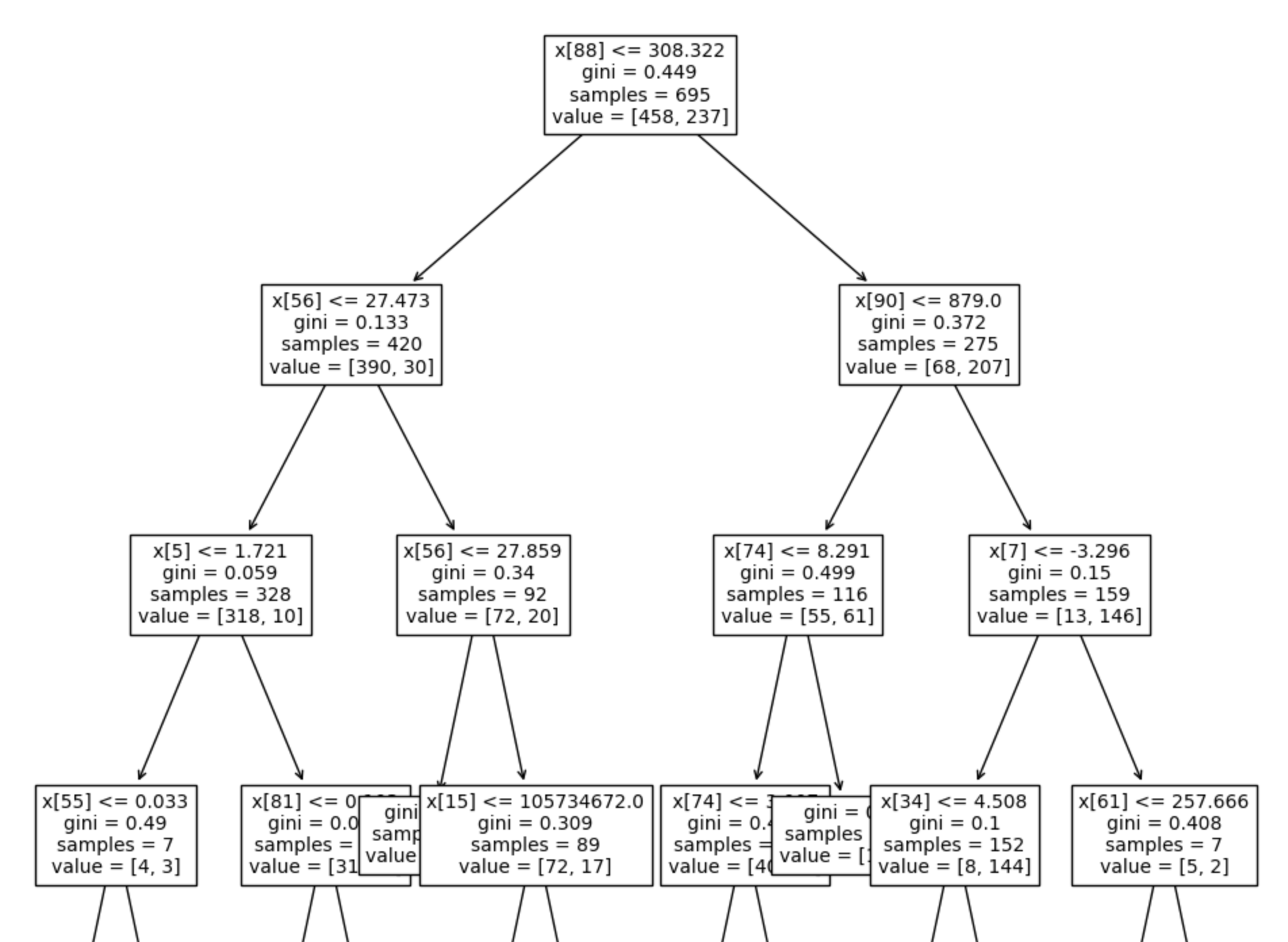


Figure 9. Overview of the generated DT model.

As we can see from this example, in the first test the accuracy of the model is: 0.869 while in the second test the accuracy is: 0.876. DTs can create over-complex models that overfit the training data, and on the other hand DTs can also be unstable, meaning small changes in the data can lead to significantly different trees.

**Another thing that decision trees provide is a way to check which features are the ones that help the most during classification time. We briefly analyze the values of those features randomly selecting pairs of nodules for each existing case, i.e. a "Malignant Nodule (1)", which means a nodule diagnosed as malignant and with a malignancy type 1", and the same for all the malignant cases we can found in the datasource and "Benign Nodule (1)", which is a nodule diagnosed as benign with a malignancy type 1", and again the same for all the bening cases.**

The results are shown in Table 2 and from that short analysis we extract the following insights:

* It can be noticed that in general the gray leven non-uniformity feature and the gray level variance are higher for bening images and lower for malignant ones. Still, some types of nodules have lower values which can cause the models to miss some cases.
* Mean has the contrary behavior, higher values (near cero) for malignant nodules and lower values for benign ones. Something interesting to notice here is the big difference for the test case "Malignant Nodule Type 1". **This can be associated with type 1 being sometimes bening and others malignant, meaning that in the "Metadata.xls" file that has all the information about nodules it can be seen that modules with malignancy types equals to 1 were sometimes diagnosed as benign but some other were diagnosed as malignant.**
* Surface area does not say too much for us but somehow the decision tree is also considering it as one of the top features for splitting.
* Lastly, for our selected test cases we can not give a conclusion for Kurtosis because it is around the same value for all of them except the "Benign Nodule Type 3".

A more comprehensive analysis could be conducted across additional test cases to uncover further similarities or differences, providing deeper insights into the behavior of these features.

| **Feature (extracted with Pyradiomics)** | **Malign Nodule (1)** | **Malign Nodule (2)** | **Malign Nodule (3)** | **Malign Nodule (4)** | **Malign Nodule (5)** | **Benign Nodule (1)** | **Benign Nodule (2)** | **Benign Nodule (3)** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Surface area | 1.07 | 0.69 | 0.99 | 0.89 | 0.64 | 0.85 | 1.27 | 1.20 |
| Gray level non-uniformity | 1.52 | 1.74 | 1.54 | 1.32 | 1.15 | 1.56 | 5.07 | 3.38 |
| Voxel volume | 1.32 | 0.91 | 1.34 | 1.18 | 0.82 | 1.36 | 2.96 | 2.65 |
| Kurtosis | 0.27 | 0.60 | 0.25 | 0.36 | 0.40 | 0.28 | 0.28 | 2.69 |
| Gray level variance | 1.23 | 5.55 | 0.33 | 0.41 | 0.32 | 0.74 | 9.91 | 7.99 |
| Mean | 1258.06 | -1.38 | -0.50 | -0.54 | -0.63 | -9.27 | -1.49 | -1.65 |

Table 2. Feature values for malignant nodules.

**Support Vector Machines (SVMs)**

In order to train and test a SVM model we used skit-learn library [28]. An overview of the code is included in Image 13.

clf = svm.SVC()

clf = clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

Image 13. SVM implementation

After the execution of the code we got the following results:

* Accuracy: 0.883
* Sensitivity: 0.813
* Specificity: 0.921

**K-Nearest Neighbor**

In order to train and test a SVM model we used skit-learn library [29]. An overview of the code is included in Image 14.

clf = KNeighborsClassifier(n\_neighbors=3)

clf = clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

Image 14. K-Nearest Neighbor implementation

After the execution of the code, started with n\_neigbors param equals to 3, we got the following results:

* Accuracy: 0.863
* Sensitivity: 0.822
* Specificity: 0.885

Therefore, in order to get better results we try some other n\_neighors values. The results were improved until n\_neighors reached sqrt(n) being "n" number of data points in the training set. The results for n\_neighors = 9 are the following:

* Accuracy: 0.883
* Sensitivity: 0.860
* Specificity: 0.896

**Random Forest**

Finally, we tried to improve our results with an embedded model. Again we used the Random Forest model from skit-learn library [30]. An overview of the code is included in Image 15.

clf = RandomForestClassifier(n\_estimators=5)

clf = clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

Image 15. Random Forest implementation

After the execution of the code, with 5 estimators, we got the following results:

* Accuracy: 0.886
* Sensitivity: 0.850
* Specificity: 0.906

In this case we also changed the param n\_estimators a few times to obtain better results. With n\_estimators = 50 we got the following results:

* Accuracy: 0.900
* Sensitivity: 0.860
* Specificity: 0.922

Running this model for just first order features and the optimal number of estimator (50) we got:

* Accuracy: 0.872
* Sensitivity: 0.785
* Specificity: 0.921

Then, changing the selection to just gray level features and completely removed the first order ones, we got:

* Accuracy: 0.886
* Sensitivity: 0.803
* Specificity: 0.932

All the previous detailed scenarios can be found in our shared colaboratory notebooks written in python [31] for other teams to reuse and reproduce our results.

**Lastly, when changing the classification target to malignancy instead of diagnosis we got the confusion matrix presented in Figure 10 with the Random Forest model and an accuracy of just 0.501. This confirms that the model performs better recognizing malignant versus benign nodules but it is not able to properly classify the nodules level of malignancy.**

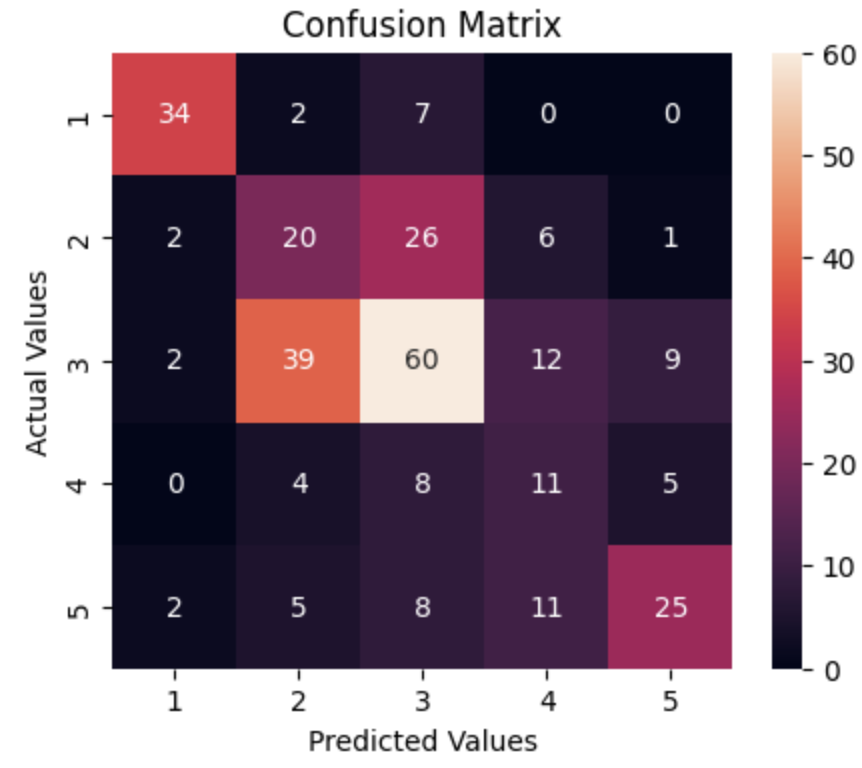


Figure 10. Confusion matrix values for multi-class classification (malignancy) **using Random Forest (n\_estimators=10).**

**Comparison of differents approaches**

For exploration purposes we ran other tests, including some other models also, using slices instead of the 3D volumes directly. In this case, we extract another CSV file for 3D volumes. Also the obtained data from CSV are different to the slices one. To approach them, we processed the same model tests with the 3D volume one. And we get the following results, which are presented in Table 4.

| **Model** | **Accuracy** | **Sensitivity** | **Specificity** |
| --- | --- | --- | --- |
| Decision tree | 0.872 | 0.810 | 0.910 |
| SVM | 0.876 | 0.745 | 0.952 |
| K-NN (k=9) | 0.866 | 0.781 | 0.915 |
| Random forest (n=50) | 0.872 | 0.755 | 0.941 |

Table 3. Slices results (including all features for the training)

All the calculations for accuracy, sensitivity, and specificity are processed using Formula 1, Formula 2, and Formula 3, as illustrated in Image 12, which is referenced in the decision tree section.

In conclusion, for the Slices case (Table 3), the accuracy across different models is quite similar. Notably, the SVM model achieved a specificity value of 0.952, indicating that it performed well as anticipated.

**In the Volume case (Table 4), we observed that the SVM model outperforms other models, exhibiting the highest accuracy and specificity.**

| **Model** | **Accuracy** | **Sensitivity** | **Specificity** |
| --- | --- | --- | --- |
| Decision tree | 0.866 | 0.869 | 0.859 |
| SVM | 0.883 | 0.813 | 0.921 |
| K-NN (k=9) | 0.883 | 0.860 | 0.896 |
| Random forest (n=50) | 0.900 | 0.860 | 0.922 |

Table 4. Volume results (including all features for the training).

**A complete overview of the metrics presented in Table 4 for the different models can be found in Figures 11, 12 and 13 respectively.**

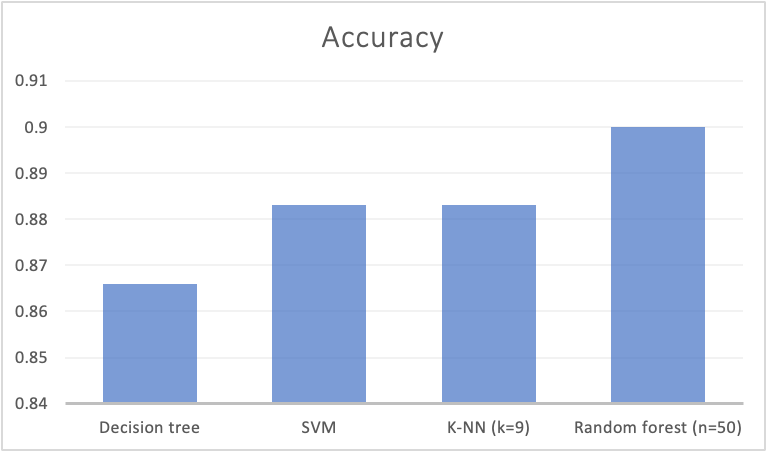


Figure 11. Accuracy comparison between models.

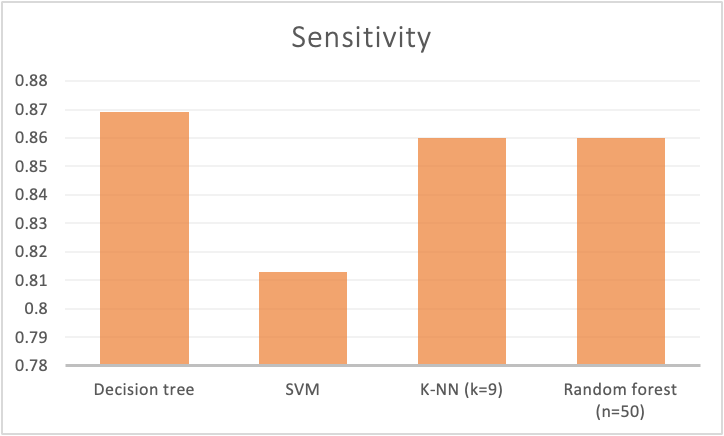


Figure 12. Sensitivity comparison between models.

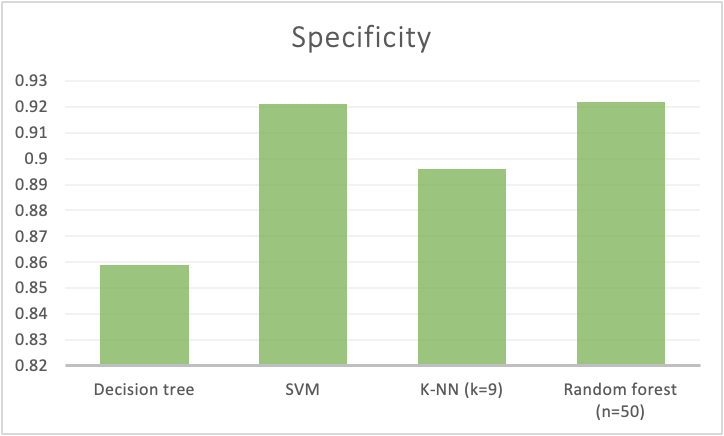


Figure 13. Specificity comparison between models.

In Table 5, we additionally present the classification report obtained for each model. This classification report shows the precision, recall and f1-score metrics for each class of the binary classification (Benign/Malignant). The column support indicates the amount of elements of each class present in the testing set.

| **Model** | **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
| --- | --- | --- | --- | --- | --- |
| DT | Benign | 0,91 | 0,92 | 0,91 | 192 |
|  | Malignant | 0,85 | 0,84 | 0,85 | 107 |
| SVM | Benign | 0,90 | 0,92 | 0,91 | 192 |
|  | Malignant | 0,85 | 0,81 | 0,83 | 107 |
| KNN | Benign | 0,92 | 0,90 | 0,91 | 192 |
|  | Malignant | 0,82 | 0,86 | 0,84 | 107 |
| Random Forest | Benign | 0,91 | 0,92 | 0,91 | 192 |
|  | Malignant | 0,85 | 0,84 | 0,85 | 107 |

Table 4. Classification report by class (bening, malignant).

Figures 14, 15 and 16 show the results for precision, recall and f1-score in a more comparative way using charts. From these numbers we can see that all models performs better for class 0 (benign) and have slightly less performance for class 1 (malignant nodules).



Figure 14. Precision comparison between models.

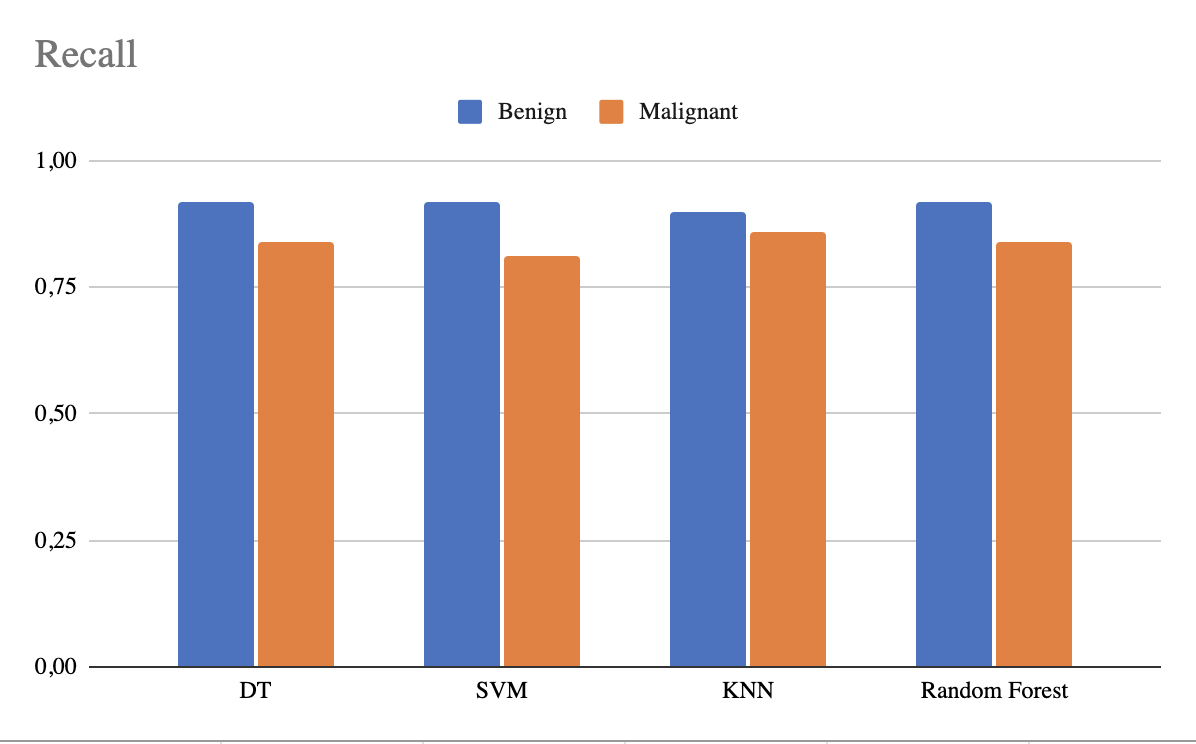


Figure 15. Recall (or sensitivity) comparison between models.

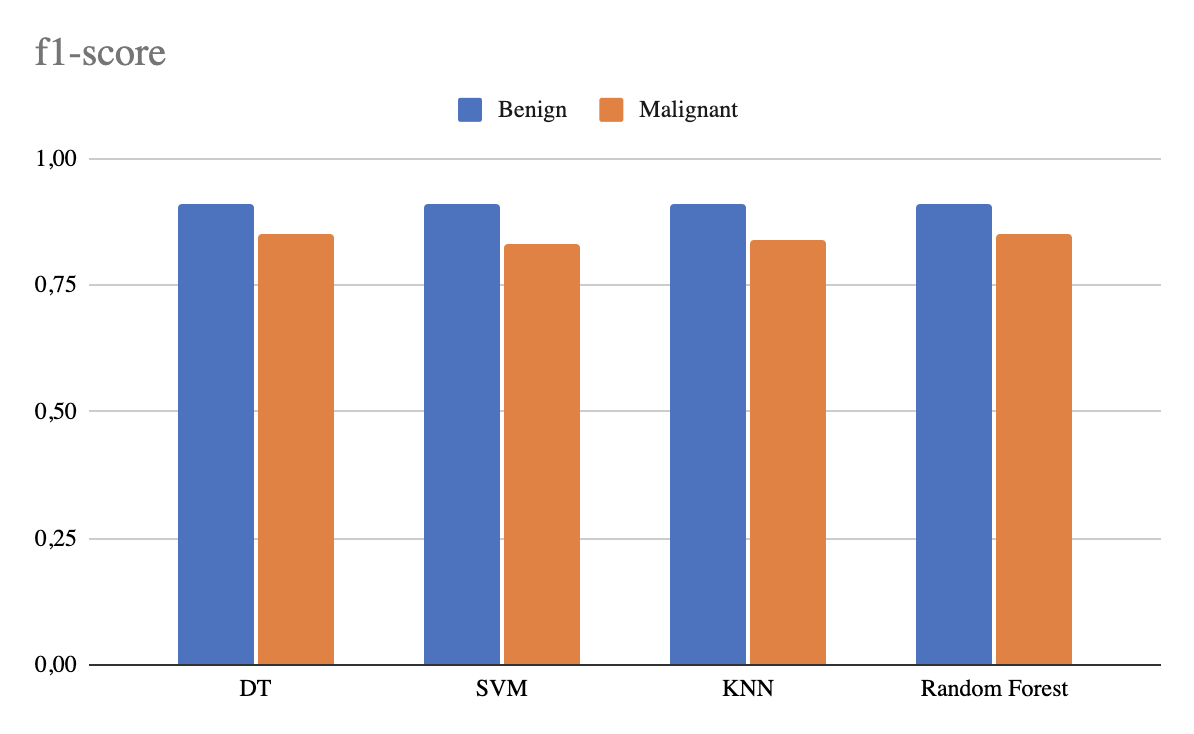


Figure 16. f1-score comparison between models.

Discussion and conclusions

The comparative table (Table 4) in terms of accuracy, specificity and sensitivity allow us to evaluate the different models and their performance.

In terms of accuracy (Figure 6) the Random Forest has the highest value, followed by the SVM. The worst accuracy was obtained by the Decision Tree model. For sensitivity (Figure 7) the Decision Tree has the highest value, followed by the K-Nearest Neighbor and Random Forest. In this case the SVMs model has the worst result. However, for specificity (Figure 8) we can see that the SVMs model is the one with the highest value, followed by the Random Forest and K-Nearest Neighbor, the worst result in this case is for the Decision Tree.

What we see with the SVM case is that this type of model might be sacrificing sensitivity results for a better specificity. The more sensitive a model is, the less likely an individual with a negative test will be diagnosed with the disease. The more specific the model, the less likely an individual with the disease will be diagnosed as healthy.

Ultimately, the Random Forest model demonstrates the highest performance, followed by the SVM. Surprisingly, despite its superior sensitivity, the Decision Tree model yields the least favorable results.

Additionally, we confirmed that changing the features included as predictor variables does not improve the model. In some other experiments, we also confirmed that using the 3D volume directly versus slices is not substantially improving the models either.

For comparison purposes, we also train and evaluate a deep learning architecture using Pytorch library [32]. Neural networks do not need to perform a feature extraction stage before the training, they can directly consume the images (and in some hybrid cases the masks) in order to train a model. This kind of architecture has several advantages but also challenges. Our biggest issue while implementing a convolutional neural network (CNN) was the heterogeneous dimensions of the images in the LIDC dataset. We research about a few topics such as resizing, resampling and interpolation but those require a more extensive knowledge on the topic. With a simple resizing of the images to have them all in the same dimension we got accuracies around 70%, which is no better than the ones obtained by classical models. Again, we believe these results can be drastically improved applying the right preprocessing techniques.

Considering our experiments and findings, it appears imperative to enhance the preprocessing stage to yield superior outcomes compared to those achieved using the Random Forest method.

All our tests, including implementation, data, extracted features and other significant results can be found in an open Drive folder [33].

Bibliography

[1] <https://arxiv.org/pdf/1612.08012.pdf>

[2] <https://www.rbht.nhs.uk/our-services/lung/lung-nodules-guide-for-patients>

[3] Liu, B., Chi, W., Li, X. *et al.* Evolving the pulmonary nodules diagnosis from classical approaches to deep learning-aided decision support: three decades' development course and future prospect. *J Cancer Res Clin Oncol* **146**, 153–185 (2020). <https://doi.org/10.1007/s00432-019-03098-5>

[4] Samuel G Armato III, Geoffrey McLennan, Luc Bidaut, Michael F McNitt-Gray, Charles R Meyer, Anthony P Reeves, Binsheng Zhao, Denise R Aberle, Claudia I Hen- schke, Eric A Hoffman, et al. The lung image database consortium (LIDC) and image database resource initiative (IDRI): a completed reference database of lung nodules on ct scans. Medical physics, 38(2):915–931, 2011.

[5] grand challenge.org. Luna16 challenge. https://luna16.grand-challenge.org/ Description. Accessed: 01-04-2023.

[6] <https://pyradiomics.readthedocs.io/en/latest/radiomics.html>

[7] <https://scikit-learn.org/stable/modules/tree.html>

[8] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2675494/

[9] Song, Yan-Yan, and L. U. Ying. "Decision tree methods: applications for classification and prediction." Shanghai archives of psychiatry 27.2 (2015): 130. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4466856/>

[10] <https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm>

[11] <https://www.jastt.org/index.php/jasttpath/article/view/65>

[12] Jakkula, Vikramaditya. "Tutorial on support vector machine (SVM)." School of EECS, Washington State University 37.2.5 (2006): 3. <https://course.ccs.neu.edu/cs5100f11/resources/jakkula.pdf>

[13] Noble, William S. "What is a support vector machine?." *Nature biotechnology* 24.12 (2006): 1565-1567. <https://www.nature.com/articles/nbt1206-1565#Fig1>

[14] Wang, Lipo, ed. Support vector machines: theory and applications. Vol. 177. Springer Science & Business Media, 2005. <https://books.google.es/books?hl=ru&lr=&id=uTzMPJjVjsMC&oi=fnd&pg=PA1&dq=Support+Vector+Machine+(SVM)&ots=GGzF7r0Ho6&sig=8uRAYGzfB3p9BWXTnfViDSFOyk0&redir_esc=y#v=onepage&q=Support%20Vector%20Machine%20(SVM)&f=false>

[15] Anguita, Davide, et al. "Model selection for support vector machines: Advantages and disadvantages of the machine learning theory." The 2010 international joint conference on neural networks (IJCNN). IEEE, 2010. <https://ieeexplore.ieee.org/abstract/document/5596450>

[16] Bezdek, James C., Siew K. Chuah, and David Leep. "Generalized k-nearest neighbor rules." Fuzzy Sets and Systems 18.3 (1986): 237-256.

[17] Peterson, Leif E. "K-nearest neighbor." Scholarpedia 4.2 (2009): 1883. <http://scholarpedia.org/article/K-nearest_neighbor>

[18] IBM.com. "What is the k-nearest neighbors algorithm?" <https://www.ibm.com/topics/knn>

[19] Batista, G. E. A. P. A., and Diego Furtado Silva. "How k-nearest neighbor parameters affect its performance." Argentine symposium on artificial intelligence. 2009. <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=b83da505b60e32469152b986cbd6199842403b11>

[20] Biau, Gérard. "Analysis of a random forests model." The Journal of Machine Learning Research 13 (2012): 1063-1095. <https://www.jmlr.org/papers/volume13/biau12a/biau12a.pdf>

[21] Breiman, Leo. "Random forests." Machine learning 45 (2001): 5-32. <https://link.springer.com/article/10.1023/a:1010933404324>

[22] Interactive Chaos. “Random forest”. <https://interactivechaos.com/es/wiki/random-forest>

[23] Hastie, Trevor, et al. "Random forests." The elements of statistical learning: Data mining, inference, and prediction (2009): 587-604. <https://link.springer.com/chapter/10.1007/978-0-387-84858-7_15>

[24] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." nature 521.7553 (2015): 436-444. <https://www.nature.com/articles/nature14539>

[25] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016. <https://books.google.es/books?hl=ru&lr=&id=omivDQAAQBAJ&oi=fnd&pg=PR5&dq=deep+learning&ots=MOM4eppDTZ&sig=mUINcWtk6D1c5V9EeHClajWLI_4&redir_esc=y#v=onepage&q=deep%20learning&f=false>

[26] Caelen, Olivier. "A Bayesian interpretation of the confusion matrix." Annals of Mathematics and Artificial Intelligence 81.3-4 (2017): 429-450. <https://link.springer.com/article/10.1007/s10472-017-9564-8#preview>

[27] scikit-learn. "1.10. Decision Trees" <https://scikit-learn.org/stable/modules/tree.html>

[28] scikit-learn. "1.4. Support Vector Machines" <https://scikit-learn.org/stable/modules/svm.html>

[29] scikit-learn. "1.6 Nearest Neighbors" <https://scikit-learn.org/stable/modules/neighbors.html>

[30] scikit-learn. "1.11.2 Ensembles: Random Forests" <https://scikit-learn.org/stable/modules/ensemble.html#random-forests-and-other-randomized-tree-ensembles>

[31] Classical models training and testing. Colaboratory notebook. <https://colab.research.google.com/drive/1osX_SlaQFnSGtkBIgdmHDSVlice31Jji?usp=sharing>

[32] Pytorch tutorials: <https://pytorch.org/tutorials/>

[33] Drive folder with all sources: <https://drive.google.com/drive/folders/1BgCM4ZWRHexlksN7y2nMobdcFvwyMtju?usp=sharing>