Machine Learning Assignment 1

Liang Shuailong 1000829

Task: Documentation classification

1. It is a multiclass classification problem. The input X is a matrix, each row of which is a feature vector corresponding to each document. The output is a column vector, and each element is the label representing the class(category) of the document. For binary classification case, the output is just 1 and -1 for the two classes.

   In class we talked about the Bag-of-Word approach to represent a document by the term frequencies. So for now we can use this approach to extract the feature vector for each document. The algorithm is described in Table 1:

   Table 1 Algorithm to extract BoW feature

```
def feature_extractor():
    features = []
    vocabulary = set()
    for each document d in training set:
            add tokens in d to vocabulary
    for each document d:
            for each token t in vocabulary:
                    count[t] = frequencies of token in d
    append feature to features
    return features
```

2. The terminate condition is that in one loop of scanning the data points, the number of mislabeled data reaches 0. The result is shown in Table 2.

   Table 2 Result of my terminating condition

	Training accuracy	Test accuracy	Iterations
Perceptron	100%	**93.7%**	80
Averaged perceptron	97.76%	91.51%	80

   If we terminate the program after 100 loops, the result is shown in Table 3.

   Table 3 Result of setting fixed interations

	Training accuracy	Test accuracy	Iterations
Perceptron	100%	**93.7%**	100
Averaged perceptron	98.5%	91.76%	100

   For Perceptron, since after 80 iterations, any data points in training set can not update theta anymore. So the theta will not be updated anymore, and causing no difference to the result. For averaged version, theta is related to the number of loops, so the test accuracy improves a little bit.

From the table above we can see that averaged version of perceptron does not improve the result. It decreases the result a little actually. Theoretically, averaged version of perceptron should generalize better, however in here it is not the case. The reason may be that the averaged perceptron over fit on the training set.

3. The result of stochastic gradient descent using hinge loss is shown in Table 4.

Table 4 Result under different learning rate

ita(learning rate)	Training accuracy	Test accuracy	Iterations	Loss(hinge loss)
0.0001	99.75%	**98.50%**	19k	2.06
0.001	100.00%	98.25%	14k	4.67
0.01	100.00%	92.32%	13k	34.65
0.1	100.00%	94.94%	9k	341.98
1	100.00%	95.19%	7k	2610.85
10	99.75%	95.51%	9k	33643.02
100	100.00%	94.88%	6k	259895.76
1000	99.75%	95.82%	6k	2987251.87
10000	100.00%	94.69%	11k	33591745.64

From the table we can see that learning rate will influence how soon the algorithm converges as well as the accuracy. In general, the higher the learning rate, the sooner the algorithm will converge, although the accuracy will decrease a little bit. But if the learning rate is too high, the algorithm will still take a long time before convergence. It is due to that the gradient descends so much toward a direction that it may not actually descend.

We can also see that stochastic gradient descent result (Table 4) is better than perceptron result (Table 2) in general.

For this algorithm, the terminating condition is that the loss does not decrease anymore. According to my implementation, since computing loss is a little expensive, I compute loss every 1000 data points. So this is a relative strong condition to indicate the convergence of the algorithm. If we just limit the number of iterations to terminate, the algorithm may return a different result, but as long as the number of iteration is large enough, the result will not be significantly different.

4. We can use one-vs-all approach to transform the multiclass classification problem to binary classification problem. Specifically, in this problem, we train four classifiers separately, and use the classifiers to compute the predicted value. Of the four predicted values we choose the maximum predicted value as the class label.

   The algorithm is described as the pseudocode below (Table 5).

Table 5 Algorithm to do multiclassification

```
def multiclassify(X):
   train (theta, theta0) for each class vs other classes
   for each feature in features:
       for each (theta, theta0):
           compute predicted value by dot product feature vector with
theta
       set the label corresponding to theta which gives the highest
predicted value
```

The result is shown in Table 6.

Table 6 Result of multiclassification

	Training accuracy	Test accuracy
Perceptron	98.25%	77.65%
Averaged perceptron	91.65%	77.37%
Stochastic gradient descent	98.63%	**92.67%**

From the table we can see that stochastic gradient descent has the best result on both training and test set.

5. The new learning algorithm is shown in the pseudocode below (Table 7).

Table 7 Algorithm of gradient descent with regularization term

```
initialize theta, theta0 to 0.
while not converge {
   random pick a data point x, y
   calculate y’ = x · theta + theta0
   if y*y’ < 0:
          theta = (1-2*lambda*ita)*theta + y*x*ita
          theta0 = (1-2lambda*ita)*theta0 + y*ita
   }
```

The result is shown in table 8.

Table 8 Result under different lambda

Lambda(regularization)	Training accuracy	Test accuracy	Iterations	Loss
0.001	100.00%	97.00%	17k	4.55
0.01	100.00%	**97.82%**	19k	4.25
0.1	100.00%	96.88%	15k	4.46
1	98.75%	89.58%	99k(early stop)	2.86
10	77.56%	74.28%	99k(early stop)	1.11

From the table we can see that as we increase lambda from 0.001 to 0.01, the generalization may improve a little. However, if lambda is too large, it will take too long to converge, and even fail to converge, which causes a low result. The reason is that if we set lambda to a too large value, the model will underfit.

6. Instead of using BoW features of a document, we can use **tf-idf** feature to better represent a document since **tf-idf** not only considers term frequencies but also considers the inverse document frequencies. Inverse document frequency deemphasizes words which appear too often in most documents, such as stop words, particles etc. Thus tf-idf is a better feature. The performance of using tf-idf feature is shown in Table 9.

   Table 9 tf-idf feature result when perceptron is used

Feature + Algorithm	Training accuracy	Test accuracy	Iterations
tf-idf + Perceptron	100.00%	97.44%	25
tf-idf + averaged perceptron	100.00%	**98.38%**	25

   From the table we can see that compared with the result of BoW approach shown in Table 2(93.7%), using tf-idf feature can improve the result by several percentages.

7. For this question, I would like to explore the influence of unbalanced data to multiclassification problem. Since we use one-vs-all approach for this problem, at training stage, the positive samples are only 1/3 as many as other samples. To investigate whether this will influence the result, we do the experiment under settings shown below.

   Feature: tf-idf
   Algorithm: perceptron, averaged perceptron, stochastic gradient descent, stochastic gradient descent with regularization

   The result is shown in table 10.

   Table 10 Result of balanced and unbalanced data

	Balanced data		Unbalanced data	
	Train accuracy	Test accuracy	Train accuracy	Test accuracy
Perceptron	98.75%	89.48%	100%	**88.39%**
Averaged perceptron	97.88%	89.98%	99.25%	82.40%
SGD	99.25%	**93.16%**	100%	84.74%
SGD with regularization	99.63%	91.17%	100%	83.49%

   From the table we can see that for each algorithm using balanced data is almost always better than using unbalanced data. Although using unbalanced data makes training accuracy higher, it does not improve test accuracy, which means that the model trained by unbalanced data does not generalize very well.