Sigurður Ágúst Jakobsson

# Project: Assignment 10 – Boosting

## Section 1.2

Here is the code snippet to use so we don’t drop Age from the data. We remove age from the drop statement. Then we find all age values that are not null and take the mean. .fillna() fills values with NaN with the given value. In this case it is replacing NaN age with the mean age.

## Text Description automatically generated

## Section 2.2

I use a vanilla RFC (n\_estimators = 100, max\_features = sqrt) and get the following results from rfc\_train\_test(X\_train, t\_train, X\_test, t\_test):

(0.8134328358208955, 0.7731958762886598, 0.7281553398058253)

Accuracy = 0.8134328358208955

Precision = 0.7731958762886598

Recall = 0.7281553398058253

## Section 2.4

A vanilla GradientBoostingClassifier gives the following results:

(0.8246268656716418, 0.8111111111111111, 0.7087378640776699)

Accuracy = 0.8246268656716418

Precision = 0.8111111111111111

Recall = 0.7087378640776699

It does a little better than the RFC on accuracy and precision, but a little worse on recall. I think in this data analysis there is no difference in cost between False Negatives and False positives, so the goal should just be to maximise accuracy.

## Section 2.5

I filled in the values like this:

Text

Description automatically generated

I got the following results for the best parameter values:

{'n\_estimators': 100, 'max\_depth': 5, 'learning\_rate': 0.08}

I wish I could input more values but the n\_iter is set to 50, and won’t go above testing 50 permutations. To have all permutations of parameters tested I choose 3 x 4 x 4 values = 48 permutations. This gives me a warning:

C:\Users\Siggi\Desktop\Skólinn\Önn 9\Gagnanám og vitvélar\Programming Assignments\dmml\_venv\lib\site-packages\sklearn\model\_selection\\_search.py:306: UserWarning: The total space of parameters 48 is smaller than n\_iter=50. Running 48 iterations. For exhaustive searches, use GridSearchCV.

I tried values that were close to the defaults for the defaults for the GradientBoostingClassifier to try to get incremental improvements.

If I tried too many values the param\_search() would not go very deep into permutations specified.

## Section 2.6

I can’t get my gb\_optimized\_train\_test() to perform better with the given RandomizedSearchCV. I could guess more parameters but that is really the purpose of the function. I don’t know if I’m allowed to update the parts in the function that are not listed. I got:

(0.8208955223880597, 0.7777777777777778, 0.7475728155339806)

Accuracy = 0.8208955223880597

Precision = 0. 7777777777777778

Recall = 0. 7475728155339806

Only the recall is improved. I did some more work that I will outline in the independent section and updated the function with new values but couldn’t improve all 3 metrics. The best I could get was

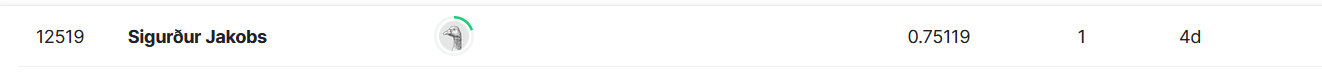
(0.8283582089552238, 0.7938144329896907, 0.7475728155339806) while the baseline was (0.8246268656716418, 0.8111111111111111, 0.7087378640776699). I couldn’t improve upon the precision. The final version of the param\_search was:

Text

Description automatically generated

## Section 3.2

This was my first submission.



## Independent Section

I wasn’t happy with the low number of permutations available from the giver param\_search function. The warning mentioned GridSearchCV, so I wanted to try this. I started by trying out all permutations from 50-100 in n\_estimators, 1-30 in max\_depth and 8 values for learning\_rate.

This started running for about 90 minutes without any info on how far it got. I shot it down and found out that it is possible to get a print out on screen of the progress and use more than one thread. I updated again and used fewer permutations. The function looked like this afterwards:

Text

Description automatically generated

This was more user friendly since I could at least get a status update on progress as well:

Graphical user interface, text

Description automatically generated

The output was {'learning\_rate': 0.1, 'max\_depth': 5, 'n\_estimators': 68}. This gave me a hint for a narrower range to use and I went back and updated the settings for section 2.6. I still couldn’t improve all metrics. Maybe this helps me with the submission to Kaggle though since that is looking only at accuracy which improved.

I updated the functions with better ranges and made the hand in function look at this.

The updated functions for this test of greedy parameter search are called gb\_optimized\_train\_test\_upd(X\_train, t\_train, X\_test, t\_test), param\_search\_upd(X, y) and \_create\_submission\_upd(). They are found at the bottom of this report. They are only slightly updated, but this was an interesting exercise in optimization. It was difficult to squeeze out better performance like this and time consuming. Maybe it’s better to take a more theoretical approach to looking at what models work well on the data in future.

I did manage to squeeze out about half a percentage increase in accuracy on a new submission like this: Graphical user interface, application

Description automatically generated

It was time consuming though and there is low marginal return on time on this method. That goes into the experience bank. Once this was done there was little time to try anything else to improve the model.

def param\_search\_upd(X, y):

    '''

    Perform randomized parameter search on the

    gradient boosting classifier on the dataset (X, y)

    '''

    # Create the parameter grid

    gb\_param\_grid = {

        'n\_estimators': list(range(50, 70, 1)),

        'max\_depth': list(range(4, 10, 1)),

        'learning\_rate': [0.095, 0.096, 0.097, 0.098, 0.099, 0.100, 0.101, 0.102]}

    # Instantiate the regressor

    gb = GradientBoostingClassifier()

    # Perform random search

    gb\_random = GridSearchCV(

        param\_grid=gb\_param\_grid,

        estimator=gb,

        scoring="accuracy",

        verbose=3,

        n\_jobs=-1,

        cv=4)

    # Fit randomized\_mse to the data

    gb\_random.fit(X, y)

    # Print the best parameters and lowest RMSE

    return gb\_random.best\_params\_

def gb\_optimized\_train\_test\_upd(X\_train, t\_train, X\_test, t\_test):

    '''

    Train a gradient boosting classifier on (X\_train, t\_train)

    and evaluate it on (X\_test, t\_test) with

    your own optimized parameters

    '''

    params = param\_search\_upd(X\_train, t\_train)

    gb\_classifier = GradientBoostingClassifier(n\_estimators=params['n\_estimators'], max\_depth=params['max\_depth'], learning\_rate=params['learning\_rate'])

    gb\_classifier.fit(X\_train, t\_train)

    predictions = gb\_classifier.predict(X\_test)

    accuracy = accuracy\_score(t\_test, predictions)

    precision = precision\_score(t\_test, predictions)

    recall = recall\_score(t\_test, predictions)

    return (accuracy, precision, recall)

def \_create\_submission\_upd():

    '''Create your kaggle submission

    '''

    (tr\_X, tr\_y), (tst\_X, tst\_y), submission\_X = get\_better\_titanic()

    params = param\_search\_upd(tr\_X, tr\_y)

    gb\_classifier = GradientBoostingClassifier(n\_estimators=params['n\_estimators'], max\_depth=params['max\_depth'], learning\_rate=params['learning\_rate'])

    gb\_classifier.fit(tr\_X, tr\_y)

    prediction = gb\_classifier.predict(submission\_X)

    build\_kaggle\_submission(prediction)