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

2. 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.

3. 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.

4. 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

2. 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

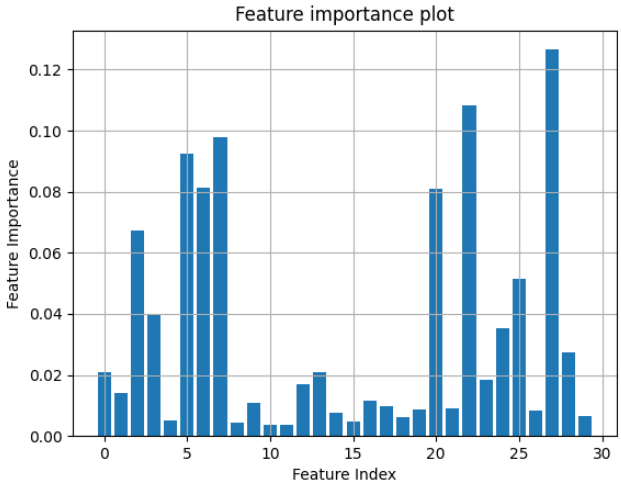
We can see that *Max Features: 4/N Estimators: 100* does well with respect to accuracy and precision. Recall is 1.0 as well, but the script takes the first run that that has the max value as the one to return. Many combinations of parameters had 1.0 for recall. The highest cross-validation accuracy was 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`



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 or here 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:

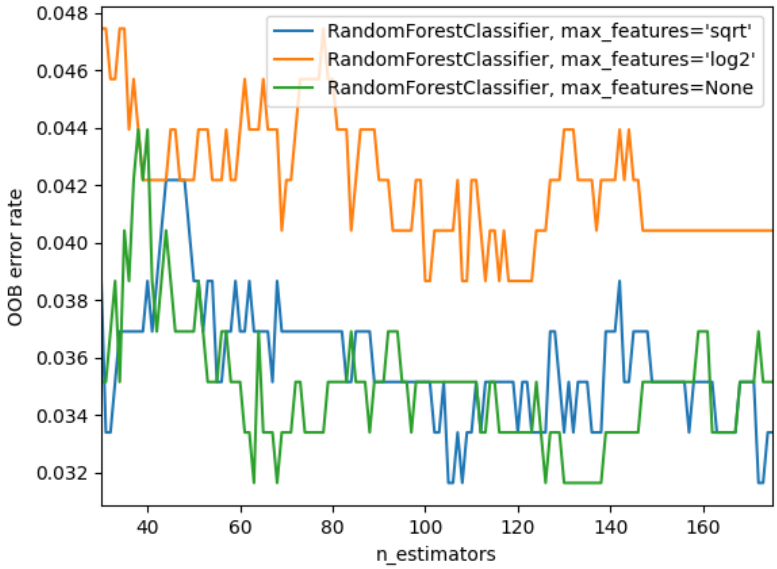
[0]radius_mean	[1]texture_mean	[2]perimeter_mean	[3]area_mean	[4]smoothness_mean	[5]compactness_mean
[6]concavity_mean	[7]concave points_mean	[8]symmetry_mean	[9]fractal_dimension_mean	[10]radius_se	[11]texture_se
[12]perimeter_se	[13]area_se	[14]smoothness_se	[15]compactness_se	[16]concavity_se	[17]concave 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`.



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?

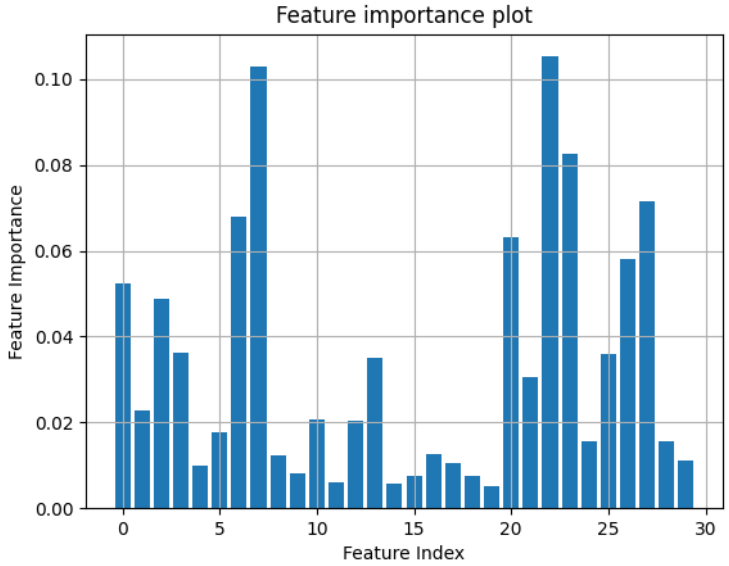
The error rate seems to decrease first as n_estimators rises and then increase again after 100-140 depending on the type of ensemble. This is not strict for all ensembles as there are a few local minima outside this range for max_features = None and max_features = 'sqrt'. This does suggest some sort 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:



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.

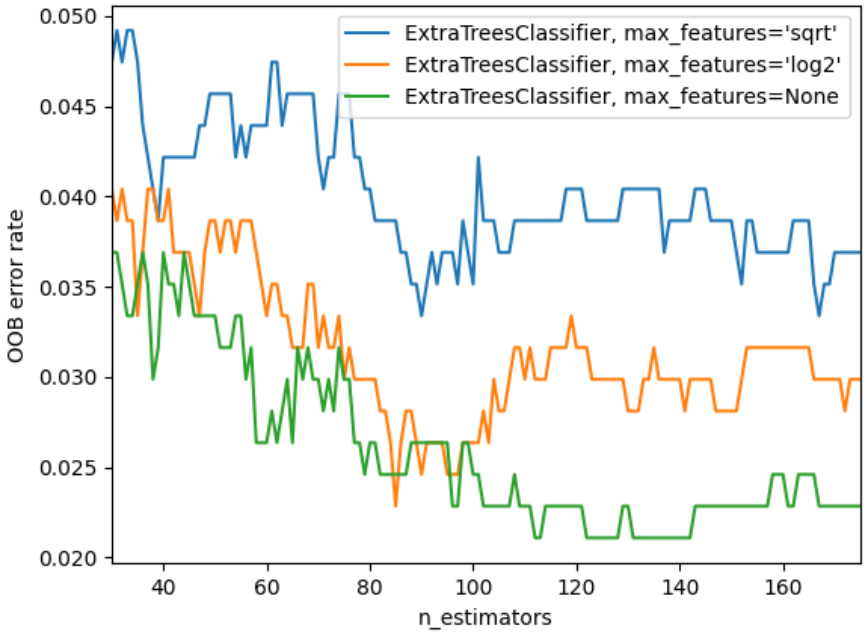
There is some similarity to the prior feature importance but it there is still a somewhat different order. This may have to do with the random split of data.

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`.



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)
```