ASSIGNMENT 2

Pattern Recognition and Machine Learning

(模式识别与机器学习)

COMP130137.01

Name: ANON

Fudan-ID: XXX

3 juni 2019

1 Overview

1.1 Introduction

In this assignment we are going to explore several well-know linear classification methods such as perceptron and logistic regression, and a realistic dataset will be provided for us to evaluate these methods.

1.2 Structure

```
assignment-2/                                                    1
     -- 16349086036/          // Users report and source code    2
          -- report.pdf                                          3
          -- source.py                                           4
     -- handout/                                                 5
          -- __init__.py      // Provided base file             6
```

1.3 Usage

```
python3 source.py --{command} {value}                           1
```

See details in source code.

1.4 Packages (Beyond default python)

The following external packages were used in the experiment: numpy, matplotlib, pandas or scipy. Which can be installed easily via pip.

```
$ sudo pip install {PACKAGE_NAME}                                1
                                                                 2
pip3 install -U scikit-learn scipy matplotlib                    3
```

1.5 Requirements

In this report, there were several requirements needed to satisfy during the estimation of the distribution. The list (The tasks in its original form) is provided below:

1. Implement the preprocess pipeline to transform the documents into multi-hot vector representation, and also the targets into one-hot representation, indicate how you implemented them in your report

2. Differentiate the loss function for logistic regression, write down how you compute $\frac{\partial \mathcal{L}}{\partial W_{ij}}$, $\frac{\partial \mathcal{L}}{\partial b_i}$, and then implement the calculation in vectorized style in numpy (meaning you should not use any explicit loop to obtain the gradient). Answer the two questions in your report: (1) Sometimes, to overcome overfitting, people also use L2 regularization for logistic regularization, if you add the L2 regularization, should you regularize the bias term? (2) how do you check your gradient calculation is correct?

3. Finish the training for the logistic regression model on the training dataset, include the plot for the loss curve you obtained during the training of the model. Answer the questions: (1) how do you determine the learning rate? (2) how do you determine when to terminate the training procedure?

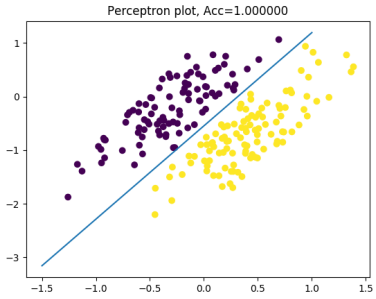
4. Somethings, other than doing a full batch gradient descent (where you take into consideration all the training data during the calculation of the gradient), people use stochastic gradient descent or batched gradient descent (meaning only one sample or several samples per update), you should also experiment with the other 2 ways of doing gradient descent with logistic regression. Answer the questions: (1) what do you observe by doing the other 2 different ways of gradient descent? (2) can you tell what are the pros and cons of each of the three different gradient update strategies?

5. Report your result for the three differently trained model on the test dataset. Do not peek into the test dataset before this requirement!

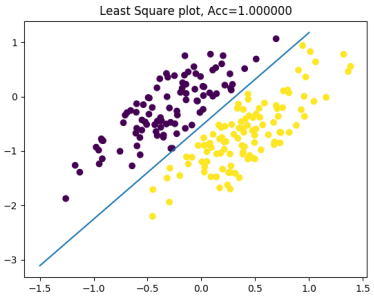
2 Experiments

3 Part 1

In this part we will the use least square model and the perceptron algorithm to learn two models to separate the dataset, and report the accuracy after when one have learned the model. We also needed to draw a decision line on top of the dataset to visually show to separate the dataset.



(a) Perception Plot with dividing line



(b) Least Square Plot with dividing line

4 Part 2

In this part of the assignment, we are required to use logistic regression to do a simple text classification task.

4.1 Task 1

Implement the preprocess pipeline to transform the documents into multi-hot vector representation, and also the targets into one-hot representation, indicate how you implemented them in your report Solution: Please observe the following code

```
// Formats the text data and filters out words that are not common
     def tokenize_data(self, data):
          for x,line in enumerate(data):
               line = line.lower()
               for c in string.punctuation:
                    line = line.replace(c, "")
               for w in string.whitespace:
                    line = line.replace(w, " ")
               words = line.split()
               data[x]=words

          word_dict = {}
          for text in data:
               for word in text:
                    if word in word_dict.keys():
                         word_dict[word] += 1
                    else:
                         word_dict[word] = 1

          word_list = list(word_dict.keys())
          for word in word_list:
               if word_dict[word] < self.min_count:
                    del word_dict[word]

          for x,line in enumerate(data):
               for word in line:
                    if word not in word_dict.keys():
                         data[x].remove(word);

          return data

     // vector representation of input and target
```

```
    def preprocessing(self):

        // Cleaning up the words
        self.text_train.data=self.tokenize_data(self.text_train.data)
        self.text_test.data=self.tokenize_data(self.text_test.data)
        self.train_vec=self.text_train.data

        // Format the vocab
        self.vocab = sorted(list(set([w for line in self.train_vec
            for w in line])))
        self.vocab_to_idx = {word: idx for idx, word in enumerate(
            self.vocab)}
        // Multi hot vector
        multi_hot_vec = np.zeros((len(self.train_vec), len(self.vocab
            )))

        for i in range(len(self.train_vec)):
             for j in range(len(self.train_vec[i])):
                  multi_hot_vec[i][self.vocab_to_idx[self.train_vec[i
                      ][j]]] = 1.0

        target_vec = np.zeros((len(self.text_train.target), 4))
        for i in range(len(target_vec)):
             target_vec[i][self.text_train.target[i]] = 1.0

        X = multi_hot_vec
        b = np.ones(len(X))
        X = np.insert(X, 0, values=b, axis=1)

        return X, target_vec
```

4.2 Task 2

Differentiate the loss function for logistic regression, write down how you compute $\frac{\partial \mathcal{L}}{\partial W_{ij}}$, $\frac{\partial \mathcal{L}}{\partial b_i}$, and then implement the calculation in vectorized style in numpy (meaning you should not use any explicit loop to obtain the gradient).

Solution:

The general linear model is defined as:

$$z = W^T \cdot x + b \tag{1}$$

where W is a weight matrix and b the bias vector. We will here define N the size of the data, K the number of classes and M as the number of features. For simplification, we will define $i = 1 \dots M$ and $j = 1 \dots K$.

We have already been given the structure of the logistic sigmoid function that we use in the softmax function where we can make the prediction through selecting a class C_k with the maximum associated probability.

We also know the cross entropy loss function (with penalizing term) which is:

$$\mathcal{L} = -\frac{1}{N}\left[\sum_{n=1}^{N} y_n \ln \hat{y_n}\right] + \lambda |W|_F^2 \tag{2}$$

$$\tag{3}$$

In this case we will have two-dimensions due to the classes, thus

$$\mathcal{L} = -\frac{1}{N}\left[\sum_{n=1}^{N}\sum_{k=1}^{K} y_{nk} \ln \hat{y_{nk}}\right] + \lambda |W|_F^2 \tag{4}$$

$$\tag{5}$$

If we use the chain rule on our two-dimension cross entropy loss function, we can find $\frac{\partial \mathcal{L}}{\partial W_{ij}}$

$$\frac{\partial \mathcal{L}}{\partial W_{ij}} = ... = -\frac{1}{N}\sum_{n=1}^{N}(y_{nj} - \hat{y_{nk}})x_{ni} + 2\lambda|W|_F^2 \tag{6}$$

$$\tag{7}$$

Where we utilise that $\sum_{k=1}^{K} y_{nk} = 1$. In the same way we can find $\frac{\partial \mathcal{L}}{\partial b_j}$ if we just augment the input vector x by defining $x_o = 1$, thus b_i can be represented as W_{io}.

$$\frac{\partial \mathcal{L}}{\partial b_j} = ... = -\frac{1}{N}\sum_{n=1}^{N}(y_{nj} - \hat{y_{nj}}) \tag{8}$$

Question 1: Sometimes, to overcome overfitting, people also use L2 regularization for logistic regularization, if you add the L2 regularization, should you regularize the bias term?

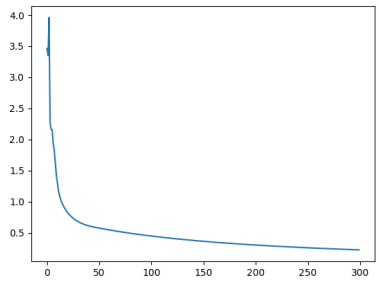
Solution: Regularization of the bias term is incorrect. In this problem, the caluclated matrix W is not that large, thus no need to add the regularization term to the gradient to simplify the matrix solution. *Thus, we should not regularize the bias.*

Question 2: how do you check your gradient calculation is correct?

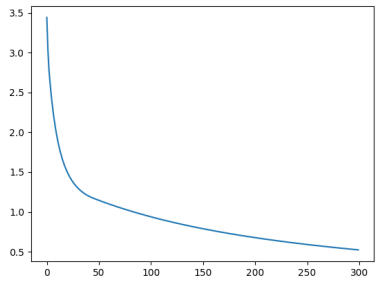
Solution: The correctness can be proved mathematically. In addition, decrease of loss can also approve the correctness in some degree.

4.3 Task 3

Finish the training for the logistic regression model on the training dataset, include the plot for the loss curve you obtained during the training of the model. Solution: We can see that we manage to reduce the error in the expected way and by variation of the learning rate we can see that



(a) Learning rate=1 and 300 iterations



(b) Learning rate=0.5 and 300 iterations

Question 1: how do you determine the learning rate?

Solution: Through testing and see how fast or slowly it converges towards a local optimal point. I decided to select the learning rate of 0.5 in the beginning with a decay rate that slowly decreases the learning rate. That is because the rate of loss decreases with each iteration and we do not want to overstep.

Question 2: how do you determine when to terminate the training procedure?

Solution: In practice, the iterations can go on indefinitely, thus I set that if the decent of loss is less than 10^3 then we stop since we have found a loss that is acceptable.

4.4 Task 4

Somethings, other than doing a full batch gradient descent (where you take into consideration all the training data during the calculation of the gradient), people use *stochastic gradient descent* or *batched gradient descent* (meaning only one sample or several samples per update), you should also experiment with the other 2 ways of doing gradient descent with logistic regression.

Question 1: what do you observe by doing the other 2 different ways of gradient descent?

Solution: When using a stochastic gradient descent, we only use one sample. Thus, we priortizing speed over finding the best possible solution (the global optimum). When using the batched gradient descent, we update the parameters with multiple samples. Thus, slower than the stochastic gradient descent but we can get a better solution. But still not as good as a full batch as we previously used.

Question 2: can you tell what are the pros and cons of each of the three different gradient update strategies?

Solution: The differences between the three different models is the balance between time spent and accuracy.

Full batch gradient descent:

Going through the whole dataset and gives an global optimum but it is time consuming.

Batched gradient descent:

Going through the dataset partially and gives an quick estimation but if the batch size is to large the data can be misleading.

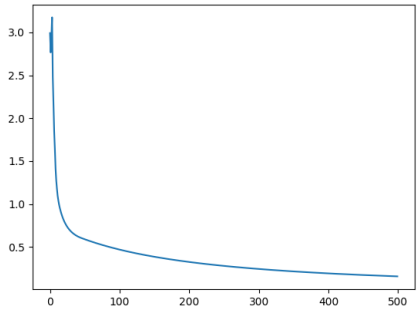
Stochastic gradient descent:

The parameters gets updated quicker beause we are randomly selecting a sample (something that can be seen in the uneven graph) but the accuracy of the model may suffer because this model can not converge to a global optimum because we are only using one sample to train the whole dataset.

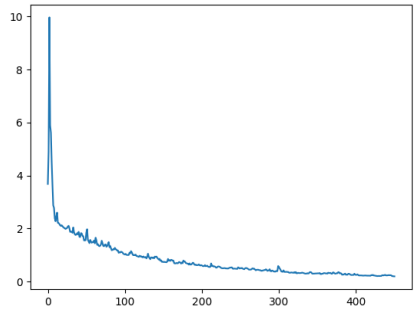
4.5 Task 5

Report your result for the three differently trained model on the test dataset.

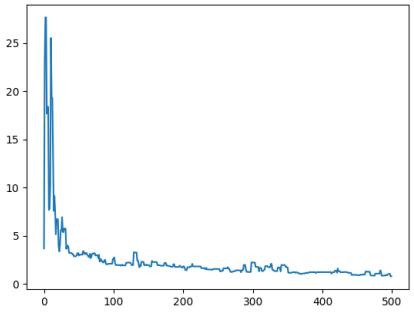
The following graphs shows the two new models with learning rate=1 (decaying) and 500 iterations. I would receive higher accuracy if I were to use more iterations but decided to stop at 500 iterations to save time and get a decent accuracy.



(a) Full batch, final loss 0.1559, acc 0.8217



(b) Batched with final loss 0.1951, acc 0.8021



(c) Stocastic with final loss 0.8669, acc 0.7720