CODE REPORT

## Name: Rishit Jain

## Division: Machine Learning

## Admission Number: 23JE0812

## Group: 1

# Timeline

### Day 1st:

Completed writing Vectorized Linear Regression code for the previous dataset provided.

Spent 3 hours learning from Andrew Ng’s Coursera course.

### Day 2nd:

Tuned hyperparameters for the dataset provided.

Implemented regularization.

Wrote basic code for polynomial regression.

Detailed study regarding regularization and its types.

### Day 3rd:

Removed outliers from linear regression and polynomial regression dataset, which made it more accurate and reduced cost.

Tested various normalization methods on an unrelated dataset.

### Day 4th – 8th:

Vacation with family.

### Day 9th:

Tuned hyperparameters for new dataset for linear regression.

Removed normalization from linear regression.

Implemented vectorized polynomial regression with new dataset provided.

Learnt logistic regression and its related concepts.

### Day 10th:

Optimized linear regression code using lambda functions, pipe function, etc. and made it faster and compact (14 lines).

Implemented logistic regression vectorized approach on standard dataset.

### Day 11th:

Added epochs in logistic regression code, hence making it faster.

Optimized hyperparameters for logistic regression code.

Learnt about KNN classification method and related concepts.

### Day 12th:

Added accuracy and cost plots in logistic regression code.

Implemented logistic regression on provided dataset.

Optimized logistic regression and tuned hyperparameters.

### Day 13th:

Tested Adam Optimizer on linear and polynomial regression codes to determine if its worth it to implement in final code (it wasn’t).

Learnt K-means model and related concepts.

### Day 14th to 16th:

Outing with friends.

### Day 17th:

Implemented vectorized approach to KNN using batches because of RAM limitations.

### Day 18th:

Flight to Dhanbad.

### Day 19th:

Implemented a special approach to KNN using only a split portion of the dataset for finding distances. (explained in further parts of the report)

Tested and studied other methods for KNN such as ANNOY, K-Tree, centroid methods, etc.

### Day 20th:

Prepared concepts and formulas for mid-evaluation interview.

### Day 21st:

Prepared basic report on paper.

Changed polynomial regression code in hopes of making it more optimized. (ruined it)

Studied Neural Networks using Tensorflow.

### Day 22nd:

Implemented Neural Networks with Relu activation as hidden layers and softmax as output layer.

### Day 23rd:

Implemented K-means algorithm on classification dataset (after removing target values).

### Day 24th to 29th:

Executed minor optimization changes in code.

Tried to repair polynomial regression code.

Wrote chunks of code report.

### Day 30th:

Optimized Neural Networks code.

Compacted and Optimized Normal KNN code. (code without time optimization)

### Day 31st:

Shortened Logistic Regression Code and made it faster.

Tuned hyperparameters of logistic regression code.

### Day 32nd:

Uploaded some files to private Github repository.

Advanced the code report.

Calculated time taken for all codes to run on Kaggle.

### Day 33rd:

Uploaded more files to Github.

Optimized K-Means code and added plot for elbow point.

Advanced the code report.

Optimized the Neural Network code.

Tried to repair the polynomial regression code (failed).

### Day 34th to 37th:

Optimized Neural Networks and uploaded to Github.

Made the code compact and fluid.

Fixed Bugs and completed the report.

Thanked the friends made along the way.

# Linear Regression Report

## Observations on previous dataset:

I observed that there were some outliers in the previous dataset provided. The majority of target values lied in the target region of +200 and -200. This was ruining accuracy and increasing cost. To combat this, I added a function to remove rows from dataset where target value lied outside the range of +200 and -200. This manoeuvre removed 50-100 values of outliers from the dataset of 50000 values.

## Experiment with Adam Optimizer:

Previously I had experimented with implementing Adam Optimizer. The implementation was successful, but the difference in accuracy or time taken wasn’t noteworthy, hence, I didn’t add that in the final code.

## Final Linear Regression Code:

[woc6.0/linreg.ipynb at main · ADizzyPython/woc6.0 (github.com)](https://github.com/ADizzyPython/woc6.0/blob/main/linreg.ipynb)

## Explanation of current Linear regression code:

### Line 1:

Imports numpy, matplotlib.pyplot and pandas libraries.

### Line 2:

Reads CSV file that stores dataset, shuffles it, resets index and splits dataset into x\_train, y\_train, y\_cvs, x\_cvs using pipe function.

### Line 3:

Defines cost function using lambda expression. Regularized cost function is outputted.

### Line 4 to 9:

Implements vectorized gradient descent using for loop. Updates values of slops and intercept based on the number of iterations. Also prints cost after every 100 iterations. The function returns the slope and intercept after execution.

### Line 10:

Calls the gradient descent function and accepts returned values in w\_final and b\_final variables.

### Line 11:

Predicts values on cross validation set using w\_final and b\_final and stores value in y\_pred.

### Line 12:

Forms overlapping scatter plots of first 100 predicted values and cross validation set target values against first 100 values of training set.

### Line 13:

Defines R2 score using lambda function.

### Line 14:

Prints R2 score on y\_cvs and y\_pred.

## Results and Hyperparameters:

Lambda: 0.01

Learning Rate: 0.00001

Number of Iterations: 10001

R2 Score: 0.9999817973330183

Time taken: 115 seconds

## Key Learnings:

Using Lambda functions efficiently.

Using (.pipe) function from pandas library.

Implementation and concept of cost function, gradient descent, outlier removal (data cleaning), regularization and testing with cross validation set.

# Polynomial Regression Report

## Sob story 😭

My polynomial regression code was successfully running on the previous dataset. On the previous dataset, I had noticed that removing outliers greatly reduced the cost and increased the accuracy. That dataset didn’t require normalization as well.

After the new dataset was provided, I switched the code to include normalization and made some other changes as well. Unfortunately, my code stopped working and I am unable to debug it despite trying innumerable times. I also rewrote the code multiple times, but to no avail.

## Final Polynomial Regression Code:

[woc6.0/polynomial\_reg.ipynb at main · ADizzyPython/woc6.0 (github.com)](https://github.com/ADizzyPython/woc6.0/blob/main/polynomial_reg.ipynb)

## Cell wise explanation of Polynomial Regression code:

### Cell 1:

Imports libraries numpy, matplotlib.pyplot, pandas.

Reads the dataset, shuffles it, resets index and splits it into training and testing datasets. [x\_train, y\_train, x\_cvs, y\_cvs]

### Cell 2:

Defines the normalize function. The function adds a small constant epsilon to avoid overflow errors. The method gives the option to choose between z-score normalization, min-max normalization or no normalization at all.

At the end, the function returns the normalized dataset, along with the values required to normalize the cross validation set.

### Cell 3:

Defines the feature scaling function which takes in the input dataset and the degree to scale to. The function copies the dataset’s transpose into another variable X to avoid manipulation of original dataset. The features variable copies the initial dataset. The previous chunk variable does the same, its function is to store the previous degree terms. The indices stores the list of len of training set.

The function then iterates over each value upto the degree specified. Then it defines an empty chunk called new\_chunk at every iteration.

Another loop iterates over the indices and values of X. This loop multiples each row of previous chunk value with its respective row in X and appends it to new\_chunk.

After this loop ends, the new\_chunk values are appended into features and prev\_chunk’s value is updated to match that of the new\_chunk.

Finally, this function reshapes the features array to a matrix matching the number of rows of X and then returns it.

This cell also tests the function by printing its result and the result’s shape.

### Cell 4:

This cell defines the compute\_cost function using lambda function.

### Cell 5:

This function defines the gradient descent function along with regularization. This is same as the linear regression function.

### Cell 6:

This function defines the R2 score using lambda function.

### Cell 7:

This function calls all other functions based on the inputs taken. It first calculates the polynomial features, of training set and cross validation set, normalizes them if norm is not defined as None. The slopes and intercept are initialized as 0. The function then calls the gradient descent function and predicts the target values from cross validation set using the slops and intercept returned. The R2 score is then calculated and printed.

A scatter plot of the first 100 values y\_cvs and y\_pred are printed against the first feature from cross validation set.

### Cell 8:

This cell finally calls the polyreg function.

## Results and Hyperparameters:

Time taken to execute: 66.61 seconds

Number of iterations: 10000

Degree: 6

Normalization used: z-score

Lambda for regularization: 0

Learning Rate: 0.1

R2 Score: -0.001097 😭

## Key learnings:

Learnt feature scaling and its importance.

Implementing polynomial regression, and its concept.

Reducing overfitting and its concept.

Normalization, its types and implementation.

# Logistic Regression Report

## Final Logistic Regression code:

[woc6.0/logregfinal.ipynb at main · ADizzyPython/woc6.0 (github.com)](https://github.com/ADizzyPython/woc6.0/blob/main/logregfinal.ipynb)

## Line wise explanation for Logistic Regression Code:

### Line 1:

Imports libraries numpy, matplotlib.pyplot and pandas.

### Line 2:

Defines softmax function using lambda expression.

### Line 3:

Defines cross entropy loss using lambda expression.

### Line 4:

Defines accuracy function using lambda expression.

### Line 5-6:

Reads the classification\_train dataset, converts it to a pandas dataframe and divides all pixel values by 255.

### Line 7-25:

Defines the logreg function which takes batch\_size, alpha, epochs and the dataframe. The function then initializes four arrays for storing values of training loss, testing loss, testing accuracy and training accuracy.

The slopes and intercept are all initialized as 0.

In each epoch, the dataframe is shuffles row wise, and then is split into training and testing sets [x\_train, y\_train, x\_cvs, y\_cvs].

The y\_train and y\_test are then one hot encoded using the pd.get\_dummies function.

Vectorized logistic regression using softmax is then implemented on the data batchwise.

Prediction sets are then formed using the obtained values of slopes and intercept.

The loss and accuracies for the sets are calculated and appended into the respective arrays and then printed as output.

The arrays are returned by the function after all iterations have been completed.

### Line 26:

Logistic regression function is called.

### Line 27-40:

Figure showing train and test loss is plotted.

Second figure displaying train and test accuracy is plotted.

## Results and Hyperparameters:

Batch size: 100

Learning rate: 0.05

Number of epochs: 50

Final Training Accuracy: 98.89%

Final Testing Accuracy: 98.70%

Time taken for execution: 67.3 seconds

## Key Learnings:

Implementation and concept of softmax function and cross entropy loss.

One hot encoding concept and implementation using pd.get\_dummies function.

# K-Nearest Neighbours Algorithm Report

## Brief:

I have implemented the KNN code using the normal method where each value of the dataset is iterated separately. If we wish to find all Euclidian distances at the same time using matrices, we face RAM shortage due to the sheer amount of data being operated in such a task. To combat this, we can iterate over the data batchwise, however, there is much faster method to implement KNN without much difference in accuracy. I have implemented this technique in the file fastknn.ipynb, while the normal KNN code lies in knnnormal.ipynb.

In fastknn.ipynb, instead of finding the Euclidian distances from all the points in the dataset for each value, I have only calculated the distances from a specifiable amount of points. This greatly reduces the time required down to mere seconds. The time could be further reduced by adding vectorized batches to this code, however, I haven’t implemented that.

## Final KNN Code:

### Normal KNN:

[woc6.0/knnnormal.ipynb at main · ADizzyPython/woc6.0 (github.com)](https://github.com/ADizzyPython/woc6.0/blob/main/knnnormal.ipynb)

### Fast KNN:

[woc6.0/fastknn.ipynb at main · ADizzyPython/woc6.0 (github.com)](https://github.com/ADizzyPython/woc6.0/blob/main/fastknn.ipynb)

## Line wise explanation of Normal KNN code:

### Line 1:

Imports libraries numpy, matplotlib,pyplot and pandas

### Line 2:

Marks start time

### Line 3:

Reads Classification dataset, shuffles it, resets index and splits it into training and testing sets [x\_train, y\_train, x\_test, y\_test] using pipe function.

### Line 4-12:

Defines the kNN function which takes x\_train, y\_train, x\_test, y\_test and k as input, where k is the number of nearest neighbours to test.

It then iterates over each value in x\_test one by one, finds the Euclidian distances of the point from each point in x\_train using the np.linalg.norm function of numpy.

The function then defines an array ‘labels’ which stores the values of the k nearest predictions.

The and value of the most occurring prediction is then appended into the array predictions.

The function finally prints the accuracy after calculating it against y\_test, and then returns it along with predictions.

### Line 13:

The line calls the function and stores the returned values in the respective variables.

### Line 14:

The line marks the end time

### Line 15:

The line prints the time taken.

## Results and Hyperparameters:

K: 7

Accuracy: 0.98

Time take: 1079 seconds

## Line wise explanation of Fast KNN code

### Line 1:

Imports libraries numpy, matplotlib.pyplot and pandas

### Line 2:

Marks the start time

### Line 3:

Reads Classification dataset, shuffles it, resets index and splits it into training and testing sets [x\_train, y\_train, x\_test, y\_test] using pipe function.

### Line 4-9:

Defines the kNN function which takes x\_train, y\_train, x\_test, y\_test and k as input, where k is the number of nearest neighbours to test.

It then iterates over each value in x\_test one by one, finds the Euclidian distances of the point from the specified amount of values in x\_train.

The value of the most occurring prediction is then appended into the array predictions.

The function calculates the accuracy against y\_test, and then returns it along with predictions.

### Line 10:

The line calls the function and stores the returned values in the respective variables.

### Line 11:

The line marks the end time

### Line 12:

The line prints the accuracy

### Line 13:

The line prints the time taken.

## Results and Hyperparameters:

K: 3

Accuracy: 0.914

Time Taken: 13.71 seconds

## Key Learnings:

Developed intuition regarding KNN and its working.

Learnt usage of np.linalg.norm function.

Learnt implementation and concept of KNN algorithm.

Studied k-trees and ANNOY algorithm as methods to reduce time taken.

# K-Means Algorithm Report

## Final K-Means Code:

[woc6.0/kmeans.ipynb at main · ADizzyPython/woc6.0 (github.com)](https://github.com/ADizzyPython/woc6.0/blob/main/kmeans.ipynb)

## Line wise explanation of K-Means code:

### Line 1:

Imports libraries numpy, matplotlib.pyplot and pandas

### Line 2:

Reads the K-Means dataset, resets index and extracts it as a numpy array.

### Line 3-6:

Defines the kmeans loss function which is required to calculate the loss after testing with each value of k.

### Line 7:

Initializes the loss array.

### Line 8-26:

Defines the kmeans function which runs the kmeans algorithm, appends the loss for each k in the loss array and plots the k means clustering for that value of k.

It initializes random centroids depending upon the number of clusters defined.

It then calculates the distances of each point from each centroid and stores the indices of the lowest distance cluster in the array called labels.

The function then recalculates all centroids including the new point added in each cluster.]

The function then plots the members of each cluster with a different colour in the same scatter plot.

### Line 27-28:

Iterates from k value 2 to 17 and finds runs the kmeans function.

### Line 29-33:

Plots the Number of clusters (k value) against the final loss for the defined range. Marks the elbow point as 6.

## Results and Hyperparameters:

Number of iterations: 25

Final K as elbow point: 6

Time taken: 8.35 seconds

## Key learnings:

Implementation and concept of kmeans loss.

Using colormaps from matplotlib.cm.

Implementation and concept of kmeans algorithm.

Finding elbow point and its concept.

Formula for silhouette score and its application (that was tested, but not implemented in final code).

# Neural Network Algorithm Report

Learning Neural Networks was the most difficult task amongst all. The courses on coursera mainly used tf.keras and pre made libraries and hence it was a challenging task. I mainly referred to CodingLane on YouTube and a project on Kaggle for the same. My friend Harshwardhan Saini’s insights helped me throughout the project.

## Final Neural Networks Code:

[woc6.0/neuralnetwork1.ipynb at main · ADizzyPython/woc6.0 (github.com)](https://github.com/ADizzyPython/woc6.0/blob/main/neuralnetwork1.ipynb)

## Cell wise explanation of code:

### Cell 1:

Imports libraries numpy, pandas, matplotlib.pyplot and time.

Reads Classification dataset, shuffles it, resets index and splits it into training and testing sets [x\_train, y\_train, x\_test, y\_test] using pipe function.

One hot encodes y\_train and y\_test using pd.get\_dummies function.

Divides pixel values in x\_train and x\_test by 255.

Defines functions relu, tanh, softmax, tanh\_derivative, relu\_derivative and compute\_cost using lambda expression.

### Cell 2:

Defines function initialize\_parameters, which uses a dictionary for defining slopes and parameters for each layer defined.

The slopes have shape such that the number of rows is equal to the number of neurons in that layer while the columns are equal to the number of neurons in the previous layer.

The intercept is a column matrix with rows equal to the number of neurons in that layer.

### Cell 3:

Defines the forward\_propagation function.

For each layer, this function multiples the values with the slops and adds the intercept. After this, the selected activation method is applied [relu or tanh] and the final output is appended in the dictionary caches. At the output layer, the function uses softmax activation.

### Cell 4:

Defines the backward\_propagation function.

This function calculates the gradients using chain rule and then updates the parameters and returns them.

The function loops over the layers from backward to forward and multiplies the derivative depending upon the activation selected. It treats the input layer separately.

Finally, this function modifies the parameters for each layer and returns them.

### Cell 5:

This function computes the accuracy of the found parameters upon the cross validation set.

The function runs the forward propagation function upon the testing data and then finds the labels of the returns values and then compares it to the real predictions and finds the accuracy and returns it.

### Cell 6:

The train\_model function combines and calls all the functions for the number of iterations mentioned.

It also prints the cost after every 100 iterations.

It prints the accuracy on the testing data and training data after completing all iterations.

The calculated time is printed at the end of the processing.

### Cell 7:

This cell finally calls the train\_model function on the layers defined.

## Results and Hyperparameters:

Train Accuracy: 99.5

Cross Validation Accuracy: 98.2

Number of iterations: 300

Training time: 275.4 seconds

Learning Rate: 0.96

Hidden layer activation: Relu

Neuron Structure: [784, 386, 64, 10]

## Key Learnings:

Concept and implementation of neural networks from scratch.

Back propagation concept and implementation.

Efficient use of dictionaries.

Working under pressure.

Making CSV file using python from parameters obtained.