Report of Assignment 1

Zou Yanyan

February 29, 2016

1

The input is the text feature matrix which contains term frequency of each word appearing in documents. In detail, for each entry of matrix $x(i, j)$, i represents i_{th} document, and j represents j_{th} word where j_{th} is the integral ID of the word, and the value of $x(i, j)$ is the term frequency. Each row represents one document, and each column represents one feature.

The output is$\{+1, -1\}$, referring to two different classifications.

To extract the feature of those documents, we can use the toolbox $'scikit'$, which can automatically convert text words to feature matrix. In this case, we will name each word an integral ID, and count its tf score (term frequency) in each document. This method is also called Bag of Words. In addition, we convert all words into lowercases, ignore punctuation marks and low frequency words.

2

Implement perceptron algorithm to perform binary classification based on the data from two sub-folders: atheism and sports and define an evaluation criteria, accuracy, which is the ratio of the number of correctly predicted points and total points. When the number of misclassified point is equal to zero, algorithm stops. In this case, we can get the accuracies of training set and test set are 100% and 98.9375% respectively, and the running time is around 0.2021 seconds. Try to change the condition in which the algorithm stops, for instance, when the number of mis-classified point is less than 10 or 20, algorithm stops, and we get the same accuracy in both sets. Moreover, we change the stop condition to 50, the accuracies of two sets decline to 95.25% and 94.44%. Hence, we can infer that perceptron algorithm is able to allow some mis-classified points in train set and still performs well in test set. On the other hand, if the allowed number of mis-classified points is too large, it may penalize the accuracy.

Implement the averaged perceptron algorithm, the accuracies of training and test sets are 87.75% and 83.5625% and the running time is 0.2788 seconds. Compared to the perceptron algorithm, this is less accurate and efficient in this case. However, in practice, the averaged perceptron is much efficient than the standard version of perceptron.

3

Implement the stochastic gradient descent algorithm that minimizes the empirical risk involving the hinge loss to tackle the same binary classification problem.

There are two ways to set learning rate. One is to set learning rate as a constant number and the other one is to set it as a variable number, which is one divided by the iteration times $\frac{1}{iteration}$, changed in each iteration time. In this case, we do both and the performances is shown in Figure 1.

As we can see from the results, the most efficient method is to use variable learning rate, which also holds the most accurate predications. Generally, the learning rate is the parameter to control how big a step the algorithm will take downhill with gradient descent. If learning rate is too small, it may take a long time to converge since the gradient descent declines slowly. If learning rate is too large, the gradient descent can overshoot the minimum, which may fail to converge or even diverge. Sometimes, the gradient descent can converge to a local minimum even with some fixed learning rate. In this case, using different learning rate holds this information well.

There are two ways to stop the algorithm. One is to compare the previous and the current loss, if the difference between them is smaller than some threshold, let say 0.0001, the algorithm stops. The other way is to set the maximum iteration times, and record the minimum loss and its responding parameters. The first method may cause false negative converge, since the stochastic gradient descent is used and the result is random. In this case, the accuracy of test set may be lower. The second method needs to test many times to estimate when the algorithm will tend to converge after some iterations. Then we use the parameter with the minimum loss to predict the training and test set. This approach may take a long time but perform well.

Compared to the performances of the perceptron, its accuracy of test set is higher but the running time is longer. On the other hand, since the stochastic gradient descent is implemented in this algorithm, the results may vary.

Hinge loss with stochastic gradient descent	Training Time	Accuracy for training set	Accuracy for test set	learn_rate
	0.527016163	1	0.99875	1/(1+iteration_time)
	89.7171061	1	0.9925	0.001
	91.05606198	1	0.99875	0.01
	0.519975185	1	0.99875	0.1
	0.505172014	1	0.999375	1
	0.487112999	1	0.996875	10
	92.13634014	0.985	0.96875	20
	92.77587795	0.9975	0.993125	50
	93.43657589	0.9975	0.993125	100

Figure 1: Performances with different lamda

4

The main problem here is how to cast a multi-class classification problem into binary

classification problem. In this case, we can regard one and the rest types as two general categories in each time and implement binary classification algorithm, such as averaged perceptron, to classify this data sets. For instance, in the first time, we can regard the $'atheism'$ as '+1' and the other three types as $'-1'$. Then in the second time, regard the $'science'$ as $'+1'$ and the others as $'-1'$, etc. Repeat binary classification algorithm until classifying all categories. In general, the repeating times is exactly the number of categories. In this case, it is four. We call averaged perceptron algorithm to solve this problem. The performance is shown in figure 2.

	Training Time	Accuracy for training set	Accuracy for test set
multi-class			
atheism	0.644557953	1	0.996
politics	0.672071934	0.99875	0.9846875
science	0.499661922	0.99875	0.9159375
sports	0.494127989	0.99	0.9878125

Figure 2: Performances of multi-class classification

5

Still implement the stochastic gradient descent algorithm to minimize this regurarized loss function. The update equation is modified to:

$\theta^{(k+1)} = \theta^{(k)} + \alpha x^{(i)} y^{(i)} + \lambda\theta$,

$\theta_0^{(k+1)} = \theta_0^{(k)} + \alpha y^{(i)}$, where α is learning rate, set as $\frac{1}{iteration}$.

The performances on both training and test sets are shown in Figure 3. As we can see, the accuracies of both training and test set are good with small regularization parameter, but it is terrible when using large ones. The regularization term is to avoid to overfit and try to keep theta parameters small. If λ is too large, the fit function could be like a line which is underfitting, since the value of θ tends to zero. But if λ is too small, it does not work so much.

Regularized hinge loss	Training Time	Accuracy for training set	Accuracy for test set	lambda
	46.03077102	1	0.990625	0.0001
	48.41287613	1	0.99625	0.001
	47.093817	0.99	0.978125	0.01
	49.62924886	1	0.9975	0.1
	75.71235204	0.5	0.5	1
	75.94687796	0.5	0.5	10
	80.04547596	0.5	0.5	100

Figure 3: Performances with different *lambda*

6

In addition to term frequency, we can apply term-frequency-inverse-document-frequency (shortly, tf-idf) score to represent the value of each word feature and add some stop words.

The tf-idf penalizes words which appear in most documents and stop words can also ignore some high frequency words. Using this approach, we rerun perceptron algorithm and averaged perceptron algorithm. The accuracies of training and test sets are 100% and 98.8125% respectively. For averaged perceptron algorithm, the accuracies are 100% and 96.0625% respectively.

N-grams is also a good way to get text feature vector. In this case, we only consider 2 to 4-gram to get feature vector. The accuracies of training and test sets are 100% and 99.75% respectively. For averaged perceptron algorithm, the accuracies are 98.5% and 97.625% respectively.

Compared to the term-frequency way, the performances on the test set basically become better with these two alternative representations. The reason may be that the new methods reward principle features and penalize or even ignore noisy features, for example words appearing in all or most documents, like 'a', 'the'. In this way, it can avoid overfit in training set and perform well in test set.

7

Recall what we have discussed in class, the logistic regression is a type of algorithm to solve classification algorithm. In this case, we can introduce *sigmoid function* to replace hinge loss function, also using stochastic gradient descent. The loss function is defined as:

$Loss = \frac{1}{N}\sum_{i=1}^{N} \log \frac{1}{1+e^{-y^{(i)}(x^{(i)}\theta+\theta_0)}}$,

and the gradient descent update steps is:

$\theta^{(k+1)} = \theta^{(k)} + \alpha \frac{x^{(i)}y^{(i)}}{1+e^{-y^{(i)}(x^{(i)}\theta^{(k)}+\theta_0)}}$,

$\theta_0^{(k+1)} = \theta_0^{(k)} + \alpha y^{(i)}$, where α is learning rate.

After running the algorithm, we get the accuracies of training and test sets, 100% and 98% respectively. The running time is around 11.4022 seconds. In this case, comparing to implementing stochastic gradient descent to minimize hinge loss function, this method is quickly to converge.