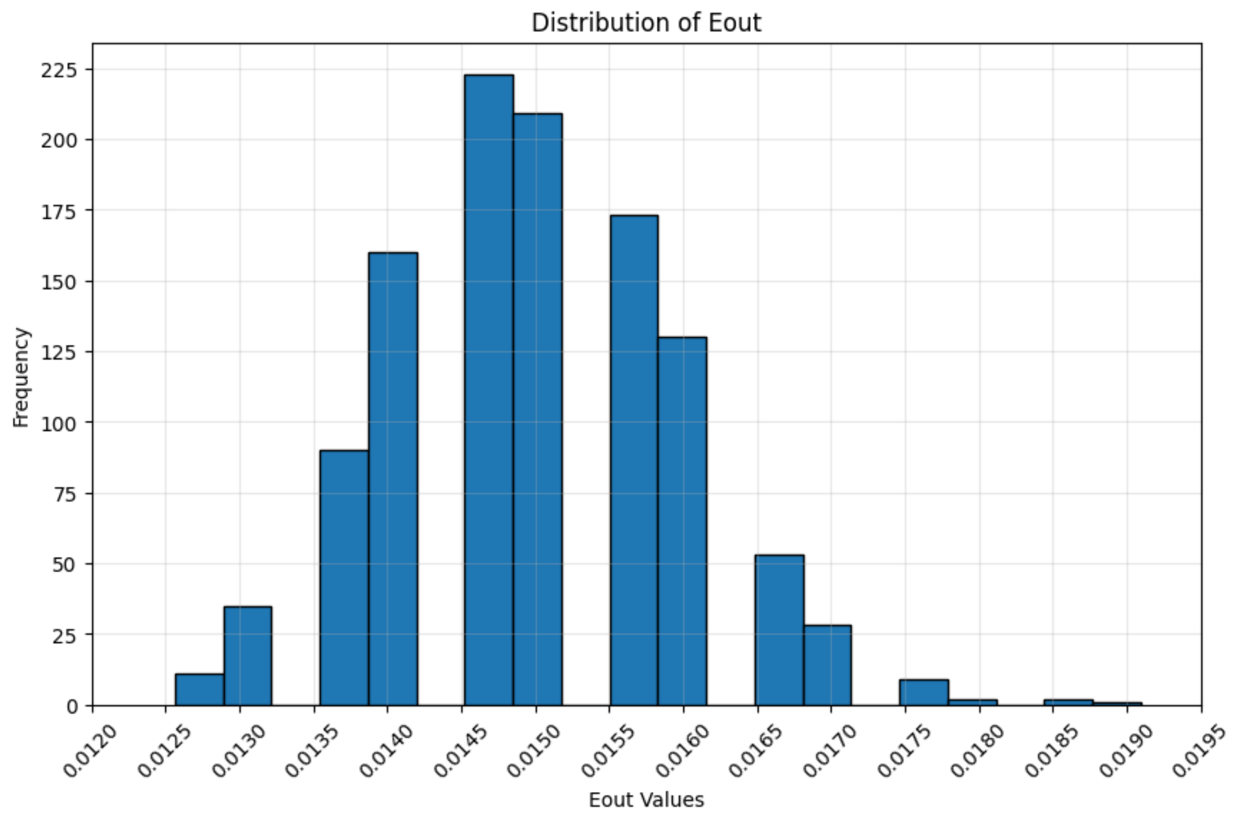
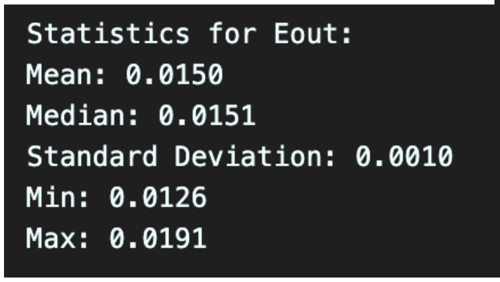
HTML homework 5: q11 report

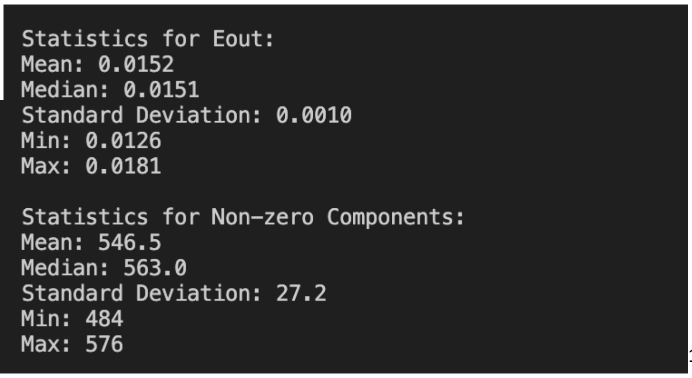
The resulting picture is as follows:



In order to compare with the previous question, some statistics are:



```
Statistics for Eout:
Mean: 0.0150
Median: 0.0151
Standard Deviation: 0.0010
Min: 0.0126
Max: 0.0191
```



```
Statistics for Eout:
Mean: 0.0152
Median: 0.0151
Standard Deviation: 0.0010
Min: 0.0126
Max: 0.0181

Statistics for Non-zero Components:
Mean: 546.5
Median: 563.0
Standard Deviation: 27.2
Min: 484
Max: 576
```

[1]

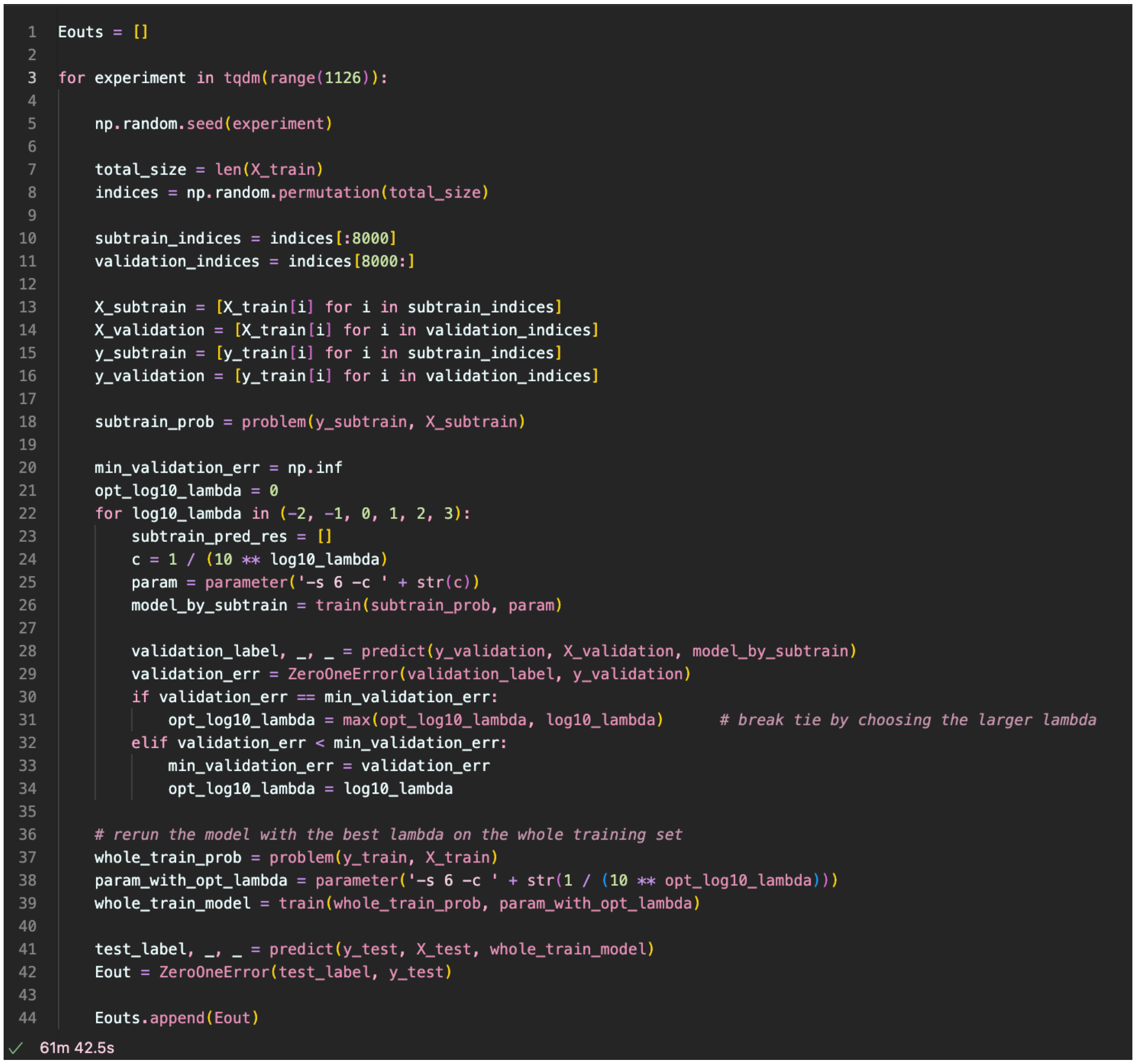
[1] Left: statistics of q11 result; Right: statistics of q10 result

In this problem, the difference from the previous is that we further divide the training set into a subtrain set (with 8000 exampls), and a validation set (3876 examples).
From the statistics, we can see that most of them are quite similar, I think part of the reason is that, even though 3876 examples are taken out from the original training set, so one may expect that as we train on a smaller subtrain set, the out of sample error should be bigger than in q10 while we have the whole training set used, we got an expected value slightly smaller because we retrain the whole training set after the optimal λ is chosed. We did not use g^- to predict the test set, but g instead.

Apart from this reason, I think this result may also due to the precision, if more digits are taken into consideration, maybe we can observe the difference.

Another cause may be the limited number of possible λ, this constraints the variability of our model, even though there are infinitely many possible models, most of them are alike, while only slightly different by the λ value chosen.

Code:



```python
Eouts = []

for experiment in tqdm(range(1126)):

    np.random.seed(experiment)

    total_size = len(X_train)
    indices = np.random.permutation(total_size)

    subtrain_indices = indices[:8000]
    validation_indices = indices[8000:]

    X_subtrain = [X_train[i] for i in subtrain_indices]
    X_validation = [X_train[i] for i in validation_indices]
    y_subtrain = [y_train[i] for i in subtrain_indices]
    y_validation = [y_train[i] for i in validation_indices]

    subtrain_prob = problem(y_subtrain, X_subtrain)

    min_validation_err = np.inf
    opt_log10_lambda = 0
    for log10_lambda in (-2, -1, 0, 1, 2, 3):
        subtrain_pred_res = []
        c = 1 / (10 ** log10_lambda)
        param = parameter('-s 6 -c ' + str(c))
        model_by_subtrain = train(subtrain_prob, param)

        validation_label, _, _ = predict(y_validation, X_validation, model_by_subtrain)
        validation_err = ZeroOneError(validation_label, y_validation)
        if validation_err == min_validation_err:
            opt_log10_lambda = max(opt_log10_lambda, log10_lambda)      # break tie by choosing the larger lambda
        elif validation_err < min_validation_err:
            min_validation_err = validation_err
            opt_log10_lambda = log10_lambda

    # rerun the model with the best lambda on the whole training set
    whole_train_prob = problem(y_train, X_train)
    param_with_opt_lambda = parameter('-s 6 -c ' + str(1 / (10 ** opt_log10_lambda)))
    whole_train_model = train(whole_train_prob, param_with_opt_lambda)

    test_label, _, _ = predict(y_test, X_test, whole_train_model)
    Eout = ZeroOneError(test_label, y_test)

    Eouts.append(Eout)
```

```
✓ 61m 42.5s
```

[2]

[2] Other part of codes that are similar to problem 10 (like reading in the data, split the data to get proper form of X and y, the ZeroOneError function…) are omitted in this report, can check for problem 10 for details.