Report for Spring 2019 PRML Assignment # 2

Date: April 2, 2019

Abstract: The purpose of this assignment was to gain experience with several well known linear classification methods such as perceptron and logistic regression. We will be experimenting with a realistic dataset and analyzing these algorithms.

1 Introduction

For this assignment, I experienced designing classifier algorithms. I followed the report requirements [1] and instructions from our T.A. Zhifeng. We were assigned a task of designing and analyzing linear regression model, perceptron algorithm, and logistic regression. This assignment consisted mainly of understanding the algorithms, coding them, and comparing the results.

This report is organized with Section 2 goes over the task assigned. Section 3 has the conclusion.

2 Assignment

2.1 Part 1

2.1.1 Least Square Model

The coding portion for this algorithm is fairly simple. Since the data is linearly separable, I took the approach by splitting the dataset and did a linear regression to both. After, I averaged the two lines to make the decision boundary. The code is shown in Figure 1. The result is fairly accurate due to no easy points and there is only two classes. The graph is shown in Figure 2.

```
def least_square_model():
    cls1x = []
    cls1y = []
    cls2x = []
    cls2y = []
    for i in range(len(data.X)):
        if(data.y[i]):
            cls1x.append(data.X[i,0])
            cls1y.append(data.X[i,1])
        else:
            cls2x.append(data.X[i,0])
            cls2y.append(data.X[i,1])
    cls1x_mean = np.mean(cls1x)
    cls1y_mean = np.mean(cls1y)
    cls2x_mean = np.mean(cls2x)
    cls2y_mean = np.mean(cls2y)
    num1 = 0
    den1 = 0
    num2 = 0
    den2 = 0
    for i in range(len(cls1x)):
        num1 += (cls1x[i] - cls1x_mean)*(cls1y[i] - cls1y_mean)
        den1 += (cls1x[i] - cls1x_mean)**2
    for i in range(len(cls2x)):
        num2 += (cls2x[i] - cls2x_mean)*(cls2y[i] - cls2y_mean)
        den2 += (cls2x[i] - cls2x_mean)**2
    m = ((num1 / den1)+(num2 / den2))/2
    c = ((cls1y_mean - m*cls1x_mean)+(cls2y_mean - m*cls2x_mean))/2
    x = np.linspace(-1.5,1.5)
    y = m*x+c
    print(m)
    print(c)
    plt.plot(x, y, '-r')
    data.plot(plt).show()
```

Figure 1. Code for Least Square Model

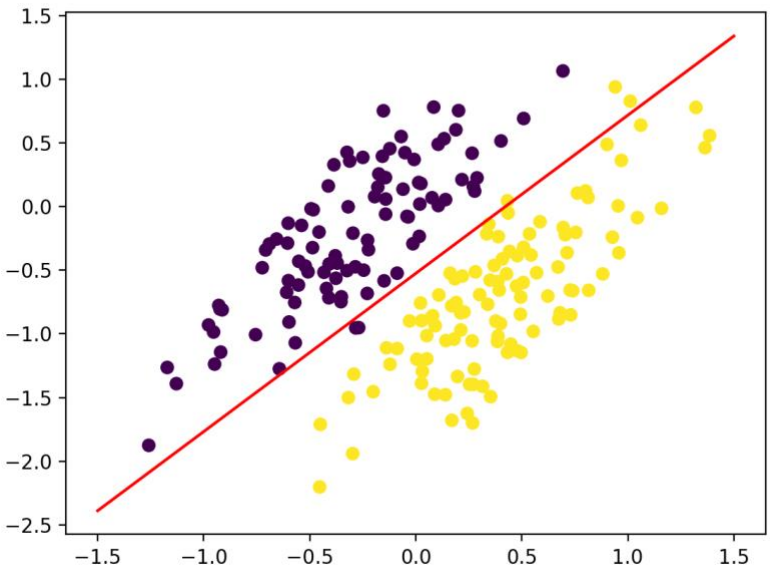

Figure 2. Graph for Least Square Model

2.1.2 Perceptron Algorithm

For this code I implemented the perceptron algorithm as instructed, shown in Figure 3.

```
class Perceptron(object):

    def __init__(self, no_of_inputs, threshold=100, learning_rate=0.001):
        self.threshold = threshold
        self.learning_rate = learning_rate
        self.weights = np.zeros(no_of_inputs + 1)

    def predict(self, inputs):
        summation = np.dot(inputs, self.weights[1:]) + self.weights[0]
        if summation > 0:
            activation = 1
        else:
            activation = 0
        return activation

    def train(self, training_inputs, labels):
        for _ in range(self.threshold):
            for inputs, label in zip(training_inputs, labels):
                prediction = self.predict(inputs)
                self.weights[1:] += self.learning_rate * (label - prediction) * inputs
                self.weights[0] += self.learning_rate * (label - prediction)
def perceptron():
    X = data.X
    y = np.zeros(len(data.y))
    for i in range(len(data.y)):
        if data.y[i]:
            y[i]=1
        else:
            y[i]=0
    p = Perceptron(2)
    p.train(X, y)
    m = -(p.weights[1]/p.weights[2])
    c = -(p.weights[0]/p.weights[2])
    x = np.linspace(-1.5,1.5)
    y = m*x+c
    print(m)
    print(c)
    plt.plot(x, y, '-r')
    data.plot(plt).show()
```

Figure 3. Code for Perceptron Algorithm

This algorithm is like a trial and error method, so it takes many iterations to get it correct. The graph is shown in Figure 4. The result is also fairly accurate.

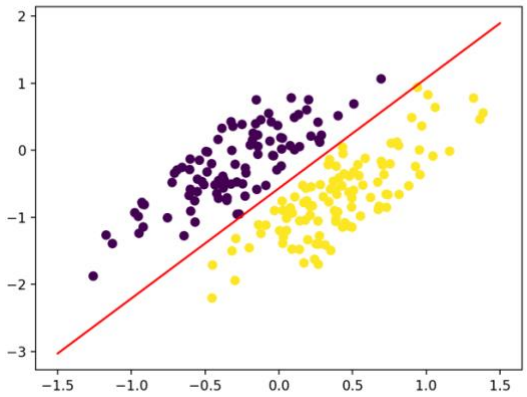

Figure 5. Graph for Perceptron Algorithm

2.2 Part 2

The second part of the lab was to incorporate logistic regression. We were give a text file to identify the categories by training it with the softmax function.

2.2.1 Transforming the Text Document

First, I needed to filter out the punctuation and whitespace creating a list of words with all lowercase letters. In order to save memory, the frequency of the word has to be at least 10 times. The function I created is in Figure 6.

```
def text_to_words(text):
    table = str.maketrans('', '', string.punctuation)
    text = str(text).translate(table).lower()
    #print(text)
    words = text.split()
    words = Counter(words)
    words10 = []
    for word in words:
        if words[word] >= 10:
            words10.append(word)
    words = sorted(words10)
    return words
```

Figure 6. Code for Changing the Text file

Then I needed to create a dictionary with all the words in alphabetical order, shown in Figure 7.

```
def creat_dict(data):
    words_train = text_to_words(data)
    my_dict = {}
    for i in range(len(words_train)):
        my_dict[words_train[i]] = i
    return my_dict
```

Figure 7. Code for Creating Dictionary

Then I need to represent the data in vector form and the category in one-hot representation, shown in Figure 8.

```
def creat_labels(targets):
    one_hot_labels = []
    for t in targets:
        one_hot_label = np.zeros(4)
        one_hot_label[t] = 1
        one_hot_labels.append(one_hot_label)
    return np.array(one_hot_labels)

def get_vectors(data,dict):
    muiti_hots = []
    for subtext in data:
        subwords = text_to_words(subtext)
        muiti_hot = np.zeros(len(dict))
        for word in subwords:
            if(word in dict):
                muiti_hot[dict[word]] = 1
        muiti_hots.append(muiti_hot)
    return np.array(muiti_hots)
```

Figure 8. Code for Changing data in to Vector and Label into One-hot

2.2.2 Differentiate the Loss Function

Following the equations in the guidelines I coded the loss function using softmax function, shown in Figure 9.

```
def getLoss(w, x, y, lam):
    m = x.shape[0]
    scores = np.dot(x,w)
    prob = softmax(scores) #Next we perform a softmax on these scores to get their probabilities
    loss = (-1 / m) * np.sum(y * np.log(prob)) + (lam/2)*np.sum(w*w) #We then find the loss of the probabilities
    grad = (-1 / m) * np.dot(x.T,(y - prob)) + lam*w #And compute the gradient for that loss
    return loss,grad

def softmax(z):
    z -= np.max(z)
    sm = (np.exp(z).T / np.sum(np.exp(z),axis=1)).T
    return sm
```

Figure 9. Loss Function and Softmax Function

Then combining all the function we can differentiate the loss function, shown in Figure 10.

```
def logistic_regression():
    text_train, text_test = get_text_classification_datasets()
    labels_train = creat_labels(text_train.target)
    my_dict = creat_dict(text_train.data)
    vectors_train = get_vectors(text_train.data, my_dict)
    w = np.zeros([vectors_train.shape[1],4])
    lam = 1
    iterations = 1000
    learningRate = 0.005
    losses = []
    for i in range(0,iterations):
        loss,grad = getLoss(w,vectors_train,labels_train,lam)
        losses.append(loss)
        w = w - (learningRate * grad)
    plt.plot(losses)
    plt.show()
```

Figure 9. Code for Differentiating the Loss Function

2.2.3 Determine the Learning rate and Iterations

The learning rate determines the precision and the speed of finding the result. We can terminate the function when the loss is barely changing any more. Figure 10 shows the loss curve.

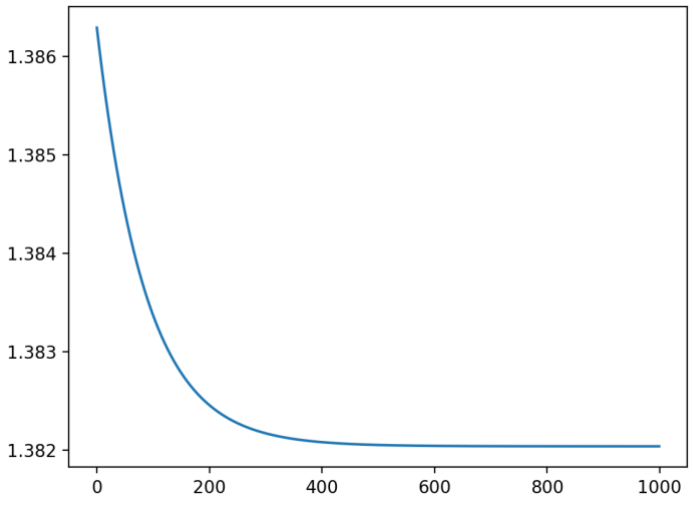

Figure 10. Loss Curve with Learning Rate = 0.005

2.2.4 Stochastic and Batched Gradient Descent

The cons of using a full batch gradient descent is that it takes longer to process. When we need an immediate result with less accuracy we a stochastic gradient descent, meaning only one sample. The in between method will be batched gradient descent, meaning several samples per data. This will be faster than full batch and more accurate than stochastic.

3 Conclusion

Overall, the assignment was successful as I was able to finish it and further understand the three algorithms. However, I was struggling during Part 2 of the assignment due to the steep learning curve and difficult material. In the end I was able to finish it.

References

1. “Assignment 2” Pattern Recognition and Machine Learning, Fudan University, Spring 2019, <https://zfhu.ac.cn/PRML-Spring19-Fudan/assignment-2/index.html>.