Technical Specification and Study Report

[Perceptron Learning, Pocket, Logistic Regression, and Linear Regression Algorithms]

Author: [Cheng-Lin Li (9799716705), Jianfa Lin (6950769029)]

Date: [Nov., 4, 2016]
Version: [1.0]

Document Control

Author

Position	Name	USC ID
Student	Cheng-Lin Li	9799716705
Student	Jianfa Lin	6950769029

Revision history

Version	Issue date	Author/editor	Description/Summary of changes
0.9	11/4/2016	Jianfa Lin	First draft release
1.0	11/5/2016	Jianfa Lin	First release

Related documents

Document	Description
NeuralNetwork.py	The implementation of Neural Network algorithm by Python 3.5
downgesture_train.list	List of training data sets for Neural Network algorithm.
downgesture_test.list	List of testing data sets for Neural Network algorithm.
gestures	Folder of training data sets and testing data sets.

Contribution

Name	USC ID	Labour
Cheng-Lin Li	9799716705	Implement the code
Jianfa Lin	6950769029	Write the document.

Table of Contents

1 INTRODUCTION

1.1 Objectives

Implement the Back Propagation algorithm for Feed Forward Neural Networks to learn down gestures from training images available in **downgesture_train.list**. After the training, use the trained network to predict the labels for the gestures in the test images available in **downgesture_test.list**.

The result should print out whether the prediction is correct for “down” gesture, the prediction result and the accuracy of prediction.

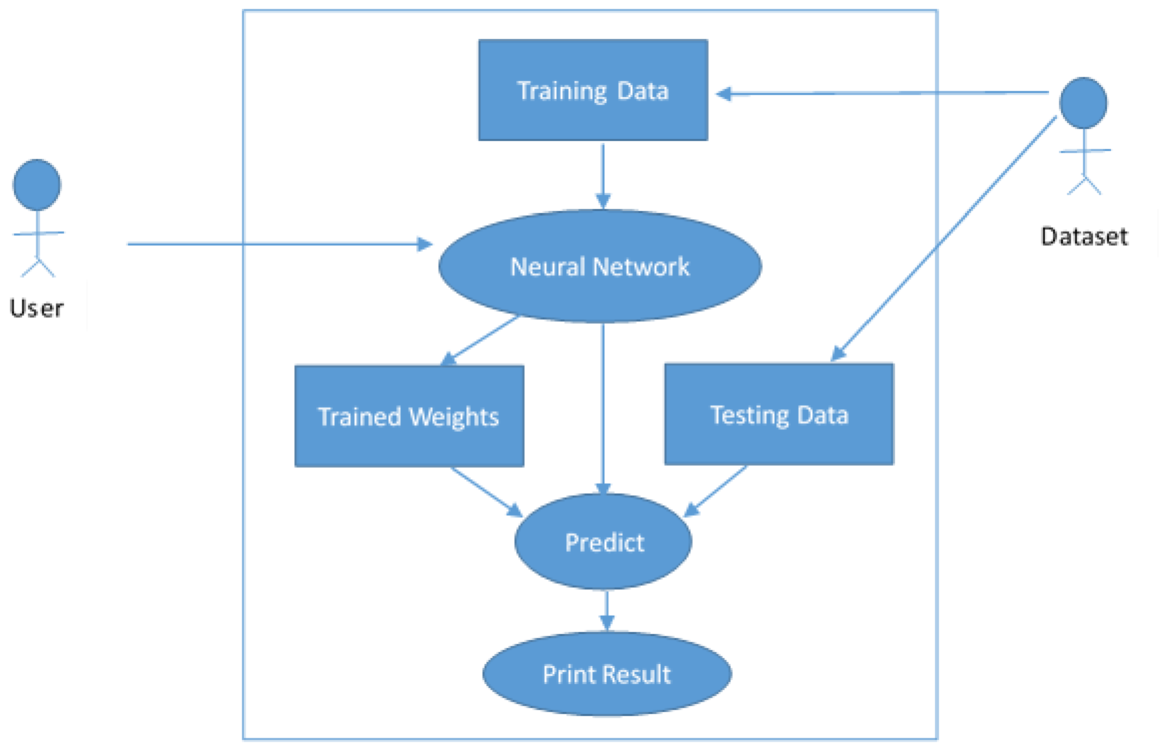
There are 184 training data sets in downgesture_train.list and 83 testing data sets in downgesture_test.list. The training data sets and testing data sets are PGM image, which is a special to present grayscale graphic image.

2 Functionality [1]

2.1 Description: Neural Network Algorithm

1. Input the dataset and signal the data with 1 or 0.
2. Assign the parameters.
3. Get an initial weight vector with [[Weights of level-01], [Weights of level-12], ..., [Weights of level-(L-1)(L)]], in which each value in the assigned range.
4. Compute $X_j^{(l)}$ in the forward direction with one random training data set.
5. Compute $\delta_j^{(l)}$ in the backward direction with corresponding result in step 4.
6. Update weight vectors $w_{ij}^{(l)}$ by $w_{ij}^{(l)} = w_{ij}^{(l)} - \eta * X_i^{(l-1)} * \delta_j^{(l)}$. Where η is learning rate.
7. Repeat step 4 ~ 6 a certain number of times. (1000 times here)
8. Predict the gesture of testing dataset with trained weights vector, and calculate the correct rate of prediction.

3 Use Case

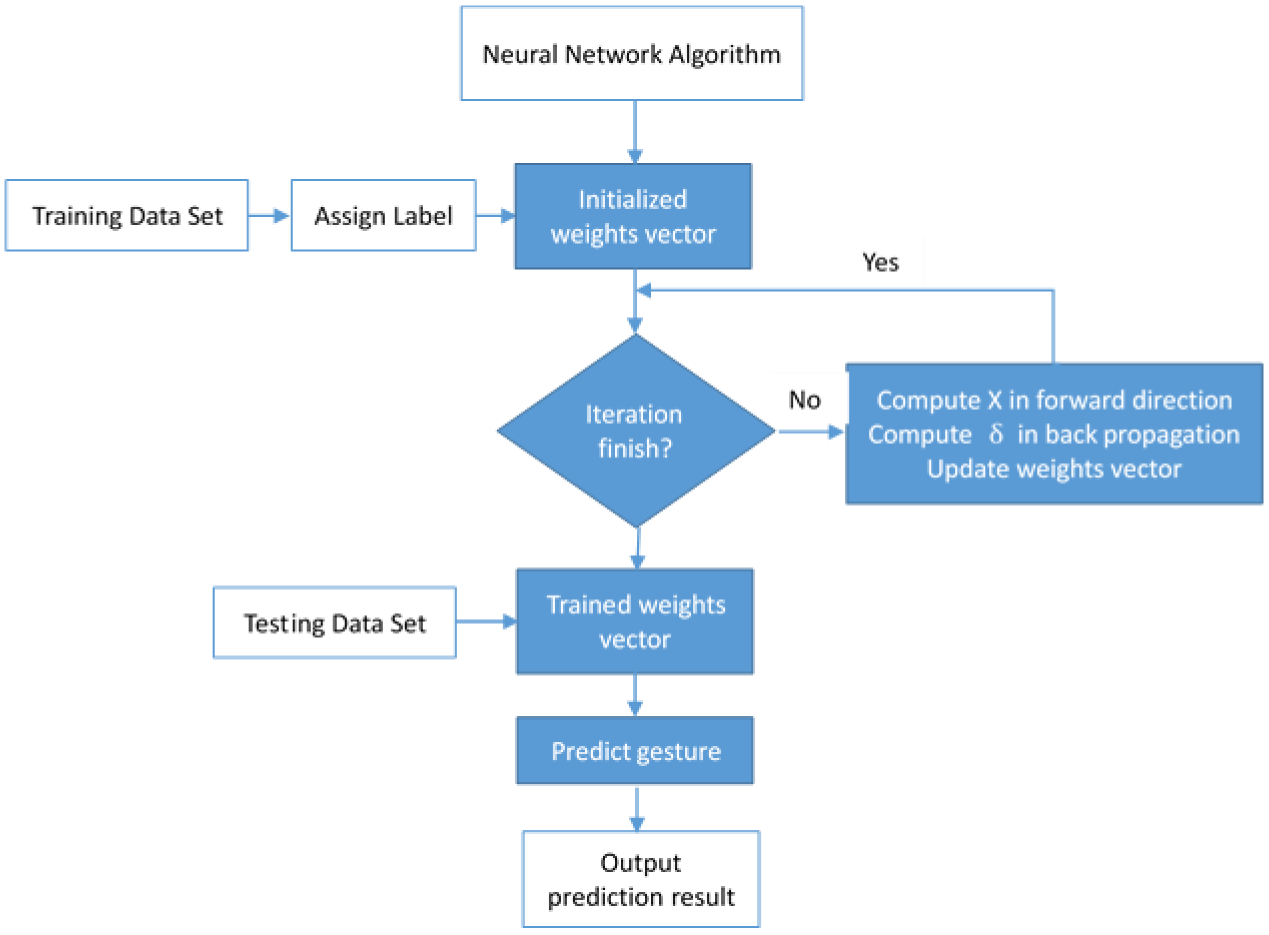


Neural Network

4 Architecture

4.1 Architecture Overview for Neural Network Algorithms.

Here is the generalized architecture diagram for Neural Network algorithms.



5 Design

5.1 Data Structure

1. Training data and testing data are stored in array like [[data1], [data2], ... , [dataN]]. For [data1] = [value_dimension_1, value_dimension_2, ..., value_dimension_d].
2. Labels of each data set are stored in list.
3. Weights store in an array like [[Weights of level-01], [Weights of level-12], ..., [Weights of level-(L-1)(L)]]. For [Weights of Level-01]=[[w01, w02, ..., w0d], [w11, w12, ..., w1d], ... [wd1, wd2, ..., wdd]]
4. The layer sizes are store in an array, in which each item corresponds to different layer.
5. Weight vector store in an array/vector.

5.2 Procedures

1. Get input training data sets from files.
2. Assign label to each data set according to its file name. If ‘down’ exists in file name, the label is 1, otherwise 0.
3. Class constructor initialize the hidden layers array, network layers array, activation function and other parameters according to user’s input.
4. Execute algorithm by execute() function.
5. Initialize weights vector randomly with each value between given range.
6. A while-loop trains the neural network with all training data sets and initial weight vector.
    a. According to Stochastic Learning approach, randomly pick up a data set from all training data sets.
    b. Compute $X_j^{(l)} = \theta\left(\sum_{i=0}^{d(l-1)} w_{ij}^{(l)} X_i^{(l-1)}\right)$ in the forward direction data set chosen in a. θ is an activation function, which can be both logistic sigmoid and hyperbolic tangent functions as what we set.
    c. Compute $\delta_j^{(l-1)} = \left(1 - \left(X_i^{(l-1)}\right)^2\right) \sum_{j=1}^{d(l)} W_{ij}^{(l)}\, \delta_j^{(l)}$ in the backward direction with corresponding result in step b.
    d. Update weight vectors $w_{ij}^{(l)}$ by $w_{ij}^{(l)} = w_{ij}^{(l)} - \eta * X_i^{(l-1)} * \delta_j^{(l)}$. η is learning rate.
7. The program terminates when the maximum iteration was reached. The default iteration here is 1000.
8. Get input testing data sets from files.
9. Predict the label of each data set with trained neural network. If the prediction finds out the “down” gesture correctly, print “Match(O)”, otherwise “Match(X)”. Calculate and output the accuracy finally.

5.3 Optimizations and Challenges

5.3.1 Optimization

- In the training procedure, we pick up the data randomly rather than sequentially from data set in every training epoch. Because we found that if putting in the data sets in sequence and repeating, the neural network would be less well-trained so that the accuracy of prediction would be lower.

- Combine the step 6.c and step 6.d in section 5.2 as mentioned above. During the calculation of $\delta_j^{(l-1)}$, we found that $w_{ij}^{(l)}$ can be updated after being leveraged in calculation, so we update it at the same loop.
- The implementation supports logistic (this assignment) and hyperbolic tangent (professor teach) activation functions with multiple hidden layers.
- The implementation supports binary classification enable or disable feature in logistic activation function. It will help to observe the gap between label and output data for parameters tuning purpose.
- According to our trail, (0,1) is a better range for the weight vectors when applying in our program, which help trained neural network get accuracy as high as 95% at most.

5.3.2 Challenge:

- What is the best initial weight matrix is the most critical for the accuracy, especially in a relative small training iterations.
- It's very confusing when dealing with array and matrix operations, especially for the weights at this time. The weights vector in the hidden layer is very complex.
- How to select and put in training data impacts the results. We choose to randomly pick up data from data sets and get a better result.
- The indexes of each X, weight, and δ are also confusing. It will be easy to make mistake when choosing which value to be calculate during implementation.

6 Usage Instruction

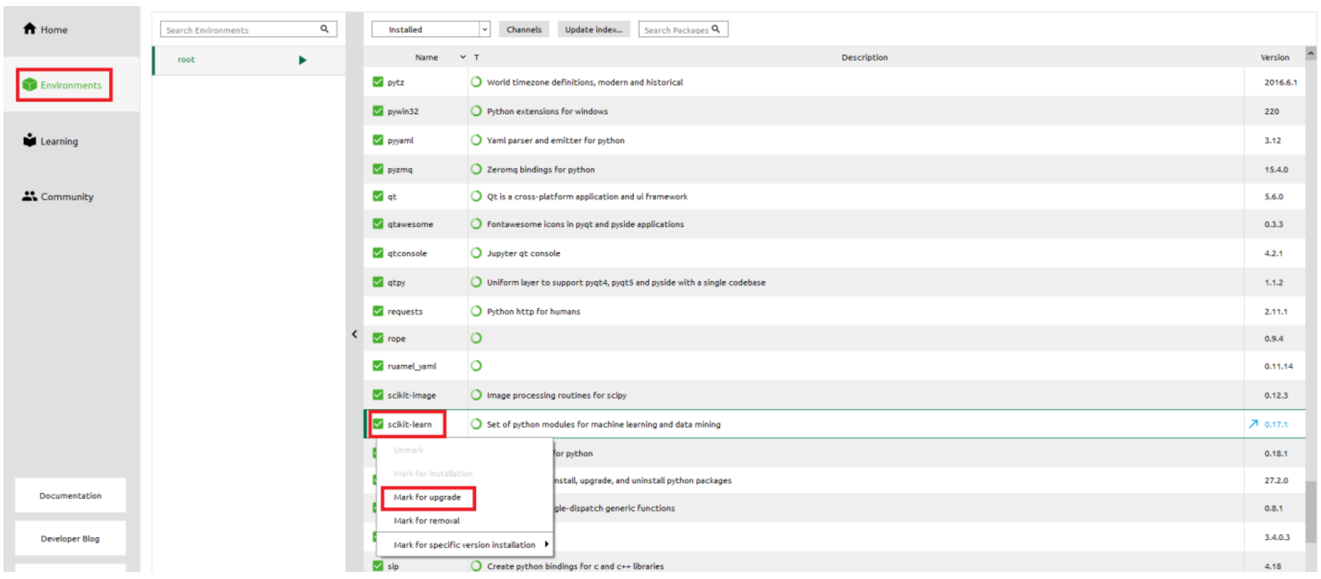
These implementations leverage Python 3.5 with a lot of libraries. We recommend the latest "Anaconda version 4.2.0" data science platform with Python version 3.5 as execution environment. It is a library management environment and collects 175 libraries for developer to build up their own code. Spyder 3.0.0 includes in the platform as IDE.

The URL of this package is https://www.continuum.io/downloads

Alternatively, these program can be executed on Python 3.5 with scikit-learn version 0.18 environment. Below instructions are based on Anaconda v4.2.0 with Spyder v3.0 or above (Spyder v3.0.1 is available).

6.1 Upgrade scikit-learn to version 0.18

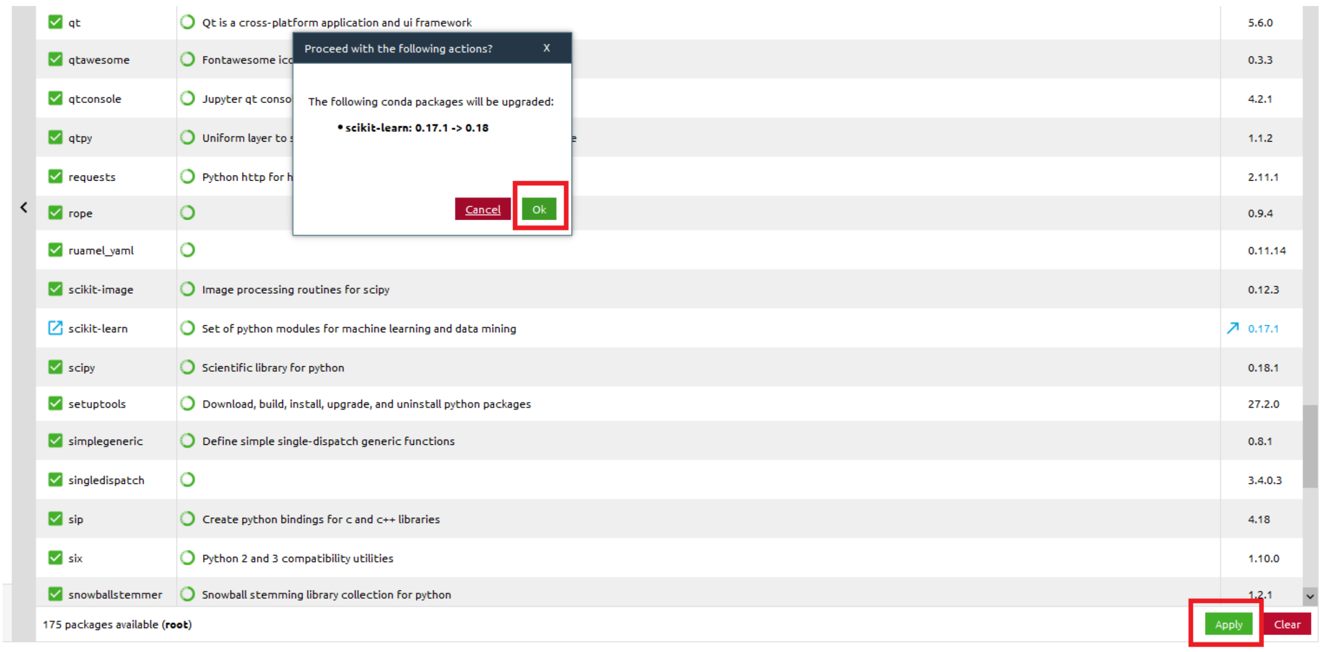
Due to a new implementation of Gaussian Mixture module was included in the latest official release of scikit-learn version 0.18 which does not include in the latest Anaconda version 4.2.0. Please upgrade scikit-learn in Anaconda environment.



Name	T	Description	Version
✓ pytz		World timezone definitions, modern and historical	2016.6.1
✓ pywin32		Python extensions for windows	220
✓ pyyaml		Yaml parser and emitter for python	3.12
✓ pyzmq		Zeromq bindings for python	15.4.0
✓ qt		Qt is a cross-platform application and ui framework	5.6.0
✓ qtawesome		Fontawesome icons in pyqt and pyside applications	0.3.3
✓ qtconsole		Jupyter qt console	4.2.1
✓ qtpy		Uniform layer to support pyqt4, pyqt5 and pyside with a single codebase	1.1.2
✓ requests		Python http for humans	2.11.1
✓ rope			0.9.4
✓ ruamel_yaml			0.11.14
✓ scikit-image		Image processing routines for scipy	0.12.3
✓ scikit-learn		Set of python modules for machine learning and data mining	↗ 0.17.1
		or python	0.18.1
		nstall, upgrade, and uninstall python packages	27.2.0
		gle-dispatch generic functions	0.8.1
			3.4.0.3
✓ sip		Create python bindings for c and c++ libraries	4.18

Apply the upgrade.

You can upgrade Spyder to v3.0.1 in the same screen.



✓ qt	Qt is a cross-platform application and ui framework	5.6.0
✓ qtawesome	Fontawesome ic	0.3.3
✓ qtconsole	Jupyter qt conso	4.2.1
✓ qtpy	Uniform layer to	1.1.2
✓ requests	Python http for h	2.11.1
✓ rope		0.9.4
✓ ruamel_yaml		0.11.14
✓ scikit-image	Image processing routines for scipy	0.12.3
scikit-learn	Set of python modules for machine learning and data mining	↗ 0.17.1
✓ scipy	Scientific library for python	0.18.1
✓ setuptools	Download, build, install, upgrade, and uninstall python packages	27.2.0
✓ simplegeneric	Define simple single-dispatch generic functions	0.8.1
✓ singledispatch		3.4.0.3
✓ sip	Create python bindings for c and c++ libraries	4.18
✓ six	Python 2 and 3 compatibility utilities	1.10.0
✓ snowballstemmer	Snowball stemming library collection for python	1.2.1

Proceed with the following actions? X

The following conda packages will be upgraded:

• **scikit-learn: 0.17.1 -> 0.18**

Cancel Ok

175 packages available (**root**)

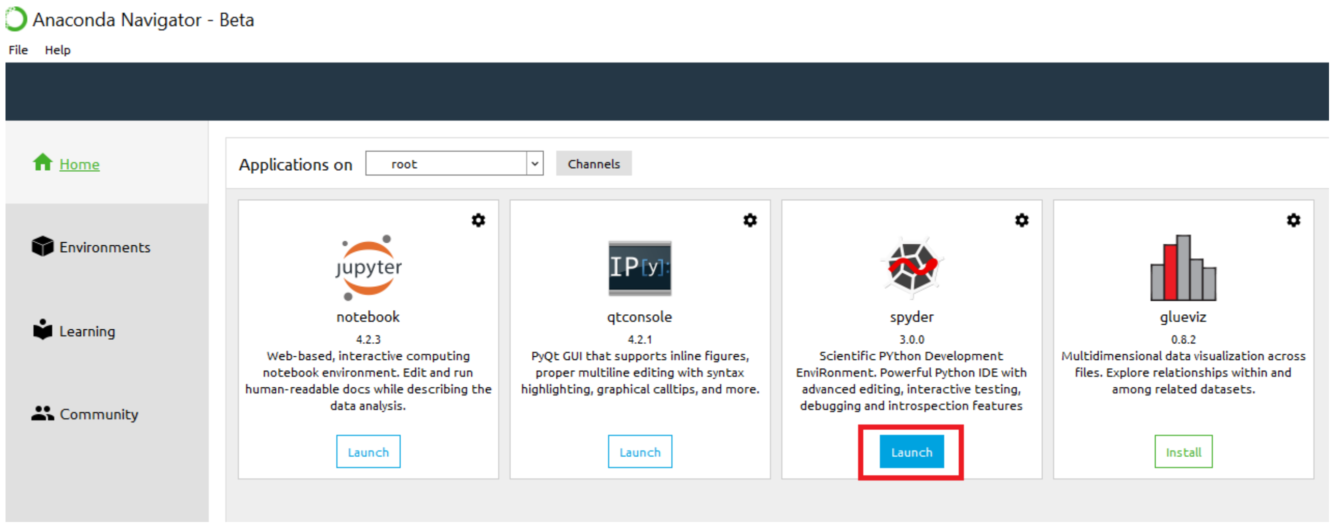
Apply Clear

6.2 Script Files and Dataset File in same folder

- NeuralNetwork.py (Implementation code)
- downgesture_train.list (List of training data sets for Neural Network algorithm)
- downgesture_test.list (List of testing data sets for Neural Network algorithm)
- gestures (Folder of data set)

6.3 Execution from Spyder version 3.0

1. Under the “Home” of Anaconda Navigator, Click “Launch” to enable spyder 3.0.0 environment.



Anaconda Navigator - Beta

File Help

Home

Environments

Learning

Community

Applications on root Channels

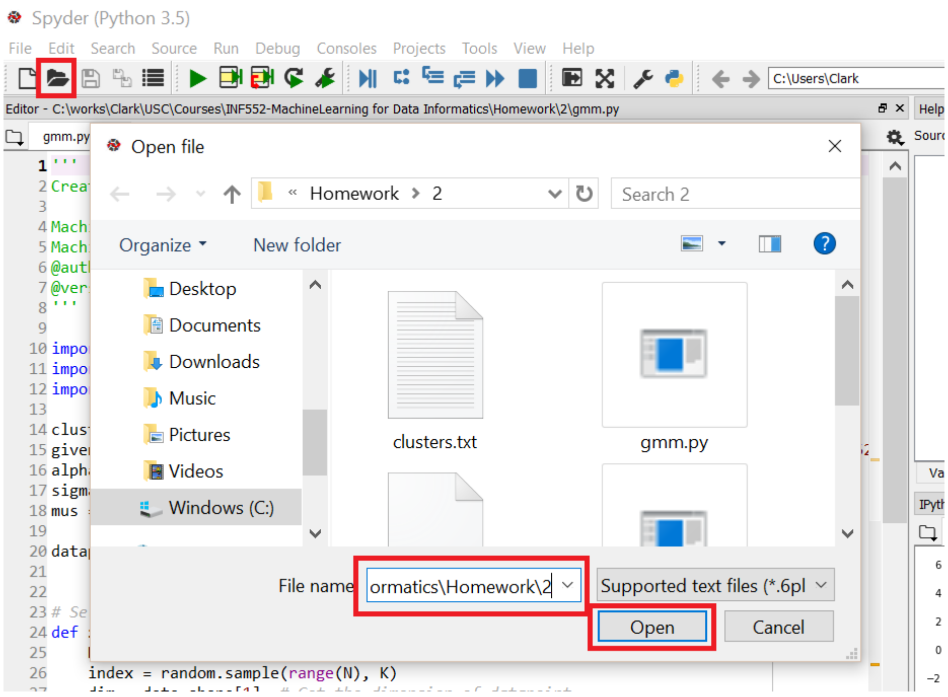
notebook
4.2.3
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch

qtconsole
4.2.1
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.
Launch

spyder
3.0.0
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features
Launch

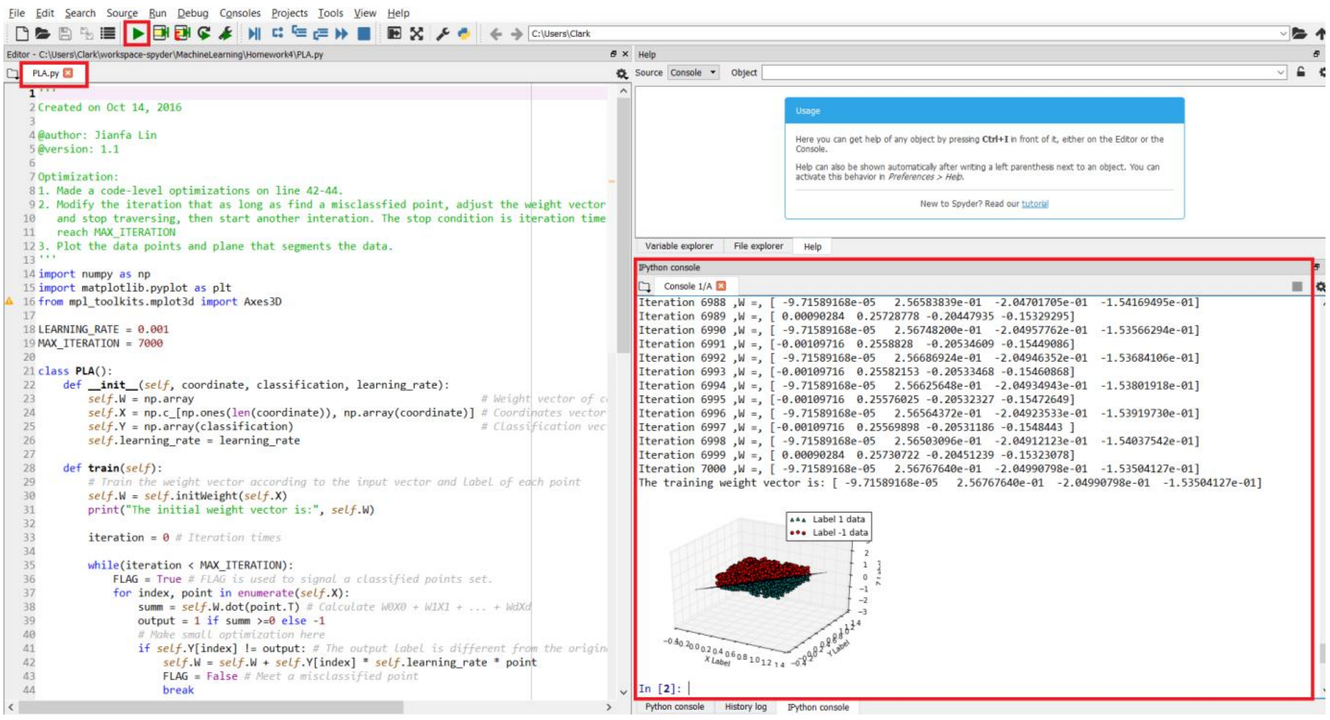
glueviz
0.8.2
Multidimensional data visualization across files. Explore relationships within and among related datasets.
Install

2. In Spyder IDE, please select open file icon on tool bar and open implementation and testing files.


Spyder (Python 3.5)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - C:\works\Clark\USC\Courses\INF552-MachineLearning for Data Informatics\Homework\2\gmm.py
Open file
Homework > 2
Search 2
Organize
New folder
Desktop
Documents
Downloads
Music
Pictures
Videos
Windows (C:)
clusters.txt
gmm.py
File name
ormatics\Homework\2
Supported text files (*.6pl
Open
Cancel

```
index = random.sample(range(N), K)
```

3. Please select the implementation code you want to execute, then click the “Run file” icon. The result will show in IPython console window.


File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Clark
Editor - C:\Users\Clark\workspace-spyder\MachineLearning\Homework4\PLA.py
PLA.py
Variable explorer
File explorer
Help
IPython console
Console 1/A
Label 1 data
Label -1 data

```
'''
Created on Oct 14, 2016

@author: Jianfa Lin
@version: 1.1

Optimization:
1. Made a code-level optimizations on line 42-44.
2. Modify the iteration that as long as find a misclassfied point, adjust the weight vector
   and stop traversing, then start another interation. The stop condition is iteration time
   reach MAX_ITERATION
3. Plot the data points and plane that segments the data.
'''
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

LEARNING_RATE = 0.001
MAX_ITERATION = 7000

class PLA():
    def __init__(self, coordinate, classification, learning_rate):
        self.W = np.array                                                  # Weight vector of c
        self.X = np.c_[np.ones(len(coordinate)), np.array(coordinate)] # Coordinates vector
        self.Y = np.array(classification)                                  # Classification vec
        self.learning_rate = learning_rate

    def train(self):
        # Train the weight vector according to the input vector and label of each point
        self.W = self.initWeight(self.X)
        print("The initial weight vector is:", self.W)

        iteration = 0 # Iteration times

        while(iteration < MAX_ITERATION):
            FLAG = True # FLAG is used to signal a classified points set.
            for index, point in enumerate(self.X):
                summ = self.W.dot(point.T) # Calculate W0X0 + W1X1 + ... + WdXd
                output = 1 if summ >=0 else -1
                # Make small optimization here
                if self.Y[index] != output: # The output label is different from the origin
                    self.W = self.W + self.Y[index] * self.learning_rate * point
                    FLAG = False # Meet a misclassified point
                    break
```

```
Iteration 6988 ,W =, [ -9.71589168e-05   2.56583839e-01  -2.04701705e-01  -1.54169495e-01]
Iteration 6989 ,W =, [ 0.00090284  0.25728778 -0.20447935 -0.15329295]
Iteration 6990 ,W =, [ -9.71589168e-05   2.56748200e-01  -2.04957762e-01  -1.53566294e-01]
Iteration 6991 ,W =, [-0.00109716  0.2558828  -0.20534609 -0.15449086]
Iteration 6992 ,W =, [ -9.71589168e-05   2.56686924e-01  -2.04946352e-01  -1.53684106e-01]
Iteration 6993 ,W =, [-0.00109716  0.25582153 -0.20533468 -0.15460868]
Iteration 6994 ,W =, [ -9.71589168e-05   2.56625648e-01  -2.04934943e-01  -1.53801918e-01]
Iteration 6995 ,W =, [-0.00109716  0.25576025 -0.20532327 -0.15472649]
Iteration 6996 ,W =, [ -9.71589168e-05   2.56564372e-01  -2.04923533e-01  -1.53919730e-01]
Iteration 6997 ,W =, [-0.00109716  0.25569898 -0.20531186 -0.1548443 ]
Iteration 6998 ,W =, [ -9.71589168e-05   2.56503096e-01  -2.04912123e-01  -1.54037542e-01]
Iteration 6999 ,W =, [ 0.00090284  0.25730722 -0.20451239 -0.15323078]
Iteration 7000 ,W =, [ -9.71589168e-05   2.56767640e-01  -2.04990798e-01  -1.53504127e-01]
The training weight vector is: [ -9.71589168e-05   2.56767640e-01  -2.04990798e-01  -1.53504127e-01]

In [2]:
```

6.4 Execution Results

This is the best result we get from our implementation. The accuracy reach around 95%.

Total input numbers= 184
Match(O)=>gestures/A/A_down_1.pgm: Predict=True, Output value=[1.]
Match(O)=>gestures/A/A_down_2.pgm: Predict=True, Output value=[1.]

```
Match(O)=>gestures/A/A_hold_1.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/A/A_hold_10.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/A/A_stop_1.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/A/A_stop_4.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/A/A_up_1.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/A/A_up_10.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/B/B_down_1.pgm: Predict=True, Output value=[ 1.]
Match(O)=>gestures/B/B_down_2.pgm: Predict=True, Output value=[ 1.]
Match(O)=>gestures/B/B_hold_1.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/B/B_hold_2.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/B/B_stop_1.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/B/B_stop_2.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/B/B_up_1.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/B/B_up_4.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/C/C_down_1.pgm: Predict=True, Output value=[ 1.]
Match(O)=>gestures/C/C_down_2.pgm: Predict=True, Output value=[ 1.]
Match(O)=>gestures/C/C_hold_1.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/C/C_hold_2.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/C/C_stop_2.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/C/C_stop_3.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/C/C_up_1.pgm: Predict=False, Output value=[ 0.]
Match(X)=>gestures/D/D_down_1.pgm: Predict=False, Output value=[ 0.]
Match(X)=>gestures/D/D_down_2.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/D/D_hold_1.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/D/D_hold_2.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/D/D_hold_6.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/D/D_stop_1.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/D/D_stop_2.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/D/D_up_1.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/D/D_up_3.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/E/E_down_1.pgm: Predict=True, Output value=[ 1.]
Match(O)=>gestures/E/E_hold_1.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/E/E_hold_5.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/E/E_stop_1.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/E/E_stop_2.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/E/E_up_1.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/E/E_up_2.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/F/F_down_1.pgm: Predict=True, Output value=[ 1.]
Match(O)=>gestures/F/F_down_4.pgm: Predict=True, Output value=[ 1.]
Match(O)=>gestures/F/F_hold_1.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/F/F_hold_2.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/F/F_stop_2.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/F/F_stop_5.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/G/G_down_2.pgm: Predict=True, Output value=[ 1.]
Match(X)=>gestures/G/G_down_3.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/G/G_hold_4.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/G/G_stop_2.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/G/G_stop_5.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/G/G_up_2.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/G/G_up_5.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/H/H_down_2.pgm: Predict=True, Output value=[ 1.]
Match(O)=>gestures/H/H_hold_10.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/H/H_hold_2.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/H/H_hold_5.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/H/H_stop_5.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/H/H_stop_6.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/H/H_up_5.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/I/I_hold_2.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/I/I_hold_5.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/I/I_down_3.pgm: Predict=True, Output value=[ 1.]
Match(O)=>gestures/I/I_stop_5.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/I/I_stop_6.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/I/I_up_2.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/I/I_up_3.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/J/J_down_5.pgm: Predict=True, Output value=[ 1.]
Match(X)=>gestures/J/J_down_6.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/J/J_hold_2.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/J/J_hold_3.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/J/J_stop_7.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/J/J_stop_8.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/J/J_up_1.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/J/J_up_2.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/K/K_down_2.pgm: Predict=True, Output value=[ 1.]
Match(O)=>gestures/K/K_down_3.pgm: Predict=True, Output value=[ 1.]
Match(O)=>gestures/K/K_hold_1.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/K/K_hold_2.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/K/K_hold_3.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/K/K_stop_1.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/K/K_stop_2.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/K/K_stop_1.pgm: Predict=False, Output value=[ 0.]
Match(O)=>gestures/K/K_stop_2.pgm: Predict=False, Output value=[ 0.]
Accuracy: correct rate: 95.18072289156626%
```

7 Software Familiarization

7.1 Overview

As previous describe, we find a good data science platform, Anaconda, which includes major open source of Python libraries for data science. The latest version as of Oct., 1, 2016 is 4.2.0. The platform includes IDE (Integrated Development Environment), Python open source library management for developer to build up their own code.

Under this environment, a clustering testing code implements base on Numpy library, scikit-learn library, and matplotlib library as a comparison to our implementations.

7.2 Perceptron algorithm in scikit-learn library

7.2.1 API introduction[4]

In the linear model, a perceptron algorithm is available for use. *Perceptron* and *SGDClassifier* share the same underlying implementation. In fact, *Perceptron()* is equivalent to *SGDClassifier(loss="perceptron", eta0=1, learning_rate="constant", penalty=None)*.

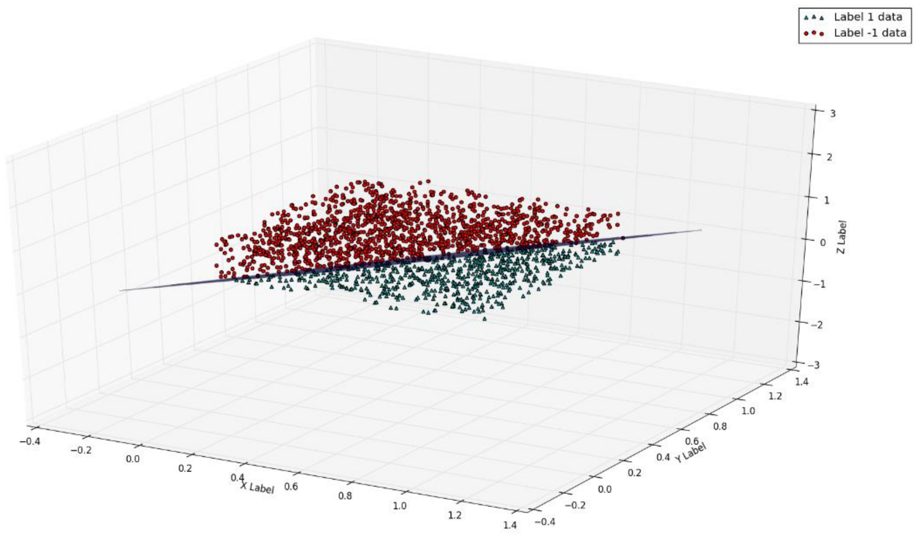
There are several interesting parameters can be assigned into this implementation.

1. "alpha": learning rate, default is 0.0001
2. "n_iter": maximum iteration number can be restricted. Default is 5.
3. "class_weight": the parameter provides user a chance to provide initial weight vector for your data. Default is weight one for all classes.
4. "warm_start": When set to True, reuse the solution of the previous call to fit as initialization. Otherwise, just erase the previous solution.
5. "score()": Returns the mean accuracy on the given test data and labels.

7.2.2 Execution results

The best result after 10 trials as below, the result is the same as the best results of our implementation:

```
{'warm_start': False, 'random_state': 0, 'shuffle': True, 'verbose': 0, 'n_jobs': 1,
'eta0': 1.0, 'penalty': None, 'fit_intercept': True, 'class_weight': None, 'n_iter': 200,
'alpha': 0.0001}
score = 1.0
W= [[  0.          54.64874766 -43.75947362 -32.77355205]]
```



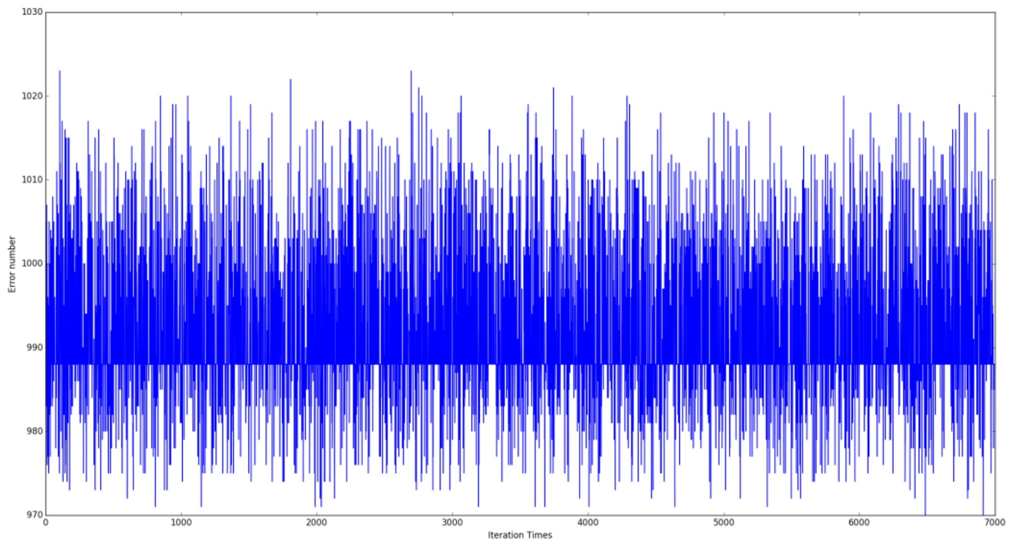
7.3 Pocket algorithm in scikit-learn library

7.3.1 introduction[4]

Due to there is no Pocket algorithm implementation in scikit-learn library. We leverage Perceptron implementation and set the warm_start as True and the n_iter = 1 for 7000 times to calculate error number via score and total data number.

The result as below for reference:

```
{'warm_start': True, 'random_state': 0, 'shuffle': True, 'verbose': 0, 'n_jobs': 1,
'eta0': 1.0, 'penalty': None, 'fit_intercept': True, 'class_weight': None, 'n_iter': 1,
'alpha': 0.0001}
score = 0.515
W= [[-1.         -0.64196622 -0.73630825 -0.63632897]]
```



7.4 Logistic Regression Algorithm in scikit-learn library

7.4.1 API Introduction[5]

There are several interesting parameters can be assigned into this implementation.

1. “tol”: Tolerance for stopping criteria, Default is 1e-4.
2. “solver”: {‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’}, different algorithm to use in the optimization problem. For small dataset, ‘liblinear’ is good choice, whereas ‘sag’ is faster for large ones. For multiclass problems, only ‘newton-cg’, ’sag’, and ‘lbfgs’ handle. Default: ‘liblinear’ is limited to one-versus-rest schemes.
3. “max_iter”: maximum iteration number can be restricted. Default is 100.
4. “warm_start”: If warm_start is true; the solution of the last fitting is used as initialization for the next call. It will speed up convergence when developer repeat called several time on similar problems. The default value is False.
5. “n_init”: the number of trials will perform. The best results are kept. Default value is 1.

7.4.2 Execution results

```
{'fit_intercept': True, 'max_iter': 100, 'solver': 'liblinear', 'C': 1.0, 'n_jobs': 1,
'dual': False, 'verbose': 0, 'penalty': 'l2', 'tol': 0.0001, 'class_weight': None,
'warm_start': False, 'intercept_scaling': 1, 'multi_class': 'ovr', 'random_state': None}
score = 0.5295
W= [[-0.01550526 -0.17376308  0.11159028  0.07474653]]
```

7.5 Linear Regression Algorithm in scikit-learn library

7.5.1 API Introduction[6]

There are several interesting parameters can be assigned into this implementation. But there is no weight vector can be printed by the API, instead, a predict(X) function provides for results prediction and plot the diagram.

6. “fit_intercept”: Boolean, optional. Where to calculate the intercept for this model.
7. “normalize”: Boolean, optional, default is False. If it is True, the regressors X will be normalized before regression. This parameter is ignored when the fit_intercept is set to Faluse.
8. “coef_”: Estimated coefficients for the linear regression problem.
9. “fit(X, y, sample_weight=None)”: X is training data, y is target values, sample_weight can provide by user or not.
10. “predict(X)”: Predict using the linear model
11. “score()”: Returns the coefficient of determination R^2 of the prediction.

7.6 How to improve our implementation

7.6.1 Improvement opportunities on Perceptron Learning and Pocket Algorithms

1. Improve the random point selection. We can improve our code to automatic generate different weight vectors and learning rate to get the best result in different variables combination.

7.6.2 Improvement opportunities on Logistic Regression Algorithm

1. We may implement the coverage function which taught in class and do a comparison between the two different coverage functions and the results.
2. Add an option to user to choose different functions.

7.6.3 Improvement opportunities on Linear Regression Algorithm

3. We may implement the coverage function for singular matrix data input to make the solution more complete.

8 Application

8.1 The applications on Perceptron Learning and Pocket Algorithm.

Except on traditional binary data classification, like credit card application question. Perceptron learning generalizes naturally to multiclass classification. In recent years, perceptron training has become popular in the field of natural language processing for such tasks as text categorization [7].

8.2 The application on Logistic Regression Algorithm.

Logistic regression algorithm is widely applied in business, like measure the success rates of marketing campaigns. An interesting application is in GIS-based application. Especially in regional hazard management, multivariate statistical analysis in the form of logistic regression was used to produce a landslide susceptibility map in specific areas[8], sometimes frequency ratio model is used to compare with logistic regression algorithm [9].

8.3 The applications on Linear Regression Algorithm.

Linear regression algorithm is an important numerical method for linear least squares problem because it is one of the most important types of model in statistics. A lot of business questions like “evaluating trends and sales estimation” or “analysing the impact of price changes and sales”. Biometrical procedure for testing the equality of measurements [10], researcher will use linear regression procedure as a method comparison studies in clinical chemistry.

9 Reference

[1] Alpaydin E. Introduction to machine learning[M]. MIT press, 2014.

[2] Andrew Ng, Machine Learning Course Materials. URL http://cs229.stanford.edu/notes/cs229-notes1.pdf, page 11~13.

[3] Chih-Hao's Blog, URL http://zhihaozhang.github.io/2016/03/17/logisticRegression/

[4] scikit-learn, version 0.18, Perceptron, URL http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html

[5] scikit-learn, version 0.18, Logistic Regression, URL http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[6] scikit-learn, version 0.18, Linear Regression, URL http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

[7] Ng, Hwee Tou, Wei Boon Goh, and Kok Leong Low. "Feature selection, perceptron learning, and a usability case study for text categorization." *ACM SIGIR Forum*. Vol. 31. No. SI. ACM, 1997.

[8] Ayalew, Lulseged, and Hiromitsu Yamagishi. "The application of GIS-based logistic regression for landslide susceptibility mapping in the Kakuda-Yahiko Mountains, Central Japan." *Geomorphology* 65.1 (2005): 15-31.

[9] Lee, Saro, and Biswajeet Pradhan. "Landslide hazard mapping at Selangor, Malaysia using frequency ratio and logistic regression models." *Landslides* 4.1 (2007): 33-41.

[10] Passing, H., and W. Bablok. "A new biometrical procedure for testing the equality of measurements from two different analytical methods. Application of linear regression procedures for method comparison studies in clinical chemistry, Part I." *Clinical Chemistry and Laboratory Medicine* 21.11 (1983): 709-720.