REPORT ON THE PROJECT

SHORT DESCRIPTION OF THE DIFFERENT ALGORITHMS IMPLEMENTED:

1. LINEAR REGRESSION:

   It is used to solve a regression problem and assumes that the label value is linearly dependent on the feature values.

2. POLYNOMIAL REGRESSION:

   It is also used to solve a regression problem but assumes that the label value is also dependent on the higher powers of the feature values.

3. LOGISTIC REGRESSION:

   It is used to solve classification problems. More frequently it is used for binary classification but can also be used for multi class classification using the concept of one vs all which was the aim of the project.

4. KNN:

   It can be used for both classification and regression problems but is more frequently used on classification problems. It is based on the concept that points which are closer to each other will belong to the same class. We have used Euclidean distance to find the closeness of the points.

5. NEURAL NETWORK:

   It can be used for both regression and classification. It consists of multiple layers of neurons which take input from the previous layer of neurons and output to the next layer. This helps in developing complexity in the model to fit the data. We were required to do classification using an n layer neural network.

DETAILED IMPLEMENTATION DETAILS:

- ### LINEAR REGRESSION:

  - How the implementation progressed:

    - I built a fully vectorised Linear Model in the beginning but had problems as the MSE on the training data was not decreasing.
    - This was because I had not scaled my feature values. Afterwards I applied Z score standardization and the MSE started decreasing with the iterations.
    - Later on to prevent overfitting I split the data into a training set and a validation set. The model was trained on the training set with different hyperparameters until the model performed best on the validation set.
    - Also L2 regularization step was implemented on the data but as the model was not overfitting to the data lambda(L2 Regularization term) was set to zero.
    - To find the label corresponding to new test data the predict function can be used.

  - Features of my algorithm:

    - The user can split the data into training and validation sets using a function of the class. They can decide the size of the training set on their own.
    - The user can run batch gradient descent where the feature scaling method:
      - Min-max-normalization
      - Z-score standardization

      Can be decided.
    - The user can also decide the value of lambda ,L2-regularization term.
    - Exponential learning rate decay can also be used.

  - Results which the model prints for a given data:

- Learning curves showing change of mean squared error and R2 accuracy with iterations for both training set and validation set is shown.
- A scatter plot showing predicted and actual value of label vs a single feature is shown for both training and validation set shown.
- A table showing actual and predicted value for both training and validation set is shown.
- Final MSE and R2 for both training and development sets are shown.

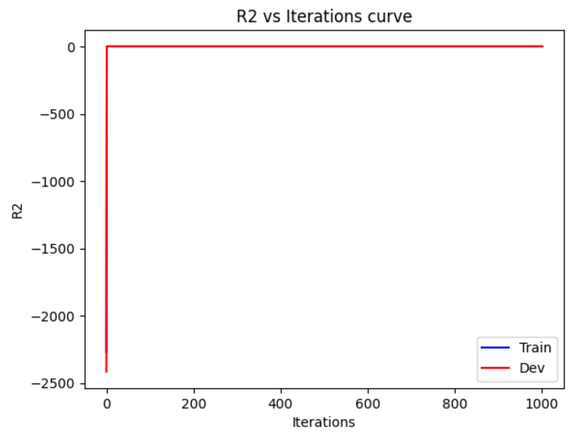
➢ Further improvements:

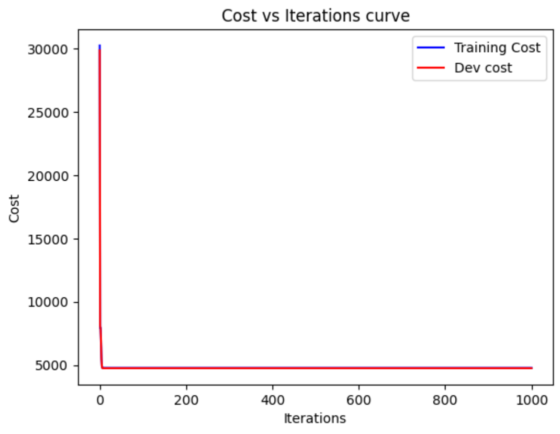
- Other optimization methods like mini batch,rms-prop,Adam can be implemented.
- A basic hyperparameter tuner can be included which will give the hyperparameters for the best results.
- Option to use other regularization methods and normalization can be included.

➢ Some things I rejected:

- Did Not include other optimization methods as the algorithm was already working really fast and well on the given data and other optimization methods were not really proving useful.
- Tried only L2 regularization as on the given data there was not really any overfitting.

➢ Results On The Given Data:





```
Final MSE of training data is:  4774.749201291516
```

```
Final MSE of dev data is:  4753.919818698941
Final R2 of training data is:  0.8139222564136669
```

```
Final R2 of dev data is:  0.81228636886605
```

POLYNOMIAL REGRESSION:

Progress Of The Implementation:

- At first I included polynomial terms using 3 for loops which added all the terms till a specified degree by taking input only the highest degree of the polynomial.
- This method has a problem as this method of adding features was not generalized to any number of features i.e. it was utilizing the fact that we had originally only three features, and wouldn't have worked with any other number of features.
- The further steps were straightforward as we only had to apply the linear regression algorithm on the new feature value with higher degree terms.
- But I faced some problems finding the suitable degree and the number of iterations and learning rate but after some tuning the cost decreased gradually.
- At first I only implemented Batch gradient descent but then I added mini batch gradient descent to it too.
- Changed the method used to add higher degree terms so that it would work with any number of features by using recursion now it can add features to any given input degree for any number of original features.
- L2 regularization and Z score normalization was implemented for the given data.

Features Of My Algorithm:

- All the features of the linear regression algorithm.
- Can add higher degree features up to any degree and for data containing any number of initial features.
- Also mini batch gradient descent can be implemented.

Results Which The Model Prints For A Given Data:

- Same results are printed as the linear regression algorithm is printed.

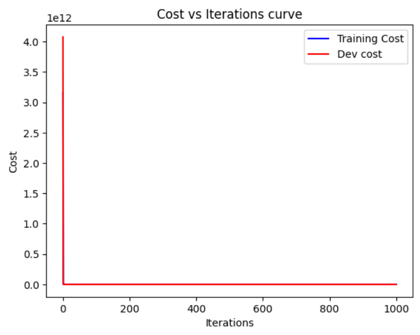
Further Improvements:

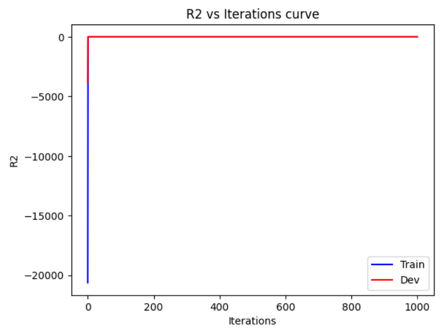
- All the improvements mentioned in the linear regression algorithm can be implemented except that mini batch gradient descent is applied here.

➢ Methods Rejected:

- Same as the linear regression algorithm

➢ Results On The Given Data:





```
Final MSE of training data is:  0.11501939174583997
```

```
Final MSE of dev data is:  0.20005718990934016
```

```
Final R2 of training data is:  0.9999999999999635
```

```
Final R2 of dev data is:  0.9999999999999509
```

❖ <u>LOGISTIC REGRESSION:</u>

➢ Progress Of The Implementation:

- I faced difficulty implementing logistic regression to multiclass classification as I had learnt only binary classification.
- Then with the help of mentors I learnt about one vs all method and implemented it but the code was not fully vectorised and was taking too much time so I worked on making it fully vectorized and then the algorithm worked.
- During this process I also learnt about one hot encoding.
- I also learnt how to derive the gradient terms during the process.
- I also added a mini batch gradient descent method to my algorithm to speed up the code.
- Basic steps like regularization, train dev split, feature scaling was also implemented.

➢ Features Of The Algorithm:

- The user can split the data into training and validation sets.
- The user can run batch gradient descent and mini batch gradient descent where the feature scaling method:
    - Min-max-normalization
    - Z-score standardization
- The value of lambda, L2 regularization term can also be set.
- Exponential learning rate decay can be implemented.

➢ Results Which The Model Prints For A Given Data:

- Learning curves showing change of mean squared error and Accuracy with iterations for both training set and validation set is shown.
- A table showing actual and predicted value for both training and validation set is shown.
- Final MSE and Accuracy for both training and development set is shown.

➢ Further Improvements:

- Other optimization methods like Adam can be used.
- Other regularization methods and feature scaling methods can also be used.

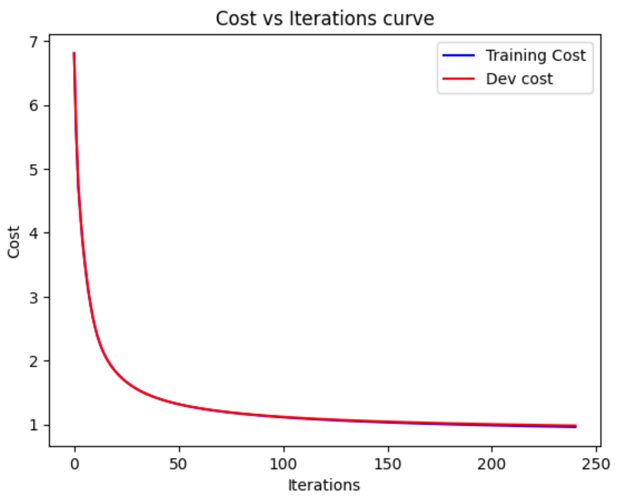
➢ Results

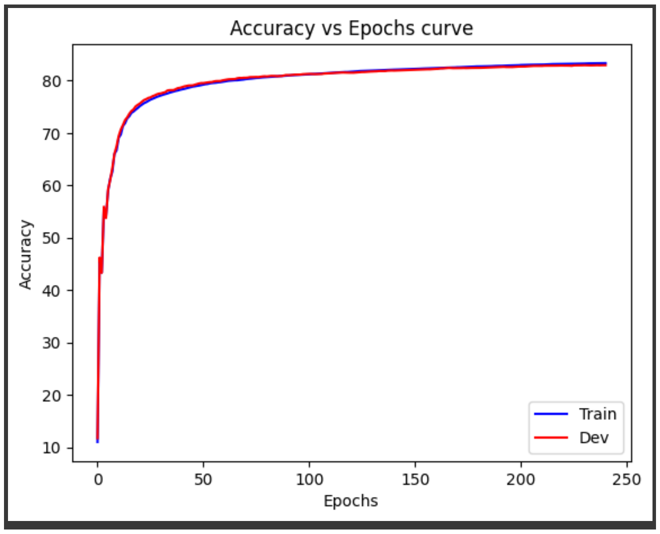
```
Final Cost of training data is:  0.9618382692696313

Final Cost of dev data is:  0.9841647389876375


Final Accuracy of training data is:  83.32083333333333 %


Final Accuracy of dev data is:  82.93333333333334 %
```





❖ <u>N-LAYER NEURAL NETWORK:</u>

➢ Progress Of The Implementation:

- Initially I faced a problem with the backprop calculation to find the gradient but eventually I calculated it.
- Further difficulties were with hyperparameter tuning and handling the large code.

➢ Features Of The Algorithm:

- The user needs to input a list containing a list containing the number of neurons in all the hidden layers.
- The user can decide between three optimization algorithms Batch gradient descent, mini batch gradient descent, adam optimization.
- The user can split the data into training and validation sets.
- ReLU activation is used for the hidden layer and softmax for the output layer.
- The user can run batch gradient descent and mini batch gradient descent where the feature scaling method:
    - Min-max-normalization
    - Z-score standardization

Can be used

- The value of lambda, L2 regularization term can also be set.
- Exponential learning rate decay can be implemented.
- Dropout regularization can also be used and the dropout probability can be given by the user.

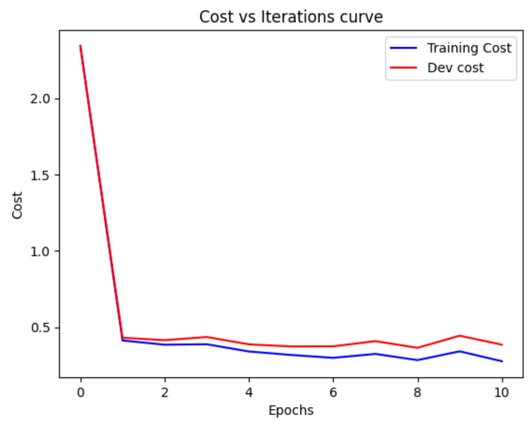
➢ Results Which The Model Prints For A Given Data:

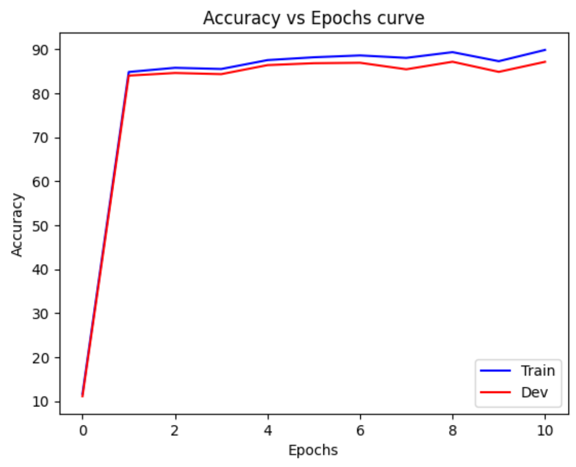
- Same as the results of the logistic regression method.

➢ Further Improvements:

- Option to change the activation of the hidden layer according to the users needs like tanh, sigmoid.
- Batch norm could also have been implemented.
- More regularization methods and better initialisation methods for weights and bias can be used.
- More feature scaling methods should have been explored.

➢ Results:





```
Final Cost of training data is:  0.27647021578817205
```

```
Final Cost of dev data is:  0.3848383210572901
```

```
Final Accuracy of training data is:  89.82083333333334 %
```

```
Final Accuracy of dev data is:  87.11666666666666 %
```

❖ <u>K NEAREST NEIGHBORS:</u>

➢ Progress Of The Implementation:

- The algorithm I first constructed was running too slow due to it having two loops.
- Then to decrease the runtime I implemented a single loop but that didn't work too well.
- Then with help of a article on google I was able to implement the algorithm without any loops in which the euclidean distance by using the axis wise sum of the square of the train set,test set and matrix multiplication of the train and test set which reduced the runtime to less than a minute.
- To find the appropriate value of k I ran a loop going through different k values for the best accuracy.
- Z-score standardization was implemented.

➢ Features Of The Algorithm:

- The algorithm runs very quickly and has a good enough accuracy on the validation set.
- Feature scaling methods:
    - Min-max normalization
    - Z-score standardization

can be used

➢ Results Which The Model Prints For A Given Data:

- The accuracy on the validation set.

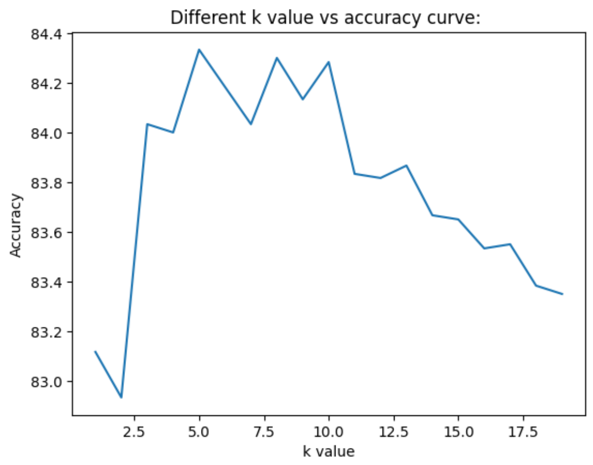
➢ Further Improvements:

- A function to find the value of k for best validation accuracy can be used.

➢ What I rejected:

- Rejected the method containing loops as it was very inefficient.

➢ Results: The highest accuracy is obtained for k=5 i.e. 84.33% in the validation set which is 20% of the given data.


Different k value vs accuracy curve:
Accuracy
k value
84.4
84.2
84.0
83.8
83.6
83.4
83.2
83.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5