INF 552
Sudeeptha Mouni Ganji (ID: 2942771049)
Vanessa Tan (ID: 4233243951)

Programming Assignment 3 Report

PART 1: IMPLEMENTATION

Perceptron Learning Algorithm:

Data Structures:

- **Pandas dataframe:** I used the read_csv() function to originally read in the input file into a dataframe. I put the first three columns into one dataframe to hold the coordinates of all the data points. I put the fourth column into its own dataframe to hold all the labels of the data points.
- **Numpy array:** I converted both of the dataframes into numpy arrays for easier manipulation and performing mathematical functions.

Code-level optimizations:

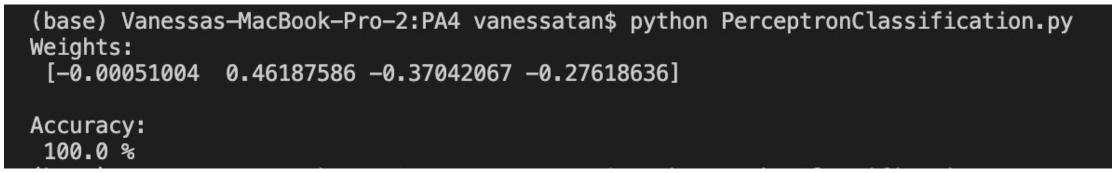
- I utilized a very small learning rate value of 0.0001. With a larger learning rate value like 0.01 or 0.001, it sometimes took longer for the number of violated constraints to reach zero. The weight vector gets updated by the learning rate, current weight vector values, and data vector values. The larger the learning rate, the more the weight vector values fluctuate.
- I used numpy arrays instead of lists or other data types in order to improve the efficiency of the program. Most of the data manipulations and calculations were made easier and quicker because of this. I was able to quickly calculate the dot product between the data vectors and weight vector.

Challenges:

- It was difficult to find the right learning rate to use with the perceptron learning algorithm. I figured out through trial and error that using the value of 0.0001 allowed my program to get rid of the violated constraints in the fastest way in most cases. Since the weight vector values are initialized randomly, sometimes using a smaller learning rate takes longer as well.

Output:

- The weights after the final iteration of the perceptron learning algorithm are:
  [-0.00051004, 0.46187586, -0.37042067, -0.27618636]
- The accuracy was 100%. All of the outputs were correctly classified after the weights had been updated.



```
(base) Vanessas-MacBook-Pro-2:PA4 vanessatan$ python PerceptronClassification.py
Weights:
 [-0.00051004  0.46187586 -0.37042067 -0.27618636]

Accuracy:
 100.0 %
```

Pocket Algorithm:

Data Structures:

- **Pandas dataframe:** I used the read_csv() function to originally read in the input file into a dataframe. I put the first three columns into one dataframe to hold the coordinates of all the data points. I put the fifth column into its own dataframe to hold all the labels of the data points.
- **Numpy array:** I converted both of the dataframes into numpy arrays for easier manipulation and performing mathematical functions.

Code-level optimizations:

- By updating the minimum number of misclassifications, we can keep track of the best weight vector we see as we iterate through the pocket algorithm.

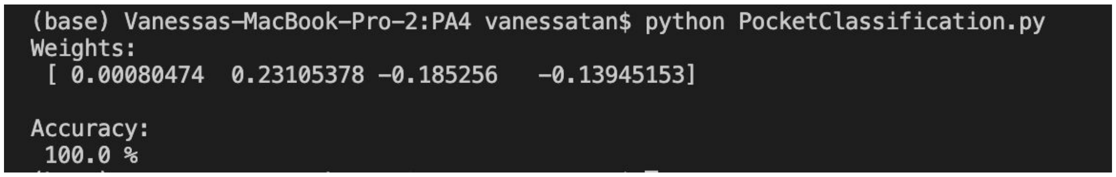
- I utilized a very small learning rate value of 0.0001. With a larger learning rate value like 0.01 or 0.001, it sometimes took longer for the number of violated constraints to reach zero. The weight vector gets updated by the learning rate, current weight vector values, and data vector values. The larger the learning rate, the more the weight vector values fluctuate.
- I used numpy arrays instead of lists or other data types in order to improve the efficiency of the program. Most of the data manipulations and calculations were made easier and quicker because of this. I was able to quickly calculate the dot product between the data vectors and weight vector.

Challenges faced:

- It was difficult to find the right learning rate to use with the perceptron learning algorithm. I figured out through trial and error that using the value of 0.0001 allowed my program to get rid of the violated constraints in the fastest way in most cases. Since the weight vector values are initialized randomly, sometimes using a smaller learning rate takes longer as well.
- When using a learning rate of 0.01 or 0.001, the pocket algorithm usually had to run through all 7000 iterations so it took a while to get the output. There was also less accuracy.

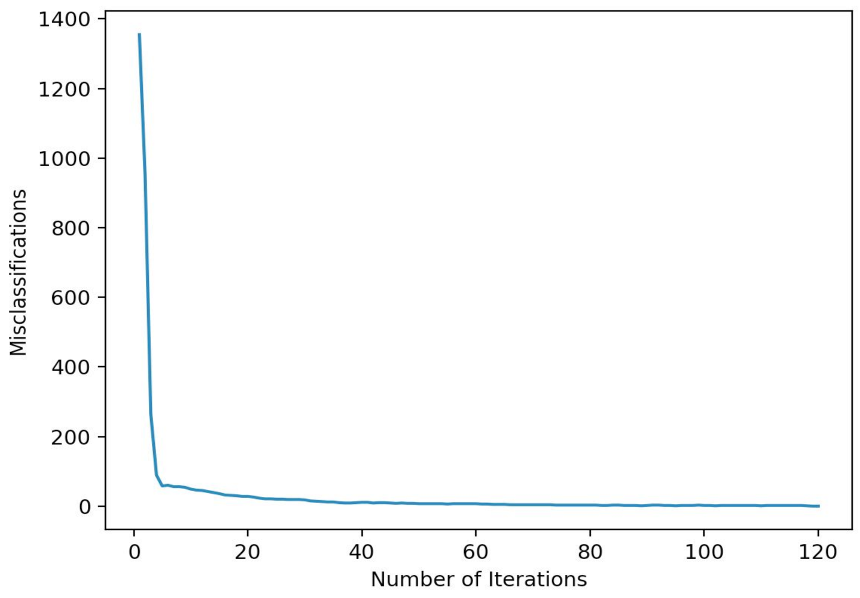
Output:

- The weights after the final iteration of the perceptron learning algorithm are: [-0.00051004, 0.46187586, -0.37042067, -0.27618636]
- The accuracy was 100%. All of the outputs were correctly classified after the weights had been updated.



```
(base) Vanessas-MacBook-Pro-2:PA4 vanessatan$ python PocketClassification.py
Weights:
 [ 0.00080474  0.23105378 -0.185256   -0.13945153]

Accuracy:
 100.0 %
```



INF 552
Sudeeptha Mouni Ganji (ID: 2942771049)
Vanessa Tan (ID: 4233243951)

Linear Regression Algorithm:

Data Structures:

- **Pandas csv and dataframe:** I have used read_csv function from pandas in order to read the input data into a dataframe.
- **Numpy arrays:** I have extensively used numpy arrays throughout the program as they let us manipulate the data easily. I have also used functions from numpy in order to perform data manipulations like finding transpose, inverse and dot product during the calculation of weights.

Code-level optimizations:

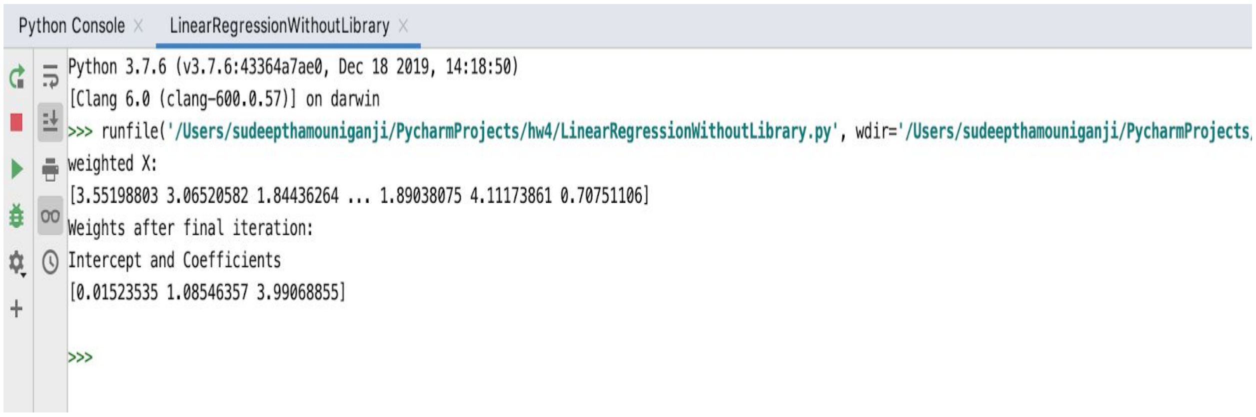
- I have used numpy arrays instead of lists or other data types in order to improve the efficiency of the program. Most of the data manipulations and calculations were made easier and quicker because of this.
- I have padded the initial input with ones in the first column to provide generality on the function for ease of computation.
- I have used genfromtxt function from numpy to read the input. This gives the input in a mathematical format and helps to achieve a greater output accuracy.

Challenges faced:

- Initially, the program calculations were going wrong. But after I introduced X0=1 which padded the first column of D with ones I was able to get better results.
- However the results weren't so accurate and exactly the same as the ones obtained in the library program. After I used the genfromtxt function to read the data rather than directly reading the data to a dataframe, I was able to obtain accurate results.

Output:

- The weights after the final iteration obtained in the program without libraries are as follows: [0.01523535 1.08546357 3.99068855]
  My function shows the intercept and coefficients respectively in a single array.



```
Python 3.7.6 (v3.7.6:43364a7ae0, Dec 18 2019, 14:18:50)
[Clang 6.0 (clang-600.0.57)] on darwin
>>> runfile('/Users/sudeepthamouniganji/PycharmProjects/hw4/LinearRegressionWithoutLibrary.py', wdir='/Users/sudeepthamouniganji/PycharmProjects,
weighted X:
[3.55198803 3.06520582 1.84436264 ... 1.89038075 4.11173861 0.70751106]
Weights after final iteration:
Intercept and Coefficients
[0.01523535 1.08546357 3.99068855]


>>>
```

Logistic Regression Algorithm:

Data Structures:

- **Pandas csv and dataframe:** I have used read_csv function from pandas in order to read the input data into a dataframe.
- **Numpy arrays:** I have converted the above obtained dataframe into a numpy array for ease of usage such as extracting and manipulating data. I have also intialized the weights and gradients

using numpy random functions and zeros respectively which gives numpy arrays with the specified values.

Code-level optimizations:

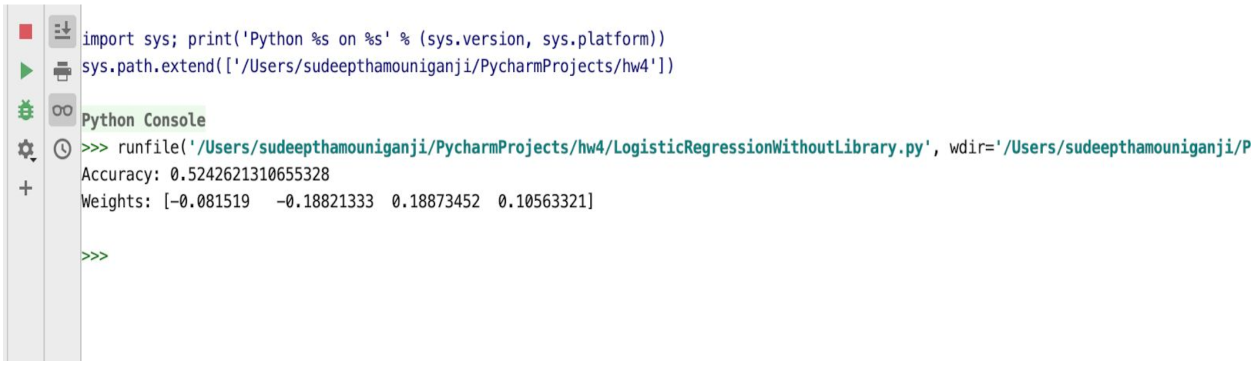
- I have used numpy arrays instead of dataframes as they provide for easy manipulation of data.
- I have used in-built functions wherever needed instead of assigning values or doing unnecessary calculations. This helped to improve the speed and accuracy of the program.
- After a lot of trial and error, I have decided on a learning rate of 0.01 while calculating the weights for the algorithm. This helped to achieve an accuracy of 0.524 in my implementation which is almost similar to the library implementation which gave an accuracy of 0.529.

Challenges faced:

- Initially I have tried to use a mathematical notation of the numbers in order to get higher accuracy levels. However, this gave me less accuracy than the normal notation, so I reverted back to the normal notation by using pandas read_csv function.
- Deciding on the learning rate was a bit difficult and required a lot of trial and error.
- I found a bit of difficulty in mapping the algorithm to the code especially with gradients.

Output:

- The output obtained by implementation is as follows:



```
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['/Users/sudeepthamouniganji/PycharmProjects/hw4'])

Python Console
>>> runfile('/Users/sudeepthamouniganji/PycharmProjects/hw4/LogisticRegressionWithoutLibrary.py', wdir='/Users/sudeepthamouniganji/P
Accuracy: 0.5242621310655328
Weights: [-0.081519   -0.18821333  0.18873452  0.10563321]

>>>
```

The accuracy obtained was similar to the accuracy obtained in the library implementation. However the weights obtained were sometimes similar and sometimes different. This is because the weights are initialized randomly in the beginning.

PART 2: SOFTWARE FAMILIARIZATION

<u>Linear Classification Algorithm</u>

Libraries

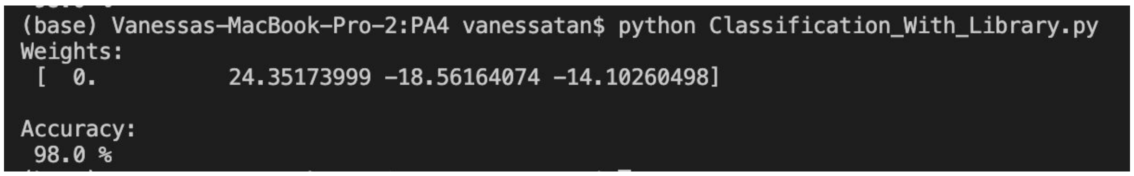
- **Sklearn.linear_model:** I utilized the Perceptron class from the linear models provided by sklearn. It fits the model given the training data and labels, and is able to provide the prediction outputs.
- **Pandas dataframe:** I used the read_csv() function to originally read in the input file into a dataframe. I put the first three columns into one dataframe to hold the coordinates of all the data points. I put the fourth column into its own dataframe to hold all the labels of the data points.
- **Numpy array:** I converted both of the dataframes into numpy arrays for easier manipulation and performing mathematical functions.

Implementation Comparison

- I read in the classification.txt file and created numpy arrays in the same way that I did for the perceptron algorithm implementation without a library. I initialized the Perceptron class from sklearn by passing in the max iterations to be 7000 and I left the remaining default parameters as is since it used a default learning rate of 0.0001 like how I did in my implementations.
- I just had to call the fit() function with the training data and labels and use predict() to get the predicted outputs.

Result Comparison

- The weights after the final iteration of the perceptron learning algorithm are:
  [0, 24.35173999,-18.56164074, -14.10260498]
- The accuracy was 98%.



```
(base) Vanessas-MacBook-Pro-2:PA4 vanessatan$ python Classification_With_Library.py
Weights:
 [  0.          24.35173999 -18.56164074 -14.10260498]

Accuracy:
 98.0 %
```

Possible Improvements

- We could also plot the number of misclassifications against the number of iterations for the Perceptron algorithm like we did for the Pocket algorithm so that we could see how the gradient descent gradually moves to a local minima.

<u>**Linear Regression Algorithm**</u>

Libraries

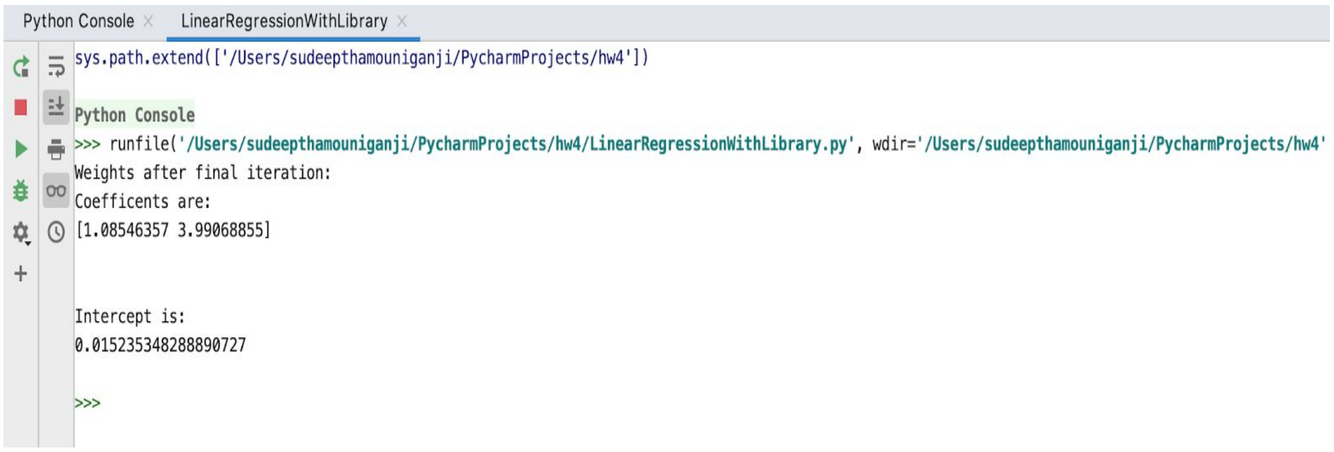
- **Numpy:** I have used the in-built function in numpy to read the input data. I have also performed data manipulations like converting to arrays and arranging values in a specific range by using the arrange function and converting the data into vectorized coordinates using meshgrid function for use in plots.
- **Matplotlib.pyplot:** I have used matplotlib library to plot the output for the libraries' implementation by using pyplot module.
- **Sklearn.linear_model:** The sklearn.linear_model provides the implementation for a variety of linear models. I have used the LinearRegression class from this module in order to implement the program. This class implements the ordinary least squares linear regression algorithm which minimizes the residual sum of squares between observed targets and computed targets.

Implementation Comparison

- The in-built Linear Regression library function provides a quick and efficient way to implement ordinary linear regression. Using it, we can also find out the coefficients, intercept etc. as well as performing prediction.
- In my implementation, I have performed various calculations and data manipulations such as finding transpose, inverse and dot product etc in order to calculate the weights and find weighted inputs. Though this process involved extensive calculations, the outputs obtained were similar.

Result Comparison

- The results obtained in both implementations were similar.
- The library implementation output is as follows:



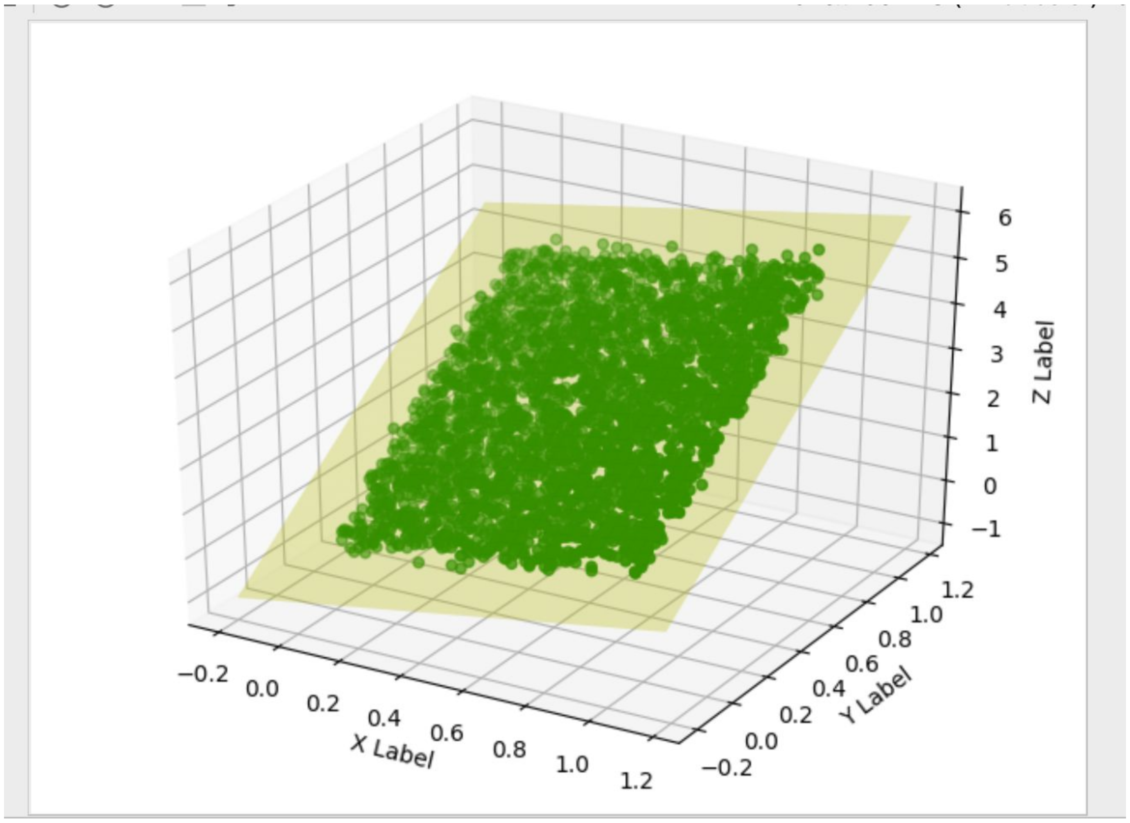
```
sys.path.extend(['/Users/sudeepthamouniganji/PycharmProjects/hw4'])

Python Console
>>> runfile('/Users/sudeepthamouniganji/PycharmProjects/hw4/LinearRegressionWithLibrary.py', wdir='/Users/sudeepthamouniganji/PycharmProjects/hw4'
Weights after final iteration:
Coefficents are:
[1.08546357 3.99068855]


Intercept is:
0.015235348288890727

>>>
```

- The plot obtained is:



Possible Improvements

- Having a direct way of plotting multidimensional data would be a very useful addition as it would allow us to view higher dimensional data and their curves much easily.
- If we could give a more defined output like probability of a value occurring or a short range of values for prediction output rather than a continuous value, it might be more useful.

Logistic Regression Algorithm

Libraries

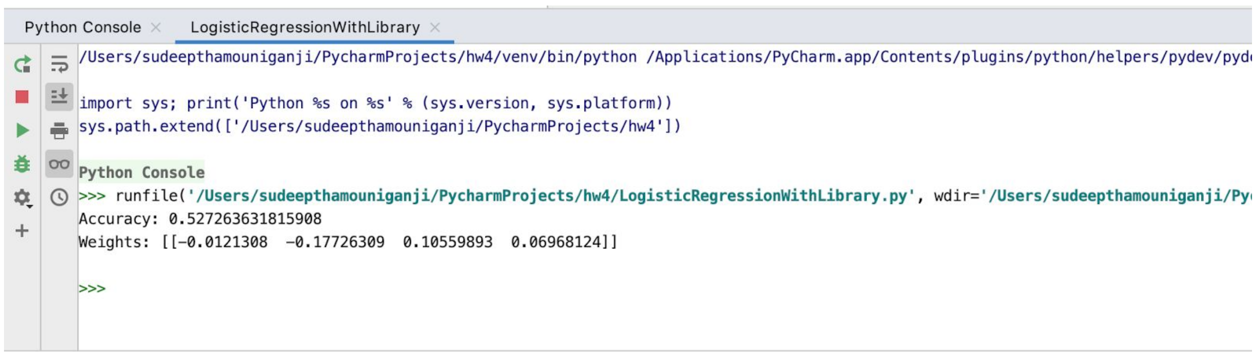
- **Numpy:** I have used numpy library to access its array data structure and also for using its various mathematical functions.
- **Pandas:** I have used Pandas for reading the input as a csv file to convert into a dataframe object by using read_csv function. I further converted that dataframe into numpy arrays
- **sklearn.linear_model:** I have used the sklearn.linear_model library function in order to access its in-built LogisticRegression class. This class can be used to run the logistic regression algorithm on any particular data which needs to be fitted to it. It can be used to calculate the weights, accuracy etc as well as perform prediction.

Implementation Comparison

- The implementation using libraries was much easier and quicker than doing it without libraries.
- In the libraries implementation, I read the input data using pandas read_csv and added the first column using ones before fitting the whole data to the algorithm.
- In the implementation without using libraries, I read and performed various data manipulations and mathematical calculations to find the weights and accuracies.
- As we specified the number of iterations to be done as 7000, the time taken was higher when compared to the library implementation.

Result Comparison

- The results obtained by using the library function is as follows:



```
/Users/sudeepthamouniganji/PycharmProjects/hw4/venv/bin/python /Applications/PyCharm.app/Contents/plugins/python/helpers/pydev/pyd

import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['/Users/sudeepthamouniganji/PycharmProjects/hw4'])

Python Console
>>> runfile('/Users/sudeepthamouniganji/PycharmProjects/hw4/LogisticRegressionWithLibrary.py', wdir='/Users/sudeepthamouniganji/Py
Accuracy: 0.527263631815908
Weights: [[-0.0121308  -0.17726309  0.10559893  0.06968124]]

>>>
```

- The results obtained in both the implementations were quite similar in both the cases sometimes where both the weights and accuracy were approximately equal. Sometimes they were different. This was because the weights are assigned randomly in the implementation.

Possible Improvements

- The library function allows us to vary the way algorithm is implemented i.e via incrementally trained logistic regression or through built-in cross validation. I think this would be a good addition to my program.
- The library implementation is faster than mine. I think improving the speed of the program without reducing the number of iterations would be good.

PART 3: APPLICATIONS

<u>Linear Classification Algorithm</u>

- Linear classification can be used to determine whether or not an applicant should be given a line of credit with a bank. Each applicant has various factors that determine their eligibility like age, salary, credit score, etc. These different factors can be weighted to help determine if the applicant is eligible to open a credit card or not with the bank.
- Mail sites also use linear classification algorithms to determine whether an email is spam or not. Raw data from emails are processed into vectors and the relative frequencies of common words and punctuation are calculated. They are compared against a set of 57 common words and punctuation that are used to help determine if an email is spam or not. This set is created from a training sample of 4061 emails that have already been classified as spam or not.

Linear Regression Algorithm

- Car insurance companies can use linear regression to create a premium table that uses predicted claims to get an insured declaration value ratio. The risk is determined by different factors about the driver and car like the driver's age, car model year, mileage, etc.
- Linear regression can also be used to measure the effectiveness of product marketing, pricing, and sales promotions. Companies can check if investing in a certain marketing tactic has given them substantial return on that investment. It enables companies to measure the effectiveness of isolated factors that impact different marketing strategies as well.

Logistic Regression Algorithm

- Logistic Regression is used by stores to decide their opening and closing hours. For example, a store manager intuitively decides to keep a store open at night and makes a decision to hire staff for the nigh shifts as well. But by using logistic regression, we can predict the probability of getting profits when compared to the costs of maintaining the staff and store at night. Thus logistic regression helps in decision making and correcting errors of intuitive judgement.
- Logistic regression can also be used in geographic information systems for disease-mapping a particular area. Here, logistic regression is used as a tool to identify and analyse the relative likelihood and its socio-environmental determinants of a disease. Satellite imagery is used to map the spatial patterns of land use and cover while the model is generated by multiple logistic regression in which environmental factors were considered as independent variables while the number of cases above a threshold becomes the dependent variable.

Group Members and Contributions

- Our group members for this programming assignment were Sudeeptha Mouni Ganji (ID: 2942771049) and Vanessa Tan (ID: 4233243951).
- Vanessa and Sudeeptha worked together to research and understand the algorithms.
- After implementation, Sudeeptha and Vanessa worked together to write the report.

Programming:

Classification algorithms ● Without libraries ● With libraries	 Vanessa Tan Vanessa Tan
Linear and Logistic regression algorithms	

● Without libraries ● With libraries	Sudeeptha Mouni Ganji Sudeeptha Mouni Ganji

Report Writing:

Part 1: ● Perceptron Learning algorithm ● Pocket algorithm ● Linear Regression algorithm ● Logistic Regression algorithm	 Vanessa Tan Vanessa Tan Sudeeptha Mouni Ganji Sudeeptha Mouni Ganji
Part 2: ● Linear Classification algorithm ● Linear Regression algorithm ● Logistic Regression algorithm	 Vanessa Tan Sudeeptha Mouni Ganji Sudeeptha Mouni Ganji
Part 3: ● Linear Classification algorithm ● Linear Regression algorithm ● Logistic Regression algorithm	 Vanessa Tan Vanessa Tan Sudeeptha Mouni Ganji