Categorical Feature Encoding Challenge

Cynthia Stephen, Shalini Ragothaman

1 Introduction

Categorical data are discrete values which belong to a specific finite set of categories or classes. Since machine learning algorithms can only process numerical data, categorical variables have to be encoded so that they can be used to extract information from a given data set.

In this challenge, a data set that contained only categorical features was presented. We were given the task of using various encoding schemes on this data and compare their performance.After exploring different encoding techniques we used different classification models and regularization techniques to obtain the most optimal model and achieve increased prediction accuracy.

2 Data Description

For this project two sets of data namely train.csv and test.csv containing categorical variables was given. The training set consisted of 300,000 rows while the test set consisted of 200,000 rows. Each set contained 24 and 25 columns of data respectively, containing binary features, nominal features, ordinal features as well as (cyclical) day and month features. The train.csv file has an additional binary [0,1] target variable whose probability has to be predicted.

Data Pre-processing
Before encoding, we performed data analysis to understand the data such as the target variable distribution, testing for cardinality and analysing the binary features present in the data.

Types of Data Present
In this project we used a data set that contained only categorical features. As stated earlier categorical variables take on qualitative values.

1. **Binary** Features
   As the name suggests, the binary features data variables contain either a 0 or 1.

2. Low and high Cardinality **Nominal** features:
   Nominal data are categorical variables that do not have any order amongst the values of that attribute

3. Low and high Cardinality **Ordinal** features:
   Ordinal data are categorical variables in which some order prevails among their values.
   Examples:
   a. Priority in terms of High, Medium, Low.
   b. Product satisfaction ranked on a scale of one to ten.

   ******Cardinality***
   Refers to the uniqueness of data values contained in a column.
   High cardinality denotes large percentage of totally unique values.
   *Low cardinality denotes lower percentage of unique values and the values repeat in its data range***

4. **Cyclical** features:
   Data is said to be cyclic when the data exhibit rises and falls that are not of fixed period.

3 Data Preparation: Encoding Techniques

Most regression or classification models require the data to be algebraic. Since categorical data mainly consist of labels, it is important to perform pre-processing on the data to convert it to numeric data. Although certain models can work directly on categorical data such as decision trees, which can learn directly from categorical data, many machine learning algorithms cannot operate on labeled data and require them to be converted to numbers. One of the key components of the Categorical Feature Encoding Challenge is to explore the various encoding techniques present to convert the categorical features to equivalent numbers for a model to operate on.

A few encoding techniques were explored and implemented in this project depending on the kind of data that is present in the data set. The advantages and disadvantages of these encoding techniques were examined and the right encoding techniques applied on the right data in the data set. All of the encoding techniques in purview were implemented using simple built-in library methods from scikit-learn using Python.

3.1 Binary Encoding:

The simplest of encoding techniques is the Binary Encoding where the categorical data predominantly consists of two different levels and can be encoded into 0 or 1. For the given dataset in the Feature Encoding Challenge, the columns, bin_0, bin_1, bin_2, consist of 1s and 0s - the data present is already encoded in binary. For the categorical columns bin_3 and bin_4, the number of levels in the data is 2, i.e., the data in bin_3 consists of either T and F representing True or False. Similarly, the bin4 data consists of Y and N - representing Yes or No. These data columns can be converted to binary data by encoding T to 1 and F to 0 while encoding Y to 1 and N to 0. This technique of converting a two-level categorical data into numerics is called Binary Encoding.

The code snippet for performing Binary Encoding on the given data:

```python
def binary_encoding(train_encoded):
    map_bin = {'T': 1, 'F': 0, 'Y': 1, 'N': 0}
    train_encoded['bin_3'] = train['bin_3'].map(map_bin)
    test_encoded['bin_3'] = test['bin_3'].map(map_bin)
    train_encoded['bin_4'] = train['bin_4'].map(map_bin)
    test_encoded['bin_4'] = test['bin_4'].map(map_bin)
    return train_encoded
```

Listing 3.1. Binary Encoding

3.2 One Hot Encoding:

One Hot Encoding is a method of mapping each category to a vector containing 0 and 1 - denoting if a particular feature is present or absent. The size of the feature vector depends on the number of categories present in the particular feature. For e.g., for a categorical variable data, particularly the nominal data present in the dataset, One Hot Encoding can be performed. The nominal data columns: nom_0, nom_1, nom_2, nom_3 and nom_4, one hot encoding is done to obtain the categorical feature representation for each level in the data column. One Hot Encoding was chosen to be implemented on this particular set of data due to several factors. The code snippets for each encoding technique described below shows the encoding technique for the training dataset only.

One important factor depends on the **Cardinality** of the data - the number of unique values present in the categorical data. For a large number of unique values or high Cardinality, One Hot Encoding fails to be efficient - more the number of unique categories in the data, the more the number of columns that will be used to encode the data, thereby occupying a large amount of space and also leads to large sparse matrices. For the mentioned nominal data columns, the cardinality was found to be less than 15, thus ensuring that One Hot Encoding does not lead to memory errors. The code snippet for performing One Hot Encoding in python is given below:

```python
from sklearn.preprocessing import OneHotEncoder
def one_hot_encoding(train_encoded):
    for col in ['nom_0', 'nom_1', 'nom_2', 'nom_3', 'nom_4']:
        onehot = OneHotEncoder()
        x = onehot.fit_transform(train_encoded[[col]]).toarray().astype('int64')
        for idx, val in enumerate(onehot.categories_[0]):
            train_encoded[col+str(val)] = x[:, idx]
    return train_encoded
```

Listing 3.2. One Hot Encoding

3.3 Feature Hashing:

As mentioned in Section 3.1, One Hot Encoding proves to be an efficient encoding technique for data that has low Cardinality. The given dataset consists of several nominal data that has a very high Cardinality, some ranging up to the value of approximately 200. One Hot Encoding does not work well in these conditions and a better encoding technique is required to be implemented in place of One hot encoding. Feature hashing is a method which converts categorical variables into a high dimensional space of integers. For this technique, the **Hash** function was used to map the categorical data of any size to a fixed size value called **hashes**. A Hash table is used internally to process the data in a computationally fast manner. Each categorical data is then identified as a particular integer value upon performing the Hash Encoding. The simple hash function was used in python to determine the hash codes for the categorical data present in the rest of the nominal data columns in the dataset. The following columns of data were used for the feature hashing: nom_5, nom_6,nom_7,nom_8,nom_9:

```python
def hashing_encoded(train_encoded):
    for col in ['nom_5', 'nom_6', 'nom_7', 'nom_8', 'nom_9']:
        train_encoded[f'hash_{col}'] = train[col].apply(lambda x: hash(str(x)) %
    5000)
    return train_encoded
```

Listing 3.3. Feature Hashing

3.4 Ordinal Encoding by Mapping:

Ordinal data represents ordered data where every category of the data has a ordered levels in the data e.g., in the data: ord_2, the categories present are:'Cold', 'Hot', 'Lava Hot', 'Boiling Hot', 'Freezing', 'Warm'. It is clear from the categories present in the data that each category has a level or an order associated with it: If the categories are ordered in terms of their levels, the data's categories can be written as: 'Freezing', 'Cold', 'Warm', 'Hot', 'Boiling Hot', 'Lava Hot'. Thus, using the Ordinal Encoder, the encoding for the categories is mapped to an integer based on the level or the order it holds in the category set. A value of 0 is set to 'Freezing' while a value of 5 is set to 'Lava Hot' and every other level in the category is given similar range of values. The following shows the mapping of the Ordinal data to their codes using python:

```python
def ordinal_encoding_1(train_encoded):
    ord_1_values = dict({'Novice': 0, 'Contributor': 1, 'Expert': 2, 'Master': 3, '
    Grandmaster': 4})
    ord_2_values = dict({'Freezing': 0, 'Cold': 1, 'Warm': 2, 'Hot': 3, 'Boiling Hot'
    : 4, 'Lava Hot': 5})
    train_encoded['ord_1'] = train['ord_1'].map(ord_1_values)
    train_encoded['ord_2'] = train['ord_2'].map(ord_2_values)
    return train_encoded
```

Listing 3.4. Ordinal Encoding by Mapping

3.5 Label Encoding:

Another technique of encoding ordinal data is using Label Encoding. It is a simple technique where the different values in the categorical data is labelled to an integer value. The data column: ord_3 consists of alphabets with lower case and ord_4 consists of alphabets in upper case. A label encoder was chosen to encode the two columns mentioned as the cardinality of the two columns is larger than the cardinality of the columns before these in ordinal data. Hence, a dictionary mapping of the ordinal data would be inefficient and computationally expensive. In Python, scikit-learn provides a library containing a method that implements the label encoding. The label encoder in python is implemented in an alphabetical order: the letter that is early in the alphabet series receives a lower label integer value while the letter higher up in the alphabet series receives a higher label integer. Thus, a label encoder from scikit-learn simplifies the process of encoding for these columns of data. The code snippet executing the encoding process is shown below.

```python
from sklearn.preprocessing import LabelEncoder
def label_encoding(train_encoded):
    for col in ['ord_3', 'ord_4']:
        labelencoder = LabelEncoder()
        train_encoded[col] = labelencoder.fit_transform(train[col])
    return train_encoded
```

Listing 3.5. Label Encoding

3.6 Encoding using an in-built Ordinal Encoder:

It was found that each value under the ord_5 column consisted of a string of two alphabet characters with some having the first character in lower case and the second in upper case and vice versa. A label encoder would under perform when encoding data consisting of a string of characters. Since this data is also a part of the ordinal data in the dataset, an Ordinal Encoder was used to encode this data. An Ordinal Encoder is another library function present in scikit-learn that performs ordinal encoding internally; it performs a similar dictionary mapping as in Section 3.4, but on a larger scale as the cardinality of the data is very large. The code snippet below demonstrates the encoding technique.

```python
from sklearn.preprocessing import OrdinalEncoder
def ordinal_encoding_2(train_encoded):
    ordencoder = OrdinalEncoder()
    train_encoded['ord_5'] = ordencoder.fit_transform(train['ord_5'].values.reshape
    (-1, 1)).astype('int64')
    return train_encoded
```

Listing 3.6. Ordinal Encoding

3.7 Cyclic Data Encoding:

The rest of the data columns present in the dataset are: 'Day' and 'Month'. The data in these columns represent cyclic data, i.e., they are periodic. 'Day' is represented by the days of the week while month are the months in consideration. For the cyclic data, the values in the column were converted to a sine and cosine data format as shown below:

```python
def cyclic_encoding(train_encoded):
    for col in ['day', 'month']:
        max_val = len(train[col].unique())
        train_encoded[col +'_sin'] = np.sin(2*np.pi*(train[col]/max_val))
        train_encoded[col+'_cos'] = np.cos(2*np.pi*(train[col]/max_val))
    return train_encoded
```

Listing 3.7. Cyclic Data Encoding

4 Classification Models

¡cite logistic regression reference¿ Once the data was processed, different classification models were explored to obtain the probability of the target value for a given set of predictor variables' values to be either [0, 1]. The Kaggle challenge provided the training set that consisted of the predictor categorical values and that target values that were either 0 or 1 for every set of predictor variables' values. Since the output, which is the target variable, was a binary 0 or 1 value, Logistic Regression is used. Logistic Regression is a model to find the probability of the target belonging to class 0 or 1 which is based on the basic logistic function given as:

$$log\frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdot + \beta_m x_m \tag{1}$$

The equation (1) holds good for multiple explanatory variables. Generally, the logit function is written as the inverse of the standard logsitic function. The function is thus written as:

$$g(p(x)) = \sigma^{-1}(p(x)) = logit(p(x)) = ln\frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdot + \beta_m x_m \tag{2}$$

The model for the given data is the case of a binary classification as only two classes exist for the data to be classified into. Here, the given sets of categorical data are the predictor variables (which are independent). The target variables can be called as the dependent variables as they depend on the combination and the values of the predictor variables. The term p(x) denotes the probability that the dependent variable equals a case given some linear combination of the predictors.

The given equation's RHS quantity consists of a number of co-efficients $\beta_0, \beta_1, \cdot, \beta_m$ where each term represents the value of the criterion to decide when the predictor will be 0 or 1. The terms β_0 represents the intercept of the regression equation line.

The project gave us an opportunity to apply the various logistic regression techniques to predict the probability of the target variable to be 0 or 1. The Kaggle Challenge expected that the final submission .csv file to contain the probabilities of the target variables for a given set of predictor variables present in the test set. Thus, the output submission file consisted of the list of IDs present in the test dataset and the corresponding target variables' probabilities.

The following subsections contain a detail of the models implemented on the dataset using packages available in python, especially scikit-learn. The models were trained on 80% of the processed data procured from the train.csv file provided and the rest 20% were used for determining the validation accuracies and losses.

4.1 Metrics Used to Evaluate the Models:

While models are designed and implemented, it is essential to determine the performance of the model on the datasets. The models are trained on the training set and the following metrics are applied on the validation sets to determine how well the particular model has been designed. However, the metrics were not applied to the test set as the test data did not consist of the target variables in order to compare the outputs obtained from the models. The predicted probabilities of the test set were directly uploaded to the Kaggle competition and the scores for the submissions were obtained. The various metrics that were used to determine the performance of a classification model implemented are listed as follows:

1. Log Loss: The logistic loss or the cross entropy loss is used in logistic regression which determines the negative log-likelihood of the true labels given a probabilistic classifier's predictions. The log loss can be defined as:
$$-log(P(y_t|y_p)) = -(y_t log(y_p) + (1 - y_t)log(1 - y_p)) \tag{3}$$
2. Accuracy: The term accuracy determines how many of the predicted classes match the true classes present in the validation set.
3. Classification Report: An overall classification report indicating the quantities: Precision, f-score, Recall were printed for most models that supported this function. Detail analysis of the terms in the classification report was not performed and only printed to provide a comparison of the reports obtained from various other models.
4. Confusion Matrix: The Confusion matrix consists of a 2x2 matrix describing how many classifications were predicted correctly and incorrectly by the classification model in the form of a numeric matrix. $\begin{bmatrix} TN\ FN \\ TP\ \ FP \end{bmatrix}$ where TN = True Negatives, FN = False Negatives, TP = True Positives and FP = False Positives. The Confusion matrix was computed to all models that supported this function.

4.2 Regularization

The concept of Regularization was explored to determine models that would classify the data best. As the models are bound to be complex with many predictor terms present, regularization is essential to help curb overgeneralizing of the data. Regularization, in simple terms, imply that a penalty is imposed on a model against complexity. Increasing the regularization strength leads to penalizing of large weight co-efficients of predictor variables, thereby disallowing the model picking up any noise present in the data or an unwarranted pattern from the data. Regularization is the idea of increasing the bias if the model suffers from high variance, or, overfits the training data. However, adding an increasing amount of regularization leads to underfitting of the data - due to a large bias - which is also not preferred.

Regularization is added to an unregularized model to minimize the cost function. In general, the regularization concept can be written in terms of the cost function equation as follows:

$$J(w) = \sum_{i}^{n}[-y^i log(\phi(z^i)) - (1 - y^i)log(1 - \phi(z^i))] + \lambda R(f) \tag{4}$$

where the first portion of the equation represents the loss and the regularization terms is R(f) which is added to the loss function with a proportion of λ.
L2 Regularization: The L2 regularization model is known as Ridge Regression where the squared magnitude of the coefficient is added as the penalty to the loss function mentioned in the above equation. It is represented as:

$$J(w) = \sum_{i}^{n}[-y^i log(\phi(z^i)) - (1 - y^i)log(1 - \phi(z^i))] + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{5}$$

If $\lambda = 0$, the model would function as one without any regression.
L1 Regularization: Models implementing the L1 regularization are known as Lasso Regression, which stands for Least Absolute Shrinkage and Selection Operator, add an absolute value of the magnitude of the coefficient as the penalty term to the loss function:

$$J(w) = \sum_{i}^{n}[-y^i log(\phi(z^i)) - (1 - y^i)log(1 - \phi(z^i))] + \lambda \sum_{j=1}^{p} |\beta_j| \tag{6}$$

It is important to note that the Lasso helps in selecting the right features that are important and shrinks away those that do not contribute to the performance of the model by literally shrinking the corresponding coefficients to zero. Thus, the acronym and it can be seen in further sections, the working of the two regularization parameters on the models.

4.3 Cross Validation

Cross Validation is used for estimating the performance accuracy of a predictive model, thereby helping us assess the quality of the model. In Cross Validation we seek to define a data set to test the model in the training phase. The best model will perform well in both, the testing set and in the training set and will overcome overfitting and underfitting data.

A function for the regression models was written where the given model was initialized and fit to the training data. The fit data was used to make predictions on the validation set and the metrics were printed. The probabilities of the target variables were saved into the corresponding submission.csv file as required by the Kaggle competition.

```python
def perform_regression(X, X_val, Y, Y_val, X_test, model, model_name):
    model.fit(X, Y)
    Y_pred = model.predict(X_val)
    # Print statistics of the fit model
    metrics = print_metrics(model, Y_val, Y_pred)
    # Store the predicted target values into a csv file
    predicted_class_prob = model.predict(X_test)
    # Write the predicted class probabilities into a .csv file
    pd.DataFrame({"ID": X_test["id"], "Target:": predicted_class_prob}).to_csv("submission_"+str(model_name)+".csv", index = True)
    # Write the metrics into a csv file
    df = pd.DataFrame.from_dict(metrics, orient="index")
    df.to_csv("metrics_"+str(model_name)+".csv", index = True)
    return
```

Listing 4.1. Classification model fitting and prediction

4.4 Logistic Regression with L2 Penalty

A simple logistic regression with the L2 penalty (which is a default feature in scikit-learn's Logistic Regression function) was implemented. The amount of the regularization - the parameter λ in scikit-learn is represented as C, which is translated as $C = \frac{1}{\lambda}$. For this model, the value of C was set to 1 (which is the default value). The following were the results of the metrics obtained.

```
C = 1, maxiter = 100,  penalty = l2

Log Loss: 10.741000417517276

MSE: 0.31098333333333333

Accuracy: 0.6890166666666667

Classification Report:

                precision  recall  f1-score  support
      0            0.70     0.97     0.81     41677
      1            0.42     0.05     0.09     18323
   accuracy                          0.69     60000
  macro avg        0.56     0.51     0.45     60000
weighted avg       0.61     0.69     0.59     60000
Confusion Matrix:

[40431 1246]
[17413  910]
```

4.5 Logistic Regression with L1 Penalty

In place of the L2 regularization technique, the L1 penalty was imposed onto the model with C = 1. The model internally changes its solver for every regularization technique implemented. The results of the L1 penalty Logistic Regression is as follows:

```
C = 1, maxiter = 100, penalty = l1

Log Loss: 8.778363401242154

Accuracy: 0.727

Classification Report:

               precision recall f1-score support
      0           0.77     0.90    0.83    41677
      1           0.64     0.39    0.48    18323
   accuracy                        0.75    60000
  macro avg       0.71     0.65    0.66    60000
weighted avg      0.73     0.75    0.73    60000
R-squared value: -0.19269662514298447

Confusion Matrix:

[37702 3975]
[11205 7118]
```

4.6 Logistic Regression with L2 Penalty and Cross Validation

The Logisitic Regression model as above was implemented with cross validation - by splitting the training data into 3 and 5 folds using the module LogisticRegressionCV imported from the scikit-learn library. For the 3 fold cross validation implementation, the following results were obtained:

```
C = 0.00077426, maxiter = 100, penalty = l2

Log Loss: 10.62183690800337

Accuracy: 0.6924666666666667

Classification Report:

               precision recall f1-score support
      0           0.7      0.98    0.82    41677
      1           0.46     0.04    0.08    18323
   accuracy                        0.69    60000
  macro avg       0.58     0.51    0.45    60000
weighted avg      0.63     0.69    0.59    60000
Confusion Matrix:

[40786 891]
[17561 762]
```

The model picks out the best C hyperparamter values and it was found to be C = **0.00077426**. For the 5 fold cross validation, the results obtained were not drastically different or improved. An accuracy of **0.689** and a loss of **10.713** was obtained.

4.7 Logistic Regression with L1 Penalty and Cross Validation

Using the Lasso Regression with CV package in scikit-learn, the models were fit and evaluated. The results were obtained as follows for a 3 fold Cross Validation implementation:

```
C = 0.0625, maxiter = 1000, l1 penalty

Log Loss: 8.74181722557515

MSE: 0.2531

Accuracy: 0.7469

Classification Report:

              precision recall f1-score support
      0         0.77     0.90    0.83    41677
      1         0.64     0.39    0.48    18323
   accuracy                      0.75    60000
  macro avg     0.71     0.65    0.66    60000
weighted avg    0.73     0.75    0.73    60000
Confusion Matrix:

[37706 3971]
[11215 7108]
```

The Lasso Regression CV model selected a model whose C value was equal to 0.5.

4.8 Other Models

Lars was implemented using scikit-learn on the given data, however, it was seen that the score or the R^2 value computed was very low, almost equal to 0. Log Loss: 0.5632982758875197 Score: 0.18193051854419262 Another similar model - Elastic Net - was implemented which also showed a very low R^2 value.

5 Analysis of Implemented Models

From the described classification models implemented, it is evident that regularization played a vital role in the classification performance of the various models. With a decreased regularization parameter, the accuracy scores were seen to be lower and the loss higher. As the amount of regularization increased, the bias was added proportionally that improved the performance of the models. The table below summarizes the classification models with varying regularization parameters.

Model	$C = \frac{1}{\lambda}$	**Score**	**Loss**
Logistic Regression with L2 Penalty	1	10.741	0.689
Logistic Regression with L2 Penalty and CV	0.25	10.62183	0.6924
Logistic Regression with L1 Penalty	1	8.77	0.727
Logistic Regression with L1 Penalty and CV	0.0625	8.7418	0.7469

Table 1. Summary of metrics of the classification models

Due to the large complexity of the model and its data, it is seen that the validation scores do not reach higher than a certain value of 0.74 for the particular sets of training and validation sets. The large amount of sparse data present in the training and validation sets could lead to the misclassification rates that are prevalent when evaluating the models. As the value of C is decreased, the amount of bias added to the models increases, thereby reducing the overfitting of the models to an extent - this is evident in the results obtained. It is also important to note that the Lasso regularization terms helps in reducing the overfitting by a large extent when compared to the Ridge Regression term. Thus, Lasso helps in reducing the complexity of this overly complex data and models to obtain an efficient representation of the model.

It is unclear to note which of the predictor variables show a higher statistical importance to the model. According to the Kaggle's challenge data description, the data does not necessarily have an interpretable meaning associated with it; it is difficult to understand the meaning of the co-efficients obtained when fitting the model to the data as the data is a synthetic set of data without any real-world meaning associated with it to make meaningful inferences to help interpret the classification obtained from the models.

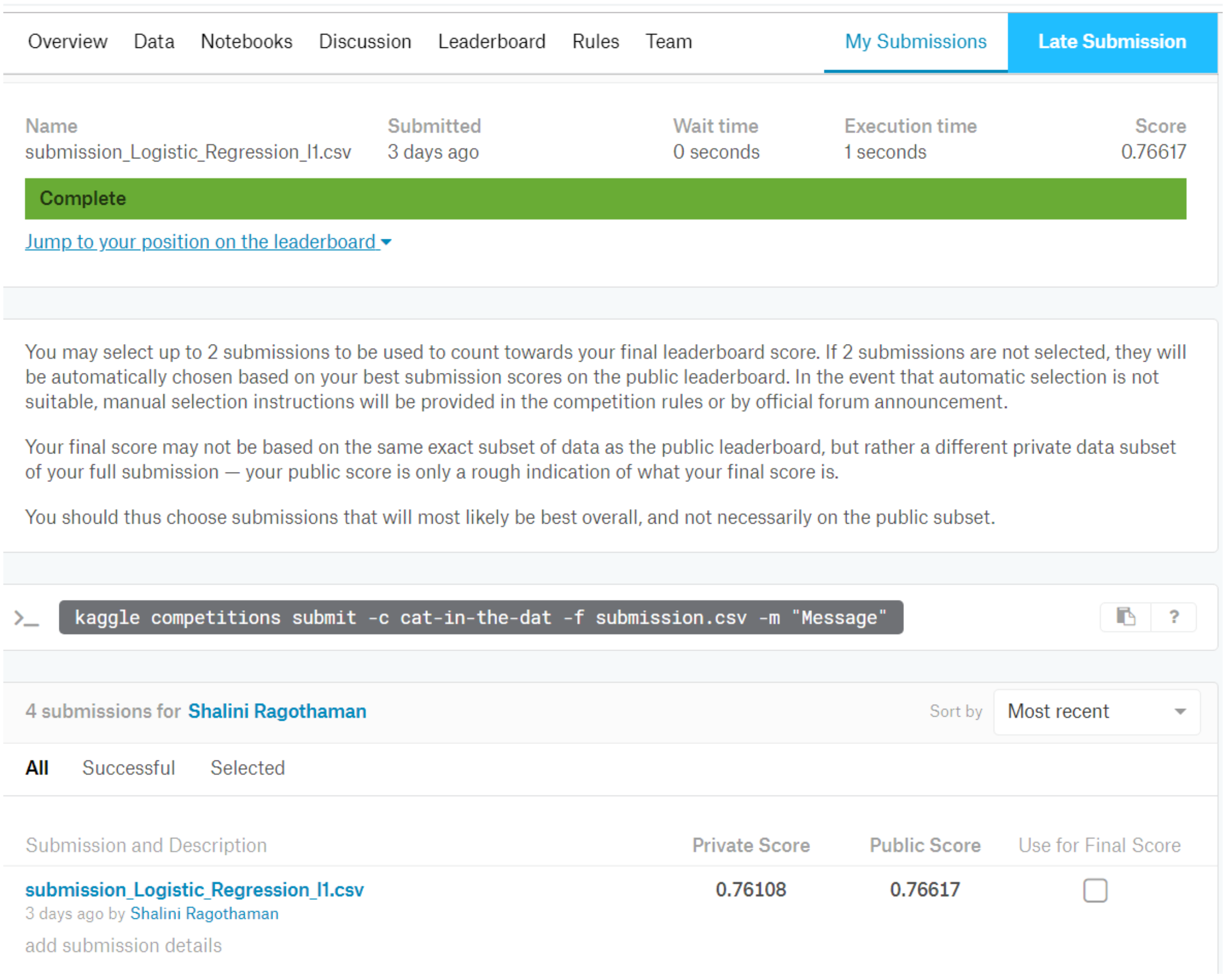
5.1 Use of Interaction Terms

The models fit on the data provide a fair accuracy score. While the data on the whole is synthetic and does not have any real meaning with each column, except for a few ordinal data columns, it would be interesting to note the performance of the models that include interaction terms between the predictor variables. In order to identify interaction terms we evaluated the correlation between each of the pairs of variables. However, our results showed that there were no pairs of variables whose correlation was statistically significant. In fact, the value of correlation between each of the data columns was insignificant that it can be concluded that the interaction between the variables is nearly nil. Hence, we concluded that there could be **No** interaction terms, and adding a few would not improve the current models' fits, and this claim is supported by the Kaggle community. The Kaggle challenge's data description page also states that there is no relation between the predictor variables.

6 Kaggle Results

The submission.csv files were uploaded to the Kaggle Challenge and the top two models were evaluated on the test set owned by the Challenge and picked. For the Logistic Regression with Cross Validation using the L2 norm, a score of 0.5809 was obtained. The top score was obtained by the Logistic Regression with Cross Validation and L1 norm - a public score of 0.76617 was obtained on the test set.

The scores depend on several factors - several predictor variables may not have been statistically significant but for all the models implemented, all of the predictor variables were retained. Simple encoding techniques were used to encode the categorical data and complex encoding methods were avoided.



Name	Submitted	Wait time	Execution time	Score
submission_Logistic_Regression_l1.csv	3 days ago	0 seconds	1 seconds	0.76617

Complete

Jump to your position on the leaderboard

You may select up to 2 submissions to be used to count towards your final leaderboard score. If 2 submissions are not selected, they will be automatically chosen based on your best submission scores on the public leaderboard. In the event that automatic selection is not suitable, manual selection instructions will be provided in the competition rules or by official forum announcement.

Your final score may not be based on the same exact subset of data as the public leaderboard, but rather a different private data subset of your full submission — your public score is only a rough indication of what your final score is.

You should thus choose submissions that will most likely be best overall, and not necessarily on the public subset.

```
>_ kaggle competitions submit -c cat-in-the-dat -f submission.csv -m "Message"
```

4 submissions for Shalini Ragothaman

Sort by Most recent

All Successful Selected

Submission and Description	Private Score	Public Score	Use for Final Score
submission_Logistic_Regression_l1.csv 3 days ago by Shalini Ragothaman add submission details	0.76108	0.76617	☐

Fig. 1. Kaggle Submission Results

The top score on Kaggle was 0.80283 for this challenge. Several other encoding techniques such as Temperature Encoding and others were implemented by the top teams in the leaderboard.

7 Conclusion

The Categorical Feature Challenge was a learning platform to explore and implement classification models on categorical data. Various Logistic Regression models were thoroughly analysed and regularization metrics were examined. With the help of cross validation, the final result proved to be higher than the average score submitted in the challenge. The project, overall, helped understand the significance of hyperparameters and model finetuning techniques that are vital in any real world scenario - along with model fitting and analysing the results on the data, optimization of the fit models is equally challenging and interesting.

References

D, C., et al. (2017). "A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers." International Journal of Computer Applications 175(4): 7-9.

Fitkov-Norris, E., et al. (2012). "Evaluating the Impact of Categorical Data Encoding and Scaling on Neural Network Classification Performance: The Case of Repeat Consumption of Identical Cultural Goods", Berlin, Heidelberg, Springer Berlin Heidelberg

Peng, C.-Y. J., et al. (2001). "Modeling Categorical Variables by Logistic Regression." American Journal of Health Behavior 25(3): 278-284.

https://www.kaggle.com/c/cat-in-the-dat/data

https://medium.com/hugo-ferreiras-blog/dealing-with-categorical-features-in-machine-learning-1bb70f07262d

https://www.techopedia.com/definition/18/cardinality-databases

https://www.kdnuggets.com/2016/08/include-high-cardinality-attributes-predictive-model.html

https://towardsdatascience.com/understanding-feature-engineering-part-2-categorical-data-f54324193e63

https://towardsdatascience.com/smarter-ways-to-encode-categorical-data-for-machine-learning-part-1-of-3-6dca2f71b159

https://towardsdatascience.com/encoding-categorical-features-21a2651a065c