Programming Assignment 4

All codes are written in Python3 and can be executed by python XXX.py

Part 1: Implementation

(a) Perceptron Learning algorithm

·Software & libraries:
Python3.6 + numpy

·Output:

```
perception weights are:
[ 0.         0.07381896 -0.05973903 -0.04698182]
perception accuracy:
0.984
```

·Implementation code:
python Perceptron.py

·Data Structure:
Use numpy.Array to store weights
Use lists to store each points' coordinators
Use lists to store each points' original prediction value

·Description:
I implemented Perceptron as a class in perceptron.py. Though iterating each row, the perceptron will keep updating its weights to classify better. After going through the entire dataset, the error of classification for the training dataset is 0.016.

·Optimization:
Use a function digest() to train data.Add a parameter:rate,since the diff between predict value is 0,or 2.I use a function to return 1/-1 according to the value of original diff and multiply diff with learning rate and coordinators of points to get new weights.

·Challenge:
I was thinking about how to train the data and use the calculated weights to make prediction.I planned to use a function only once to train the entire data.Finally,I thought I build a function and for each line in the txt file I call the function to recompute weights.

(b) Pocket algorithm

·Software & libraries:
Python3.6 + numpy

·Output:

```
pocket perceptron weights are:
[ 0.5        -0.42190234  0.1499017  -0.52735952]

pocket perceptron accuracy:
0.522
```

·Implementation code:
python Pocket.py

·Data Structure:
Use numpy.Array to store weights
Use lists to store each points' coordinators
Use lists to store each points' original prediction value

·Description:
I implemented Pocket Algorithm as a class extend Perceptron.I stored all row and values in two lists.This time I only call the digest() function once to compute weights. After going through the entire dataset, the error of classification for the training dataset is 0.478.

·Optimization:
Instead of call function digest() to train data for each data point,I stored all the data in a list.Then I called digest() for 7000 times to get weights.

·Challenge:
The error rate is much higher compared with Perceptron even if I loop for 7000 times to get error rate lower.

(c) Logistic Regression algorithm
·Software & libraries:
Python3.6 + numpy+ pandas

·Output:
```
unstandardized weights: [[ 7.16345868]
 [-8.39649979]
 [ 5.33505871]
 [ 3.74746008]]
accuracy: 0.493
```

·Implementation code:
Logistic_regr.py (attached in the file)

·Data Structure:
Use the mat data type to store the data and labels. It is easier to calculate. "train_x" is a mat data type, and each row stands for one sample . "train_y" is mat data type too, and each row is the corresponding label.

·Description:

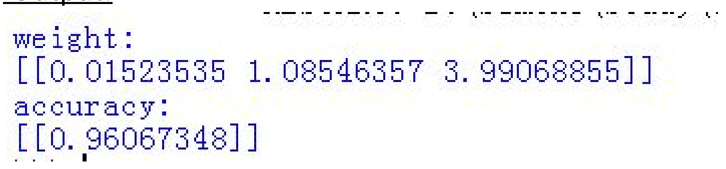
Step 1: Load data and transfer data into mat type
Step 2: Logistic Regression training
Step 3: Test the data on the model

·Optimization:
In the data-loading phase, we first keep the original labels, which is "-1" and "+1", but it cause some small trouble during the test phase. So we change the labels into 0 and 1, so that we can use bool operator to compare the actual labels and the predicted labels. The algorithm is slightly faster in time.

·Challenge:
The logistic regression is simple to calculate. So once we figure out the logic of gradient descent process, the algorithm is easier to implement.

(d) Linear Regression algorithm
·Software & libraries:
Python3.6 + numpy + pandas

·Output:



```
weight:
[[0.01523535 1.08546357 3.99068855]]
accuracy:
[[0.96067348]]
```

·Implementation code:
linear_regr.py (attached in the file)

·Data Structure:
Use the mat data type to store the data and labels. It is easier to calculate. "train_x" is a mat data type, and each row stands for one sample . "train_y" is mat data type too, and each row is the corresponding label.

·Description:
Step 1: Load data and transfer data into mat type
Step 2: Linear Regression after 2000 iterations (learning rate=0.07)

·Optimization:
We first implement the matrix calculation in hand-written code. Then we read through the numpy library code, and use matrix dot product to calculate.

·Challenge:
There are many matrix calculations in the algorithm, so the challenge is how to calculate efficiently. I am not familiar to the matrix calculation, and after some debugging time, the calculation is implemented in a neat and efficient way using numpy.

Part 2: Software Familiarization

(a) Classification(Perceptron Learning algorithm & Pocket)

<u>·Software & libraries:</u>

Python3.6 + sklearn. linear_model

<u>·Output:</u>

prediction weights :
[0.0, 0.078091246382972768, -0.062617009141169619, -0.045854555098796192]

Accuracy   99%(It may varies each time you execute around 99%)

<u>·Implementation code:</u>

python classify_lib.py (attached in the file)

<u>·Comparison:</u>

The sklearn library uses stochastic gradient descent,and it is more accurate than our codes.

<u>·How to improve our code:</u>

For Pocket.py we can iterate more times to increase our accuracy rate.

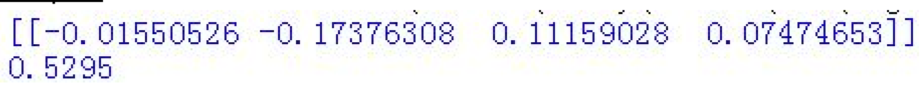
For Perceptron.py we could also make the digest() function to loop for the entire data instead of call it for each data point.

(b) Logistic Regression algorithm

<u>·Software & libraries:</u>

Python3.6 + pandas + sklearn. linear_model

<u>·Output:</u>



```
[[-0.01550526 -0.17376308  0.11159028  0.07474653]]
0.5295
```

(line 1: standardized weight)
(line 2: accuracy)

<u>·Implementation code:</u>

logistic_regr_lib.py (attached in the file)

<u>·Comparison:</u>

The sklearn library have multiple optimization methods, such as newton method, stochastic gradient descent, while our algorithm only optimize according to gradient descent method. The sklearn library also provides binary and multi-class logistic regression.

<u>·How to improve our code:</u>

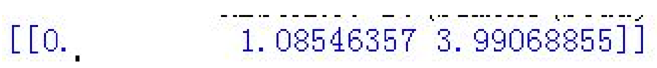
We can pre- process our data to avoid overfitting situation. In addition, we can provide multiple optimization methods for different amount of data.

(c) Linear Regression algorithm

<u>·Software & libraries:</u>

Python3.6 + pandas + numpy + sklearn. linear_model

<u>·Output:</u>



```
[[0.          1.08546357 3.99068855]]
```

<u>·Implementation code:</u>
linear_regr_lib.py (attached in the file)

<u>·Comparison:</u>
The sklearn library can choose whether to centralize the data, while our code do not have the pre-preprocessing step. “n_jobs” indicate the number of jobs, if it is more than one, then this will help accelerate the speed of the algorithm on large data sets.

<u>·How to improve our code:</u>
We can add the step of centralizing data. In addition, we can run the algorithm in parallel structure, to accelerate the speed of the algorithm on large data sets.

Part 3: Applications

(a) Perceptron Learning algorithm & Pocket algorithm

PLA could be used to decide whether an applicant could apply for a credit card. Banks could take each applicant as a vector x, which is a combination of applicant’s various dimension features such as applicant’s age, annual salary, length of service and so on. Each dimension would impose different positive or negative impacts on the decision whether banks would approve the credit card request. In this case, we can integrate these dimensions and give applicants a score. If the score exceeds a certain threshold, we can know an applicant could get the credit card. Otherwise, an applicant could not get the card.

(b) Logistic Regression algorithm

There is an application of logistic regression algorithm in the field of financial prediction. It uses logistic regression to overcome multiple co-linearity among variables and meanwhile retain useful information of original variables. This is a useful tool for forecast firm failure for policymakers and others interested in prediction system.

(c) Linear Regression algorithm

Linear Regression involves showing how the variation in the “dependent variable” can be captured by change in the “independent variables”.

In Business, this dependent variable can also be called the predictor or the factor of interest for eg., sales of a product, pricing, performance, risk etc. Independent variables are also called explanatory variables as they can explain the factors that influence the dependent variable along with the degree of the impact which can be calculated using “parameter estimates” or “coefficients”. These coefficients are tested for statistical significance by building confidence intervals around them so that the model that we are building is statistically robust and based on objective data. The elasticity based on the coefficient can tell us the extent to which a certain factor explains the dependent. Further, a negative coefficient can be interpreted to have a negative or an inverse relation with the dependent variable and positive coefficient can be said to have a positive influence. The key factor in any statistical models is the right understanding of the domain and its business application.

Part 4: Group Members

Yidan Xue	Perceptron Learning, Pocket, Application
Wenjie Shi	Logistic & Linear Regression, Report

Part 5: Reference

[1] http://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html
[2]
http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html