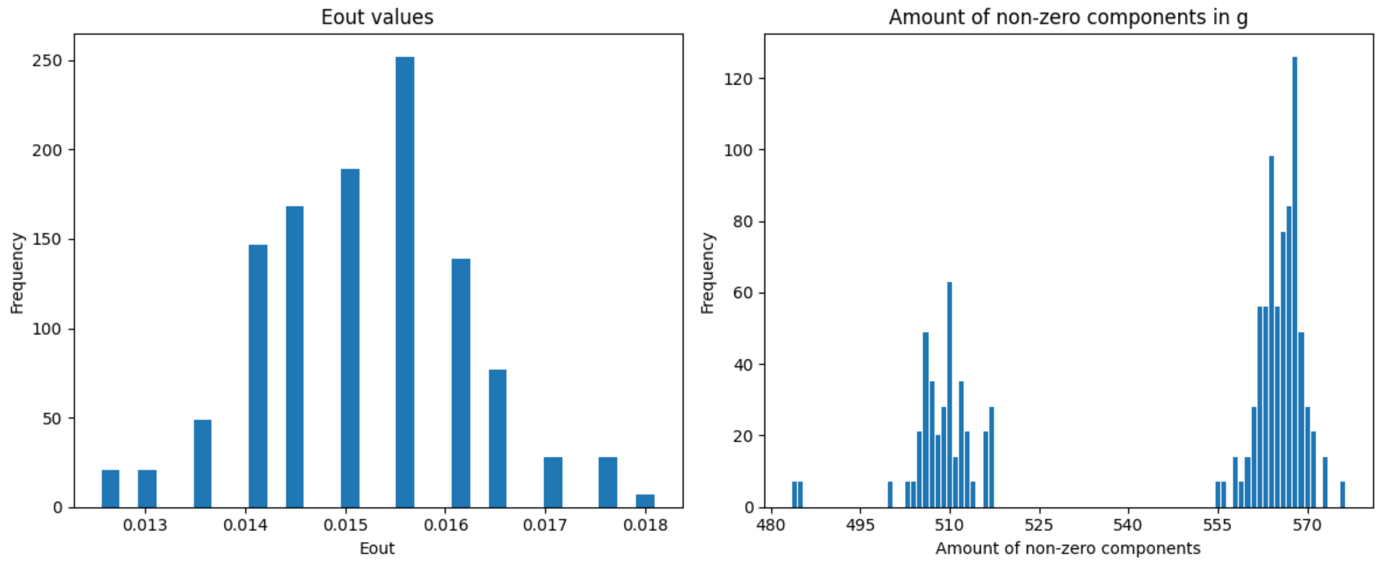
HTML homework 5: q10 report

The resulting plot is as follows:



Some additional information that I printed in order to do the comparison between this question and question 11:

```
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
```

Code[1]:

Preprocessing the dataset:

```python
from liblinear.liblinearutil import *
from itertools import combinations_with_replacement
import numpy as np
import scipy.sparse
import matplotlib.pyplot as plt
from tqdm import tqdm
from joblib import Parallel, delayed

```

[28] ✓ 0.4s

Dataset

- `mnist.scale` for training
- `mnist.scale.t` for testing

```python
train_data = 'mnist.scale'
test_data = 'mnist.scale.t'
```

[2]

Empty markdown cell, double-click or press enter to edit.

Read and save data

The training data is saved in the `X_train` list, with each element being a dictionary of features. example format: {153: 0.0117647, 154: 0.0705882, 155: 0.0705882, ...}

The corresponding label list is saved in `y_train`.

Similarily, the testing data is saved in `X_test` and the corresponding label list is saved in `y_test`.

```python
def read_linear_format(file_path):
    X, y = [], []
    with open(file_path, 'r') as f:
        for line in f:
            parts = line.strip().split()
            if parts[0] == '2' or parts[0] == '6':
                y.append(int(parts[0]))
                features = {}
                for item in parts[1:]:
                    index, value = item.split(":")
                    features[int(index)] = float(value)
                X.append(features)
    return X, y

X_train, y_train = read_linear_format(train_data)
X_test, y_test = read_linear_format(test_data)

```

Error function:

$0/1$ error

```python
def ZeroOneError(predictions, y):
    return sum(p_i != y_i for p_i, y_i in zip(predictions, y)) / len(predictions)
```

[6]

[1] Most of the screenshots here are from my .ipynb file, since I wrote the explanations here, however, I ran the code in a .py file (which has equivalent contents), and part of the screenshot is from this file.

Explanation of deriving the relationship between C, λ:

Regularized logistic regression

Original Formulation in README.md

As the explanation in the liblinear README file, for L1-regularized logistic regression `(-s 6)`, we solve

$$\min_{\vec{w}} \sum |w_j| + C \sum \log(1 + \exp(-y_i \vec{w}^T \vec{x}_i))$$

Components of this equation

In this equation, the L1 regularizer is defined as:

$$\Omega(\vec{w}) = \sum |w_j| = ||\vec{w}||_1$$

> info in Lec14 slide 18

- Note: using L1 regularization encourages sparsity, which means fewer parameters.

And the in-sample error for logistic regression is:

$$E_{in}(\vec{w}) = \frac{1}{N} \sum \log(1 + \exp(-y_i \vec{w}^T \vec{x}_i))$$

> in textbook p.91

Derivation of C

Since we knew that:

$$E_{aug}(\vec{w}) = E_{in}(\vec{w}) + \frac{\lambda}{N} \Omega(\vec{w})$$

We can derive that:

$$E_{aug}(\vec{w}) = E_{in}(\vec{w}) + \frac{\lambda}{N} \sum |w_j|$$

For minimization:

$$\begin{aligned} \arg\min_{\vec{w}} E_{aug}(\vec{w}) &= \arg\min_{\vec{w}} (E_{in}(\vec{w}) + \frac{\lambda}{N} \sum |w_j|) \\ &= \arg\min_{\vec{w}} \frac{N}{\lambda} (\frac{1}{N} \sum \log(1 + \exp(-y_i \vec{w}^T \vec{x}_i)) + \frac{\lambda}{N} \sum |w_j|) \\ &= \arg\min_{\vec{w}} \frac{1}{\lambda} \sum \log(1 + \exp(-y_i \vec{w}^T \vec{x}_i)) + \sum |w_j| \end{aligned}$$

Compare to the original equation marked with ($\star$), we can see that:

$$C = \frac{1}{\lambda}$$

Some additional information:

Select the best λ^*

Selecting optimal λ^* using the following equation:

$$\lambda^* = \arg\min_{\log_{10}(\lambda) \in \{-2,-1,0,1,2,3\}} E_{in}(\vec{w}_\lambda)$$

get amount of non-zeros

To get the amount of non-zero components in the model, we can use the function `.get_decfun()`:

```
[W, b] = model_.get_decfun()
```

Main code:

```python
def run_single_experiment(experiment):
    np.random.seed(experiment)
    min_Ein = np.inf
    opt_log10_lambda = 0

    for log10_lambda in (-2, -1, 0, 1, 2, 3):
        c = 1 / (10 ** log10_lambda)
        prob = problem(y_train, X_train)
        param = parameter('-s 6 -c ' + str(c))
        model = train(prob, param)

        train_label, _, _ = predict(y_train, X_train, model)
        Ein = ZeroOneError(train_label, y_train)
        if Ein == min_Ein:
            opt_log10_lambda = max(opt_log10_lambda, log10_lambda)
            if opt_log10_lambda == log10_lambda:
                opt_model = model
        elif Ein < min_Ein:
            min_Ein = Ein
            opt_log10_lambda = log10_lambda
            opt_model = model

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

    W = np.array(opt_model.get_decfun()[0])
    non_zero_count = np.count_nonzero(W)

    return Eout, non_zero_count

experiment_amount = 10
results = Parallel(n_jobs=-1)(                                              # use all cores by setting n_jobs to -1
    delayed(run_single_experiment)(i) for i in tqdm(range(experiment_amount))
)

Eouts, non_zero_count_list = zip(*results)
Eouts = list(Eouts)
non_zero_count_list = list(non_zero_count_list)
```

Plotting:

```python
# aim: plotting
# subaim: First subplot (Eout values)
ax1.hist(Eouts, bins=30)
ax1.set_title('Eout values')
ax1.set_xlabel('Eout')
ax1.set_ylabel('Frequency')

# subaim: Second subplot (non-zero components)
min_val = int(min(non_zero_count_list))
max_val = int(max(non_zero_count_list))
integer_bins = np.arange(min_val, max_val + 2) - 0.5  # +2 to include max_val, -0.5 for bin edges

ax2.hist(non_zero_count_list,
         bins=integer_bins,
         align='mid',
         rwidth=0.8)

ax2.xaxis.set_major_locator(plt.MaxNLocator(integer=True))
ax2.set_title('Amount of non-zero components in g')
ax2.set_xlabel('Amount of non-zero components')
ax2.set_ylabel('Frequency')

plt.tight_layout()
plt.show()
```