Sigurður Ágúst Jakobsson

# Project: Assignment 9 – Random Forests

A few things to keep in mind when evaluating output from Sklearn functions and this dataset.

Usually the default values for random forests are: max\_feature\_array Default = sqrt and n\_estimator\_array Default = 100. The dataset has 30 features so Sqrt(30) = 5,48 and log2(30) = 4,91. When reading from decision trees the output is described as follows in the documentation: “Thus in binary classification, the count of true negatives is C\_0,0, false negatives is C\_1,0, true positives is C\_1,1 and false positives is C\_0,1.“

## Section 1.2

Now run CancerClassifier with a DecisionTreeClassifier. and evaluate the performance with the methods that you have finished implementing. Answer the following questions:

1. Show the result for each metric (confusion matrix, accuracy, precision, recall, cross validation accuracy)

Here are the results from one such fitting with default parameters.

Confusion Matrix [true (rows), predicted (columns)]:

[[ 59 4]

[ 2 106]]

Accuracy: 0.9649122807017544

Precision: 0.9636363636363636

Recall: 0.9814814814814815

Cross Validation Accuracy: 0.913941102756892

1. What does the precision and recall tell us that the accuracy can't?

Since breast cancer is counted as counted as a positive/one we can use the formula TP / (TP+FP) to calculate precision. Since breast cancer is counted as counted as a positive/one we can use the formula TP / (TP+FN) to calculate recall. Precision and recall therefore tell us what kind of errors we are making in classification. They are a ratio of true positives over false negatives and false positives. In classification the different errors can have different costs associated with them. For example, a false negative (missed diagnosis) might be more serious than a false positive (diagnosed but no cancer). Further testing could rule out cancer on the false positive, but the false negative might not be examined further resulting in a patient with cancer that nothing is done about. Therefore, false negatives can be counted as more serious and we would probably want to maximize recall in this classification, even at the cost of accuracy or precision.

1. What could possibly explain the difference between accuracy and cross validation accuracy?

Accuracy is a measure on one split of test to training data. There is a random element involved in the split. Cross validation accuracy splits all the data in ten parts in this case, calculates the accuracy on each one against the remaining nine. The accuracies of the splits are then averaged. This is simply a different method of calculating explaining the difference between them. It probably gives a better estimate of how the classifier will do on different datasets than only calculating the accuracy on one split. Another single random split may give different results, but taking an average makes up for some of this variance.

1. How would you suggest a confusion matrix, precision and recall for cross validation would be formulated?

I would suggest using a similar methodology to how cross validation accuracy is calculated. Calculate precision, recall and a confusion matrix for each of the ten data splits against the remaining nine. Then take the mean of these results and return them.

## Section 2.1

Implement a vanilla random forest and apply to the problem in the same way that the decision tree was evaluated, using your CancerClassifier.

1. Show the result for each metric (confusion matrix, accuracy, precision, recall, cross validation accuracy)

Here are the results from a test with the default values of the Random forest classifier:

Testing:

Max Features: sqrt

N Estimators: 100

Confusion Matrix [true (rows), predicted (columns)]:

[[ 59 4]

[ 0 108]]

Accuracy: 0.9766081871345029

Precision: 0.9642857142857143

Recall: 1.0

Cross Validation Accuracy: 0.9631265664160402

1. What is the best combination of a total number of trees in the forest (n\_estimators) and the maximum number of features considered in each split (max\_features) that you can find? What are the metric results for this parameter selection?

I made a script to test all combinations of the following parameters:

max\_feature\_array = [3, 4, "sqrt", "log2", 6, 7, 8, 9, 10] #Default = sqrt

n\_estimator\_array = [10, 20, 40, 80, 100, 160, 320] #Default = 100

This is a procedure in the hand-in code called \_test\_2\_1(). The relevant output is as follows:

Best results:

Accuracy:

Value: 0.9883040935672515

Max Features: 4

N Estimators: 100

Precision:

Value: 0.9818181818181818

Max Features: 4

N Estimators: 100

Recall:

Value: 1.0

Max Features: 3

N Estimators: 20

Cross Validation Accuracy:

Value: 0.9736842105263157

Max Features: 6

N Estimators: 80

We can see that *Max Features: 4/N Estimators: 100* does well with respect to accuracy and precision. Recall is 1.0 as well, but the script takes the first run that that has the max value as the one to return. Many combinations of parameters had 1.0 for recall. The highest cross-validation accuracy was with *Max Features: 6/N Estimators: 80*. The metric results for these parameters can be seen below. This is an interesting problem for the independent section - it would make sense to do cross validation for precision and recall as well. Maybe the good result on *Max Features: 4/N Estimators: 100* was due to a lucky single split.

Testing:

Max Features: 4

N Estimators: 100

Confusion Matrix:

[[ 61 2]

[ 0 108]]

Accuracy: 0.9883040935672515

Precision: 0.9818181818181818

Recall: 1.0

Cross Validation Accuracy: 0.9649122807017545

Testing:

Max Features: 6

N Estimators: 80

Confusion Matrix:

[[ 59 4]

[ 0 108]]

Accuracy: 0.9766081871345029

Precision: 0.9642857142857143

Recall: 1.0

Cross Validation Accuracy: 0.9736842105263157

## Section 2.2

Turn in your plot in your PDF as 2\_2\_1.png

Chart, histogram

Description automatically generated

## Section 2.3

1. Describe how feature importance is calculated
2. Which feature is the most important and which is the least important. Use information from either [assignment 8](https://file+.vscode-resource.vscode-cdn.net/c%3A/Users/Siggi/Desktop/Sk%C3%B3linn/%C3%96nn%209/Gagnan%C3%A1m%20og%20vitv%C3%A9lar/Programming%20Assignments/T809DATA_2022/08_SVM/README.md) or [here](https://www.kaggle.com/uciml/breast-cancer-wisconsin-data) to name these features.

The feature importance in RandomForestClassifier.feature\_importances\_ is calculated using the reduction in the Gini Impurity, according to the documentation. The higher the reduction a feature produces, the more important it is.

Below is the output of importance from one data fitting on a single split of data:

Feature Importance:

[27 22 7 5 6 20 2 25 3 24 28 13 0 23 12 1 16 9 17 21 19 26 14 29 18 4 15 8 11 10]

The key for the cancer data from assignment 8 is:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0]radius\_mean | [1]texture\_mean | [2]perimeter\_mean | [3]area\_mean | [4]smoothness\_mean | [5]compactness\_mean |
| [6]concavity\_mean | [7]concave points\_mean | [8]symmetry\_mean | [9]fractal\_dimension\_mean | [10]radius\_se | [11]texture\_se |
| [12]perimeter\_se | [13]area\_se | [14]smoothness\_se | [15]compactness\_se | [16]concavity\_se | [17]concave points\_se |
| [18]symmetry\_se | [19]fractal\_dimension\_se | [20]radius\_worst | [21]texture\_worst | [22]perimeter\_worst | [23]area\_worst |
| [24]smoothness\_worst | [25]compactness\_worst | [26]concavity\_worst | [27]concave points\_worst | [28]symmetry\_worst | [29]fractal\_dimension\_worst |

The three most important features from this training are [27]concave points\_worst, [22]perimeter\_worst and [7]concave points\_mean in decreasing order.

The three least important features from this training are [8]symmetry\_mean, [11]texture\_se and [10]radius\_se in decreasing order.

## Section 2.4

Turn in your plot in your PDF as 2\_4\_1.png.

Chart

Description automatically generated

## Section 2.5

1. What can be said about the relationship between the OOB error rate and the number of estimators?
2. Do all three types of ensembles follow this correlation?

The error rate seems to decrease first as n\_estimators rises and then increase again after 100-140 depending on the type of ensemble. This is not strict for all ensembles as there are a few local minima outside this range for max\_features = None and max\_features = ‘sqrt’. This does suggest some sort of sweet spot for n\_estimators though even though it is not totally conclusive. There is definitely a range where all of them can operate at their best level.

## Section 3.1

Plot the same feature importance bar plot as before. Upload it as 3\_1\_1.png.

1. Show the result for each metric (confusion matrix, accuracy, precision, recall, cross validation accuracy)
2. What is the most important feature and the least important feature?

The vanilla ExtraTreesClassifier produces the results shown below:

Chart, histogram

Description automatically generated

Feature Importance:

[22 7 23 27 6 20 26 0 2 3 25 13 21 1 10 12 5 24 28 16 8 29 17 4 9 18 15 11 14 19]

The feature importance can be seen above. The key was:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0]radius\_mean | [1]texture\_mean | [2]perimeter\_mean | [3]area\_mean | [4]smoothness\_mean | [5]compactness\_mean |
| [6]concavity\_mean | [7]concave points\_mean | [8]symmetry\_mean | [9]fractal\_dimension\_mean | [10]radius\_se | [11]texture\_se |
| [12]perimeter\_se | [13]area\_se | [14]smoothness\_se | [15]compactness\_se | [16]concavity\_se | [17]concave points\_se |
| [18]symmetry\_se | [19]fractal\_dimension\_se | [20]radius\_worst | [21]texture\_worst | [22]perimeter\_worst | [23]area\_worst |
| [24]smoothness\_worst | [25]compactness\_worst | [26]concavity\_worst | [27]concave points\_worst | [28]symmetry\_worst | [29]fractal\_dimension\_worst |

The three most important features from this training are [22]perimeter\_worst, [7]concave points\_mean and [23]area\_worst in decreasing order.

The three least important features from this training are [11]texture\_se, [14]smoothness\_se and [19]fractal\_dimension\_se in decreasing order.

There is some similarity to the prior feature importance but it there is still a somewhat different order. This may have to do with the random split of data.

The results from the metrics are below:

Confusion Matrix [true (rows), predicted (columns)]:

[[ 60 3]

[ 0 108]]

Accuracy: 0.9824561403508771

Precision: 0.972972972972973

Recall: 1.0

Cross Validation Accuracy: 0.9683897243107771

## Section 3.2

Turn in your plot in your PDF as 3\_2\_1.png.

Chart, line chart

Description automatically generated

## Independent Section

I want to try to do cross validation on recall, precision and a confusion matrix with a RandomForestClassifier and *Max Features: 4/N Estimators: 100* and *Max Features: 6/N Estimators: 80* to analyze the data from question 2.1 better. The plot from 2.4 suggests that sqrt does better than log2. 6 is closer to sqrt and 4 is closer to log2. Maybe I can get a clearer picture of which set of parameters really performs the best if I do cross-validation across all metrics.

I do this with script \_indep() in the handin code which uses helper function x\_fold\_split(features: np.ndarray, targets: np.ndarray, x: int).

The output is:

Run - n\_estimators: 100 , max\_features: 4

Cross Validation Accuracy: 0.9665659340659339

Cross Validation Precision: 0.965209880067871

Cross Validation Recall: 0.9815448600520444

Cross Validation CM:

[[20. 1.2]

[ 0.7 35. ]]

Run - n\_estimators: 80 , max\_features: 6

Cross Validation Accuracy: 0.9650274725274725

Cross Validation Precision: 0.9657973759332455

Cross Validation Recall: 0.9791058356618005

Cross Validation CM:

[[20. 1.2]

[ 0.8 34.9]]

There is some randomness involved with the permutations here, but anecdotally after a few runs it seems that *n\_estimators: 100 , max\_features: 4* is actually doing better than *n\_estimators: 80 , max\_features: 6*. This is the opposite of what I expected. The cross-validation accuracy is now lower for *n\_estimators: 80 , max\_features: 6*. Some further statistical analysis would probably be necessary to make a decision on what parameters are best to use with more certainty. Maybe also a second opinion and more testing of the script. The additional code that I made can be seen below:

def x\_fold\_split(features: np.ndarray, targets: np.ndarray, x: int):

    '''Shuffle the features and targets in unison. Split into x sections for cross validation.'''

    #Standard code from split\_train\_test

    p = np.random.permutation(features.shape[0])

    features = features[p]

    targets = targets[p]

    section\_len = features.shape[0] // x

    feature\_data\_splits = []

    target\_data\_splits = []

    for section in range(x):

        if section == x-1:

            features\_sec, targets\_sec = features[section\_len\*section:, :], targets[section\_len\*section:]

        else:

            features\_sec, targets\_sec = features[section\_len\*section:section\_len\*(section+1), :], targets[section\_len\*section:section\_len\*(section+1)]

        feature\_data\_splits.append(features\_sec)

        target\_data\_splits.append(targets\_sec)

    return feature\_data\_splits, target\_data\_splits

def \_indep():

    cancer = load\_breast\_cancer() #D=30

    X = cancer.data  # all feature vectors

    t = cancer.target  # all corresponding labels

    validation\_fold = 10

    feature\_data\_splits, target\_data\_splits = x\_fold\_split(X, t, validation\_fold)

    #Parameters to test

    runs = [(100, 4), (80, 6)]

    for i in runs:

        accuracies = []

        precisions = []

        recalls = []

        cms = []

        my\_classifier = RandomForestClassifier(n\_estimators=i[0], max\_features=i[1])

        #Calculate stats for each fold

        for fold in range(validation\_fold):

            test\_X = feature\_data\_splits[fold]

            test\_t = target\_data\_splits[fold]

            #Make arrays

            if fold == 0:

                train\_x = feature\_data\_splits[1]

                train\_t = target\_data\_splits[1]

                for others in range(2, validation\_fold):

                    train\_x = np.append(train\_x, feature\_data\_splits[others], axis=0)

                    train\_t = np.append(train\_t, target\_data\_splits[others], axis=0)

            else:

                train\_x = feature\_data\_splits[0]

                train\_t = target\_data\_splits[0]

                for others in range(1, validation\_fold):

                    if fold != others:

                        train\_x = np.append(train\_x, feature\_data\_splits[others], axis=0)

                        train\_t = np.append(train\_t, target\_data\_splits[others], axis=0)

            #Train, predict and analyze for fold

            my\_classifier.fit(train\_x, train\_t)

            predictions = my\_classifier.predict(test\_X)

            accuracies.append(accuracy\_score(test\_t, predictions))

            precisions.append(precision\_score(test\_t, predictions))

            recalls.append(recall\_score(test\_t, predictions))

            cms.append(confusion\_matrix(test\_t, predictions))

        #Calculate mean for run

        mean\_accuracy = 0

        mean\_precision = 0

        mean\_recall = 0

        mean\_cm = 0

        for fold in range(validation\_fold):

            mean\_accuracy += accuracies[fold] / validation\_fold

            mean\_precision += precisions[fold] / validation\_fold

            mean\_recall += recalls[fold] / validation\_fold

            mean\_cm += cms[fold] / validation\_fold

        #Print run stats

        print('\nRun - n\_estimators:', i[0], ', max\_features:', i[1])

        print('Cross Validation Accuracy:', mean\_accuracy)

        print('Cross Validation Precision:', mean\_precision)

        print('Cross Validation Recall:', mean\_recall)

        print('Cross Validation CM:', mean\_cm)