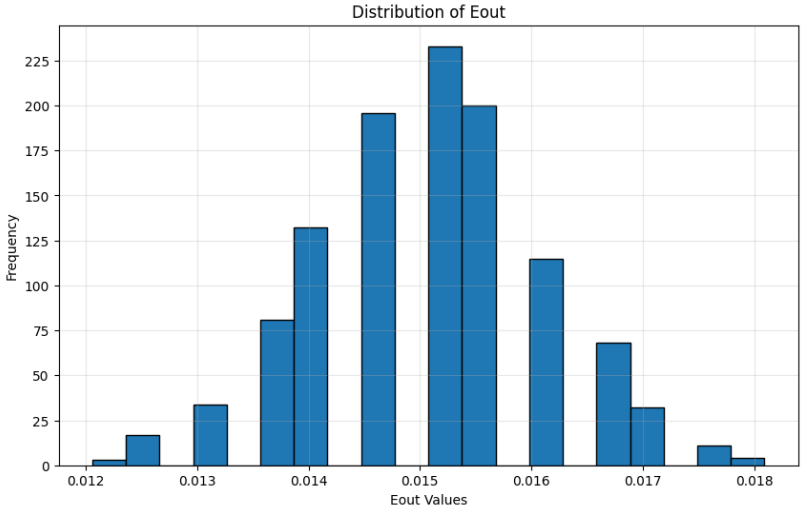
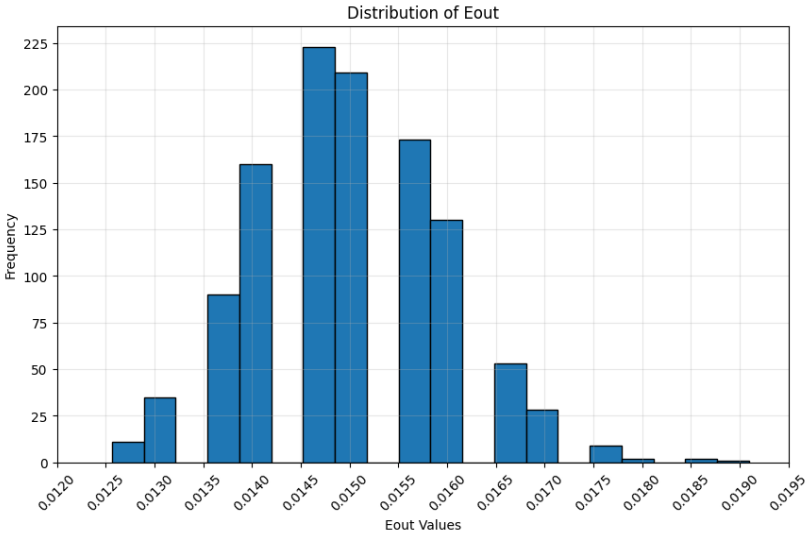
HTML homework 5: q12 report

The resulting picture is as follows:



(q12)

To find the difference, the plot below is the result of the previous problem:



(q11)

From the two plots we can see that the subtrain / validation split is more dispersed, and the variability of the 3-fold cross validation is smaller.

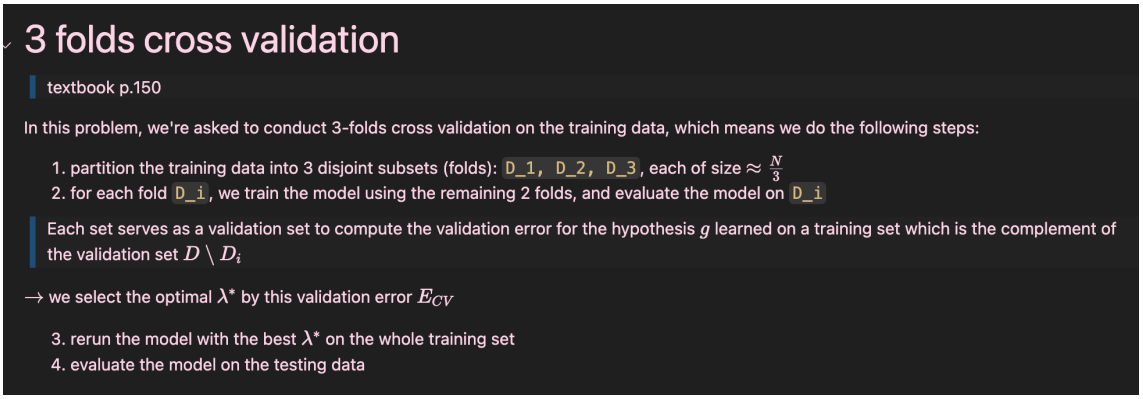
The reason why the subtrain / validation split is more dispersed may due to the fact that the validation set size is not that big compared to the 3-fold version. In the previous question, the validation set size is 3876, and in this question, using 3-fold splits the training set into 3 sets,

with each about 3959 examples, which is slightly higher. Also, $\mathrm{E_{CV}}$ is calculated by the mean error over using each of the 3 sets as the validation set, therefore these 2 reasons introduce stability in the 3-fold version.

On the other hand, the subtrain / validation split relies only on one split, if the validation set is not that representative, our selection of λ may be biased, this would magnify the effect of the unstableness in the previous question.

Code:

The markdown screenshot below describes the modification of this problem, other parts like reading in the data and the error function…, are the same as problem 10:



```python
def run_single_experiment(experiment):
    np.random.seed(experiment)

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

    fold_size = total_size // 3

    fold1_indices = indices[:fold_size]
    fold2_indices = indices[fold_size:2*fold_size]
    fold3_indices = indices[2*fold_size:]

    X_fold1 = [X_train[i] for i in fold1_indices]
    X_fold2 = [X_train[i] for i in fold2_indices]
    X_fold3 = [X_train[i] for i in fold3_indices]

    y_fold1 = [y_train[i] for i in fold1_indices]
    y_fold2 = [y_train[i] for i in fold2_indices]
    y_fold3 = [y_train[i] for i in fold3_indices]

    min_Ecv = np.inf
    opt_log10_lambda = 0
    for log10_lambda in (-2, -1, 0, 1, 2, 3):
        each_fold_as_valid_err = []
        c = 1 / (10 ** log10_lambda)

        for i in range(1,4):
            if i == 1:
                X_subtrain = X_fold2 + X_fold3
                y_subtrain = y_fold2 + y_fold3
                X_validation = X_fold1
                y_validation = y_fold1
            elif i == 2:
                X_subtrain = X_fold1 + X_fold3
                y_subtrain = y_fold1 + y_fold3
                X_validation = X_fold2
                y_validation = y_fold2
            else:
                X_subtrain = X_fold1 + X_fold2
                y_subtrain = y_fold1 + y_fold2
                X_validation = X_fold3
                y_validation = y_fold3

            subtrain_prob = problem(y_subtrain, X_subtrain)
            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)
            each_fold_as_valid_err.append(validation_err)

        Ecv_each_lambda = np.mean(each_fold_as_valid_err)
```

```
            if Ecv_each_lambda == min_Ecv:
                opt_log10_lambda = max(opt_log10_lambda, log10_lambda)  # break tie by choosing larger lambda
            elif Ecv_each_lambda < min_Ecv:
                min_Ecv = Ecv_each_lambda
                opt_log10_lambda = log10_lambda

        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)
        return ZeroOneError(test_label, y_test)

n_jobs = -1
Eouts = Parallel(n_jobs=n_jobs)(
    delayed(run_single_experiment)(experiment)
    for experiment in tqdm(range(1126))
)
```

✓ 166m 7.7s