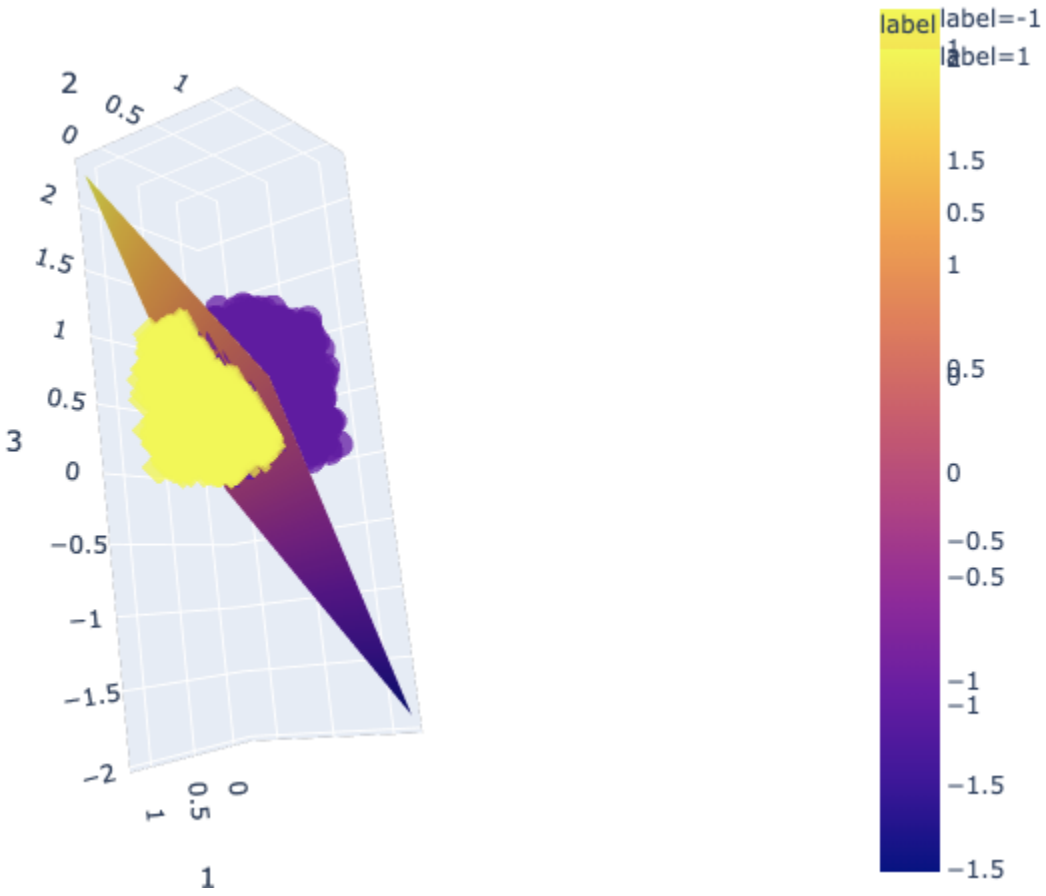
Lisa Meng
Shagun Gupta
INF 522
Homework 4

REPORT

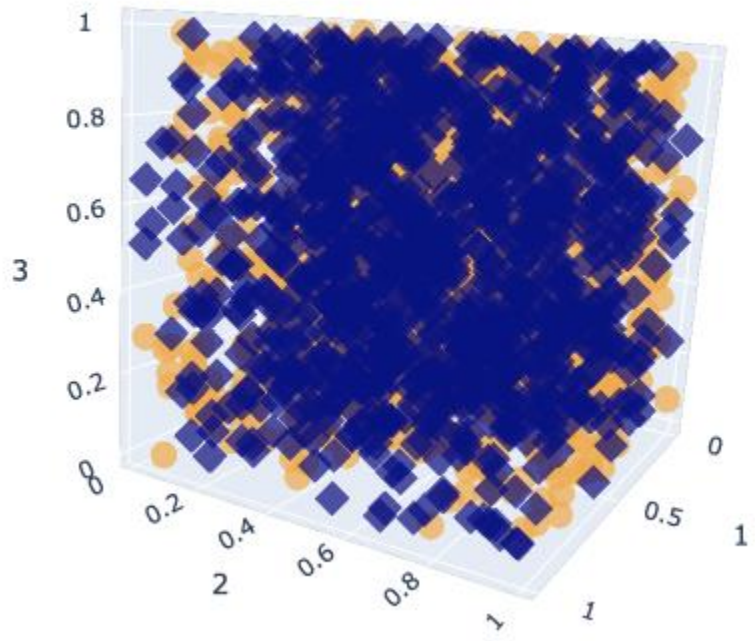
Part 1: Implementation

1. Perceptron Learning: The data structures used to implement this algorithm were mainly pandas dataframe and lists. Functions were defined to optimise the code and separate different parts of the code. Initially w is assigned randomly and changed till convergence occurs, that is no more constraints left. Due to the simplicity of the algorithm no actual challenges were faced.
   Accuracy: 99.6%
   Final w: `[0.07084665141189883,22.560361091559837,-18.105849962817768, -13.593466418008965]`



2. Pocket Algorithm: The data structures used to implement this algorithm were lists and pandas dataframe. Functions were assigned to optimise the code and separate the different parts of the code. The main challenge faced was how to tweak the Perceptron

Learning Algorithm to implement this Algorithm while simultaneously recording every iteration's in-sample Error. This was finally implemented by defining a list that records the number of in-sample Errors at every iteration. The functions were modified to calculate the total number of constraints in every iteration instead of just returning the first randomly found constraint. The final w is chosen on the basis of minimum number of constraints. The data is not linearly separable.



Accuracy: 50.6%

Final w: `[-0.5666243508093995,-0.1849718134444034,0.03844359022092908, -0.29647423527524214]`

3. Logistic Regression: Gradient Descent was used to implement logistic regression. Pandas was used to store the data and numpy was used to implement matrix multiplication. Random library was used to generate a random initial w. Lists were also used for the implementation of the algorithm. 7000 iterations were run. The only challenge faced was deciding on a value of alpha- the learning rate. Multiple values were used to finalise the value giving the best accuracy. All learning rates gave the same result.
    a. When learning rate = 0.5

    Accuracy: `52.95%`

    Final w: `[-0.031500751673216275, -0.17769619354271532, 0.11445234509554042,0.07670125569483043]`

b. When learning rate = 0.8
Accuracy: `52.95%`
Final w: `[-0.031500751673216275, -0.17769619354271532, 0.11445234509554042,0.07670125569483043]`

c. When learning rate = 0.3
Accuracy: `52.95%`
Final w: `[-0.031500751673216275, -0.17769619354271532, 0.11445234509554042,0.07670125569483043]`

Since the data is not linearly separable, the accuracy is similar as the one achieved by the pocket algorithm.

4. Linear Regression: Although Pandas DataFrame was used to import the text file, NumPy array was the data structure used to manipulate the data and calculate the weights. Functions were defined to optimise and separate different parts of the code. Initially, we calculated the least-square solution to the linear system of equations by matrix factorization by using NumPy to determine the weights, which returned the results below:

```
intercept: 0.015235348288884565
weights: [1.08546357 3.99068855]
```

Due to the simplicity of this algorithm, no actual challenges were faced.

Then out of curiosity, we decided to calculate the weights by using Gradient Descent, which returned the results below:

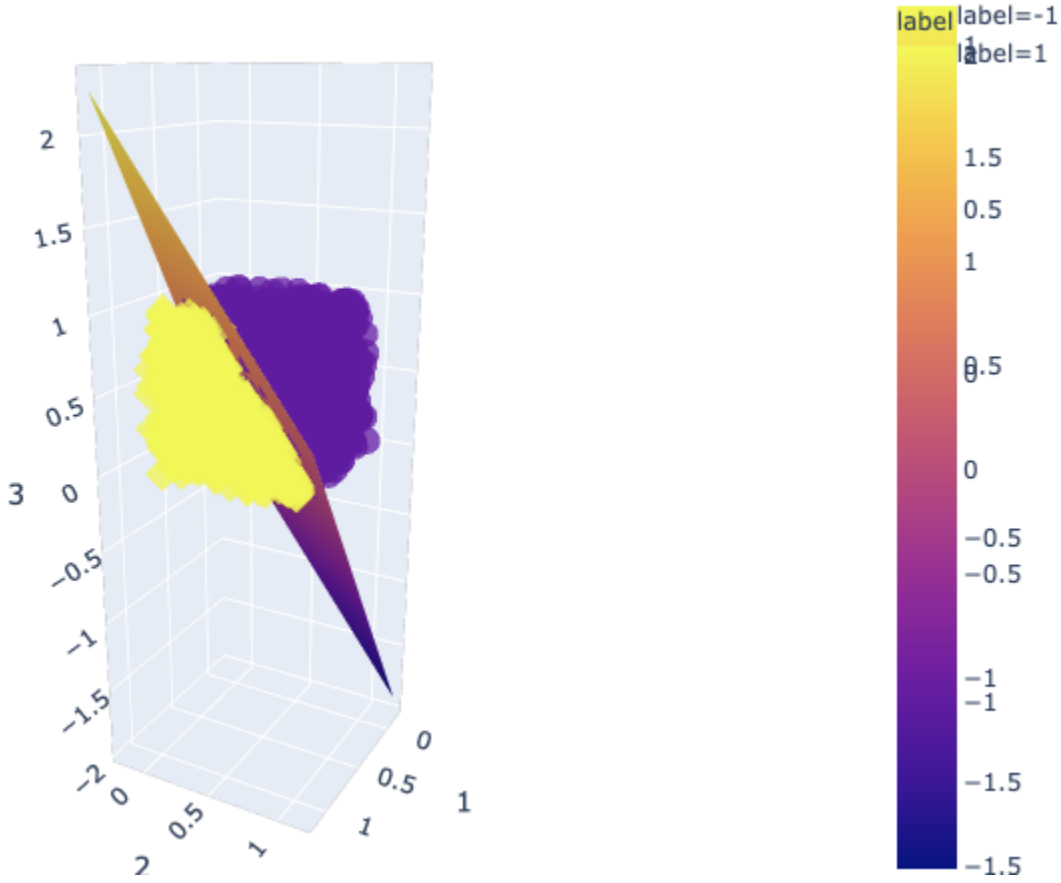
```
intercept: 0.01523534828890486
weights: [1.08546357 3.99068855]
```

As you can see, the results are identical. But with the Gradient Descent method, we did face a challenge: determining the best alpha value. Because alpha, the learning rate, is a hyper-parameter that is external to the model, we do not know the best value and cannot estimate it from the dataset. Therefore, we searched for the most sufficient value by trial and error.

Part 2: Software Familiarization

1. Linear Classification
We used the sklearn.linear_models.Perceptron model to apply to the first 4 columns of the classification.txt file (the same as in Part 1).
The parameters that made a change in the accuracy was a number of iterations to wait in case of convergence = 30 (unclear why this made a change of 1.3% in the accuracy as the model clearly converged before 60 iterations were completed). **The accuracy achieved was 100% while w produced was:** `array([[ 0. , 33.15648072, -26.55520294, -19.88226152]])` which is different from the result received in part 1. It clearly states that the data is linearly separable. The difference in implementation exists in the weight values initially. The library sets all weights as 1 initially unlike our

implementation which takes random weights. Although this can be tuned, we did not experiment with this part. Also the library only took 57 iterations to reach convergence while our model took more than 1000 iterations. The difference exists in how many constraints are solved in 1 iteration.



2. Linear Regression

Upon investigation, we decided sklearn.linear_model.LinearRegression() had the best implementation for the linear regression algorithm. When calculating the weights by solving a system of linear equations (using NumPy), we obtained the same results as the sklearn library with a few lines of code. Thus, we believe there is no further improvement necessary.

But when solving for the weights using Gradient Descent, a key difference between our code from scratch and that of using the sklearn library, the library uses Ordinary Least Squares. Although the sklearn library uses a different statistical method, our code from scratch using Gradient Descent achieved the same results as the library. Because our code using the Gradient Descent method calculates the changing values of the weights (`w = w - alpha * gradient`), we need to take into account alpha, the learning rate. And because alpha is a hyper-parameter, it is not possible to know the best value for the model by looking at the data. Thus, it needs to be

externally/manually set based on trial and error. As explained above, we decided to use 0.1 as the value for alpha because it showed the least difference when it was being adjusted. Therefore, to improve our code, we can slow down the learning rate and increase the iteration to avoid underfitting/overfitting. In more general words, it would be beneficial to determine a balance between the alpha and iteration values or a way to calculate the best values to improve our accuracy.

Below are the results from the sklearn library:

```
weights: [1.08546357 3.99068855]
intercept: 0.015235348288884953
R^2: 0.9722515684162281
```

Note that the weights and intercept are the same as that of Part 1 (for both the Least Squares and Gradient Descent methods).

3. Logistic Regression:

   Upon investigation, `sklearn.linear_model.LogisticRegression` was used to implement the model. The library uses the logistic function to implement this algorithm. It offers multiple solvers to implement the logic. Unlike part 1 where we used simple gradient descent, this library offers multiple solvers like newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'. Upon experimentation we derived the following results:

   a. Solver: lbfgs (uses gradient)
      Weights: `[[-0.17357511,  0.1116996 ,  0.07491538, -0.01487118]]`
      Accuracy: `52.949999999999996%`

   b. Solver: liblinear (linear classification that supports logistic as well)
      Weights: `array([[-0.17376308,  0.11159028,  0.07474653, -0.01550526]])`
      Accuracy: `52.949999999999996%`

   c. Solver: saga (uses stochastic gradient descent)
      Weights: `array([[-1.73580337e-01,  1.11770196e-01, 7.49317605e-02, -1.06053833e-04]])`
      Accuracy: `52.949999999999996%`

   Multiple solvers return the same accuracy even though there is a difference in the weights which indicates the difference in the calculation process of each of these solvers. This accuracy is the same as the one we achieved in Part 1 indicating that no real improvement is needed in the optimisation of the code as the data is not linearly separable except that our code does not converge as fast as the library does and can be changed by tuning the parameters better.

Part 3: Applications

1. Linear Classification
    a. Many surveys and data collecting organisations use linear classification to divide, organise and observe data trends, differences and impact.
    b. Spam Detection
    c. Serves as a building block for multiple neural networks and deep learning algorithms
2. Linear Regression:
   Linear Regression refers to a model that can show a relationship between two variables and how one can impact the other. It is an extremely popular model applied in many industries:
    a. Linear regressions can be used in business to evaluate trends and make estimates or forecasts. It can also be used to analyze the marketing effectiveness, pricing and promotions on sales of a product or to assess risk in financial services or insurance domains.
    b. Studying engine performance from test data in automobiles
    c. Least squares regression is used to model causal relationships between parameters in biological systems
    d. OLS regression can be used in weather data analysis
    e. Linear regression is used in observational astronomy commonly enough. A number of statistical tools and methods are used in astronomical data analysis, and there are entire libraries in languages like Python meant to do data analysis in astrophysics.
3. Logistic Regression
    a. Image Segmentation and Categorization
    b. Geographic Image Processing
    c. Handwriting recognition
    d. Healthcare : Analyzing a group of over million people for myocardial infarction within a period of 10 years is an application area of logistic regression.
    e. Prediction whether a person is depressed or not based on bag of words from the corpus

Contributions:

- For Part 1, Lisa and Shagun collectively discussed the concepts of all algorithms (linear classification, pocket, linear regression, and logistic regression) and decided to initially

code the algorithms individually. From the individual work, we then picked the best code to submit, which was Shagun's code for the linear classification and pocket algorithms, Lisa's code for linear regression and both for logistic regression.

- For Part 2, Lisa and Shagun collectively researched all possible libraries for this assignment. Lisa implemented the library for linear regression and Shagun implemented the libraries for linear classification and logistic regression.
- Part 3 was researched and completed by Shagun.
- The report was compiled by Lisa and Shagun.