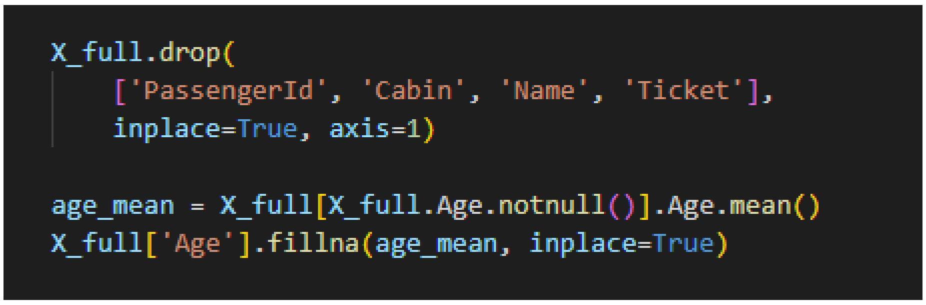
Sigurður Ágúst Jakobsson

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.



```
X_full.drop(
    ['PassengerId', 'Cabin', 'Name', 'Ticket'],
    inplace=True, axis=1)

age_mean = X_full[X_full.Age.notnull()].Age.mean()
X_full['Age'].fillna(age_mean, inplace=True)
```

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

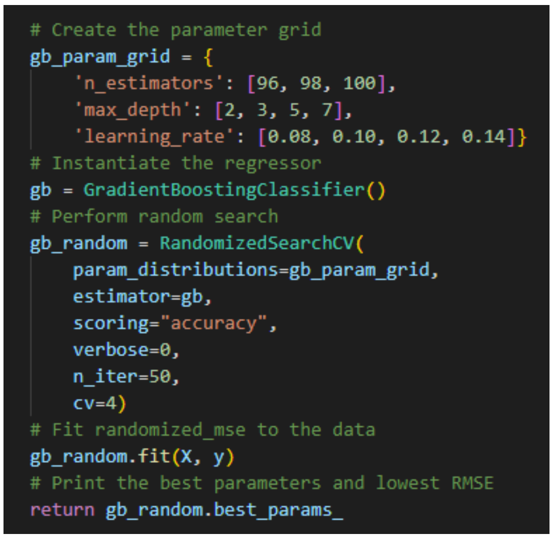
Precision = 0.8111111111111111

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:



```
# Create the parameter grid
gb_param_grid = {
    'n_estimators': [96, 98, 100],
    'max_depth': [2, 3, 5, 7],
    'learning_rate': [0.08, 0.10, 0.12, 0.14]}
# Instantiate the regressor
gb = GradientBoostingClassifier()
# Perform random search
gb_random = RandomizedSearchCV(
    param_distributions=gb_param_grid,
    estimator=gb,
    scoring="accuracy",
    verbose=0,
    n_iter=50,
    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_
```

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)

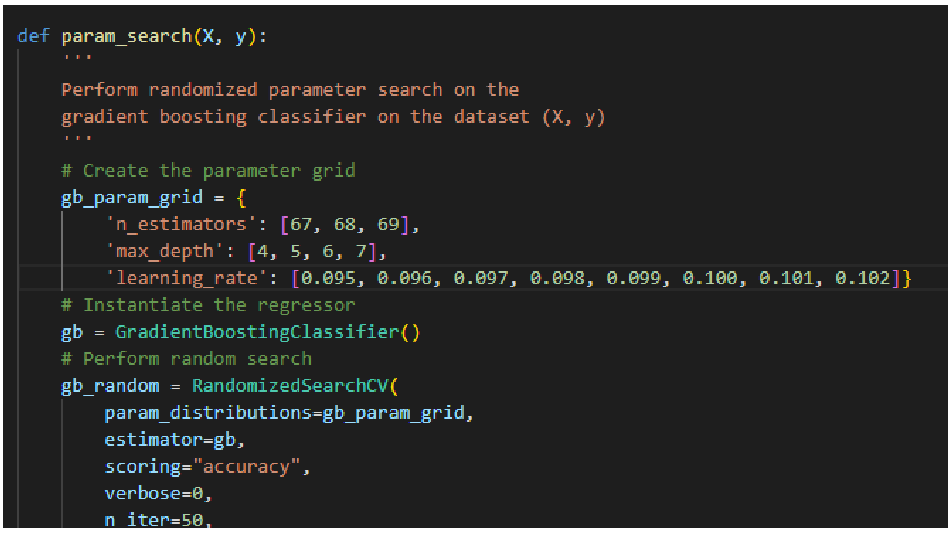
Accuracy = 0.8208955223880597

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

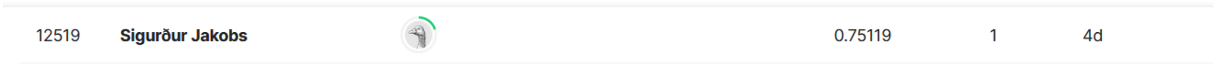
(0.8283582089552238, 0.7938144329896907, 0.7475728155339806) while the baseline was (0.8246268656716418, 0.8111111111111111, 0.7087378640776699). I couldn't improve upon the precision. The final version of the param_search was:



```
def param_search(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': [67, 68, 69],
        'max_depth': [4, 5, 6, 7],
        '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 = RandomizedSearchCV(
        param_distributions=gb_param_grid,
        estimator=gb,
        scoring="accuracy",
        verbose=0,
        n_iter=50,
```

Section 3.2

This was my first submission.

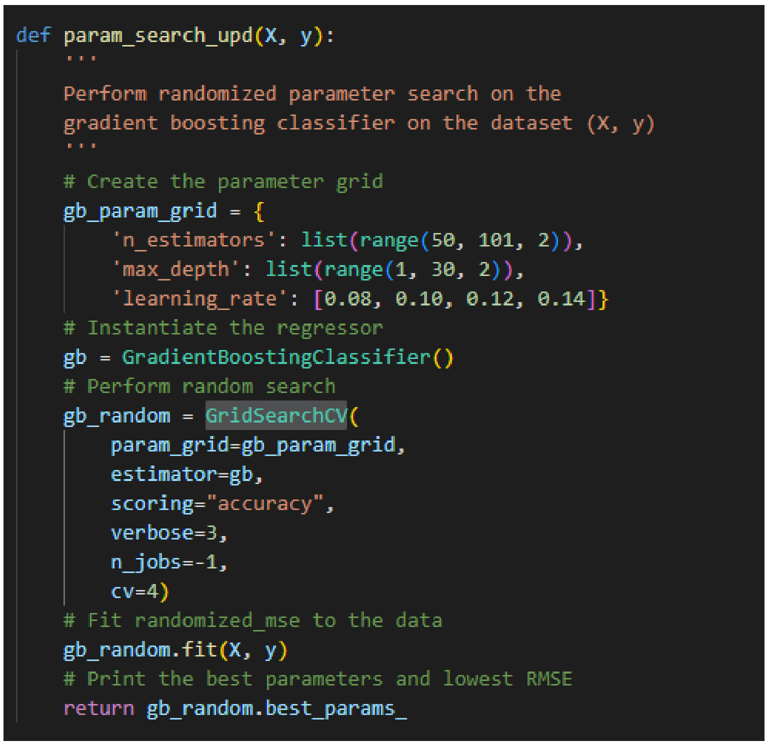


12519	**Sigurður Jakobs**	0.75119	1	4d

Independent Section

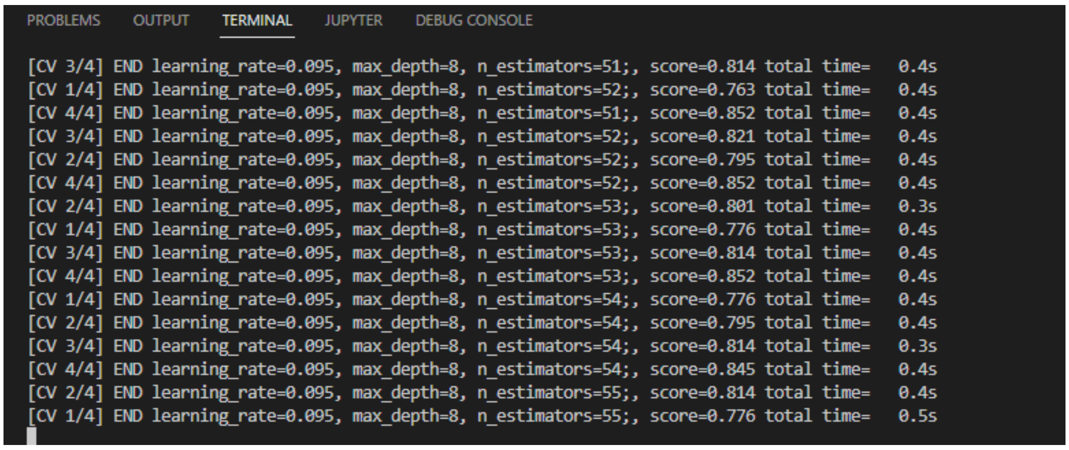
I wasn't happy with the low number of permutations available from the giver param_search function. The warning mentioned GridSearchCV, so I wanted to try this. I started by trying out all permutations from 50-100 in n_estimators, 1-30 in max_depth and 8 values for learning_rate.

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:



```python
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, 101, 2)),
        'max_depth': list(range(1, 30, 2)),
        'learning_rate': [0.08, 0.10, 0.12, 0.14]}
    # 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_
```

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



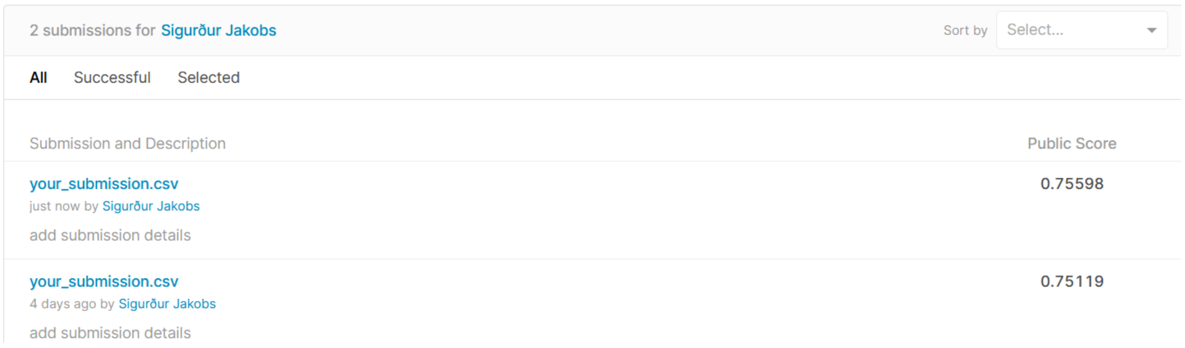
```
[CV 3/4] END learning_rate=0.095, max_depth=8, n_estimators=51;, score=0.814 total time=   0.4s
[CV 1/4] END learning_rate=0.095, max_depth=8, n_estimators=52;, score=0.763 total time=   0.4s
[CV 4/4] END learning_rate=0.095, max_depth=8, n_estimators=51;, score=0.852 total time=   0.4s
[CV 3/4] END learning_rate=0.095, max_depth=8, n_estimators=52;, score=0.821 total time=   0.4s
[CV 2/4] END learning_rate=0.095, max_depth=8, n_estimators=52;, score=0.795 total time=   0.4s
[CV 4/4] END learning_rate=0.095, max_depth=8, n_estimators=52;, score=0.852 total time=   0.4s
[CV 2/4] END learning_rate=0.095, max_depth=8, n_estimators=53;, score=0.801 total time=   0.3s
[CV 1/4] END learning_rate=0.095, max_depth=8, n_estimators=53;, score=0.776 total time=   0.4s
[CV 3/4] END learning_rate=0.095, max_depth=8, n_estimators=53;, score=0.814 total time=   0.4s
[CV 4/4] END learning_rate=0.095, max_depth=8, n_estimators=53;, score=0.852 total time=   0.4s
[CV 1/4] END learning_rate=0.095, max_depth=8, n_estimators=54;, score=0.776 total time=   0.4s
[CV 2/4] END learning_rate=0.095, max_depth=8, n_estimators=54;, score=0.795 total time=   0.4s
[CV 3/4] END learning_rate=0.095, max_depth=8, n_estimators=54;, score=0.814 total time=   0.3s
[CV 4/4] END learning_rate=0.095, max_depth=8, n_estimators=54;, score=0.845 total time=   0.4s
[CV 2/4] END learning_rate=0.095, max_depth=8, n_estimators=55;, score=0.814 total time=   0.4s
[CV 1/4] END learning_rate=0.095, max_depth=8, n_estimators=55;, score=0.776 total time=   0.5s
```

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:


2 submissions for Sigurður Jakobs

Sort by Select...

All Successful Selected

Submission and Description	Public Score
your_submission.csv just now by Sigurður Jakobs add submission details	0.75598
your_submission.csv 4 days ago by Sigurður Jakobs add submission details	0.75119

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.

```python
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)
```