Otto Group Product Classification

Introduction to Machine Learning Applications: MGMT 6560-02

Professor: Thomas Morgan
Author: Brendan Donnelly
Date: December 13, 2020

Contents

Executive Summary:

Supply chains in the e-commerce age have faced a massive overhaul due to the revolutionizing of the traditional brick and mortar tiered pathways to reach the customer. The modern e-commerce supply chain lead by groups such as Alibaba, Amazon, and Otto Group have reached remarkable success in gaining value through the improvement of distribution to the customer through direct transportation from their warehouse networks. The role of warehousing management, and product tracking is critical for these companies as the warehouses grow and pathways for customer specific delivery increase.

The Otto GmbH & Co KG (Otto Group) is an e-commerce company based in Germany. As one of the largest mail order companies in the world, this company has amassed a significant number of retail subsidiaries including About You, Manufactum, OTTO, Crate and Barrel, Eddie Bauer Japan, 3 Suisses, and more. At this level, one of the main goals for the Otto Group is in optimizing their supply chain among their subsidiaries. Due to the Otto Group's global structure and scale there are issues in consistent classification of their goods. Language difference, differing company procedure and differing enforcement in the tagging of products, and more can be bottlenecking their products and reducing their sales. As a result Otto Group developed a product classification challenge. Given an obfuscated example training set of features and class groupings of their products and a test data set with only product features, the group was looking to find the best model to classify their products correctly into class categories.

Benchmarking:

The performance of this challenge was determined through a multi-class logarithmic loss function per class prediction. As the training dataset was 42% the size of the testing set there was a significant challenge to not over-fit the model determined with the training set. The logarithmic loss function was utilized to measure performance of the models. By minimizing the "Log Loss" the algorithm is equivalently maximizing the accuracy however it should be noted that it assigns higher penalties to more confident incorrect classifications [Collier]. The highest performing notebooks on the leaderboard utilized highly complex approaches, extensive hyper parameter tweaking loops and ensemble methods to yield an impressive minimal log loss for the top 50 ranging from .38 to .411. The following benchmarked notebooks represent some of the highest achieving models that have an available training and testing performance metrics spanning a variety of feature and model approaches:

Table 1: Benchmarked Solutions

Notebook Name	**Feature Approach**	**Model Approach**	**Train/Test Perf (Log Loss)**
Feature Extraction TFIDF	TFIDF	LightGBM	.461 , 0.441
Otto NeuralNetwork	LabelEncoding, StandardScaler	Tensorflow, StratifiedKFold	0.508, 0.483
TC1-project OTTO XGBoost	StandardScaler	KGBoost	.473, 0.435

Varying Feature Approaches:

The first benchmark, “Feature Extraction TFIDF” utilized an information retrieval, term frequency-inverse document frequency, feature approach. This approach is often used to reflect on word importance in documents for Natural Language Processing areas. Applying this feature extraction commonly used for text for the product features was particularly savvy. This feature approach is commonly utilized for classifying text such as consumer complaints into predefined categories and should work well in a product classification objective.

The following benchmarks, “Otto NeuralNetwork” and “TC1-project OTTO XGBoost” both use a scaling feature approach. They each scale the data in preprocessing through standardization based around a mean and standard deviation. The StandardScaler function is applied feature-wise to multivariate data; independently for each column of the data. Given the distribution of the data, each value in the dataset will have the mean value subtracted, and then divided by the standard deviation of the whole dataset (or feature in the multivariate case). This scaling only occurred through the subset of feature columns in both notebooks. In addition, label encoding was used for the KFold model to convert the target of type string indicating the class to a machine readable numeric form.

Varying Model Approaches:

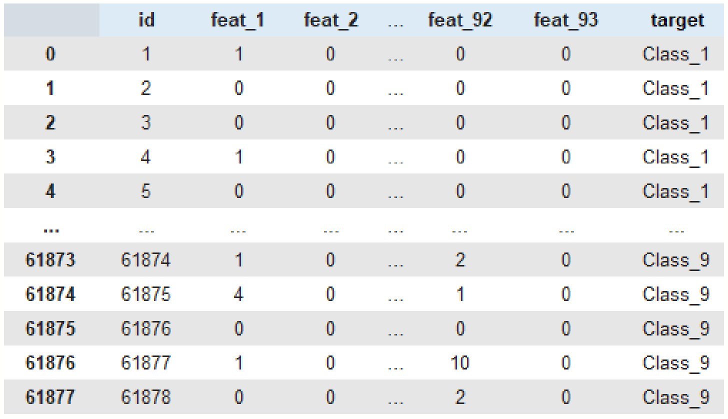
A LightGBM (Light Gradient Boosting Machine) model approach was taken for the first benchmarked notebook. Compared to other Gradient Boosting methods such as XG Boost, Light GBM grows trees leaf-wise rather than level-wise. A light GBM model determines the leaf that it believes yields the largest decrease in loss develops the tree from there. After applying a TFIDF transformation, this model split the training data 70-30 into a train and validation dataset. It than developed a LGB model through 500 rounds of boosting leading to a best training iteration of .46 log loss. This model yielded insight into their transformed feature importance listing the TFIDF transformed feat_25, feat_67, feat_86, feat_24, and feat_48 as the most important features determined from the LGB model.

The Tensorflow model used a sequential stack of linear arrays developed through batch normalization, and modifying densities from the input shape, and input number of classes. The feature columns from the train data were Standard Scaled and id column was dropped. A KFold model in conjunction was looped to find the optimal number of n folds and together a batch of simulations were tested resulting in log loss of around .7 to .5.

The XGBoost model was the last benchmarked model using the XGBClassifier from sklearn and the random over sampler function from imblearn to test an overfitting of the model. The data was scaled and label encoded similarly to the previous models and then the training data was split 80-20 with a stratified shuffle split function. PCA was actually considered for this particular notebook however with 95% of the variance explained by 77 of the 93 components the XGBoost method was applied. After parameter tuning done through Grid Search, the XGBoost model improved on the training set from a log loss of .64 to yielding .43.

Data Description and Initial Processing:

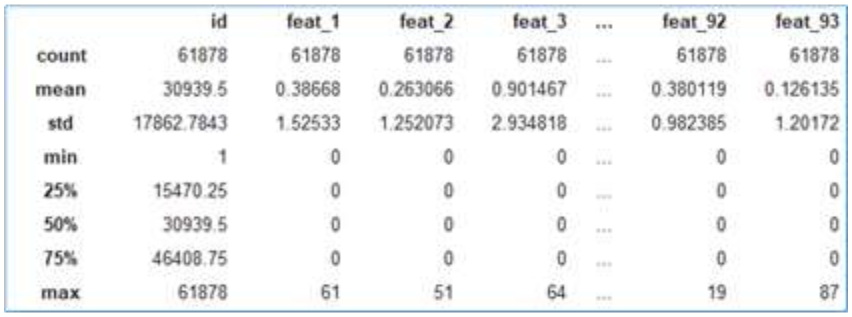
The data is composed of three csv files: test, train, and a sample submission. The train file has columns: id, feat_1 through feat_99, and a target. Each row represents one product in the holdings of an Otto Group subsidiary. As the data is read through the pd.read_csv() function the dataset becomes a panda.core.frame.DataFrame. Each column name starting with 'id' is a pandas core series. The data within this first series is of type numpy.int64. This 'id' column represents the product id and is simply the row number from 0 through 61877; the total number of rows within the train dataset. The features formatted 'feat_n' of n 1 through 93 house values indicating the levels of a specific feature in the product and are also a pandas series housing numbers of type numpy.int64. These features were purposefully obfuscated to add to the challenge, as were the target classes. The 'target' column in the training dataset is a pandas series holding string values formatted as 'Class_n' with n being a number from 1 to 9. These classes can be assumed to be of categories similar to fashion, electronic goods, and furniture based upon the graphic from the description and the nature of Otto group's subsidiaries.



	id	feat_1	feat_2	...	feat_92	feat_93	target
0	1	1	0	...	0	0	Class_1
1	2	0	0	...	0	0	Class_1
2	3	0	0	...	0	0	Class_1
3	4	1	0	...	0	0	Class_1
4	5	0	0	...	0	0	Class_1
...	...	...	...	...	...	...	...
61873	61874	1	0	...	2	0	Class_9
61874	61875	4	0	...	1	0	Class_9
61875	61876	0	0	...	0	0	Class_9
61876	61877	1	0	...	10	0	Class_9
61877	61878	0	0	...	2	0	Class_9

Figure 1.Train Data

Above, one can see a scaled down view of the Train dataset. This variables of this train dataset were than broken down by percentiles, counts, means, and standard deviations to get a better perspective on the breadth of this data. Overall, the sparsity of certain areas as a high amount of zero values were apparent from these early visualizations.



	id	feat_1	feat_2	feat_3	...	feat_92	feat_93
count	61878	61878	61878	61878	...	61878	61878
mean	30939.5	0.38668	0.263066	0.901467	...	0.380119	0.126135
std	17862.7843	1.52533	1.252073	2.934818	...	0.982385	1.20172
min	1	0	0	0	...	0	0
25%	15470.25	0	0	0	...	0	0
50%	30939.5	0	0	0	...	0	0
75%	46408.75	0	0	0	...	0	0
max	61878	61	51	64	...	19	87

Figure 2.Train Data Descriptors (pre removal of ‘id’)

This is further shown by the significant amount of variables that have zeroes listed through their 75th percentile.

The testing data has all of the same series types in the panda core data frame however it does not have a target value listing the class of the item and it contains significantly more products with a total of 144368 rows. The output to be sent in to this competition is a csv file with columns: id, Class_1, Class_2, Class_3… Class_9. For each id, a probability from 0 to 1 inclusive has to be set per each Class variable. The sum of the probabilities through all of the classes has to equal 1 for each product id.

In initial preprocessing, the datasets were searched and found to have no null values. The unique target values were found to be Class’s 1 through 9 formatted as ‘Class_n’. The head and tail visualizations of the train dataset yielded what was close to a binary table as most of the features appeared to be 1 or 0 however there were some variants with occasionally higher integer values. The class variable was changed through a label encoder to integer from 0 to 8. This enabled a quick visualization which counted the total number of each class in the train data frame with a seaborn countplot.

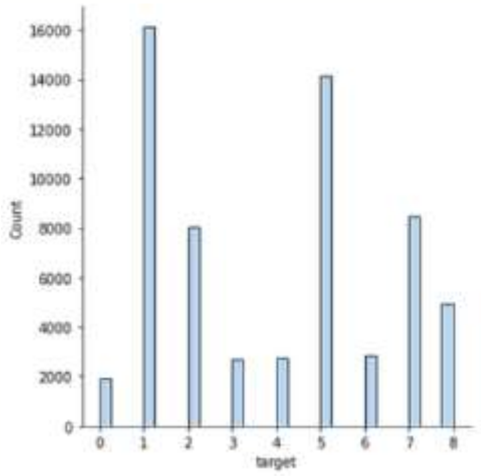


Figure 3. Count of Class Targets in Train Dataset

This plot yields some key intuition that Class_2 and Class_6 (here listed as target 1, 5) accounted for a large amount of the products in the training set. With Class_3 and Class_8 also accounting for around 8 to 10 thousand of the 61,877 products in the training dataset.

Analysis of Relevance of Independent Variables:

Analysis of Feature Variables Towards Target

To gain more insight into the correlation of the many features, the data was plotted into a heat map based on correlation of Training variables including all of the features, and the target value.

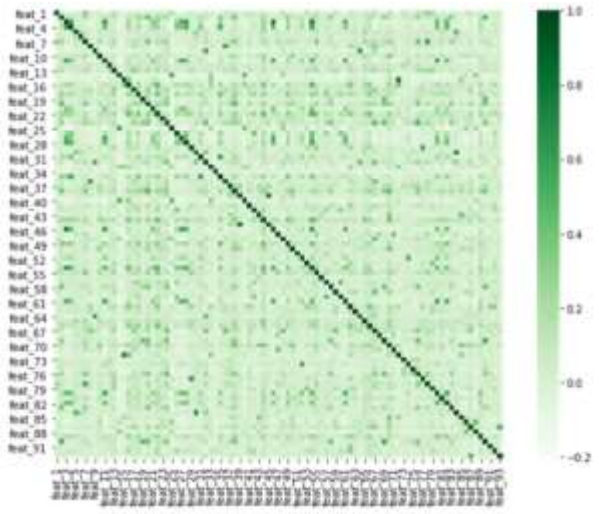


Figure 4.Correlation Plot of Train Dataset

An additional correlation matrix was plotted which indicated only correlations greater than an absolute value of .25. The following result showed that these correlations were relatively small and randomly distributed throughout the data set.

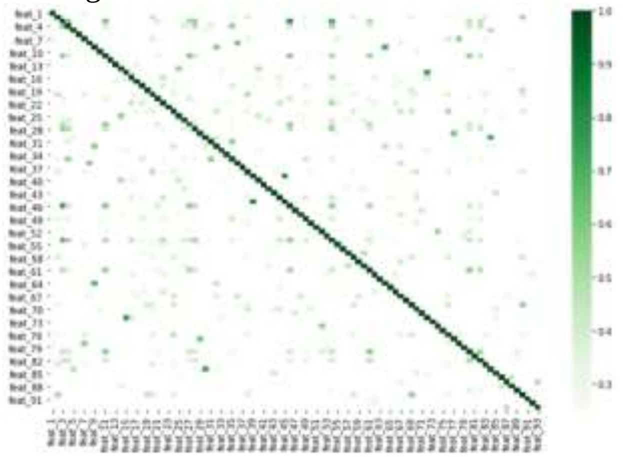


Figure 5.Correlation Plot of Correlation > .25 in Train Dataset

These obfuscated features showed no signs of strong correlation towards target/class values so steps in preprocessing had to occur to gain more insight on the features.

Pre-processing, Model Specific Feature Importance:

For the generated LightGBM model, features were processed with TFIDF (term frequency-inverse document-frequency) pre-processing. The importance of each feature in relation to the model output was plotted after the model was trained for 200 iterations with a validation set. The feature values are plotted by importance below.

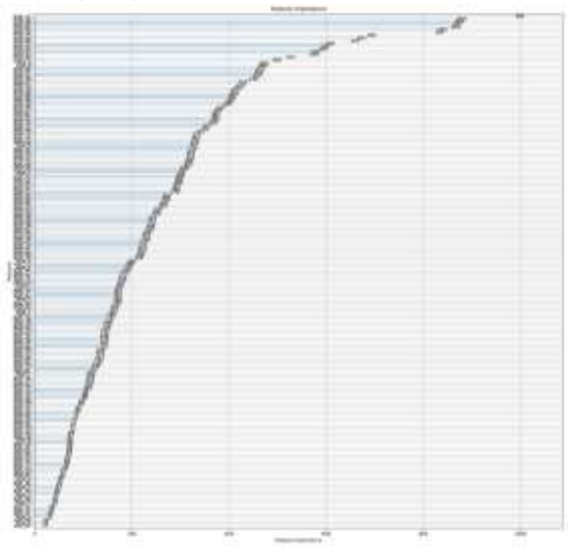


Figure 6. LightGBM Model Feature Importance (Whole Set)

This plot serves to outline the differences in the feature importance throughout every feature. This importance is mostly smooth and curved, and has a few sudden jumps around the 10 most important preprocessed features towards the model.

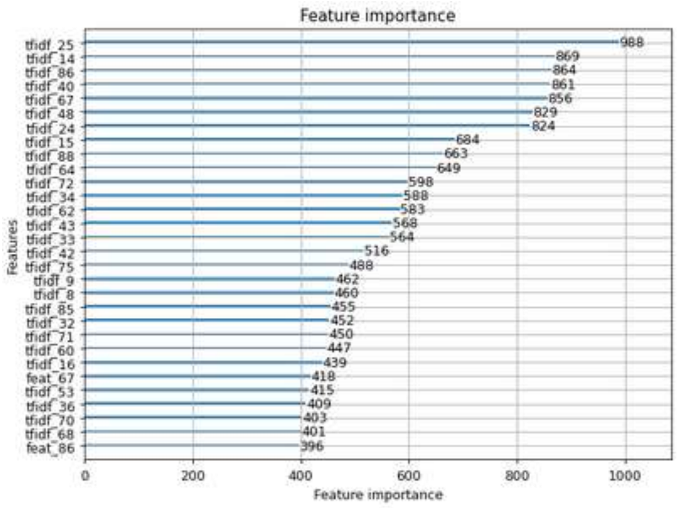


Figure 7. LightGBM Model Feature Importance (Top 25)

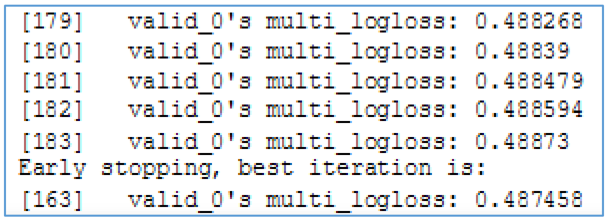
This top feature importance plot demonstrated a high importance on the feature values (25, 14, 86, 40, 67, 48, and 24). These values all seem scattered throughout the range of the data set and with the obfuscation, and low correlations they could not have been found without the LightGBM model's utilization of the plot_importance function.

Analysis of Performance of Different Model Types:

The models were each chosen due to their intermediate sophistication and incorporation of neural networks. A few alternative, more sophisticated, models such as XGBoost and similar methods were tinkered with and were attempted however difficulties in the domain such as in GPU, and CPU allocation led to the selection of more manageable classification models. Overall these models performed comparatively to the neural network benchmarked solutions.

Light Gradient Boosting Machine

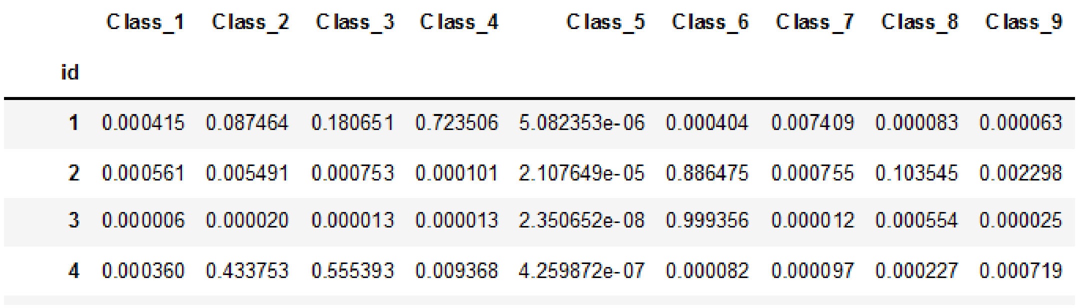
The initial modeling was done with a LightGBM model and utilized a TFIDF feature approach. The hyper parameter tuning of the previous model was scratched and the LightGBM model was used with mainly default values with the exception of the tinkering of the feature_fraction parameter from values 0 to 1. Surprisingly even with this introductory level tuning the model still performed highly and these changed yielded a delta of only -.02 in the log loss performance.



```
[179]   valid_0's multi_logloss: 0.488268
[180]   valid_0's multi_logloss: 0.48839
[181]   valid_0's multi_logloss: 0.488479
[182]   valid_0's multi_logloss: 0.488594
[183]   valid_0's multi_logloss: 0.48873
Early stopping, best iteration is:
[163]   valid_0's multi_logloss: 0.487458
```

Figure 8: Model Training of LightGBM Model

Early stopping was utilized and after 20 iterations without an improvement of the multi_logloss function the model stopped at 183 iterations. Overall this model performed at .487 logarithmic loss and it improved in the test set decreasing to .477 in the test's public logarithmic loss score. This improvement in logarithmic loss indicates that this model had not over fit to the training data which was a large concern in this application.



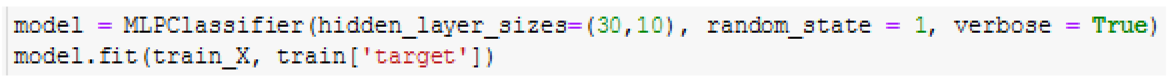
id	Class_1	Class_2	Class_3	Class_4	Class_5	Class_6	Class_7	Class_8	Class_9
1	0.000415	0.087464	0.180651	0.723506	5.082353e-06	0.000404	0.007409	0.000083	0.000063
2	0.000561	0.005491	0.000753	0.000101	2.107649e-05	0.886475	0.000755	0.103545	0.002298
3	0.000006	0.000020	0.000013	0.000013	2.350652e-08	0.999356	0.000012	0.000554	0.000025
4	0.000360	0.433753	0.555393	0.009368	4.259872e-07	0.000082	0.000097	0.000227	0.000719

Figure 9: Model Test Class Predictions

A submission data frame with the results was then made of these predictions following the format of the submission guidelines and converted into a csv. This model notably predicted high probabilities of one or two classes while only distributing some small probabilities towards the others for the initial products in which they predicted class confidence. As the log loss function more heavily punishes the wrong and more confident predictions this might indicate a drawback to this model.

Multi-layer Perceptron Classifier

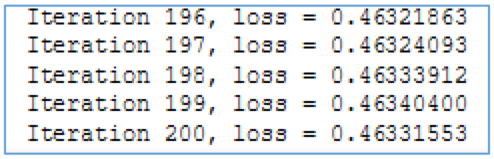
After exploring some alternatives to XGBoost, the Multi-layer Perceptron (MLPClassifier) packaged in Sci-kit-Learn was discovered. The MLPClassifier proved to be both relatively easy to implement and a strong enough Neural Network framework for classification. Unlike any other classification algorithm in sklearn such as a Naïve Bayes Classifier, the MLP relies on an underlying neural network. In terms of technical set up and tuning, this method only allowed for slight tinkering with the hidden layer sizes. Overall this model generation surprisingly came down to two lines of code



```
model = MLPClassifier(hidden_layer_sizes=(30,10), random_state = 1, verbose = True)
model.fit(train_X, train['target'])
```

Figure 10: The Two Lines of Code

Trial and error was used in modifying the number of layers and nodes. Ultimately, a network with two layers with 30 nodes in the first layer and 10 nodes in the second was successful and generated impressive results in the training set.



```
Iteration 196, loss = 0.46321863
Iteration 197, loss = 0.46324093
Iteration 198, loss = 0.46333912
Iteration 199, loss = 0.46340400
Iteration 200, loss = 0.46331553
```

Figure 11: Model Training of MLPClassifier Model

With the training data, the MLPClassifier model performed better than the LightGBM with a training loss of .463. The model had significant drops in log loss pretty quickly in the first 25 epochs and a slightly diminishment over the following 175 epochs before the maximum 200 iterations were reached, which was a limitation of the algorithm. Below this log loss over the stretch of epochs can be seen.

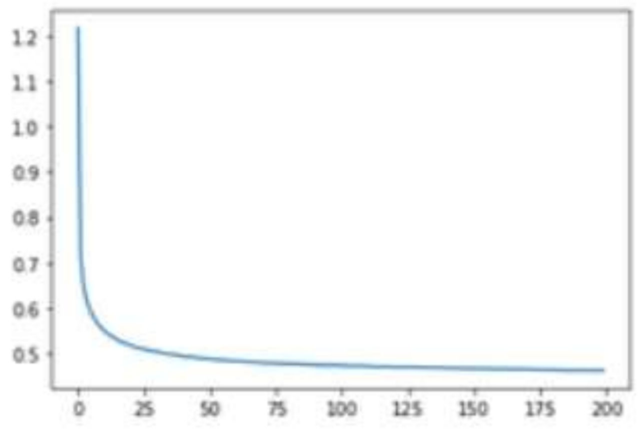


Figure 12: Logarithmic Loss per Epoch, MLPClassifier Model

Following these successful neural network backed classifiers, a simpler random forest method was used in the final model as a somewhat of a control. This was done to see the true difference in quality of results of model types and to determine whether or not these more sophisticated methods were worth the added effort. In business solutions, such as this test case sometimes a simpler and more time efficient method such as linear regression model is more worthwhile to pursue than a neural network. In particular, when there is a time limit and the business decisions determined by the models are comparable.

Random Forest Classifier:

Random decision forests are a classic ensemble learning method for classification in addition to regression, etc. For this model a random forest classifier was generated with 100 estimators, and 50 max features. This classifier was then calibrated using the isotonic approach so it wouldn't overfit, and utilized a five fold cross validation. Overall the log loss for the training was low at 0.124 indicating an overfitting of the training data. The training accuracy was 0.998 which also helped justify this claim. Following this analysis, feature importance was graphed to compare with the initial LightGBM model.

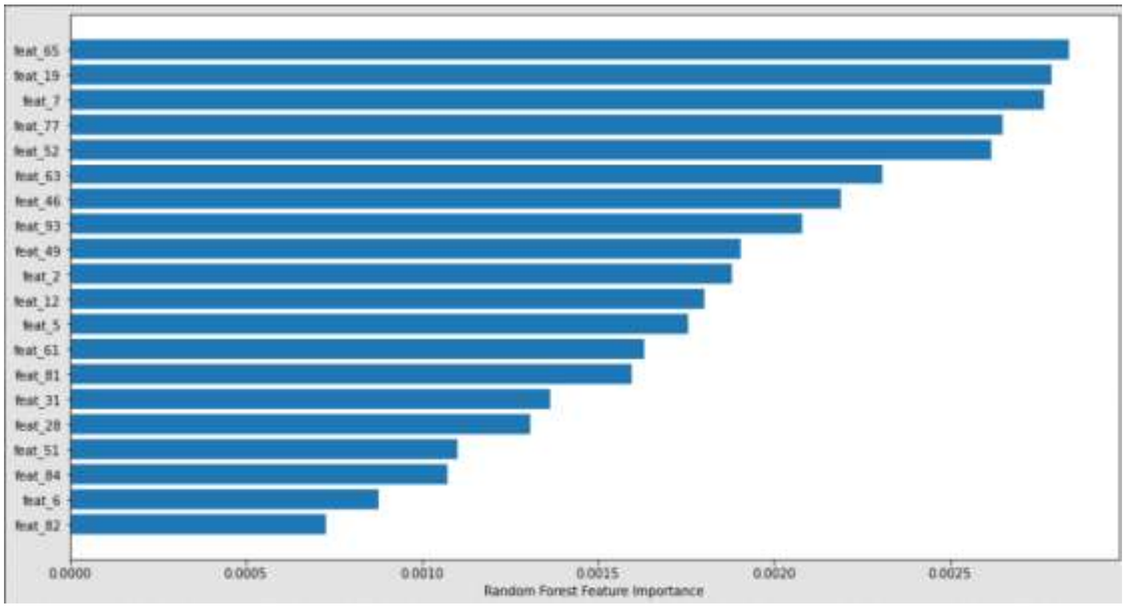


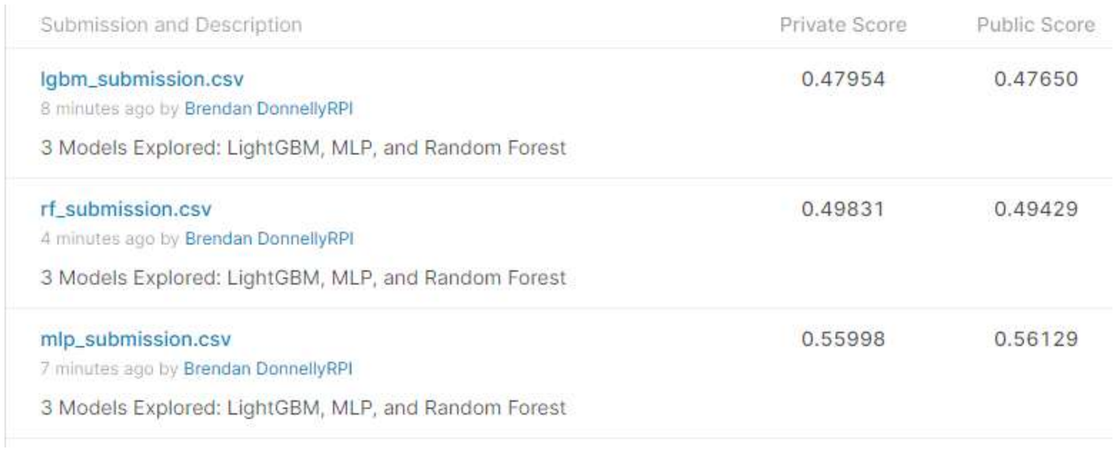
Figure 13: Feature Importance Output Random Forest

Compared to the LightGBM model's feature importance there were very many differences. The spread of importance for this model was most notably more uniform at the top performing variables. The top features for the random forest classifier were features (65, 19, 7, 77, 52, and 63) the highest non-tfidf transformed features of importance for the LGBModel were features (67, 86, 48, and 25). Interesting that the model selection made such an impact.

The predictions were then generated from the test data and outputted to the a submission csv in which it performed with a log loss of 0.561. Overall each of these models performed well comparatively with the benchmarked solutions of the leaderboard's 3,505 submissions. Of these three models, the highest performing model at 0.477 logarithmic loss would have placed at 1100/3505 placing the performance in the top third in the competition participants.

Table 2: Developed Model Solutions

Model Approach	**Feature Approach**	**Train/Test Perf (Log Loss)**
Light GBM	TFIDF	.487 , 0.477
MLPClassifier	Label Encoder	0.463, 0.494
Random Forest, Calibrated Classification	Label Encoder	0.124 , 0.561



Submission and Description	Private Score	Public Score
lgbm_submission.csv 8 minutes ago by Brendan DonnellyRPI 3 Models Explored: LightGBM, MLP, and Random Forest	0.47954	0.47650
rf_submission.csv 4 minutes ago by Brendan DonnellyRPI 3 Models Explored: LightGBM, MLP, and Random Forest	0.49831	0.49429
mlp_submission.csv 7 minutes ago by Brendan DonnellyRPI 3 Models Explored: LightGBM, MLP, and Random Forest	0.55998	0.56129

Figure 14: Kaggle Scores

Appendix:

https://github.com/BrendanDonnelly/MachineLearning/blob/main/ML_Final_Project_Brendan_Donnelly.ipynb

Works Cited:

Collier, Andrew B. "Making Sense of Logarithmic Loss." *Datawookie*, 14 Dec. 2015, datawookie.netlify.app/blog/2015/12/making-sense-of-logarithmic-loss/.

Works Consulted:

Nagomiso. "Feature Extraction - Tfidf." *Kaggle*, Kaggle, 25 May 2020, www.kaggle.com/nagomiso/feature-extraction-tfidf.

Wakamezake. "[Tf-Keras] Otto NeuralNetwork." *Kaggle*, Kaggle, 3 Apr. 2020, www.kaggle.com/wakamezake/tf-keras-otto-neuralnetwork.

Awwabtahir. "Using Random Forest." *Kaggle*, Kaggle, 26 Apr. 2017, www.kaggle.com/awwabtahir/using-random-forest.

Otto Group: Geschäftsberichte, www.ottogroup.com/en/about-us/daten-fakten/Annual_Reports.php.

Cmdline. "How To Filter Pandas Dataframe By Values of Column?" *Python and R Tips*, 27 Nov. 2020, cmdlinetips.com/2018/02/how-to-subset-pandas-dataframe-based-on-values-of-a-column/.

Nair 20/06/2019, Amal, et al. "A Beginner's Guide To Scikit-Learn's MLPClassifier." *Analytics India Magazine*, 21 Nov. 2020, analyticsindiamag.com/a-beginners-guide-to-scikit-learns-mlpclassifier/.