**Machine Learning Project Documentation**

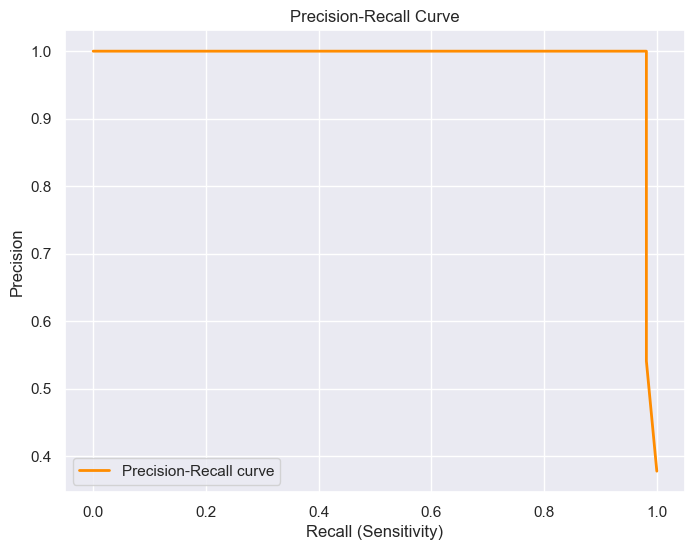
**Model Refinement**

**1. Overview**

In the model refinement phase, our objective is to enhance the performance of the Random Forest model designed for predicting the benign or malignant nature of breast cancer masses. This phase involves a systematic evaluation process, employing refinement techniques to optimize the model's predictive capabilities.

**2. Model Evaluation**

Building on the initial evaluation, our Random Forest model exhibited promising results. Precision, recall, and F1-score were key metrics assessed during the evaluation phase. The randomness that comes with Random Forests gives the ability to generalize better and also the ensemble advantage of it gives us all the benefits of a decision tree while minimizing its drawbacks. The accuracy, weighted precision, f1-score and precision are all 0.99 so there is not much left in terms improvement. But we would still try to manipulate the hyperparameters and observe what happens.



**Figure 1:** Precision-Recall Curve for the Random Forest

**3. Refinement Techniques**

For refining the model, feature selection based on its importance and hyperparameter tuning were used.

**4. Hyperparameter Tuning**

The RandomForestClassifier from sklearn library has the following hyperparameters:

**{'bootstrap': True,  
'criterion': 'mse',  
'max\_depth': None,  
'max\_features': 'auto',  
'max\_leaf\_nodes': None,  
'min\_impurity\_decrease': 0.0,  
'min\_impurity\_split': None,  
'min\_samples\_leaf': 1,  
'min\_samples\_split': 2,  
'min\_weight\_fraction\_leaf': 0.0,  
'n\_estimators': 10,  
'n\_jobs': 1,  
'oob\_score': False,  
'random\_state': 42,  
'verbose': 0,  
'warm\_start': False}**

According to Breiman [1] the most important hyperparameters are:

* **Number of trees(**n\_estimators**)**: The number of trees in the forest. Increasing the number of trees decreases the variance of the forest estimator and improves its accuracy, but also increases the computational cost.
* **Max depth**: The maximum depth of each tree in the forest. Increasing the maximum depth of the trees increases the complexity of the model and can lead to overfitting.
* **Minimum samples per leaf**: The minimum number of samples required to be at a leaf node. Increasing this value can help prevent overfitting.
* **Minimum samples per split**: The minimum number of samples required to split an internal node. Increasing this value can help prevent overfitting.
* **Bootstrap**: Whether or not to use bootstrap samples when building trees. Bootstrap samples are random samples with replacement from the original dataset. Using bootstrap samples can help reduce the variance of the forest estimator.
* **Criterion**: The function used to measure the quality of a split. The two most common criteria are Gini impurity and entropy. Default value is Gini.
* **Max features**: The maximum number of features to consider when looking for the best split. This is usually set to the square root of the total number of features in the dataset.
* **Random state**: The seed used by the random number generator. This can be used to ensure reproducibility of the results.

We first did a Random Search with Cross Validation to have a rough idea about the hyperparameter values. The parameters we decided to try were the following:

{'bootstrap': [True, False],

'max\_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],

'max\_features': ['log2', 'sqrt'],

'min\_samples\_leaf': [1, 2, 4],

'min\_samples\_split': [2, 5, 10],  
 'n\_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}

The code to run the Random Search:

rf\_random = RandomizedSearchCV(estimator = rf, param\_distributions =

random\_grid, n\_iter = 100, cv = 3, verbose=2, random\_state=42, n\_jobs = -1)

The best parameters:

{'n\_estimators': 1000,

'min\_samples\_split': 5,

'min\_samples\_leaf': 2,

'max\_features': 'sqrt',

'max\_depth': 10,

'bootstrap': True}

When we compared the result obtained from a Random Forest model with these parameters and one with default hyperparameters values, except n\_estimators set to 100, the results were identical.

The Parameters:

{'bootstrap': True,

'max\_depth': None,

'max\_features': 'sqrt',

'max\_leaf\_nodes': None,

'min\_samples\_leaf': 1,

'min\_samples\_split': 2,

'n\_estimators': 100}

Random Forest Accuracy (Base Model): 0.993006993006993

Classification Report:

precision recall f1-score support

0 0.99 1.00 0.99 89

1 1.00 0.98 0.99 54

accuracy 0.99 143

macro avg 0.99 0.99 0.99 143

weighted avg 0.99 0.99 0.99 143

The Parameters:

{'bootstrap': True,

'max\_depth': 10,

'max\_features': 'sqrt',

'min\_samples\_leaf': 2,

'min\_samples\_split': 5,

'n\_estimators': 1000}

Random Forest Accuracy (RandomSearchCV): 0.993006993006993

Classification Report:

precision recall f1-score support

0 0.99 1.00 0.99 89

1 1.00 0.98 0.99 54

accuracy 0.99 143

macro avg 0.99 0.99 0.99 143

weighted avg 0.99 0.99 0.99 143

After this search, we narrowed down on the hyperparameters that could be tuned to possibly gain a better result. Therefore, we decided to try a Grid Search with Cross Validation as well. The parameters chosen for the search were:

param\_grid = {

    'bootstrap': [True],

    'max\_depth': [80, 90, 100, 110],

    'max\_features': [2, 3],

    'min\_samples\_leaf': [3, 4, 5],

    'min\_samples\_split': [8, 10, 12],

    'n\_estimators': [100, 200, 300, 1000]

}

The code to run the search:

rf = RandomForestClassifier()

grid\_search = GridSearchCV(estimator = rf, param\_grid = param\_grid,

                          cv = 3, n\_jobs = -1, verbose = 2)

The best parameters and the result obtained:

{'bootstrap': True,

'max\_depth': 90,

'max\_features': 2,

'min\_samples\_leaf': 3,

'min\_samples\_split': 10,

'n\_estimators': 200}

Random Forest Accuracy (GridSearchCV): 0.993006993006993

Classification Report:

precision recall f1-score support

0 0.99 1.00 0.99 89

1 1.00 0.98 0.99 54

accuracy 0.99 143

macro avg 0.99 0.99 0.99 143

weighted avg 0.99 0.99 0.99 143

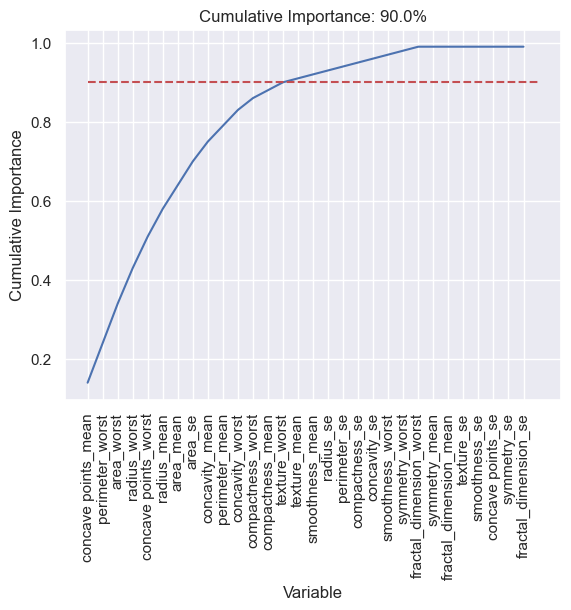
**5. Feature Selection**

Feature selection based on importance yielded no better result. But, in hopes of bettering the performance of training the model we did do it.

In total, we have 30 features. Their importance are as follows:

| Variable | Importance |  
|------------------------------|------------|  
| concave points\_mean | 0.14 |  
| perimeter\_worst | 0.1 |  
| area\_worst | 0.1 |  
| radius\_worst | 0.09 |  
| concave points\_worst | 0.08 |  
| radius\_mean | 0.07 |  
| area\_mean | 0.06 |  
| area\_se | 0.06 |  
| concavity\_mean | 0.05 |  
| perimeter\_mean | 0.04 |  
| concavity\_worst | 0.04 |  
| compactness\_worst | 0.03 |  
| compactness\_mean | 0.02 |  
| texture\_worst | 0.02 |  
| texture\_mean | 0.01 |  
| smoothness\_mean | 0.01 |  
| radius\_se | 0.01 |  
| perimeter\_se | 0.01 |  
| compactness\_se | 0.01 |  
| concavity\_se | 0.01 |  
| smoothness\_worst | 0.01 |  
| symmetry\_worst | 0.01 |  
| fractal\_dimension\_worst | 0.01 |  
| symmetry\_mean | 0.0 |  
| fractal\_dimension\_mean | 0.0 |  
| texture\_se | 0.0 |  
| smoothness\_se | 0.0 |  
| concave points\_se | 0.0 |  
| symmetry\_se | 0.0 |  
| fractal\_dimension\_se | 0.0 |

The Cumulative Importance graph is as follows:



The red line in the figure shows the 90% cumulated importance and that equals to 14 features.

A model trained with these features yields the same result as our best model:

Random Forest Accuracy with 14 features: 0.993006993006993

precision recall f1-score support

0 0.99 1.00 0.99 89

1 1.00 0.98 0.99 54

accuracy 0.99 143

macro avg 0.99 0.99 0.99 143

weighted avg 0.99 0.99 0.99 143

The performance is measure over 100 trainings of each model. The performance gain was about 1% in my 6-core machine (Ryzen 5 6600H)  
Average Execution time with all of the features: 0.1912687879999612 seconds  
Average Execution time with 14 important features: 0.18599525699997685 seconds  
Performance Gain: 1.0283530402067458 %

**Test Submission**

**1. Overview**

The test submission phase involved preparing the model for deployment and evaluating its performance on a dedicated test dataset. The primary steps included data preparation, model application, and metric evaluation.

**2. Data Preparation for Testing**

Our test dataset is actually derived from the main dataset. The main dataset was split into two datasets, training and testing datasets with a 75/25 split after all of the preprocessing was done.

**3. Model Application**

The trained Random Forest model, obtained after a grid search, was applied to the test dataset using the .predict method. The best model from the grid search, referred to as best\_grid, was utilized for predictions.

best\_grid = grid\_search.best\_estimator\_

best\_grid\_pred = best\_grid.predict(X\_test)

**4. Test Metrics**

Evaluation metrics such as accuracy classification score, precision-recall, and F1-score were employed to assess model performance. The Random Forest model achieved an accuracy of 99.3%. Detailed classification reports provided insights into precision, recall, and F1-score for both classes. The performance of our model is:

Random Forest Accuracy: 0.993006993006993

Classification Report:

precision recall f1-score support

0 0.99 1.00 0.99 89

1 1.00 0.98 0.99 54

accuracy 0.99 143

macro avg 0.99 0.99 0.99 143

weighted avg 0.99 0.99 0.99 143

**5. Code Implementation**

The code implementation involved refining the Random Forest model using hyperparameter tuning through grid and random searches. The final model's performance was compared with the base model and the best model obtained through hyperparameter tuning.

For model refinement:

# MODEL REFINEMENT

# Defining the grid search parameters for each model

# Number of trees in random forest

n\_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]

# Number of features to consider at every split

max\_features = ['log2', 'sqrt']

# Maximum number of levels in tree

max\_depth = [int(x) for x in np.linspace(10, 110, num = 11)]

max\_depth.append(None)

# Minimum number of samples required to split a node

min\_samples\_split = [2, 5, 10]

# Minimum number of samples required at each leaf node

min\_samples\_leaf = [1, 2, 4]

# Method of selecting samples for training each tree

bootstrap = [True, False]

# Create the random grid

random\_grid = {'n\_estimators': n\_estimators,

               'max\_features': max\_features,

               'max\_depth': max\_depth,

               'min\_samples\_split': min\_samples\_split,

               'min\_samples\_leaf': min\_samples\_leaf,

               'bootstrap': bootstrap}

# Use the random grid to search for best hyperparameters

# First, the base model to tune

rf = RandomForestClassifier()

# Random search of parameters, using 3 fold cross validation,

# search across 100 different combinations

rf\_random = RandomizedSearchCV(estimator = rf, param\_distributions = random\_grid, n\_iter = 100, cv = 3, verbose=2, random\_state=42, n\_jobs = -1)

rf\_random.fit(X\_train, y\_train)

# Random Forest Classifier

base\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

base\_model.fit(X\_train, y\_train)

base\_model\_pred = random\_forest\_model.predict(X\_test)

accuracy\_random\_forest = accuracy\_score(y\_test, base\_model\_pred)

print("Random Forest Accuracy:", accuracy\_random\_forest)

print("Classification Report:")

print(classification\_report(y\_test, base\_model\_pred))

print(base\_model.get\_params())

rand\_model\_pred = rf\_random.best\_estimator\_.predict(X\_test)

accuracy\_random\_forest = accuracy\_score(y\_test, rand\_model\_pred)

print("Random Forest Accuracy:", accuracy\_random\_forest)

print("Classification Report:")

print(classification\_report(y\_test, rand\_model\_pred))

# GridSearch

from sklearn.model\_selection import GridSearchCV

# Create the parameter grid based on the results of random search

param\_grid = {

    'bootstrap': [True],

    'max\_depth': [80, 90, 100, 110],

    'max\_features': [2, 3],

    'min\_samples\_leaf': [3, 4, 5],

    'min\_samples\_split': [8, 10, 12],

    'n\_estimators': [100, 200, 300, 1000]

}

# Create a based model

rf = RandomForestClassifier()

# Instantiate the grid search model

grid\_search = GridSearchCV(estimator = rf, param\_grid = param\_grid,

                          cv = 3, n\_jobs = -1, verbose = 2)

grid\_search.fit(X\_train, y\_train)

best\_grid = grid\_search.best\_estimator\_

best\_grid\_pred = best\_grid.predict(X\_test)

best\_grid\_accuracy = accuracy\_score(y\_test, best\_grid\_pred)

print("Random Forest Accuracy:", best\_grid\_accuracy)

print("Classification Report:")

print(classification\_report(y\_test, best\_grid\_pred))

**Conclusion**

The model refinement and test submission phases demonstrated the effectiveness of the Random Forest classifier in accurately predicting breast cancer malignancy. Although hyperparameter turning did not result in a more accurate model it was worth practicing a portion of what we learned in FTL Program. The final Random Forest model, after grid search optimization, achieved a remarkable accuracy of 99.3% on the test dataset. The precision, recall, and F1-score metrics further validated the robustness of the model. The project's success can be attributed to meticulous model refinement, leveraging scikit-learn's RandomForestClassifier, and drawing insights from external resources such as Koehrsen's hyperparameter tuning guide and Jolliffe's work on Principal Component Analysis.

**References**

1. Jolliffe, I. T. (2002). Principal Component Analysis in Exploratory Data Analysis. Springer.
2. Koehrsen, Will (2018). Hyperparameter Tuning the Random Forest in Python. (<https://towardsdatascience.com/>)
3. scikit-learn. (n.d.). RandomForestClassifier. Retrieved 12/11/2023, from <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>