Abanob Tawfik z5075490
Nayeon Lee z5164405
Riley Sutton z5164867
Daniel Gordos z5208992

Traditional ML Approaches for Consumer Product Classification

Link to drive:
https://drive.google.com/drive/folders/12DXaloGil8eYdglJbty6tYKvxMAC9KFI?usp=sharing

Introduction

As online shopping platforms grow increasingly dominant in the retail market, automated product categorisation has become invaluable to e-commerce companies. Accurate classification is required for effective targeted advertising to be developed, as well as for consistency in internal analytics. This matter is further complicated by the popularity of platforms facilitating both consumer-to-consumer business-to-consumer transactions, which often results in inconsistent and noisy data. In this report we evaluate the effectiveness of several traditional supervised learning algorithms on a product dataset provided by Otto Group, a leading online retail firm.

Implementation

The dataset utilized was provided by Otto Group as part of a public Kaggle competition. It consists of 200k unique products, with each represented by a sparse feature vector containing 93 boolean product features. Of the 200k data points, 61k are labelled as belonging to one of 10 product categories.

As per the Kaggle competition, the main metric used for model evaluation was multi-class log loss, which is defined as

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij})$$

where N and M are the number of test samples and class labels respectively, y is 1 if the sample lies in the class and 0 otherwise, and p is the predicted probability of the sample belonging to the class.

For our second evaluation method, we calculated the accuracy of each of the models to further evaluate how accurately each model classifies the data. The test set that was provided in the public Kaggle competition did not have any labels for us to test the accuracy ourselves. Hence we decided to split the training data in 8:2 ratio to training and test data. Then we used the *metric.accuracy_score* function in sklearn to calculate the accuracy of the models. The accuracy is defined as:

$$accuracy = \frac{1}{|Test|} I[\ \widehat{c}(x) = c(x)\ ]$$

where *Test* is the test set and $I[]$ is the indicator function. (referencing Classification1 Lecture Notes)

For further and more powerful testing locally we had also done a K-Cross-Fold validation with 5 groupings. K-Cross-Fold validation allowed us to perform more test/validation testing and make greater use out of our training data set. By using local testing we were able to get another view on our model performance. The accuracy for K-Cross-Fold validation was given as

$$accuracy = \frac{1}{k} * \sum_{i=1}^{k} fold\, accuracy(i)$$

Where *k* is the number of folds, and *fold accuracy* is the accuracy measured by performing a test validation split on that fold, similar to the calculation for accuracy_score.

The approaches evaluated on the dataset are: KNN (K-Nearest Neighbour), decision trees and random forests, and naive bayes.

Experimentation

1. Baseline logistic regression

A multinomial logistic regression was used as a baseline for comparison to other methods. This model was chosen as it is commonly used as a baseline, owing to its relative simplicity and fair performance. The model also does not require the independent variables to be statistically independent of each other, making it effective in cases with large numbers of binary variables. A ‘one vs rest’ scheme was used to handle the multi-class nature of the problem.

This baseline produced a log loss score of 0.674 via Kaggle scoring, with an accuracy of 0.759 when run with a local train / test split.

2. KNN

We had implemented a KNN model using the scikit package and attempted to see the effect in changing the parameters of KNN on our results. KNN was initially chosen as our classification model due to its simplicity, and in theory its accuracy on large datasets. However in comparison to the baseline KNN had performed significantly worse. The baseline regression model had performed with a score of 0.67417. With the default setting of 5 neighbours the KNN model had performed with a score of 30.47336. We had thought this must have been due to using such a low number of neighbours in our model that contained an enormous amount of data. After changing the number of neighbours with set intervals of 100 up until 3000 (2% of our dataset), the average score had almost remained unaffected. We had also tried to modify the distance measure metric by changing the power used in the calculation, however similar to the results below, the scores were identical.

Figure 1 displays the effect of modifying the number of neighbours on the accuracy of our model. It was worth noting that the performance of models with a large value of k, had significantly slower computation time. The reason we think that the model performs poorly is due to the large amount of features, 93, in the dataset. Having a larger amount of features worsens the performance of KNN due to the dimensional increase and distance calculations increasing at a larger rate, leading to the so-called curse of dimensionality.

The metric used to calculate the distance between two points, x and y, is the following:

$$distance(x,\ y) = \sum_{i=1}^{k} (x_i - y_i)^p$$

Where k is the number of features, and p is the power parameter passed into our classifier for the minkowski metric. What can be seen is that as k increases, our distance values become larger, thus making our data in effect more spread out and less densely populated.

In conclusion KNN was not a viable model for the task at hand.

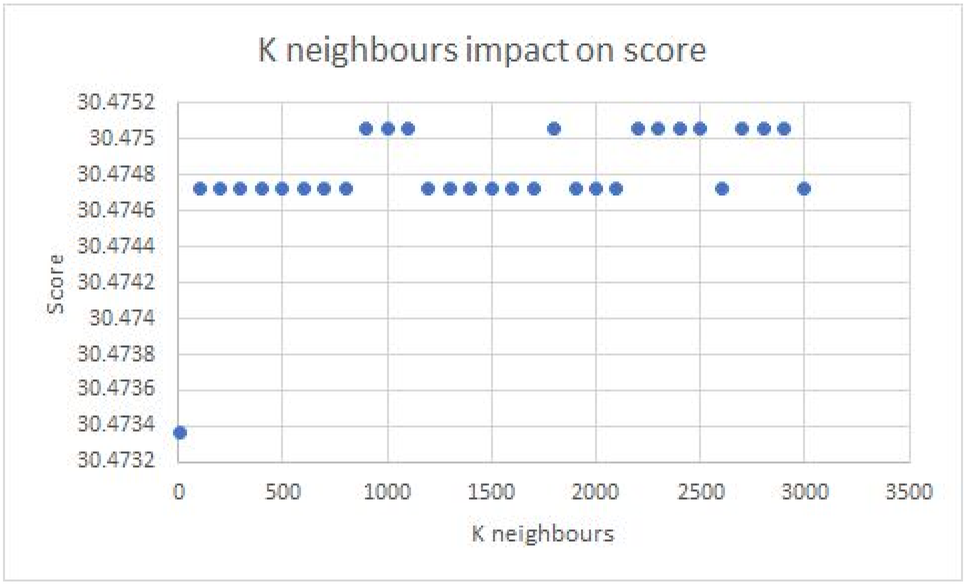

Figure 1: Effect of modifying K neighbours

3. **Decision Trees**

For our third model, we utilised decision trees and random forests to classify the products to one of the categories. We first implemented a single decision tree which had a score of 9.71427, significantly worse than the baseline. We concluded that this poor performance was likely due to overfitting, a very common issue with simple decision trees.

Hence we attempted to implement random forests to minimise the variance and fix the overfitting problem. Random forests are powerful as they have the ability to limit overfitting without substantially increasing the error due to bias. We implemented three different random forests, each with 10, 100 and 1000 trees in each forest by changing the *n_estimators* parameter. Despite increasing the number of trees in each forest does not lead to overfitting, it takes a lot of computational time hence we need to pick the most optimal number of trees. We computed each of the four different random forests, and each gave the accuracy of:

1. 10 trees: 0.7776341305753071
2. 100 trees: 0.8080155138978669
3. 300 trees: 0.8081771170006464
4. 1000 trees: 0.8103587588881707

We can observe that the tree's accuracy continuously increases as the number of trees grow, but stops increasing significantly from around 100 trees. Hence we concluded that the optimal number of trees would be 100 trees, if we take in consideration of the computational time for random forests with a larger number of trees.

Although random forests decrease the chance of overfitting significantly, they are still prone to overfitting. Hence we attempted changing the hyperparameters of the decision tree classifier in scikit package to check if we are overfitting and increase accuracy. We modified the *max_depth* parameter to find the optimal depth to maximise accuracy without overfitting. We used 8 different values, ranging from 25 to 200 by 25 and kept the *n_estimators* parameter constant as 100 trees to observe only the effect of changing *max_depth* parameter.

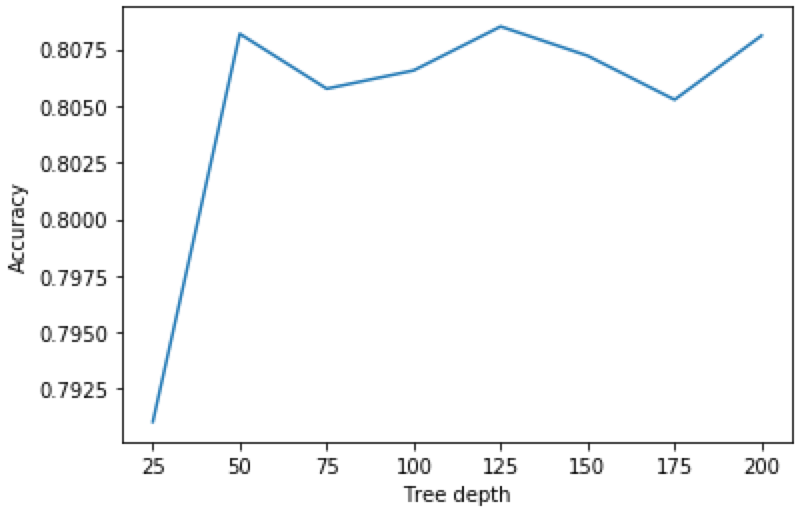

Figure 2: Tree depth effect on accuracy

From Figure 2, we can see that from depth 50, the accuracy does not seem to increase anymore, hence we can conclude that depth 50 would be the optimal max_depth.

Hence we concluded that the most optimal choice of hyperparameters would be 100 for *n_estimator* and 50 for *max_depth*. Using this random forest model, the score of log loss evaluation was 0.55148.

4. Naive Bayes

For our Naive Bayes implementation we had created multiple models using different variations of Naive Bayes classification. We had used Gaussian Naive Bayes (GNB), Bernoulli Naive Bayes (BNB) and Multinomial Naive Bayes (MNB). GNB and MNB had performed much better than KNN receiving a score of 15.69367 and 17.51665 respectively. However BNB had performed far better than all these with a score of

2.57160. Naive bayes was chosen as it performs well in classification tasks. The difference between the scores, mainly with bernoulli naive bayes, as compared to the other bayes classification models, was due to the features more closely matching independent binary values rather than continuous values (GNB) or a histograms (MNB). Since all features are kept hidden with regards to how they were created, we can assume from this outcome that each is independent and correlates well to binary conversion.

Feature Selection
In the data we had received there was a large amount of features, and for models including KNN, feature selection had been a method of increasing accuracy. We had used a tree classifier with gini impurity criterion provided in the scikit-learn package to perform the feature selection. Using feature selection had a minor effect on KNN decreasing the score by only 2 on average. We had tried many different top k features, from 10 ranging to 40 in increments of 10, however the best score we had managed to get was 27.80546 using top 40 features and 244 neighbours. Whilst being a small improvement, we suspect that the reason feature selection did not drastically improve performance is that the class relies on many features to make an accurate prediction, as seen in the case of trees the performance levels-off at depth 50, and in these depths there are many features which affects the accuracy of KNN due to the curse of dimensionality.

For Naive Bayes, feature selection had a strange effect on the accuracy of our models. Using feature selection on GNB had actually worsened the performance of the model giving us a score of 17.51742 having a worse score than originally 15.69367. On MNB feature selection slightly improved the performance of the model giving us an average score of 16.72804 compared to the original, 17.51665. On BNB however the model had a higher accuracy with an average score of 1.19122 compared to the original 2.57160. Overall feature reduction seemed to have almost no effect, or when it did it was too small a change to be considered significant and no meaningful conclusion can be drawn from these observations.

Results
Table 1 shows an aggregation of the best score achieved by each model using the automatic scoring from Kaggle:

Model	**Score (log loss)**
Baseline (Logistic Regression)	0.674
K-Nearest Neighbours (k = 243, 40 features)	27.80546
Decision Trees	9.71427
Random Forest	0.55148
Naive Bayes	1.19122

Table 1: Best score achieved from each model using automatic scoring

Table 2 shows the highest accuracy achieved by each model using the single train/validation set local testing.

Model	**Accuracy (%)**
Baseline (Logistic Regression)	75.87%
K-Nearest Neighbours (k = 243, 40 features)	70.99%
Decision Trees	71.36%
Random Forest	80.89%
Naive Bayes	92.92%

Table 2: Accuracy using local train / test split

Table 3 shows the highest accuracy achieved by each model using K-Cross-Fold validation local testing.

Model	**Accuracy (%)**
Baseline (Logistic Regression)	75 ± (0)%
K-Nearest Neighbours (k = 243, 40 features)	73 ± (0)%
Decision Trees	71 ± (1)%
Random Forest	81 ± (1)%
Naive Bayes	82 ± (0)%

Table 3: Accuracy using local K-Cross fold validation

It is worth noting that despite Naive Bayes outperforming Random Forest locally in both cases. Random Forest overall is more consistent and performs accurately in all three cases, and received the high score when submitting for the competition, Due to the assumptions of Naive Bayes, that there is no correlation between features, which doesn't always hold on the dataset, it may perform well on the training set but worse in a test set.

Conclusion

In conclusion, the model with the best performance was Random Forests. However, several other models performed significantly worse than the baseline logistic regression model. This poor performance is likely due to the high dimensionality of the data, and the resulting huge increase in the required amount of training data for accurate prediction. We can deduce that the effect of ensemble methods such as random forests, can lead to highly improved performance. Random forests minimised the effect of overfitting, which is usually a well-known issue in a single decision tree, as it reduces variance and does not increase any errors due to bias. We also concluded that the optimal hyperparameters were a tree count of 100, with a maximum tree depth of 50.

Working as a team to produce multiple models for the competition had taught us many valuable lessons. There are many different tasks we would have done differently, and other approaches we would have liked to try had we had the time, however overall we are happy with the outcome. There are various other models of supervised learning theorems which we could have explored further such as Neural Networks. Something we would have liked to have done was implement our Random Forest model (our best performing) from scratch, to allow us to have more control and really understand how the hyper-parameters impact the model accuracy for our given dataset. Also, the attributes and the labels of the provided data were numeric, such as feature_1, which limited our interpretation significantly. Despite our feature selection being processed through computation, it is hard to logically understand why some features were not significant in our model. Overall in the competition our final score we had submitted was of our Random Forest classifier which scored 0.55148 which would have ranked us 1811 had we made this submission today out of the top 3505. Unfortunately we are unable to compete in the leaderboard as the competition has expired.

References

1. Novakovic J., 2010 NOV 23, ‘The Impact of Feature Selection on the Accuracy of Naive Bayes Classifier’, available from The Impact of Feature Selection on the Accuracy of Naïve Bayes Classifier
2. Bain M., 2020 JUN, ‘Classification (1)’, available from https://moodle.telt.unsw.edu.au/pluginfile.php/5766480/mod_resource/content/1/Classification1.pdf
3. Ray S., 2015 DEC 29, ‘8 Proven Ways for improving the “Accuracy” of a Machine Learning Model’ https://www.analyticsvidhya.com/blog/2015/12/improve-machine-learning-results/
4. Koehrson W., 2018 JAN 7, ‘Improving the Random Forest in Python Part 1’ https://towardsdatascience.com/improving-random-forest-in-python-part-1-893916666cd