TECHNICAL UNIVERSITY OF MUNICH

CHAIR OF TRANSPORTATION SYSTEMS ENGINEERING

STATISTICAL LEARNING AND DATA ANALYTICS FOR TRANSPORTATION SYSTEMS

PROBLEM SET 2

Author	*Matriculation Number*
NATALIA ALLMI	03755024

Problem 1

Suppose we have trained a logistic regression model for several iterations on a tiny dataset with two features (see Table 1), and the resulting parameters are $w1 = 0.25$, $w2 = 1.01$ (slope) and $b = 0.41$ (intercept):

No.	x_1	x_2	y
1	1.4	0.9	0
2	-1.0	-1.3	1
3	-0.4	0.5	0
4	0.5	1.0	1

Table 1: Tiny dataset

1.1: Compute $p(y = 0|x1, x2)$ for all the samples using the parameters above.

For a logistic regression, the probability of the label belonging to class 1 is given by:

$$p(C_1|x) = \frac{1}{1+\exp(-z)}$$

Therefore, the probability of the label belonging to class 0 is calculated as:

$$p(C_0|x) = 1 - \frac{1}{1+\exp(-z)}$$

Where: $z = b + w_1x_1 + w_2x_2$

And in this specific case: $z = 0.41 + 0.25x_1 + 1.01x_2$

We obtain $p(y = 0|x1, x2)$ for each sample, by replacing the values of x_1 and x_2.

- 1) $p(C_0|x1, x2) = 1 - \frac{1}{1+\exp(-(0.41-0.25*1.4-1.01*0.9))} = 0.700$
- 2) $p(C_0|x1, x2) = 1 - \frac{1}{1+\exp(-(0.41-0.25*(-1)-1.01*(-1.3))} = 0.122$
- 3) $p(C_0|x1, x2) = 1 - \frac{1}{1+\exp(-(0.41-0.25*(-0.4)-1.01*(0.5))} = 0.498$
- 4) $p(C_0|x1, x2) = 1 - \frac{1}{1+\exp(-(0.41-0.25*(0.5)-1.01*(1.0))} = 0.673$

1.2: Compute the log likelihood value.

Likelihood Function:

$$p(y_1, y_2, .., y_N|x) = \prod_{i=1}^{N} p(y_i|x_i)$$

Logarithm of Likelihood function:

$$logp(y_1,y_2,..,y_N|x) = \sum_{i=1}^{N} logp(y_i|x_i) = \sum_{i=1}^{N}[y_i log(p_{1,i}) + (1-y_i)log(1-p_{1,i})]$$

Compute $[y_i log(p_{1,i}) + (1-y_i)log(1-p_{1,i})]$ for each sample :

- 1) $0 * log(1-0.700) + (1-0)log(0.700) = -0.356$
- 2) $1 * log(1-0.122) + (1-1)log(0.122) = -0.130$
- 3) $0 * log(1-0.498) + (1-0)log(0.498) = -0.697$
- 4) $1 * log(1-0.673) + (1-1)log(0.673) = -1.117$

Therefore, the value of the log-likelihood is:

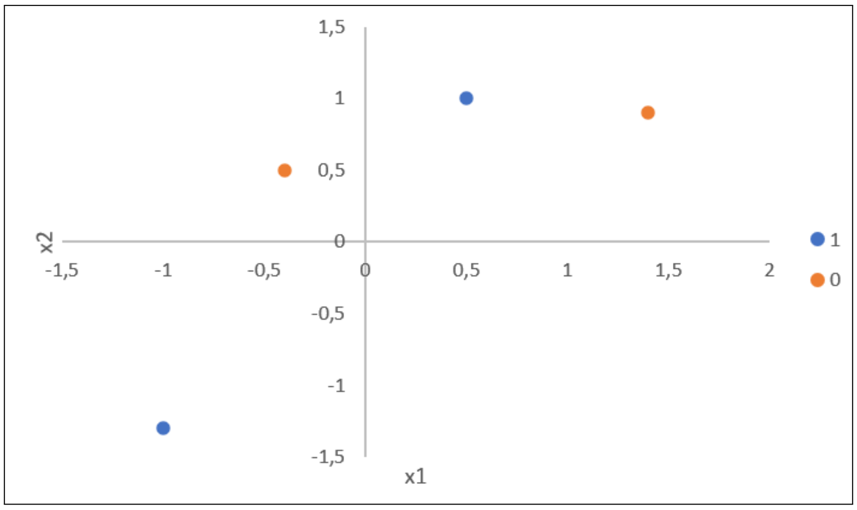
$$\sum_{i=1}^{N} logp(y_i|x_i) = \sum_{i=1}^{N}[y_i log(p_{1,i}) + (1-y_i)log(1-p_{1,i})] = -2.30$$

1.3: How can you transform the data such that they can be linearly separated?

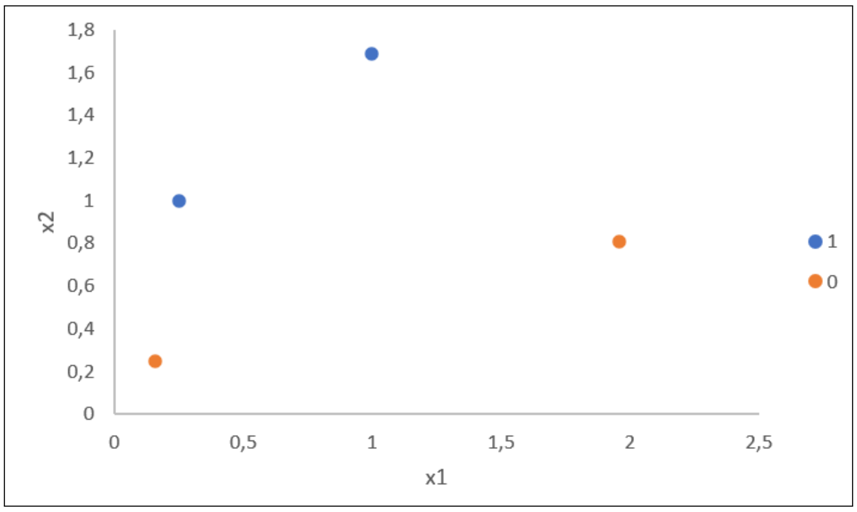
In cases where data is not linearly separable, we can apply the kernel trick to transform the data in order for it to be linearly separable. The kernel trick consists in mapping the data from lower to higher dimensional spaces using a mapping function $\phi()$, so the data can be linearly separated using a hyperplane. Some of the more common kernel functions are the polynomial kernel and the RBF kernel.

For this excercise, if we use as a mapping function$\phi() = x^2$ for both x1 and x2, we can see the data becomes linearly separable.

Before transformation:

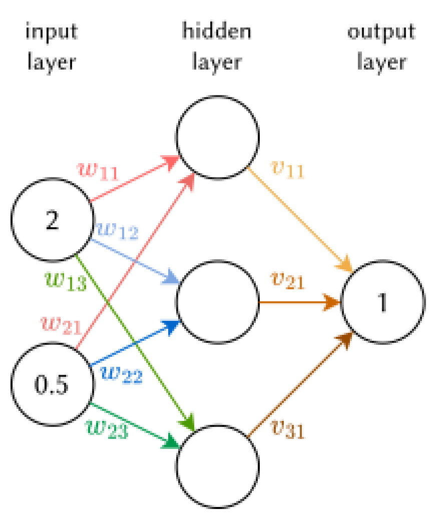


After transformation:

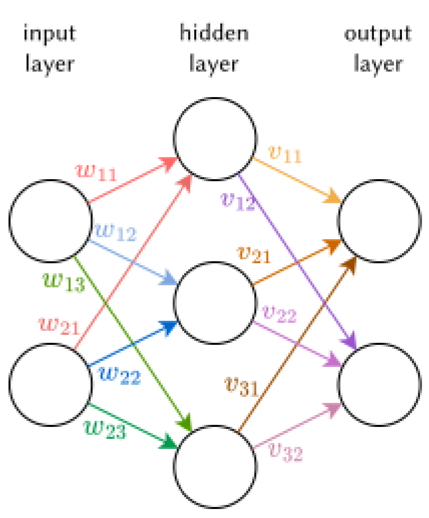


Problem 2

Your highway management department is planning to build a multilayer perceptron model for traffic monitoring. You are asked to test and demonstrate the functionality of the model using a simple test sample:



(a) Single output



(b) Two outputs

Figure 1: Two MLP models

2.1: The architecture of the model is demonstrated in Figure 1(a). The feature vector of the test sample is (2,0.5) , and the label is 1. Ignore bias terms and the activation function. Train the model until reaching zero loss. Use the stochastic gradient descent (SGD) optimizer with learning rate $\eta =$ 0.05, and the loss is calculated by $\ell(y,\hat{y}) = (y-\hat{y})^2$.

We train the model by optimizing the loss function, where the decision variables are the weights and biases of the NN.

$$W^* = arg_w minL(y,\hat{y})$$

Using forward propagation we feed input vectors to the network through all neurons and compute the output ($\hat{y}$). Taking into account that we ignore bias terms and the activation function, the output ($\hat{y}$) is calculated as:

$$z_1 = x_1 w_{11} + x_2 w_{21}$$

$$z_2 = x_1 w_{12} + x_2 w_{22}$$

$$z_3 = x_1 w_{13} + x_2 w_{23}$$
$$\hat{y} = z_1 v_{11} + z_2 v_{21} + z_3 v_{31}$$

Where x_1 and x_2 are in input feature vector.

Through backward propagation we compute errors and update the weights. The errors of outputs are calculated using the loss function as:

$$L = (y - \hat{y})^2$$

Using stochastic gradient descent we update weights using step size $\eta = 0.05$. The updated weights are calculated as:

$$w_i = w_i - \eta \frac{\partial L}{\partial w_i}$$

In this case, the partial derivatives for each weight is:

$$\frac{\partial L}{\partial w_{11}} = 2(-1)v_{11}x_1$$
$$\frac{\partial L}{\partial w_{21}} = 2(-1)v_{11}x_2$$
$$\frac{\partial L}{\partial w_{12}} = 2(-1)v_{21}x_1$$
$$\frac{\partial L}{\partial w_{22}} = 2(-1)v_{21}x_2$$
$$\frac{\partial L}{\partial w_{13}} = 2(-1)v_{31}x_1$$
$$\frac{\partial L}{\partial w_{23}} = 2(-1)v_{31}x_2$$
$$\frac{\partial L}{\partial v_{11}} = 2(-1)(x_1 w_{11} + x_2 w_{21})$$
$$\frac{\partial L}{\partial v_{21}} = 2(-1)(x_1 w_{12} + x_2 w_{22})$$
$$\frac{\partial L}{\partial v_{31}} = 2(-1)(x_1 w_{13} + x_2 w_{23})$$

Once the weights are updated we have to perform forward propagation once again with the new weights, to calculate the new loss. We repeat these steps until the loss is close to 0.

The calculations are in the attached excel sheet.

It's worth noting that we can't guarantee a global optimum. The algorithm could be stuck in a local optimum.

2.2: The department would like to investigate another variable at the same time, hence an additional output dimension in the model (see Figure 1 (b)). The square loss can be extended to $\ell(y,\hat{y}) = ||y-\hat{y}||_2^2$.Show step by step how you use the chain rule to develop a formula of the gradient with respect to w12, and discuss the relationship between this MLP model and linear regression.

The Loss function is:

$$L = ||y-\hat{y}||_2^2$$

Therefore, in our case, with two outputs:

$$L = \sqrt{(y_1-\hat{y}_1)^2+(y_2-\hat{y}_2)^2}^2 = (y_1-\hat{y}_1)^2+(y_2-\hat{y}_2)^2$$

Where the two outputs are calculated as:

$$z_1 = x_1w_{11}+x_2w_{21}$$
$$z_2 = x_1w_{12}+x_2w_{22}$$
$$z_3 = x_1w_{13}+x_2w_{23}$$
$$\hat{y}_1 = z_1v_{11}+z_2v_{21}+z_3v_{31}$$
$$\hat{y}_2 = z_1v_{12}+z_2v_{22}+z_3v_{32}$$

Using the chain rule we calculate the gradient with respect to w_{12}.

$$\frac{\partial L}{\partial w_{12}} = \frac{\partial L}{\partial \hat{y}_1}*\frac{\partial \hat{y}_1}{\partial z_1}*\frac{\partial z_1}{\partial w_{12}}+\frac{\partial L}{\partial \hat{y}_1}*\frac{\partial \hat{y}_1}{\partial z_2}*\frac{\partial z_2}{\partial w_{12}}+\frac{\partial L}{\partial \hat{y}_1}*\frac{\partial \hat{y}_1}{\partial z_3}*\frac{\partial z_3}{\partial w_{12}}+\frac{\partial L}{\partial \hat{y}_2}*\frac{\partial \hat{y}_2}{\partial z_1}*\frac{\partial z_1}{\partial w_{12}}+\frac{\partial L}{\partial \hat{y}_2}*\frac{\partial \hat{y}_2}{\partial z_2}*\frac{\partial z_2}{\partial w_{12}}+\frac{\partial L}{\partial \hat{y}_2}*\frac{\partial \hat{y}_2}{\partial z_3}*\frac{\partial z_3}{\partial w_{12}}$$

Calculating the partial derivatives:

$$\frac{\partial z_1}{\partial w_{12}} = 0,\ \frac{\partial z_2}{\partial w_{12}} = x_1,\ \frac{\partial z_3}{\partial w_{12}} = 0$$
$$\frac{\partial \hat{y}_1}{\partial z_1} = v_{11},\ \frac{\partial \hat{y}_1}{\partial z_2} = v_{21},\ \frac{\partial \hat{y}_1}{\partial z_3} = v_{31}$$
$$\frac{\partial \hat{y}_2}{\partial z_1} = v_{12},\ \frac{\partial \hat{y}_2}{\partial z_2} = v_{22},\ \frac{\partial \hat{y}_2}{\partial z_3} = v_{32}$$
$$\frac{\partial L}{\partial \hat{y}_1} = 2*(-1),\ \frac{\partial L}{\partial \hat{y}_2} = 2*(-1)$$

And therefore:

$$\frac{\partial L}{\partial w_{12}} = 2(-1)v_{21}x_1+2(-1)v_{22}x_1 = (-2)(v_{21}x_1+v_{22}x_1)$$

A linear regression can be thought of as a single layer neural network with an activation function that is the identity function (the output is the same as the input). If we build a NN with said characteristics, we have: $\hat{y} = \sum w_i x_i + b$ which is the multiplication of the weights with the features plus the bias. This is equivalent to the linear regression model where $\hat{y}$ is equal to multiplying the coefficients with the features plus the intercept.

Problem 3

The traffic management and control center in Arcisville monitors the traffic state of a highway segment shown as Figure 2. Traffic volume and speed data were automatically and continuously collected from location 1, 2 and 3 when the loop detectors work well. However, the loop detectors are sometimes offline due to either a regular maintenance or malfunction. Usually, it takes about three business days for the loop detectors to be back online. For that reason, you are asked to use supervised learning method to estimate traffic state (i.e. volume and speed) for a location when its loop detector is absent. Given one-month historical data (see Data-Problem-3.csv) from location 1, 2, and 3, you are asked to design a neural network model for traffic state estimation at location 3 when its loop detector is absent for a 3-day period. It is suggested that you split the historical data into at least two subsets; the first set is used to train the model; and the second set, which contains the last several days (e.g., 3 days) of data, is used to test prediction accuracy of your model. The time interval between each two records in the dataset is 60 minutes.

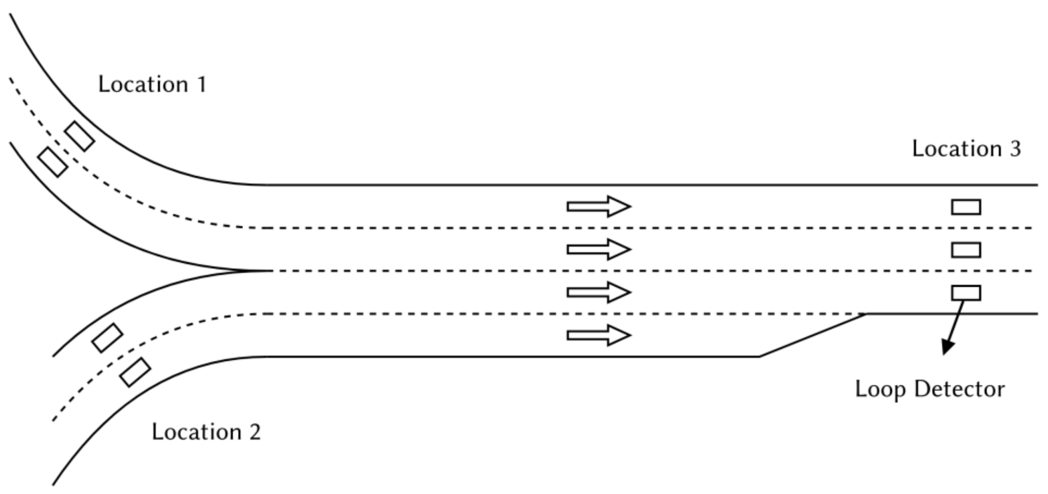
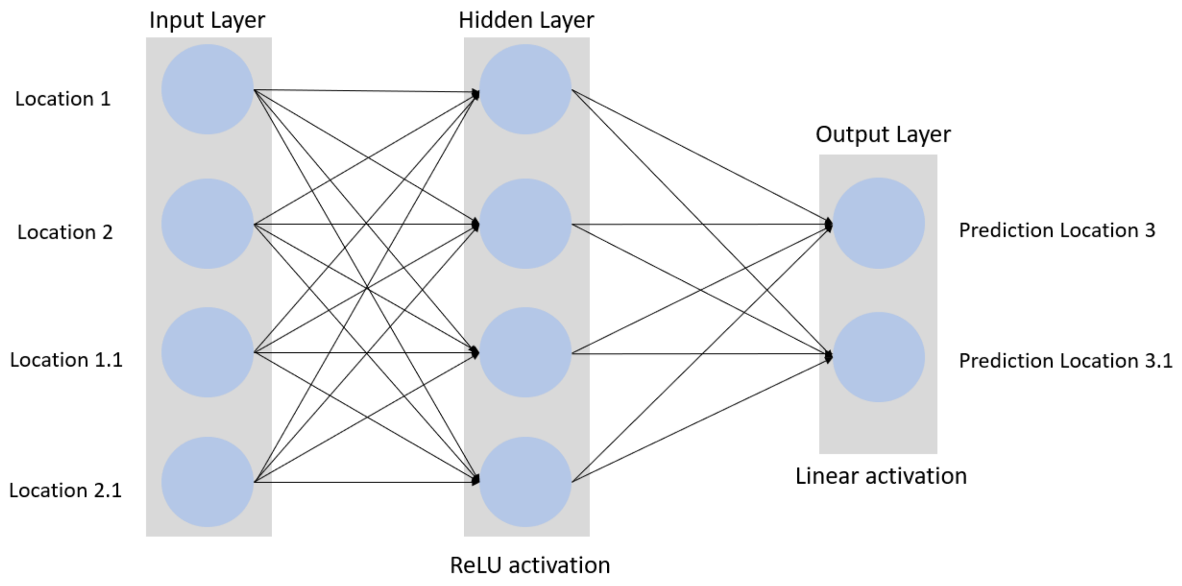


Figure 2: Highway segment and loop detector locations

3.1: Build your neural network and plot the architecture of your neural network.



The NN has one input layer of size 4. The hidden layer has the same size as the input and uses the ReLu activation function. The output layer has size 2 and uses a linear activation because it's a regression model.

3.2: Show your Python code of model construction, training and prediction. Plot your prediction vs true labels. Choose at least three different metrics to measure the accuracy of your prediction and discuss the performance.

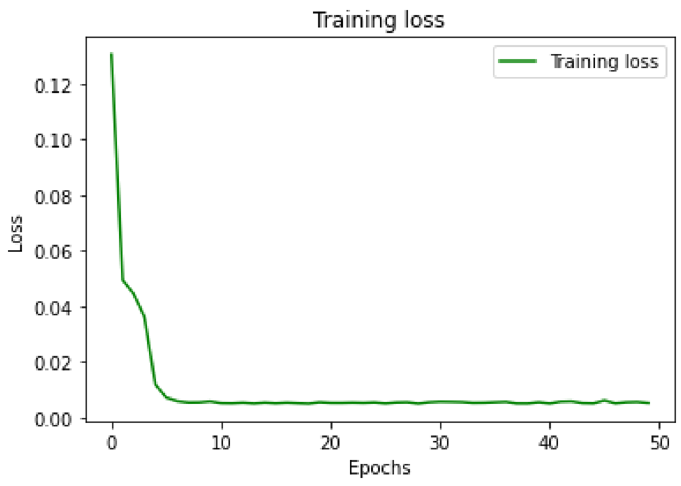
To train the model I normalize the data. Differences in the scales across the input features can make the problem more difficult to model. Large input values can lead to the model learning large weight values, which can result in an unstable model. This can also happen if the target variable has a wide range of values. Therefore, I normalize the input and output variables [1].
The one month of data is divided in two subsets. The first one hast the first 28 days and the second the last 3 days. The first subset is used as train data and the second subset is used as the test data set. Because this is a regression problem I use the mean average error (MAE), mean squared error (MSE), and the mean absolute percentage error (MAPE) [2].

[1]https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/

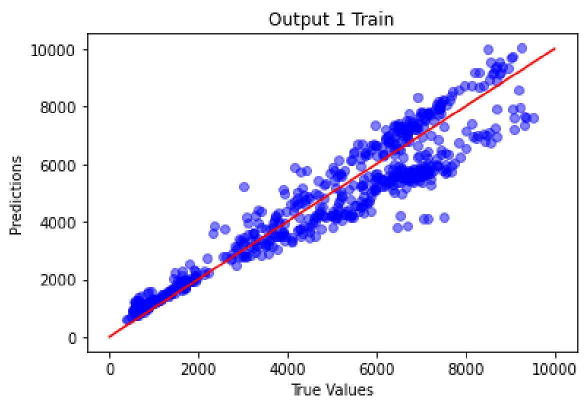
[2]https://towardsdatascience.com/4-metrics-to-evaluate-your-regression-models-885e9caeee57#: :text=One

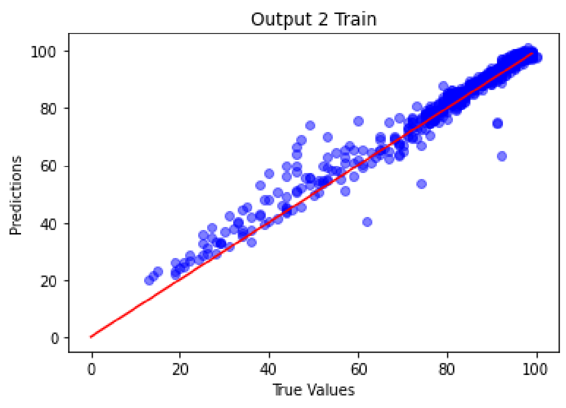
The following code shows the construction of the model, training and prediction, as well as the plot with the training loss (the Adam optimizer was used and the loss function is mean squared error):

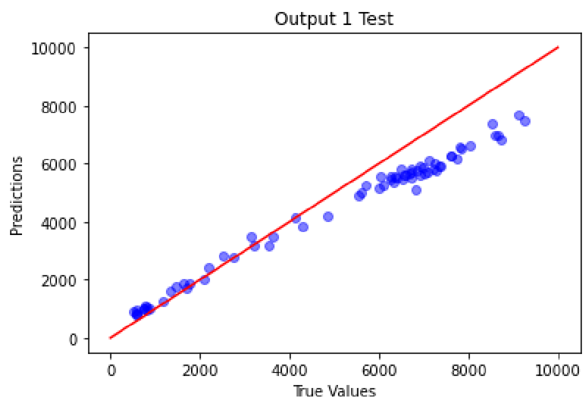
```
input_size=train_x_norm.shape[1]
output_size=train_y_norm.shape[1]
models = tf.keras.Sequential([
                              tf.keras.layers.Input(shape=(input_size,)),
                              tf.keras.layers.Dense(input_size, activation="relu"),
                              tf.keras.layers.Dense(output_size, activation='linear')
                             ])
custom_optimizer=tf.keras.optimizers.Adam(learning_rate=0.02)
models.compile(optimizer=custom_optimizer,loss='mean_squared_error')
history = models.fit(train_x_norm,train_y_norm,epochs=50, verbose =1, batch_size=10)


output_train = models.predict_on_batch(train_x_norm)

output_test = models.predict_on_batch(test_x_norm)
```

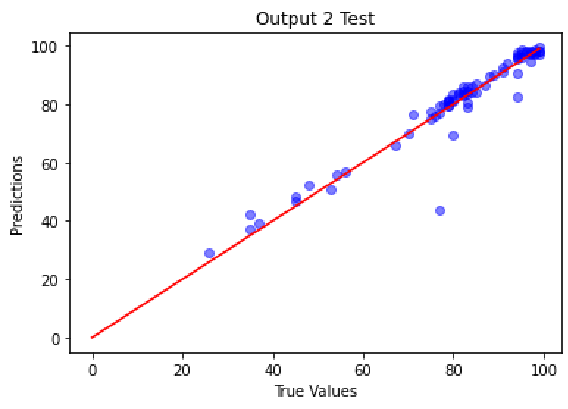


To plot values and output metrics, I use the reverse transform function to scale them back. The following plots show the predicted vs true values of the target variable for the train and test set:

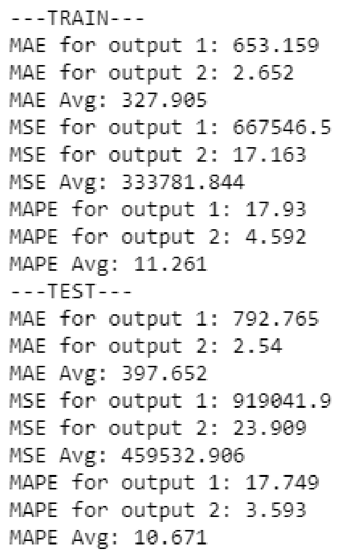








The metric values for train and test set are the following. They are showed for each output variable separately and the average.

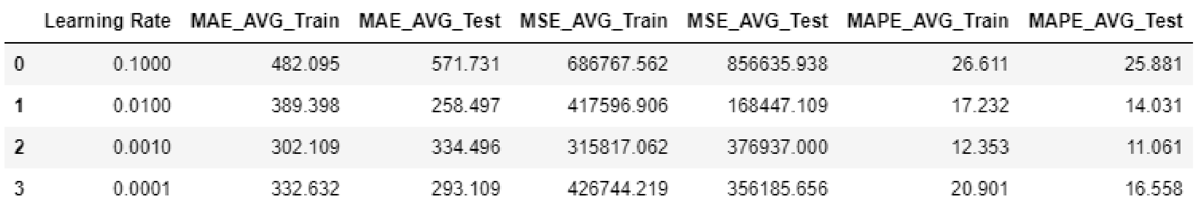


```
---TRAIN---
MAE for output 1: 653.159
MAE for output 2: 2.652
MAE Avg: 327.905
MSE for output 1: 667546.5
MSE for output 2: 17.163
MSE Avg: 333781.844
MAPE for output 1: 17.93
MAPE for output 2: 4.592
MAPE Avg: 11.261
---TEST---
MAE for output 1: 792.765
MAE for output 2: 2.54
MAE Avg: 397.652
MSE for output 1: 919041.9
MSE for output 2: 23.909
MSE Avg: 459532.906
MAPE for output 1: 17.749
MAPE for output 2: 3.593
MAPE Avg: 10.671
```

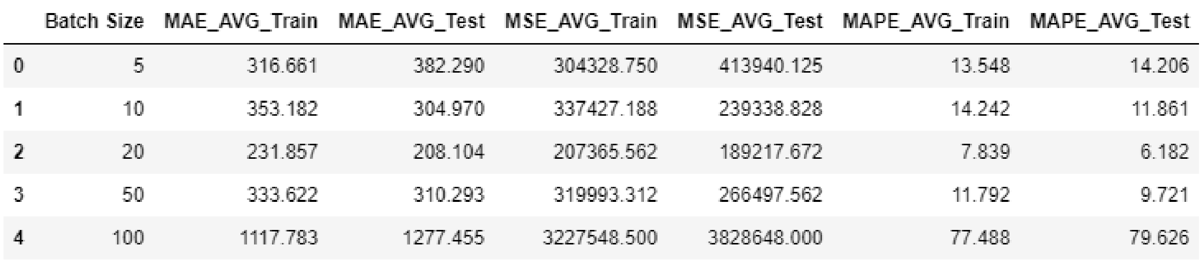
From the true value vs. prediction plots we can see that overall the predictions don't seem to extremely deviate from their true values. In particular, for output 1 the predictions tend to deviate more at higher values, and for output 2 the lower value predictions tend to deviate more from their true values.
The MAPE gives us an idea of how the error is deviating from the target in percentual terms. According to the metric values, on average the predictions for output 1 deviate by a higher percentage than for output 2. For output 2 the MAPE value is considerably low. The results indicate that the model is not over-fitting because the test set performs as well as the train set.

3.3: Adjust some of the hyperparameters of your model. Discuss the changes in performance with the assistance of tables and figures.

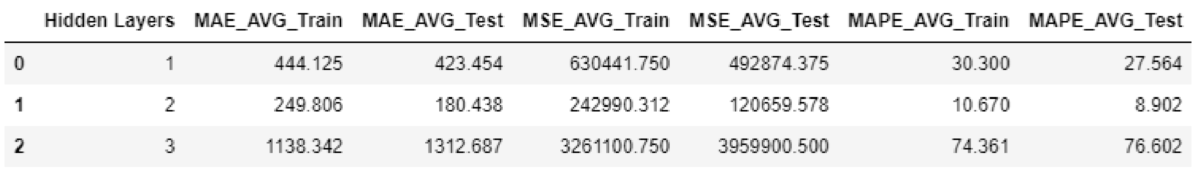
In the following tables we can see the results for the model when we vary the learning rate, batch size and number of hidden layers. The performance metrics are the same three that were previously used. The value obtained for each metric corresponds to the average for outcome 1 and outcome 2. The hidden layers that are added all have size 4 and relu activation function.



	Learning Rate	MAE_AVG_Train	MAE_AVG_Test	MSE_AVG_Train	MSE_AVG_Test	MAPE_AVG_Train	MAPE_AVG_Test
0	0.1000	482.095	571.731	686767.562	856635.938	26.611	25.881
1	0.0100	389.398	258.497	417596.906	168447.109	17.232	14.031
2	0.0010	302.109	334.496	315817.062	376937.000	12.353	11.061
3	0.0001	332.632	293.109	426744.219	356185.656	20.901	16.558



	Batch Size	MAE_AVG_Train	MAE_AVG_Test	MSE_AVG_Train	MSE_AVG_Test	MAPE_AVG_Train	MAPE_AVG_Test
0	5	316.661	382.290	304328.750	413940.125	13.548	14.206
1	10	353.182	304.970	337427.188	239338.828	14.242	11.861
2	20	231.857	208.104	207365.562	189217.672	7.839	6.182
3	50	333.622	310.293	319993.312	266497.562	11.792	9.721
4	100	1117.783	1277.455	3227548.500	3828648.000	77.488	79.626



	Hidden Layers	MAE_AVG_Train	MAE_AVG_Test	MSE_AVG_Train	MSE_AVG_Test	MAPE_AVG_Train	MAPE_AVG_Test
0	1	444.125	423.454	630441.750	492874.375	30.300	27.564
1	2	249.806	180.438	242990.312	120659.578	10.670	8.902
2	3	1138.342	1312.687	3261100.750	3959900.500	74.361	76.602

The results show the best result is achieved for learning rate 0.001, batch size 20, and 2 hidden layers. To optimize the hyper-parameters a random grid search could be performed.

Problem 4

Your local department of transportation has collected a vast number of traffic accident data. It is of great value to study whether there is correlation between accident severity and corresponding road conditions. You are asked to investigate this problem using statistical learning methods. The data1 is available in the attached zip file (Data-Problem-4.zip):

4.1:Explore and visualize the data to identify possible influencing factors, and perform data preprocessing as needed.

Influencing Factors

The first thing is to identify the different variables and whether they could be possibly influencing factors. The variables are therefore separated into those that could influence and those that would presumably not.

Target Variable: 'Accident_Severity'

Variables that are discarded from being influencing factors:

-Accident_Index: this is a unique index.

-Year: I keep day of the week and time with I think provides more information.

-Date: I keep day of the week and time with I think provides more information.

-1st_Road_Number: unclear from reference what it represents.

-2nd_Road_Number: unclear from reference what it represents.

I additionally discard Location_Easting_OSGR, Location_Northing_OSGR, and LSOA_of_Accident_Location because for location I use latitude and longitude.

Variables that can influence the problem:
'Longitude', 'Latitude', 'Police_Force','Number_of_Vehicles', 'Number_of_Casualties', 'Day_of_Week', 'Time', 'Local_Authority_(District)', 'Local_Authority_(Highway)', '1st_Road_Class', 'Road_Type', 'Speed_limit','Junction_Detail', 'Junction_Control', '2nd_Road_Class', 'Pedestrian_Crossing-Human_Control', 'Pedestrian_Crossing-Physical_Facilities', 'Light_Conditions', 'Weather_Conditions', 'Road_Surface_Conditions', 'Special_Conditions_at_Site', 'Carriageway_Hazards', 'Urban_or_Rural_Area','Did_Police_Officer_Attend_Scene_of_Accident'

Missing Values

Once this is settled, I continue the preprocessing by evaluating whether there is missing data and how to handle it.

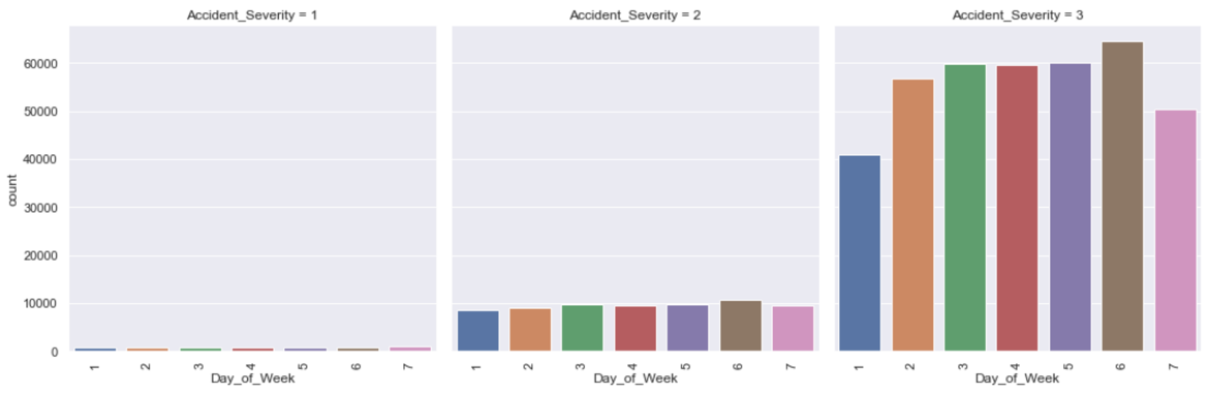
A check reveals there is missing data for the variables Time, Junction_Detail, Junction_Control, Road_Surface_Conditions, Special_Conditions_at_Site, Carriageway_Hazards, and Did_Police_Officer_Attend_Scene_of_Accident.

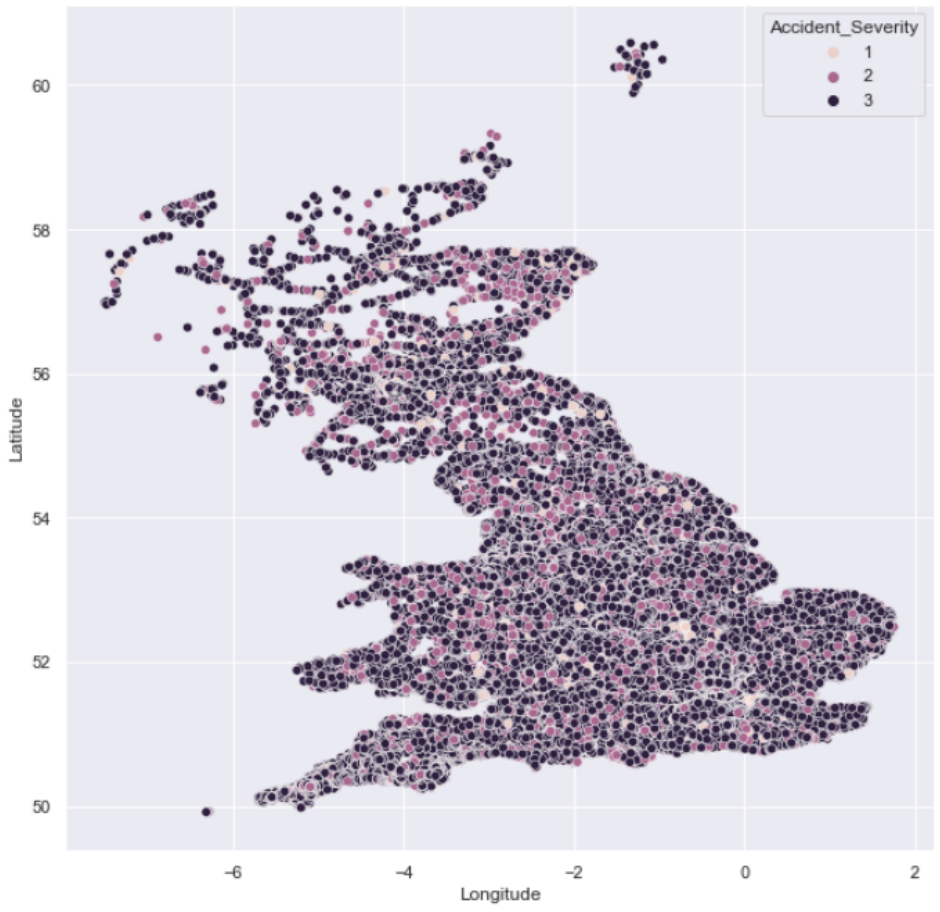
```
Longitude                                        0
Latitude                                         0
Police_Force                                     0
Accident_Severity                                0
Number_of_Vehicles                               0
Number_of_Casualties                             0
Day_of_Week                                      0
Time                                            13
Local_Authority_(District)                       0
Local_Authority_(Highway)                        0
1st_Road_Class                                   0
Road_Type                                        0
Speed_limit                                      0
Junction_Detail                             464697
Junction_Control                            178610
2nd_Road_Class                                   0
Pedestrian_Crossing-Human_Control                0
Pedestrian_Crossing-Physical_Facilities          0
Light_Conditions                                 0
Weather_Conditions                               0
Road_Surface_Conditions                        755
Special_Conditions_at_Site                       2
Carriageway_Hazards                              3
Urban_or_Rural_Area                              0
Did_Police_Officer_Attend_Scene_of_Accident      2
dtype: int64
```

- Junction_Detail has all entries as NaN, so I discard using this variable.
- To check whether Junction_Control can be an interesting value to keep, I create a sub dataframe with only Accident_Severity and Junction_Control values that are not NaN and using one hot encoding I look at the correlation between the target variable and Junction_Control. The results show that Junction_Control has almost no correlation with Accident_Severity. Therefore I drop this feature from the set.
- Regarding the remaining NaN values, because they only consitute 0.16% of the dataset I drop these rows.

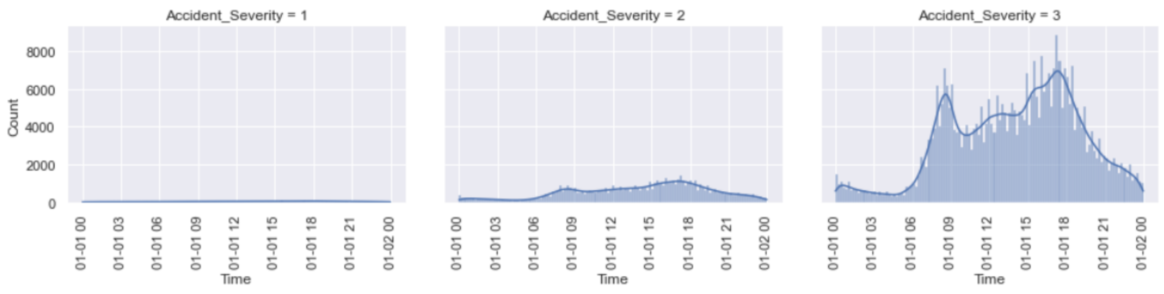
Visualization

The plots for each of the considered variables according to accident severity can be seen in the python notebook. Here I place a few examples.





For the variable time the plot is the following:



Using the plot as a guideline, I encode the time variable in 8 categories (time ranges) that comprise 3 hours each.

Encoding

Since I will be using a LGBM and this model can directly process categorical features, one-hot encoding not necessary. However, it is necessary to encode the features that have text categories.

Normalization

Normalization isn't performed because tree models don't require normalization.

Target Variable

The percentage of samples with Accident_Severity 3 is: 84 %

The percentage of samples with Accident_Severity 2 is: 14 %

The percentage of samples with Accident_Severity 1 is: 1 %

Therefore target variable is not balanced, this means we have to take special precautions when building and evaluating the model.

4.2: Build a statistical learning model to predict the accident severity using the identified features. Then, train and evaluate the model using appropriate strategies, where appropriate metrics need to be selected to measure its performance. You are free to choose any model that you believe is suitable for solving the problem.

The model selected to predict the accident severity is an LGBM. In particular, because this is an imbalanced problem there are certain particulars to take into account when training and evaluating the model.

For the model evaluation if we just look at the accuracy we may erroneously conclude the model has a high performance, because the model would predict all Accident_Severity as 3 and have an accuracy of 84%. Therefore we have to use the f1 metric to evaluate performance. I also calculate recall and plot confusion matrices.

For the training process, when I compute the train test split I use the stratify parameters so that the proportions of examples in each class in the original dataset are preserved. Otherwise, because there is a very small amount of samples for classes 1 and 2, it may result to be that there aren't any samples for these classes in the train or test set. Additionally, I use the LGBM's class_weight parameter to assign more weight to the classes with lower number of samples. I assign weights for each class as inversely proportional to the amount of samples there are for that class in the train set.

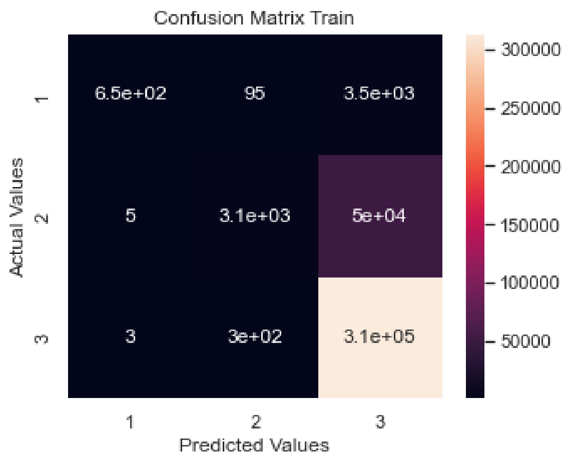
The objective of the model is set to multiclass because we have 3 different classes.

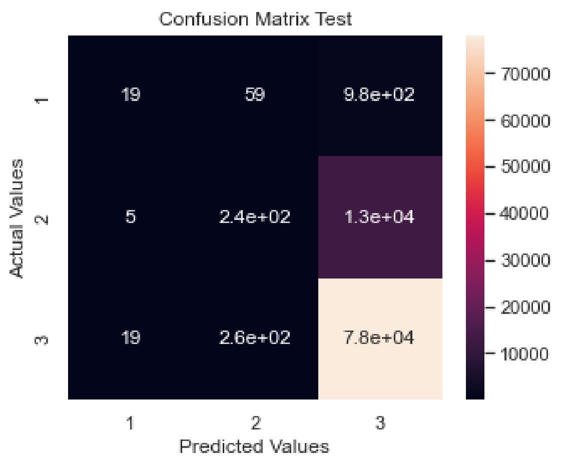
```
X = df1[cat_feat + cont_feat + location]
y = df1['Accident_Severity']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

weights = {1:(1/(y_train.value_counts().values[2]/y_train.shape[0])),
           2:(1/(y_train.value_counts().values[1]/y_train.shape[0])),
           3:(1/(y_train.value_counts().values[0]/y_train.shape[0]))}

lgb_model = lgb.LGBMClassifier(
    objective='multiclass',
    n_estimators=500,
    learning_rate=0.1,
    num_leaves=128,
    max_depth=7,
    subsample=0.9,
    colsample_bytree=0.9,
    reg_lambda=0.001,
    random_state=42,
    n_jobs=-1,
    class_weight = weights
    )

lgb_model.fit(X_train, y_train, eval_metric='logloss')

yhat_train = lgb_model.predict(X_train)
yhat_test = lgb_model.predict(X_test)

classification_metrics_report(y_train, y_test, yhat_train, yhat_test)
```

4.3: Analyze and discuss the performance of the model. Whether the accident severity can be accurately predicted? If yes, what strategies do you use to deal with the data and the model. If not, what do you think is the greatest challenge to the prediction and whether additional information might be useful to improve the performance?

The following metrics show the result for the model trained without specifying class weights.

```
===Accuracy score===
Train: 85.4
Test: 84.5
===F1 score===
Train: 0.43
Test: 0.33
===Recall===
Train: 0.40
Test: 0.34
===Confusion matrix===
```

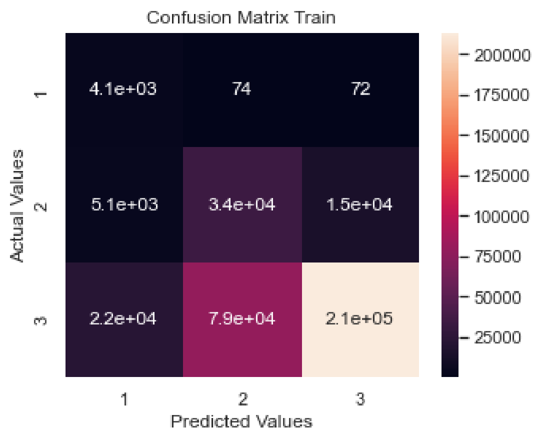


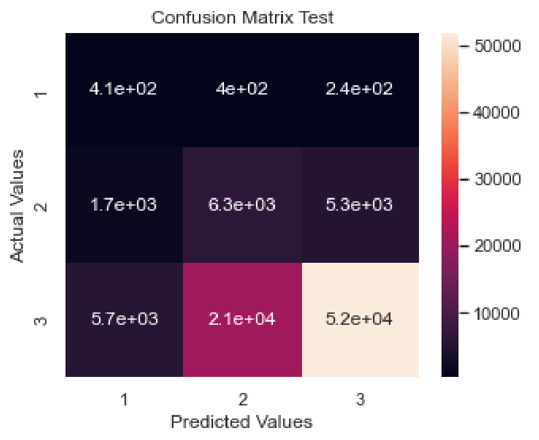


We can see how without class weights the accuracy is around 85% as was expected. However, the F1 and recall scores are much lower. The model can't classify correctly the accidents of severity 1 and 2.

The following metrics show the result for the model trained by specifying class weights.

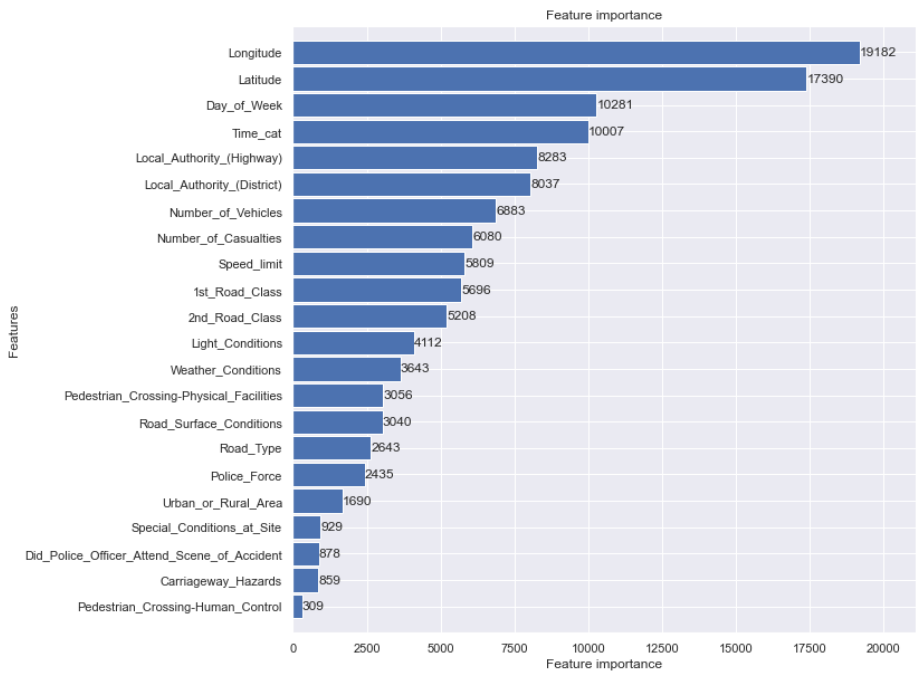
```
===Accuracy score===
Train: 67.6
Test: 63.2
===F1 score===
Train: 0.47
Test: 0.39
===Recall===
Train: 0.76
Test: 0.51
===Confusion matrix===
```





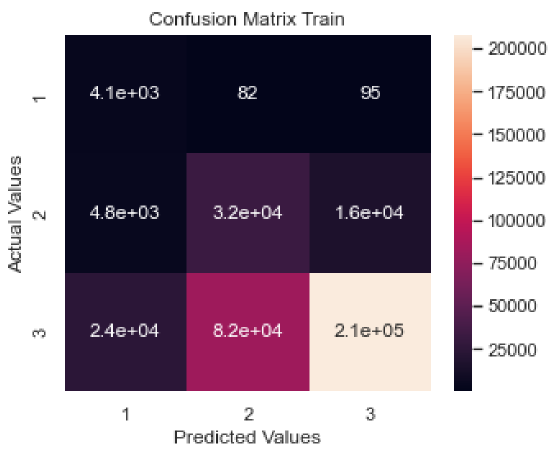
We can see how now the accuracy score has dropped to 63% for the test set, but the F1 and the recall scores have gone up. This model seems to identify better classes 1 and 2.

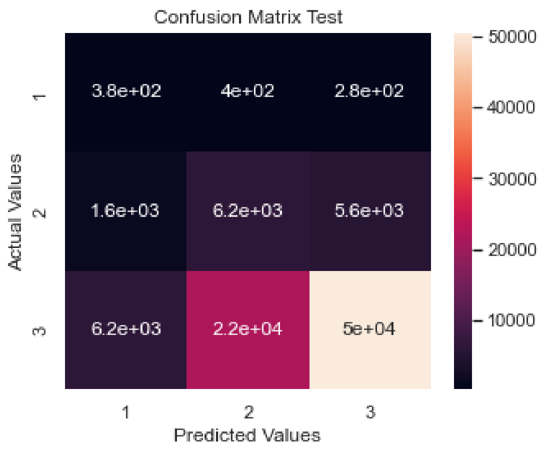
We plot the feature importances to see which ones contribute the most to the model.



I select the features with importance higher than 5000 and re-run the model to see if the scores change substantially. The results show that with these features the scores are approximately the same, which is why I only keep these features in my model.

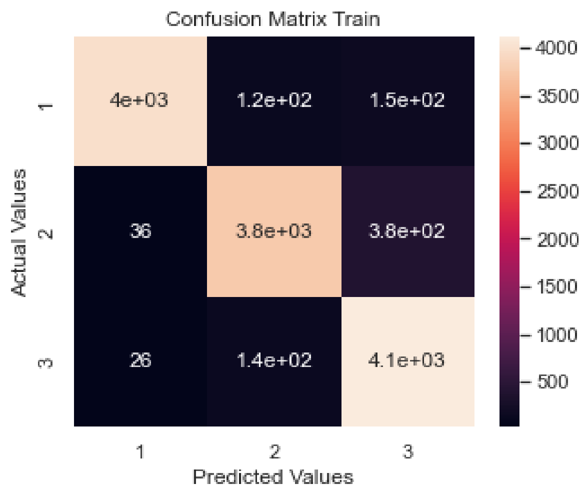
```
===Accuracy score===
Train: 65.8
Test: 61.4
===F1 score===
Train: 0.46
Test: 0.38
===Recall===
Train: 0.74
Test: 0.49
===Confusion matrix===
```

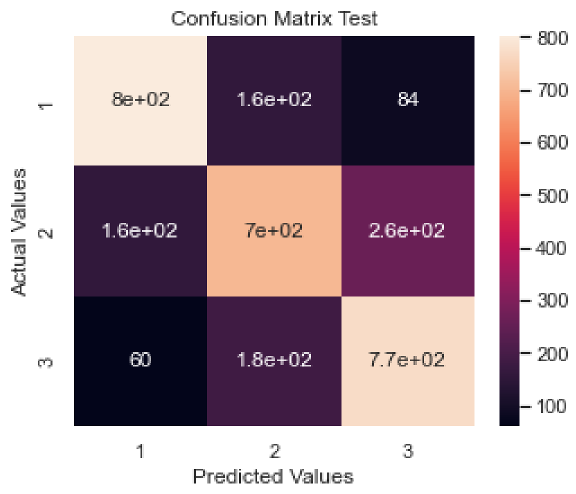




To further improve the model we can try an over- or under-sampling technique. These techniques are used for imbalanced problems. In particular, I will use the NearMiss algorithm. This is an under-sampling technique that aims to balance class distribution by randomly eliminating majority class samples. Although the class weights that we used before are in someways equivalent to applying over-/under- sampling techniques, they are different processes. So we try it to see if it improves the results.
The algorithm produces a data set with new amount of samples for the classes. The number of samples for each class is: Counter(1: 5297, 2: 5297, 3: 5297).

```
===Accuracy score===
Train: 93.2
Test: 71.7
===F1 score===
Train: 0.93
Test: 0.72
===Recall===
Train: 0.93
Test: 0.72
===Confusion matrix===
```





The metrics for performance evaluation show that the method effectively improves the results. We obtain an F1 of 0.72 for the test set.

There are other techniques and methods that can be tried to continue improving the model, although there will be a point where these methods can't improve anymore. In this case we need better data to improve the performance.