ML Bootcamp Report

The project was based on creating a library for implementing certain machine learning algorithms.

The following algorithms have been implemented using NumPy, Pandas and Matplotlib:

- Linear regression
- Polynomial regression
- Logistic regression
- N-layer Neural Network
- K-Nearest Neighbour
- K-Means clustering

Along with these,the library include many helpful functions to determine the performance of the model and data pre-processing functions.

Feature scaling functions:

- Standardization Scaling
- Normalization Scaling
- Robust Scaling

Performance metric functions:

- Accuracy Calculator
- Confusion Matrix with precision and recall values
- R2-score calculator
- Mean Squared Error Calculator

Features included in the library:

- Modularity of algorithms: Python classes have been used to create a modular library, making it flexible to further expand it.
- Vectorization: All the algorithms use NumPy's vectorized matrix methods for fast calculation.
- Hyper parameter tuning: The library gives the user full flexibility to fine tune the hyper parameters for the algorithms to get the best results according to the needs.
- Cost Plot: All the algorithms contain and optional feature to plot Cost vs Number of Iterations to keep track of the model's performance.
- Additional features to load data from CSV files and split data into training and cross validation sets.

Usage of the library:

Neural Networks

- It is initialized by creating a class instance of it while passing the layer information like number of units and its activation function.

```
my_model = NeuralNet([
    Inputs(inputs=784),
    Layer(units=30, activation=Activations.relu),
    Layer(units=10, activation=Activations.softmax)
    ])
```

- Model training is done by calling the `model.fit()` function while passing the required hyper parameters,he cost function for the model and the training dataset.
- `model.predict()` is used to predict the outcome for the required data values.

Other Algorithms

- All other algorithms use a similar kind of implementation. `model.fit()` is used to set the hyper parameter values and provide the data set.
- `model.predict()` is to provide the prediction for the required data values.

Feature Scaling

- A instance is created for the scaling object which saves the scaling variables on calling the `scaler.fit_transform()` .
- To transform data using this scaler instance, `scaler.transform()` is used.

Feature Implementation Failures and fixes:

1. Problem in creation of Polynomial Features:
   a) Initially used a recursive function which calculated the product among all combinations of variables, resulting in creation of duplicate terms. This possible fix to this was checking for duplicate terms and deleting all but one of the duplicates.
   b) This method did not work as intended and resulted in creation of too many or too less terms. This was due to floating point errors and high precision floating numbers.
   c) To fix this, a different approach was used where two nested loops are used to create all the combinations of the powers of the required terms. The variables are raised to these powers to create the polynomial terms. The only drawback is, it will work for three variables only.
2. NaN values while scaling:
   a) While using the scaling feature, sometimes it result in all values becoming “nan” in some datasets. This was caused to due division by zero.
   b) This problem was fixed by using NumPy’s `nan_to_num()` function.
3. Random occurrence of nan values in K Means Clustering:
   a) Since the initial centroids are assigned randomly in the algorithm, while running the algorithm repeatedly, in some code executions, predicted values became “nan”
   b) The root of the error was the unaccounted case for a centroid with no data points assigned to it.
   c) These was fixed by randomly re-initializing the centroid if no data points were assigned to it.
4. Complete vectorization of code to increase speed:
   a) Initially the algorithms involved use of nest loops to iterate over the elements to do the calculations.
   b) A big increase in speed of code execution was achieved by using NumPy’s vector based matrix calculation methods instead of using nested loops.

Performance and Approach of each Algorithm

1. Linear Regression:
   Approach: Uses NumPy's matrix multiplication to calculate the predicted values for a linear model and Mini Batch Gradient Descent for training.
   Diagnostics: It was seen that increasing the regularization parameter led to a worse model. Thus minimal value of regularization parameter was used.
   Performance: R2 Score of 0.99 on Cross Validation set, executed in 1.9s.

2. Polynomial Regression:
   Approach: A custom made `CreatePolynomialFeatures` function is used to create polynomial terms up to any degree. After creating polynomial terms, a Linear Regression model is used to train on the data-set.
   Diagnostics: It was seen that increasing the regularization parameter led to a worse model. Thus minimal value of regularization parameter was used. On varying the degree of polynomial, best fit was found 8^{th} degree polynomial. It was also noted that the model started over fitting after 30 epochs.
   Performance: R2 Score of 0.96 on Cross Validation set, executed in 16.8s.

3. Logistic Regression:
   Approach: Checks if the given data set is a multi-class classification problem or binary classification problem and one hot encodes the target matrix if the former.Uses NumPy's vectorized functions for fast calculations of predicted classes and Mini Batch Gradient Descent for training.
   Diagnostics: It was seen that increasing the regularization parameter led to a worse model. Thus minimal value of regularization parameter was used.
   Performance: 96.64% Accuracy on Cross Validation set with no regularization, in 52.5s.

4. K Nearest Neighbour:
   Approach: Uses NumPy's matrix based functions to find the 'K' nearest training data points from the test data point, and based on the maximum classes present among the 'K' nearest neighbour, a class is assigned to the test data point.
   Performance: 96.8% Accuracy on Cross Validation set for 'K' set to 3.

5. N-Layer Neural Network:
   Approach: It requires the user to define `Layer` objects specifying the number of neurons and it activation function. A sequence of layers are passed to a `NeuralNet` object. It includes three activation functions, namely, ReLU, Sigmoid and Linear. It uses Mini Batch Gradient Descent to train the network. A random set of training examples are selected based on the batch size. Each `Layer` object has a `forward()` and `backward()` function to calculate the activation values and the gradients matrices for back propagation. It also provides the features to use a Softmax activation in the output layer. It can handle three cost functions, namely, Mean Squared Loss, Binary Cross Entropy, Categorical Cross Entropy.
   Performance: 98.2% accuracy on Cross Validation set with no regularization involved.

6. K Means Clustering:
   Approach: It initially randomly assigns 'K' centroid points and then for each data point, it finds the nearest centroid point to it. The centroid points are then moved to a new mean point of the data points assigned to it. If any centroid did not get any points assigned to it, it is randomly initialized some other point.
   Diagnostics: Elbow method was used to determine the value of K. The model was trained with values of K ranging from 2 to 25.
   Performance: As per the Elbow Method, optimal value of 'K' was found to be 22.