Yibo Liu (yl6769)

1. Split dataset

Explain why measuring the performance of your final classifier would be problematic had you not created this validation set.

When evaluating the performance of a trained model, it is supposed to use a validation set, in which the data were not used to update the parameters in training.

- If you use the data from training set to evaluate the model, the validation error would be lower than what it should be, because the data is already used to update the weights in training. When evaluating the model on the test set, the parameters on the last iteration is the one to use (because it gives the lowest error), which will result in overfitting. So you cannot pick the best configuration by the evaluation on traning set.
- If you use the test set as the validation set, it is also problemetic. Usually we choose the best configuration according to the validation error. If we tune the parameters according to the test set, the evaluation on test set would be biased, namely better than its actual performance.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import random
import warnings
warnings.filterwarnings('ignore')
```

```
def tokenize(line):
    return line.replace('\n', '').split(' ')
```

```
# split dataset
f_train = 'spam_train.txt'
f_test = 'spam_test.txt'

dataset = []
for line in open(f_train, 'r', encoding='utf-8').readlines():
    label, email = tuple(line.split(' ', 1))
    dataset.append((label, tokenize(email)))
# random.shuffle(dataset)
train_set = dataset[0:4000]
val_set = dataset[4000:5000]

test_set = []
for line in open(f_test, 'r', encoding='utf-8').readlines():
    label, email = tuple(line.split(' ', 1))
    test_set.append((label, tokenize(email)))

print('training set size:', len(train_set), ' validation set size:', len(val_set), ' test se
```

```
training set size: 4000  validation set size: 1000  test set size: 1000
```

2. Transform all data into feature vectors

```
# build vocab
def build_vocab(dataset, min_cnt=30):
    vocab_cnt = {}
    for tgt, src in dataset:
        for word in src:
            if not word in vocab_cnt.keys():
                vocab_cnt[word] = 1
            else:
                vocab_cnt[word] += 1

    email_cnt = dict.fromkeys(vocab_cnt.keys(), 0)
    for tgt, src in dataset:
        cnted = dict.fromkeys(vocab_cnt.keys(), False)
        for word in src:
            if not cnted[word]:
                email_cnt[word] += 1
                cnted[word] = True

    vocab = []
    for word in email_cnt:
        if email_cnt[word] >= min_cnt:
            vocab.append(word)
    # sorted_vocab = sorted(vocab.items(), key = lambda kv:kv[1], reverse = True)
    return vocab, vocab_cnt, email_cnt

vocab, vocab_cnt, email_cnt = build_vocab(train_set)
print('vocab size:', len(vocab), ' all vocab size:', len(vocab_cnt), len(email_cnt))
```

```
vocab size: 2376  all vocab size: 55232 55232
```

```
# embed dataset
def emb(dataset, vocab):
    embedded = []
    for tgt, src in dataset:
        feat = [0 for i in range(len(vocab))]
        i=0
        for word in vocab:
            if word in src:
                feat[i] = 1
            else:
                feat[i] = 0
            i += 1
        tgt = 1 if tgt == '1' else -1
        embedded.append((feat, tgt))
    return embedded
```

```
emb_train = emb(train_set, vocab)
emb_val = emb(val_set, vocab)
```

3. Perceptron Train/Test Functions

```
def perceptron_train(data, max_iter=-1, p=False, test=False):
    lr = 1
    its = 0
    total_err = 0
    err = 1
    w = np.zeros(len(vocab))
#     while its != max_iter and err > 0:
    while (max_iter == -1 and err > 0) or (max_iter > 0 and its != max_iter):
        its += 1
        err = 0
        n = 0
        for src, tgt in data:
            x = np.array(src)
            y = np.dot(w, x)
            y_ = 1 if y>=0 else -1
            if np.all(x==0):
                pass
#                 print('bad sample:', n)
            elif tgt != y_:
                err += 1
                total_err += 1
                w = np.add(w, lr * tgt * x)
            n += 1
        acc = 1 - err / len(data)
        if not test:
            val_err = perceptron_test(w, emb_val)
            if p: print('iter:', its, ' acc:', acc, ' val_acc:', 1-val_err)
        else:
            if p: print('iter:', its, ' acc:', acc)
    return w, total_err, its

def perceptron_test(w, data):
    err = 0
    n=0
    for src, tgt in data:
        x = np.array(src)
        y = np.dot(w, x)
        y_ = 1 if y>=0 else -1
        if tgt != y_:
            err += 1
        n += 1
    return err/len(data)
```

4. Run train and test

```
w, total_err, its = perceptron_train(emb_train, p=True)
print('weight:\n', np.around(w, decimals=2), '\ntotal error:', total_err, '\niterations:',it
```

```
iter: 1  acc: 0.94075  val_acc: 0.972
iter: 2  acc: 0.9795  val_acc: 0.971
iter: 3  acc: 0.9905  val_acc: 0.953
iter: 4  acc: 0.991  val_acc: 0.982
iter: 5  acc: 0.99625  val_acc: 0.973
iter: 6  acc: 0.99875  val_acc: 0.976
iter: 7  acc: 0.99875  val_acc: 0.98
iter: 8  acc: 0.99675  val_acc: 0.975
iter: 9  acc: 0.99675  val_acc: 0.98
iter: 10  acc: 0.99925  val_acc: 0.98
iter: 11  acc: 1.0  val_acc: 0.98
weight:
 [-6. -6. -7. ... -1.  2.  5.]
total error: 447
iterations: 11
```

```
val_err = perceptron_test(w, emb_val)
print('validation error for perceptron:', val_err)
```

```
validation error for perceptron: 0.02
```

5. Words with greatest weights

```
def word_contrib(w):
    pos_words = []
    neg_words = []
    for i in np.argsort(w)[0:15]:
        pos_words.append(vocab[i])
    for i in np.argsort(-w)[0:15]:
        neg_words.append(vocab[i])
    return pos_words, neg_words
```

```
pos_words, neg_words = word_contrib(w)
print('top15 words with most positive weights:\n', pos_words, '\ntop15 words with most negat
```

```
top15 words with most positive weights:
 ['wrote', 'prefer', 'but', 'and', 'i', 'reserv', 'on', 'technolog', 'still', 'upgrad', 'url
top15 words with most negative weights:
 ['sight', 'our', 'remov', 'yourself', 'pleas', 'these', 'guarante', 'nbsp', 'market', 'clic
```

6. Average Perceptron Train

```
def avg_perceptron_train(data, max_iter=-1, p=False, test=False):
    lr = 1
    its = 0
    total_err = 0
    err = 1
    w = np.zeros(len(vocab))
    avg_w = np.zeros(len(vocab))
    n = 0
    while (max_iter == -1 and err > 0) or (max_iter > 0 and its != max_iter):
        its += 1
        err = 0
        for src, tgt in data:
            x = np.array(src)
            y = np.dot(w, x)
            y_ = 1 if y>=0 else -1
            if np.all(x==0):
                pass
#                 print('bad sample:', n)
            elif tgt != y_:
                err += 1
                total_err += 1
                w = np.add(w, lr * tgt * x)
            avg_w = (avg_w * n + w)/(n+1)
            n += 1
        acc = 1 - err / len(data)
        if not test:
            val_err = perceptron_test(avg_w, emb_val)
            if p: print('iter:', its, ' acc:', acc, ' val_acc:', 1-val_err)
        else:
            if p: print('iter:', its, ' acc:', acc)
    return avg_w, total_err, its

avg_w, total_err, its = avg_perceptron_train(emb_train, p=True)
print('avgerage weight:\n', np.around(avg_w, decimals=2), '\ntotal error:', total_err, '\nit
```

```
iter: 1  acc: 0.94075  val_acc: 0.98
iter: 2  acc: 0.9795  val_acc: 0.982
iter: 3  acc: 0.9905  val_acc: 0.983
iter: 4  acc: 0.991  val_acc: 0.983
iter: 5  acc: 0.99625  val_acc: 0.984
iter: 6  acc: 0.99875  val_acc: 0.983
iter: 7  acc: 0.99875  val_acc: 0.982
iter: 8  acc: 0.99675  val_acc: 0.982
iter: 9  acc: 0.99675  val_acc: 0.983
iter: 10  acc: 0.99925  val_acc: 0.984
```

```
iter: 11  acc: 1.0  val_acc: 0.982
avgerage weight:
 [-5.16 -5.3  -6.12 ... -1.11  1.68  3.8 ]
total error: 447
iterations: 11
```

```
avg_val_err = perceptron_test(avg_w, emb_val)
print('validation error for avg_perceptron:', avg_val_err)
```

```
validation error for avg_perceptron: 0.018
```

7. Dataset size N affects validation error

```
# change the size of dataset
Ns = [100,200,400,800,2000,4000]

errs = []
iters = []
for N in Ns:
    w, total_err, its = perceptron_train(emb_train[0:N], p=False)
    iters.append(its)
    val_err = perceptron_test(w, emb_val)
    errs.append(val_err)

avg_errs = []
avg_iters = []
for N in Ns:
    w, total_err, its = avg_perceptron_train(emb_train[0:N], p=False)
    avg_iters.append(its)
    val_err = perceptron_test(w, emb_val)
    avg_errs.append(val_err)

%matplotlib inline
# plt.scatter(Ns, errs)
plt.scatter(Ns, errs, marker='o', label="perceptron")
plt.scatter(Ns, avg_errs, marker='^', label="avg_perceptron")
plt.xlabel('training set size N')
plt.ylabel('validation error')
plt.legend(loc='best')
plt.show()
```

As we can see in the figures above, validation error decreases as the dataset size N increases.

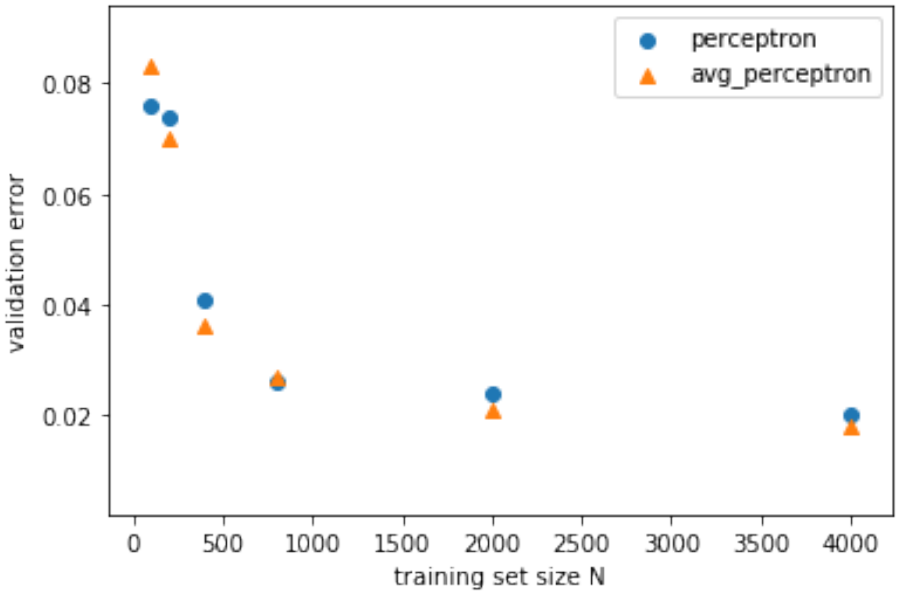
perceptron
avg_perceptron
validation error
0.08
0.06
0.04
0.02
0
500
1000
1500
2000
2500
3000
3500
4000
training set size N

Figure 1: png

8. Dataset size N affects iterations to converge

```
# change the size of dataset
Ns = [100,200,400,800,2000,4000]

errs = []
iters = []
for N in Ns:
    w, total_err, its = perceptron_train(emb_train[0:N], p=False)
    iters.append(its)
    val_err = perceptron_test(w, emb_val)
    errs.append(val_err)

avg_errs = []
avg_iters = []
for N in Ns:
    w, total_err, its = avg_perceptron_train(emb_train[0:N], p=False)
    avg_iters.append(its)
    val_err = perceptron_test(w, emb_val)
    avg_errs.append(val_err)

%matplotlib inline
plt.scatter(Ns, iters, marker='o', label="perceptron")
plt.scatter(Ns, avg_iters, marker='^', label="avg_perceptron")
plt.xlabel('training set size N')
plt.ylabel('iterations to converge')
plt.legend(loc='best')
plt.show()
```

As we can see in the figures above, the number of iterations to converge increases in general as the dataset size N increases.

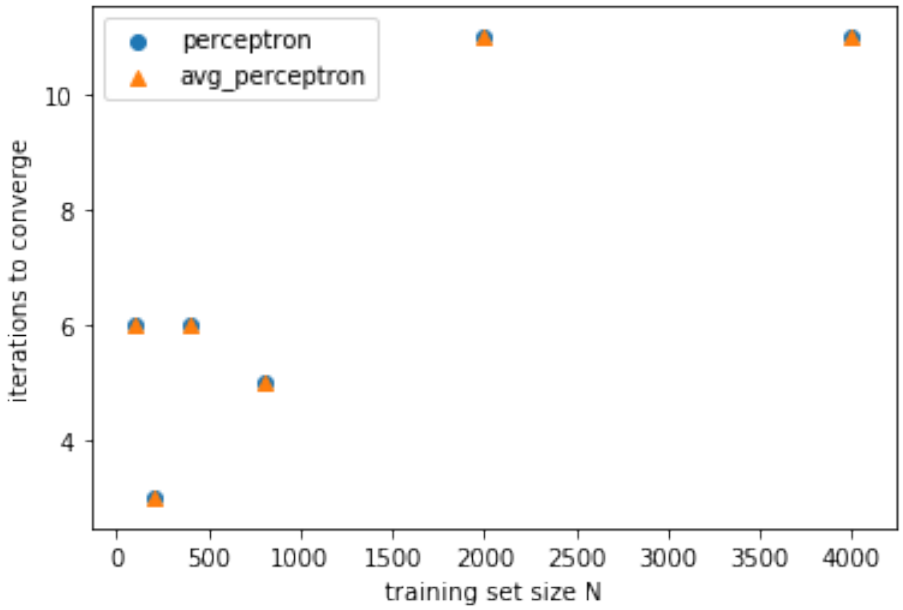
perceptron
avg_perceptron
iterations to converge
10
8
6
4
0
500
1000
1500
2000
2500
3000
3500
4000
training set size N

Figure 1: png

9. Explore the best configuration

Specifically, we implement 'perceptron_train' and 'avg_perceptron_train' respectivelly with different maximun iterations from 1 to 11. (11 is the maximum iterations to converge according to the previous experiments.)

```
# try different iterations and different algorithms(perceptron/average perceptron)
val_errs = []
for i in range(11):
    w, train_err, its = perceptron_train(emb_train, max_iter=i+1, p=False)
    val_err = perceptron_test(w, emb_val)
    print('max iter:', i+1, ' validation error:', val_err)
    val_errs.append(val_err)
```

```
max iter: 1  validation error: 0.028
max iter: 2  validation error: 0.029
max iter: 3  validation error: 0.047
max iter: 4  validation error: 0.018
max iter: 5  validation error: 0.027
max iter: 6  validation error: 0.024
max iter: 7  validation error: 0.02
max iter: 8  validation error: 0.025
max iter: 9  validation error: 0.02
max iter: 10  validation error: 0.02
max iter: 11  validation error: 0.02
```

```
avg_val_errs = []
for i in range(11):
    w, train_err, its = avg_perceptron_train(emb_train, max_iter=i+1, p=False)
    val_err = perceptron_test(w, emb_val)
    print('max iter:', i+1, ' validation error:', val_err)
    avg_val_errs.append(val_err)
```

```
max iter: 1  validation error: 0.02
max iter: 2  validation error: 0.018
max iter: 3  validation error: 0.017
max iter: 4  validation error: 0.017
max iter: 5  validation error: 0.016
max iter: 6  validation error: 0.017
max iter: 7  validation error: 0.018
max iter: 8  validation error: 0.018
max iter: 9  validation error: 0.017
max iter: 10  validation error: 0.016
max iter: 11  validation error: 0.018
```

```
plt.scatter([i+1 for i in range(11)], val_errs, marker='o', label="perceptron")
plt.scatter([i+1 for i in range(11)], avg_val_errs, marker='^', label="avg_perceptron")
plt.xlabel('iterations')
```

```
plt.ylabel('validation error')
plt.legend(loc='best')
plt.show()
```

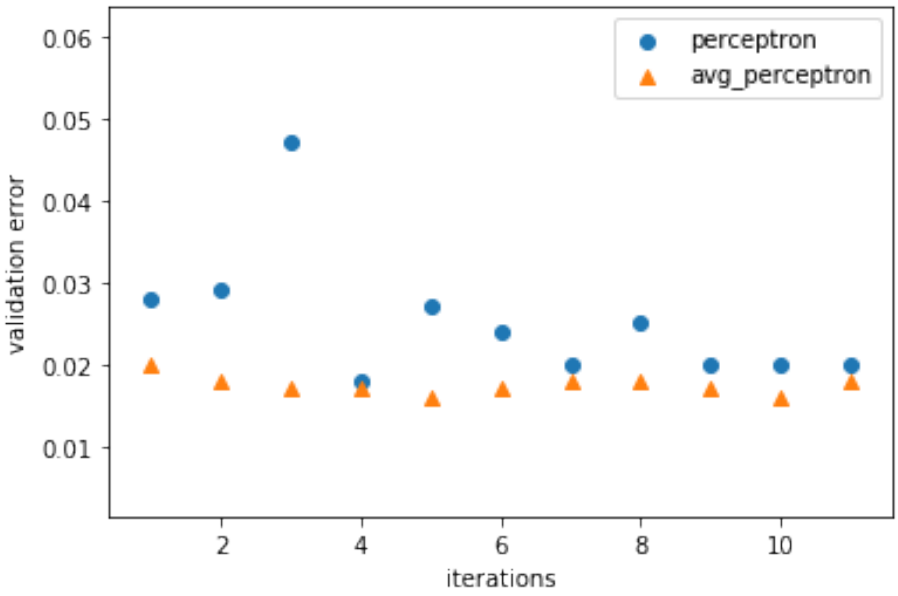

Figure 1: png

The figure shows that average perceptron algorithm performes better than perceptron algorithm.

```
# best config
it = np.argsort(val_errs)[0]
print('best config for perceptron - iteration:',it+1,'validation error:', val_errs[it])
it = np.argsort(avg_val_errs)[0]
print('best config for avg_perceptron - iteration:',it+1,'validation error:', avg_val_errs[i
```

```
best config for perceptron - iteration: 4 validation error: 0.018
best config for avg_perceptron - iteration: 5 validation error: 0.016
```

Therefore, the best configuration is using 'avg_perceptron' with maximum iteration 5.

11. Test error

```
# build new vocab and embed dataset
new_train_set = train_set+val_set
vocab, vocab_cnt, email_cnt = build_vocab(new_train_set)
emb_new_train = emb(new_train_set, vocab)
emb_new_test = emb(test_set, vocab)
```

```
print(len(vocab))
```

```
2769
```

```
# train on the full training set
w, train_err, its = avg_perceptron_train(emb_new_train, max_iter=5, p=True,test=True)
test_err = perceptron_test(w, emb_new_test)
print('test error:', test_err)
```

```
iter: 1  acc: 0.9472
iter: 2  acc: 0.9828
iter: 3  acc: 0.9918
iter: 4  acc: 0.994
iter: 5  acc: 0.9954
test error: 0.018
```