CS419 Assignment 1

- **Shubham Lohiya, 18D100020**

Note to the grader: Refer to `report.ipynb` for better demonstration of tasks and results. main() run output is also present in the last cell of the notebook.

Data Pre-processing:

The following features were present in the given data:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4500 entries, 0 to 4499
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   name           4500 non-null   object
 1   year           4500 non-null   int64
 2   selling_price  4500 non-null   int64
 3   km_driven      4500 non-null   int64
 4   fuel           4500 non-null   object
 5   seller_type    4500 non-null   object
 6   transmission   4500 non-null   object
 7   owner          4500 non-null   object
 8   mileage        4364 non-null   object
 9   engine         4364 non-null   object
 10  max_power      4367 non-null   float64
 11  torque         4363 non-null   object
 12  seats          4364 non-null   float64
dtypes: float64(2), int64(3), object(8)
memory usage: 457.2+ KB
```

Pre-processing to get features for training the model:

- The brand name and the model name of the car were extracted as separate features and both were one-hot encoded
- 'fuel', 'seller_type', 'transmission' and 'owner' features were also one-hot encoded
- For 'torque', only the maximum torque value is considered and all values have been converted to Nm. Rpm has been ignored
- 'engine' and 'mileage' were parsed simply to extract the numerical value as feature
- All the numerical features (features which were not one-hot-encoded) have undergone 0-1 normalization so as to bring them all in the same range. Standardization was also tried but didn't give better results. Also, there was not obvious reason to assume a gaussian distribution of feature data, so standardization was not considered for feature scaling
- Finally, 252 features were obtained per sample for training the regression model

Task 1:

a)

i. Closed form RMSE on dev set: 228195.7197465307
ii. Batch-GD RMSE on dev set (lr =0.1, 218k epochs): 226615.3705898587
iii. Absolute difference: **1580.349156671989**

Thus, closed form solution has higher RMSE than batch GD. This is because the closed form solution considers only the training set and hence overfits to it whereas the linear model trained through gradient descent does a better job of generalising.

b) I have implemented the early stopping criteria to check GD convergence. The model stops training if this criterion is satisfied or if the user-set cap of 500k epochs is reached, whichever happens first. I have outlined the early stopping implementation below:

- Evaluate current performance on the dev set after every 100 epochs and compare it with the previous evaluation
- If current performance is better (lower RMSE), save current weights as the best weights and continue training
- If the current performance is worse, increment a counter variable by one. The early stopping criteria has a patience parameter (p) set by the user (currently 5).
- If the counter variable hits this patience threshold (p) before the epoch cap is reached, i.e. it continually performs worse on p consecutive checks, training will stop and the best weights (from last improvement step) will be returned

Thus, this stopping criterion prevents overfitting by stopping training if dev set RMSE starts to increase while train set RMSE is still decreasing.

c)

i. As mentioned previously, Batch GD RMSE: 226615.3705898587
ii. SGD RMSE on dev set (lr = 0.01, 985 epochs): 228591.4829743926
iii. Absolute difference: **1976.112384533888**

Thus, SGD has higher RMSE than batch GD. Annealing learning rate over time will remove this and make is as good as batch GD while taking less epochs to converge.

Task 2:

a) RMSE on dev-set with L2 regularization (lr = 0.1, λ = 1e-4): **229233.0241678528**
b) RMSE on dev-set with L4 regularization (lr = 0.1, λ = 1e-10): **218730.98015190024**

L2 regularization didn't help improve over unregularized batch GD on the dev set but L4 regularization helps improve over the unregularized model.

Task 3:

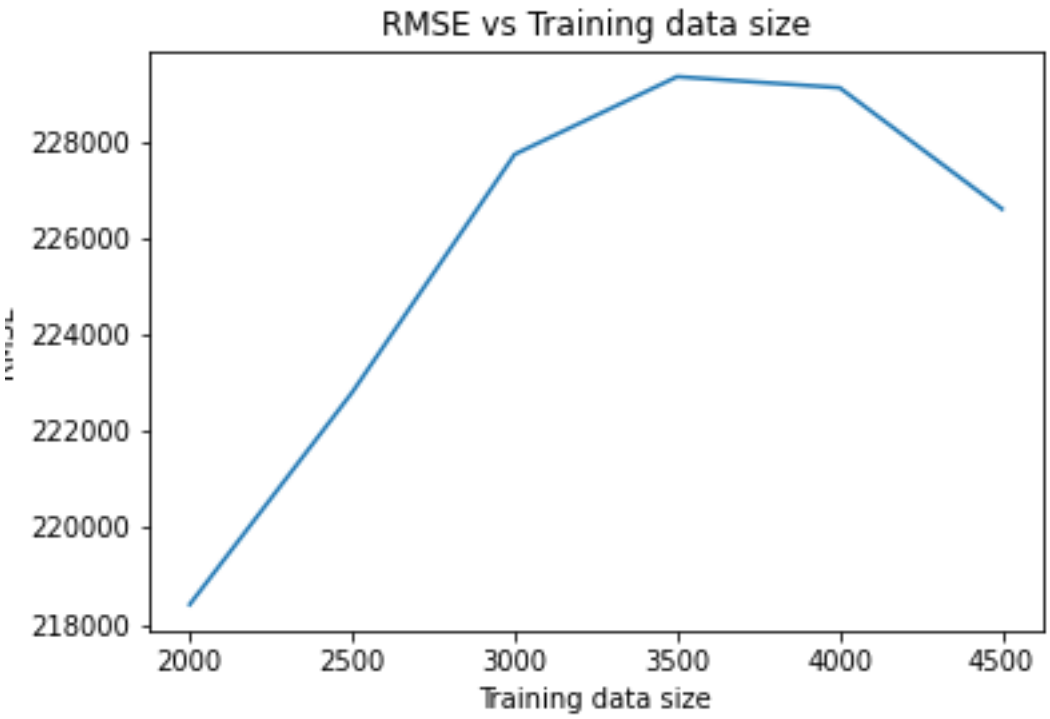
I have applied the exponential function as the basis functions to the first 7 features of the input. These features *are year, km_driven, power, seats, torque, engine and mileage*.

I have chosen the exponential function for basis as my normalization brings my data in the 0-1 range. By applying exponential function (which has a slope greater than that of linear function (>1) in this range) on this, the spread of the data is improved and this helps the model better discriminate between examples and get more accurate predictions.

RMSE on dev-set with L2 regularization and basis function implementation (lr = 0.03, λ = 1e-4): **227573.29355538535**

Thus, there was an improvement over the earlier L2 regularized model without basis functions implemented. The RMSE dropped by 1659.7306124674506. The model stopped training as it reached the cap of 500k epochs without converging. If the model is let run longer, it'll improve even more.

Task 4:



The X-axis indicates the subset size of training data use, in increments of 500. The Y-axis shows the RMSE on the dev set when model is trained with a particular subset of data.

Task 5:

['seller', 'seats', 'fuel', 'transmission', 'torque', 'mileage', 'engine', 'km', 'year', 'power', 'owner', 'name']

The list above indicates the order of features from least important to most important.

We clearly see that the **2 least important features are "seller type" and "number of seats"**

The importance of features was identified as follows: Weights corresponding to the less important features will have lower magnitudes, so there is less effect of these features on the price estimation.

The weight with the maximum magnitude was considered among weights for one-hot encoded features as they correspond to the most important information that feature is giving the model.

Task 6:

I have made use of a **Multi-Layered Perceptron** model (Prof. allowed usage of MLPs) with 2 hidden layers of 8 neurons each as an enhanced model to get the best performance on the Kaggle competition. Using this, I have achieved Rank 1 on the Leader board with a considerable margin.

After 193 epochs, we have the following result:
`Epoch 193.............RMSE on train = 98088.50331994436, RMSE on dev = 175881.39976665002`

Thus, we see that this model easily outperforms the linear models, but it's also more prone to overfitting. Tinkering around with hyperparameters and model parameters led me to the top position on the Kaggle leader board. Even without MLP, I was able to obtain around 160k RMSE which was pretty good on the leader board. After tuning the hyperparameters by considering performance on the dev set, retraining the model on train+dev set gives us more data to work with and hence better predictions.

The performance is even better when learning rate is 1e-6 and the model is trained for 300-600 epochs. The performance may also improve further by using the basis function applied features.