Capstone Project

Jeremy Jesse
2016-08-05

Machine Learning Engineer Nanodegree

Definition

Project Overview:

Retail stores such as Walmart serve various needs for their customers. One way to classify these various needs is the trip type methodology. A few trip type examples include the weekly grocery stockup trip, the pharmacy trip, the oil change trip, and the seasonal trip among many others. Walmart has established an internal process to classify each shopping trip(receipt) into one of many trip types. Walmart has made a sample of this data available on Kaggle.com, which is the world's most popular destination for data science competitions. In this data various attributes of the trip are found, such as sales volume, the department of the item, the UPC number and others. Walmart would like help in identifying new ways to classify each trip into the trip types it has identified. This could lead to new insights into the business and greater processing efficiencies.

Problem Statement

The dataset contain columns which describe the trip along with one column which defines the trip type Walmart placed it in. Each row of this data contains is at the UPC level, and will be aggregated to the trip type level. Initially, the data will be explored to gain an intuitive understanding of its structure and meaning, as well as to prepare it for modeling. Supervised learning techniques will then be employed to classify each trip into one of the Walmart defined trip types. Supervised learning is a subset of machine learning, which predicts the category a given event is part of. The effectiveness of the prediction will be measured against the actual trip type as assigned by Walmart.

Metrics

The effectiveness of the predictions made by machine learning will be measured using the multi-class logarithmic loss metric (entropy loss). This metric was specified by Walmart on the description of this project on the Kaggle website. It was also intelligently chosen, as it is evaluates a prediction not on the final result, but on the probability estimates for each of the possible classifications. This allows a finer grain of measurement, as the final result of a classification prediction is simply the class with the highest predicted probability. This metric is commonly used to measure the effectiveness of classification predictions. The equation as found in the scikit learn documentation is seen below.

For binary classification with a true label $y \in \{0, 1\}$ *and a probability estimate* $p = \Pr(y = 1)$*, the log loss per sample is the negative log-likelihood of the classifier given the true label:*

$$L_{\log}(y, p) = -\log \Pr(y|p) = -(y \log(p) + (1 - y) \log(1 - p))$$

This extends to the multiclass case as follows. Let the true labels for a set of samples be encoded as a 1-of-K binary indicator matrix Y*, i.e.,* $y_{i,k} = 1$ *if sample* i *has label* k *taken from a set of* K *labels. Let* P *be a matrix of probability estimates, with* $p_{i,k} = \Pr(t_{i,k} = 1)$*. Then the log loss of the whole set is*

$$L_{\log}(Y, P) = -\log \Pr(Y|P) = -\frac{1}{N} \sum_{i=0}^{N-1} \sum_{k=0}^{K-1} y_{i,k} \log p_{i,k}$$

Analysis

Data Exploration

We have a dataset that is 7 columns wide and 647,054 rows long. It consists of transaction data from Walmart. The TripType column is a classification of the trip as determined by Walmart. VisitNumber is a unique identifier for a specific receipt at a cash register. Weekday is simply the day of the week. UPC is the universal product code. ScanCount is the number of the given UPC that was purchased. If the value is negative then there was a return. DepartmentDescription is the name of the department, which is a logical grouping of similar UPC's. FinelineNumber is a more detailed grouping of UPC's than DepartmentDescription. The first four lines of the dataset are seen below.

TripType	**VisitNumber**	**Weekday**	**Upc**	**ScanCount \**	**DepartmentDescription**	**FinelineNumber**
999	5	Friday	68113150000	-1	FINANCIAL SERVICES	1000
30	7	Friday	60538820000	1	SHOES	8931
30	7	Friday	7410811000	1	PERSONAL CARE	4504

26	8	Friday	223840400 0	2	PAINT AND ACCESSORIES	3565
26	8	Friday	200661400 0	2	PAINT AND ACCESSORIES	1017

The count of different variables is not the same. There must be some missing data. The difference in count between these variables is quite small (647,054-642,925)/647054=0.638%. Any rows with missing data will be removed before model development begins.

Column Name	**Count**
TripType	647,054
VisitNumber	647,054
Weekday	647,054
Upc	642,925
ScanCount	647,054
DepartmentDescription	645,693
FinelineNumber	642,925

It can be seen that for a given trip type it is possible to see obvious meaning as to what the trip represents. The table below shows the number of trips which fall in a given trip type for each department.

Trip type 3 has a very large number of visits in Financial Services, while there aren't very many visits for the other departments. This is the financial services trip.

Trip type 5 must be primarily for those who are ill. Both Pharmacy OTC and Pharmacy RX have a large number of visits.
A sample of the data is found below.

TripType	**3**	**4**	**5**	**6**	**...**	**44**	**999**
DepartmentDescription							
1-HR PHOTO	2	NaN	1	1	...	NaN	71
ACCESSORIES	2	1	16	3	...	79	57
AUTOMOTIVE	20	3	40	5	...	209	364

Let us look a little closer to find out how many unique values there are for each column in the dataframe. There are 38 trip types, where trip type 999 stands for other. There were 95,674 unique visitors. What this really means is unique receipts. The same person could have shopped at Walmart more than once. Unsurprisingly, there were 7 weekdays. These text values will need to be changed to numbers to work with many classification algorithms. The scancount (POS) or number of given items purchased are all reasonable. There are 69 departments. They all seem reasonable as groupings of items. There are 5,196 finelines. Having worked at Walmart, I know that finelines are controlled by individual buyers, who all have their own way of placing UPC's within them. Therefore Finelines wouldn't show useful information, as they don't behave according to a consistent logic. Also, there are simply too many of them to create that many dummy variables. My computer would run out of RAM in the modeling stage.

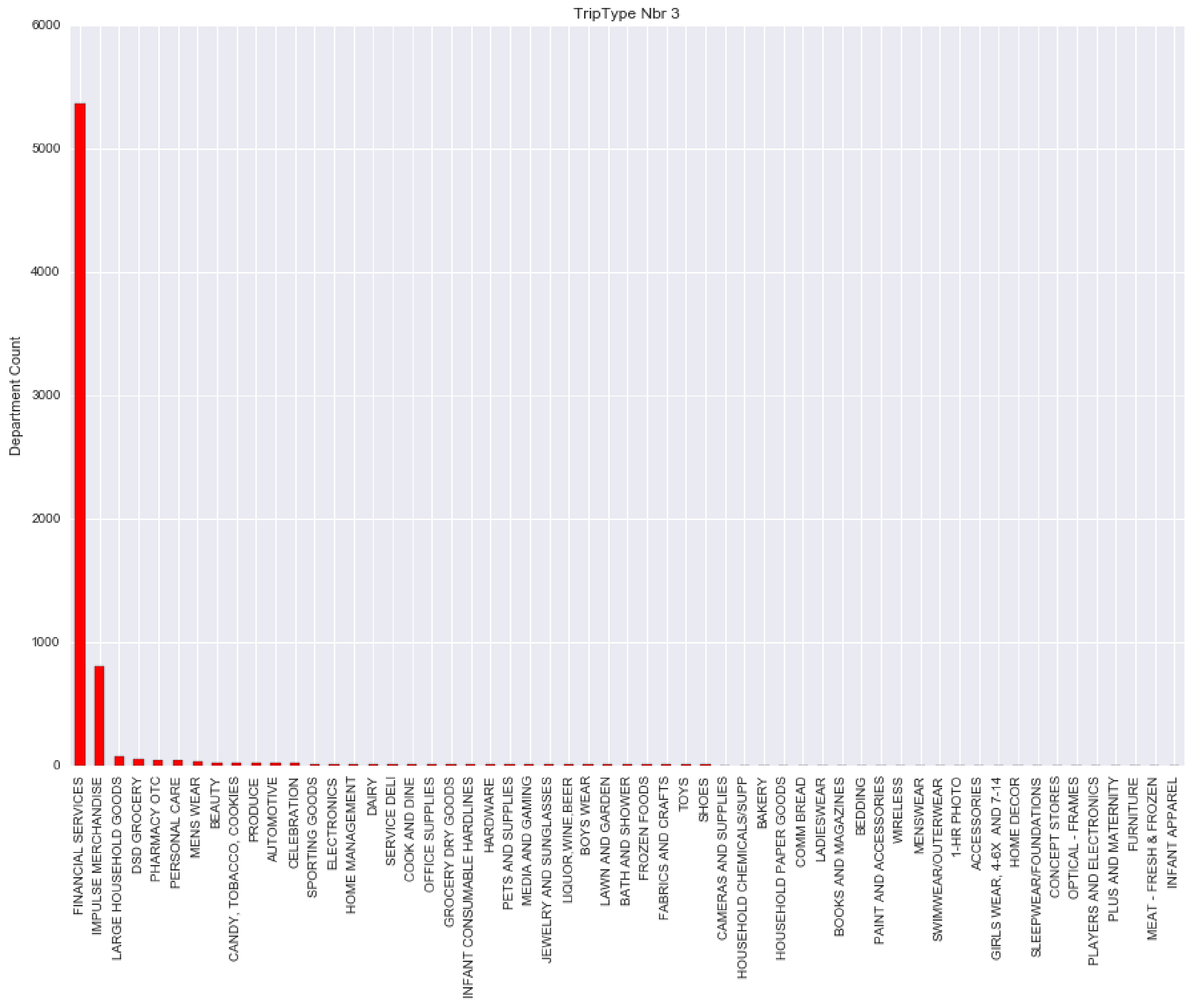
A description of each trip type is in the table below.

Trip Type	Description		Trip Type	Description
999	Financial Services/Impulse		5	Pharmacy
30	Shoes		3	Financial Services
8	DSD Grocery/Personal Care		4	Pharmacy
35	DSD Grocery		24	Food / Decoration
41	Men shopping for apparel and snacks		33	Cleaning Supplies
21	Fabrics Crafts / Office Supplies		43	Personal Care / Grocery
6	Kill Yourself Slowly		31	Cell Phone
42	Impulse/Celebration		27	Lawn
7	Grocery		34	Pets
9	Men's Apparel		18	Toys
39	Grocery		29	Toys / Sporting Goods
25	Apparrel		44	Personal Care / Grocery
38	Grocery		19	Electronics
15	Celebration		23	Electronics
36	Personal Care / Beauty		22	Electronics
20	Automotive		28	Sporting Goods
37	Produce		14	Fabrics Crafts
32	Baby		12	Grocery / Paper Goods / Lawn

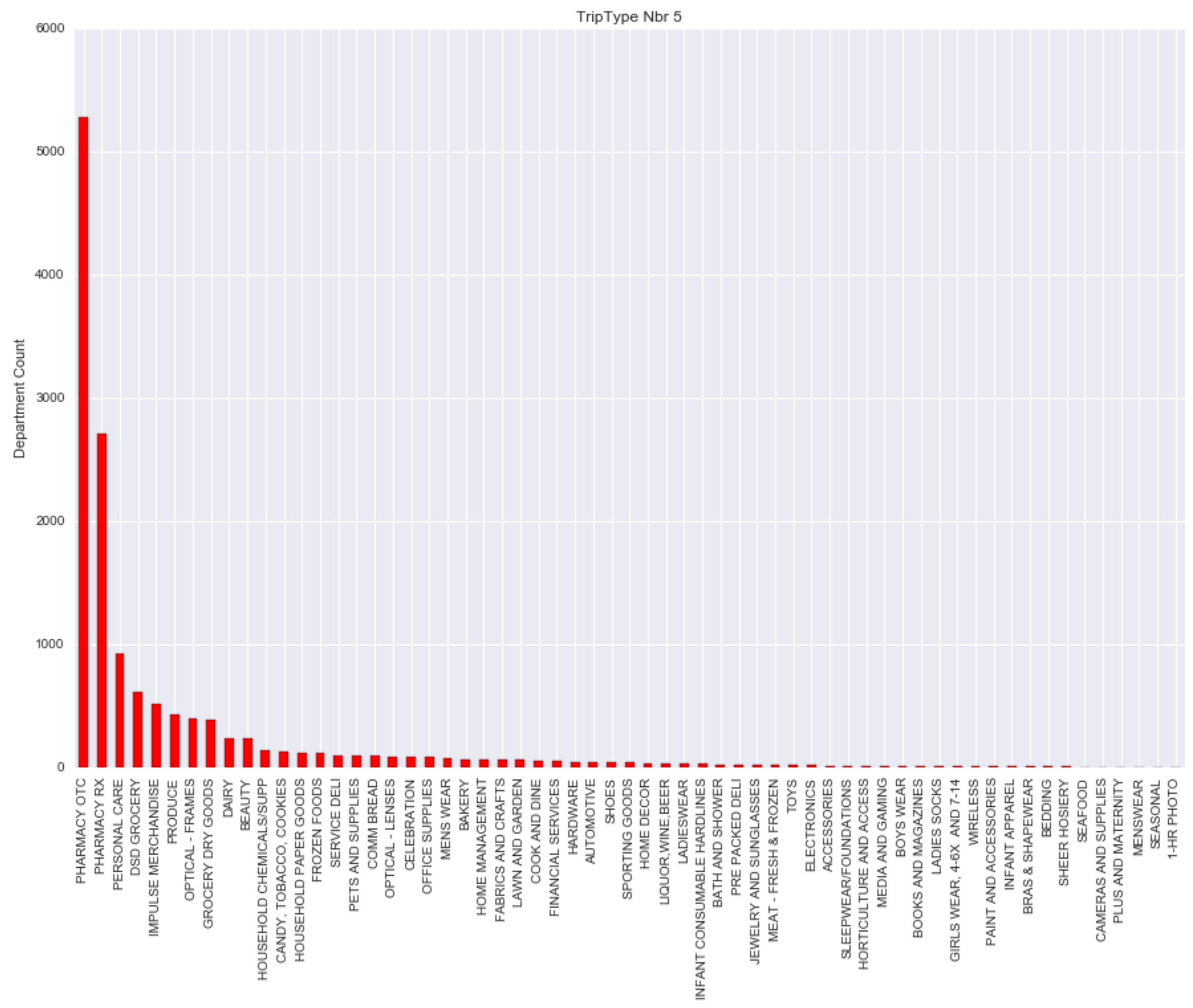
40	Grocery	⬛		

Exploratory Visualization

Now it is time to look at some plots of the data. The two plots below show the count of department numbers which show up in trips labeled as either 3, or 5. As stated previous trip type 3 is a financial services trip. This is clearly seen based on the number of department numbers which show up for financial services.



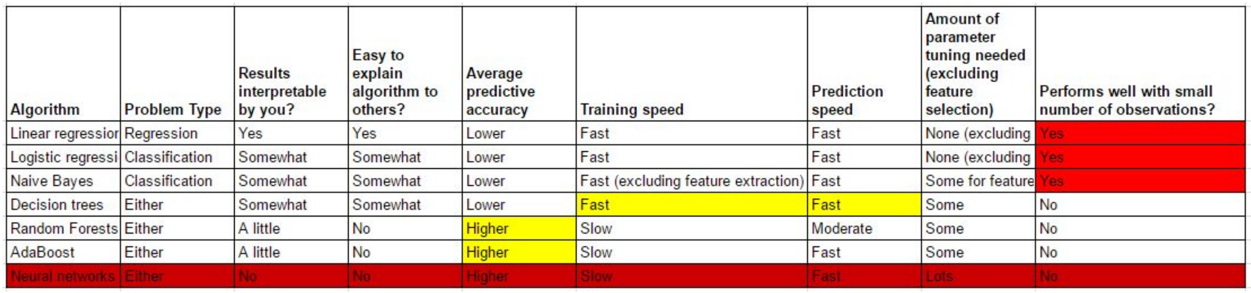
Trip type number 5 is focused on meeting health needs for either prescribed or over the counter medicine.



Algorithms and Techniques

Three different classification algorithms were selected; Random Forest, Ada Boost, and Decision Trees. These are all well known methods to solve classification problems, and are found in scikit learn, which is the preeminent machine learning package for the Python programming language. Each of these algorithms will examine the descriptive variables of the data and learn how they are related to trip type. Then a predictive forecast will be made, where the algorithm assigns a visit number to a particular trip type. After the prediction has been made, it will be compared to the trip type assigned by Walmart. The success of the predictions will be evaluated by the log_loss function in scikit learn as stated previously.

A table which compares the differences between various classification algorithms is found here and a compact sample of it is seen below.



Algorithm	Problem Type	Results interpretable by you?	Easy to explain algorithm to others?	Average predictive accuracy	Training speed	Prediction speed	Amount of parameter tuning needed (excluding feature selection)	Performs well with small number of observations?
Linear regressior	Regression	Yes	Yes	Lower	Fast	Fast	None (excluding	Yes
Logistic regressi	Classification	Somewhat	Somewhat	Lower	Fast	Fast	None (excluding	Yes
Naive Bayes	Classification	Somewhat	Somewhat	Lower	Fast (excluding feature extraction)	Fast	Some for feature	Yes
Decision trees	Either	Somewhat	Somewhat	Lower	Fast	Fast	Some	No
Random Forests	Either	A little	No	Higher	Slow	Moderate	Some	No
AdaBoost	Either	A little	No	Higher	Slow	Fast	Some	No
Neural networks	Either	No	No	Higher	Slow	Fast	Lots	No

Random Forest and Ada Boost generally have higher predictive accuracies. Neural Networks also have high predictive accuracies, but was not chosen since it is not part of scikit learn. The other chosen model, Decision Trees is not as accurate but is fast. Linear Regression, Logistic Regression, as well as Naive Bayes are also fast. They are particularly useful on a small number of observations, so they would have been tried had the dataset been smaller. These models also tend to be less effective for complicated relationships in the data. They are less flexible. Given that we have a large dataset, we want to use models which are capable of conforming to the data. These models have high bias and low variance.

Decision Trees are like flow charts. Each node is a test of the class that the instance is part of. They are the basis of the Random Forest algorithm, hence the name. Given that they are just trees, and not an entire forest they learn faster, yet the quality of their predictions is low. Training observations in the area are counted and the response is the mode of those observations.

Random Forest is a tree based learner. It works by creating a large number or ensemble of trees. It outputs the class that is the mode of the trees for classification problems. As the trees grow they are able to become more accurate without overtraining. Stated simply Random Forests are many Decision Trees working together. The trees are randomly sampled from the dataset with replacement (each observation can be selected each time a sample is taken). When making a decision about a node the split is decided on by evaluating a random sample of the features. Also, the scikit learn implementation of Random Forest is not like the standard version. "In contrast to the original publication [B2001], the scikit-learn implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class."

Ada Boost is a meta algorithm. It takes the result of weak learners (small decision trees) and combines them as a weighted sum. As the process progresses new weak learners are focused on areas that previous weak learners did not classify well.

More specifically, each algorithm will learn the data independently. Grid search and cross-validation will be used each time. For most algorithms there are parameters, the optimal value of which can't be known a priori. Therefore a range of options for each parameter are chosen and the algorithm is then executed for all possible combinations. The combination of parameter values which leads to the best outcome is chosen.

Cross-validation is a way to ensure that the algorithm isn't overfitted to the training dataset. Overfitting is when a model is tuned too closely to the training dataset, so that it believes that randomness in the training dataset, is in fact representative of all data for the problem. When this happens the model will tend to perform poorly on the test dataset. 3-fold cross-validation is used. The data is divided into 3 sections or folds. Two thirds of the data is used to train the model. The model is then tested on the remaining fold. This is repeated two more times, where each time the remaining fold is switched, so that each fold is held out once. The average of the log loss for each of the 3 folds is then taken as the representative log loss for the model.

After grid search and cross-validation have been performed for each model, the model with the lowest log loss value is chosen. It is the final model.

Benchmark

The benchmark log loss value was calculated assuming that there was an equal probability of the true value being in any one of the 38 possible classes. It is 1.5797.

Methodology

Data Preprocessing

Initially any rows with missing values were deleted, as they represented a small percentage of the total. Then the weekday's were changed from Sunday to Saturday to 1 to 7, with Sunday being 1. A new column was created which denoted whether or not the day was a weekday or on the weekend. A returned item feature was created if the scanCount (POS amount) was negative. The data was aggregated together at the visit number level. This will place all data for a given visit number on one row in the resultant dataset. This format is necessary because it is expected by most machine learning algorithms as coded in scikit learn. The department number column was broken out into dummy variables so that each department number has it's own column. For each visit number the amount of sales for a given department are placed in that department's column.

Implementation

The largest difficulty with this section, along with the entire project, was that I normally use R and not Python. Although, I have studied and used Python before, it is necessary to spend time reading documentation and searching for example code on stack overflow. Also, given that this is a classification problem, it is necessary to ensure that enough of each class is present in the dataset to be evaluated. TripType 14 has a low count. Initially when the data was split there would not be enough of this tripType such that the training and testing data each had all of the classes. Also, even if the all classes were found in both, the folds of K-fold cross validation didn't always have each of the classes present. This would cause errors after algorithms were initiated.

Class imbalances are a common problem in supervised learning classification problems. Much to my surprise scikit learn does not have a built in ability to oversample low frequency classes. Therefore I developed my own process with base Python code, which first divided the dataset into training and testing sets. Fifty percent of the data was used for training and the remaining for testing. Normally, you would want to use 80% for training, but the dataset was too large for for the available memory in my computer. This was surprising since I tuned parameters in the code to reduce memory consumption and increase run time. The specific rows selected for the training set were defined by my selection of the "random state". A section of the code follows.

```
train=grouped.sample(frac=0.5,random_state=61)
test=grouped.drop(train.index)
```

The presence of all classes in both the train and test datasets was verified. It was then found that the presence of tripType 14 was low in the train dataset, as was to be expected. The few tripType 14 rows present were replicated so that there would always be enough of them present in each cross-validation fold. This was done with custom code, as found in the attached jupyter notebook.

The data was preprocessed as in the above section. Then I evaluated the data using grid search and cross-validation in conjunction with each of the Random Forest, Ada Boost, and Decision Trees algorithms.

Refinement

There was a small process of adjusting the parameter values that were used in grid search. If one of the parameters was a numerical range and the optimal solution found used a value of that parameter in the middle of the range, I would then go back and add new parameters that were close to that value on both sides of it. This simply increased the resolution of the process, and helped to find the true maximum. Sometimes that change in the log loss metric value was quite small. This would tell me that the results of the algorithm were fairly stable in that range.

Also, it may happen that for a given set of numerical values, the one included in the optimal combination was at the top end of the parameter range entered into grid search. In this situation I would enter larger values and run the algorithm again. The change in the log loss metric was then evaluated. At some point the improvement was very small, and it became clear that given random variation the improvement was not meaningful, and did not justify longer computation times, or acquiring a more powerful computer.

In the initial runs, I didn't oversample TripType 14. This led to the errors which stopped the algorithms from running. This occurred when the each fold in cross-validation didn't contain the same number of classes. That is TripType 14 was not found in all of the folds. I would then select a different random state value when selecting the training data and start over. Different random state values worked for different algorithms and this caused the code to error out when it was first evaluated after being submitted.

All possible combinations of the below parameter grid settings were evaluated with grid search.

Random Forest:

```
param_grid = {
   'n_estimators': [200, 500, 1000, 1500, 2000, 2500],
   'max_features': ['auto', 'sqrt', 'log2', None],
   'class_weight': ['balanced', None]}
```

n_estimators = The number of trees found in the forest.
max_features = The number of features to look at when searching for the best split.
class_weight = The weight given to each class (in this example it is TripType).
Details are found in the scikit learn documentation.

Ada Boost:

```
param_grid = {
   'n_estimators': [1, 5, 10, 25, 50, 75],
   'algorithm': ['SAMME', 'SAMME.R'],
   'learning_rate': [0.01, 0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0]}
```

n_estimators = The largest number of estimators where boosting stops at
algorithm = Choose between the "SAMME" and "SAMME.R" algorithms.
learning_rate = The higher the learning rate the smaller the contribution of each classifier
Details are found in the scikit learn documentation.

Decision Trees:

```
param_grid = {
   'criterion': ['gini', 'entropy'],
   'splitter': ['best', 'random'],
   'max_features': ['auto','sqrt','log2',None],
   'min_samples_split': [2,3,4,5],
   'min_samples_leaf': [1,2,3],
   'max_leaf_nodes': [25,30,35],
   'class_weight': ['balanced',None]}
```

criterion = The function which decides the quality of the split
Splitter = The way the split is selected at each node
Max_features = The number of features that are looked at when deciding what the best split is.
Min_samples_split = The smallest number of samples needed to split an internal node
Min_samples_leaf = The smallest number of samples need to be a leaf node
Max_leaf_nodes = Grow a tree with X number of nodes in the best first fashion
Class_weight = This is the weight given to each class.
Details are found in the scikit learn documentation.

Results

Model Evaluation and Validation

The Random Forest algorithm provided the best results. It's best training dataset log loss value (with cross-validation) was 1.258. The log loss value for the test dataset was 1.24542628505. These are both better than the benchmark value of 1.5797. Although, 1.258 was the value for the best cross-validation training model, the parameters used for it were to computationally intensive to use on the test set. So a model which led to a training result of 1.283 was used to generate the test data result. The parameters for this final model were; Max Features = log2, Number Estimators = 1,500, and Class Weight = None. The code for this model is found below.

```
clf = RandomForestClassifier(n_jobs=-1,max_features= 'log2' ,n_estimators=1500, class_weight=None, oob_score =
True)
```

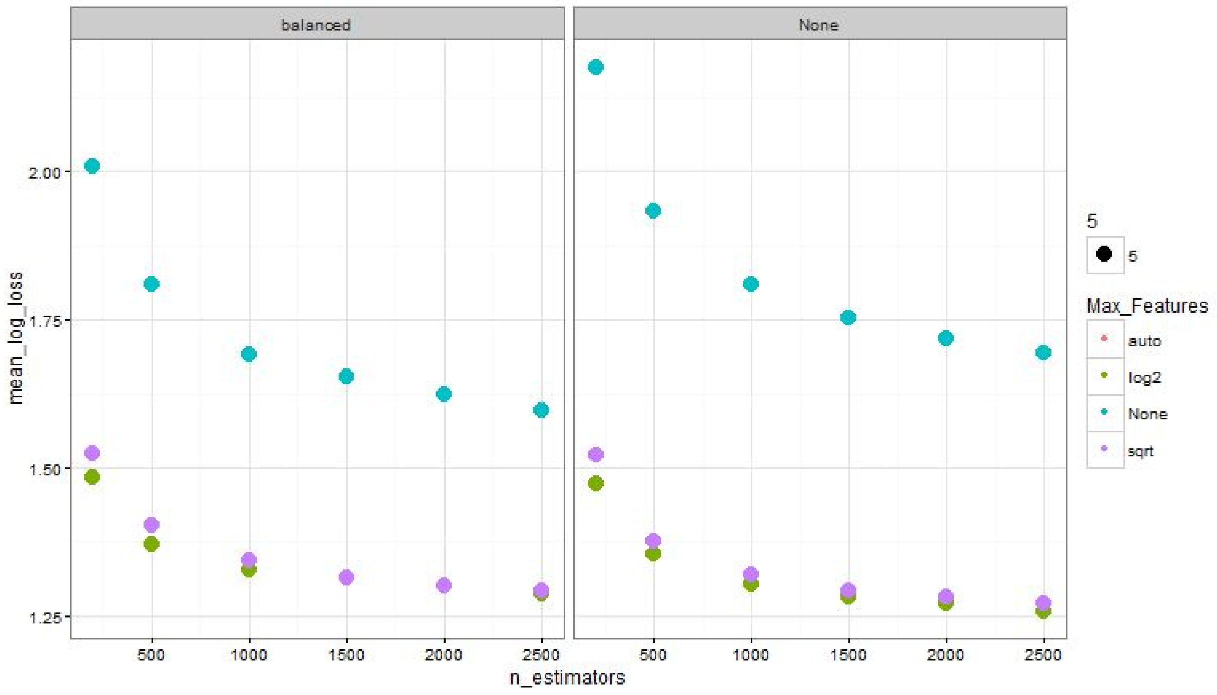
Justification

The log loss value for the test dataset was 1.24542628505, and the benchmark value was 1.5797, which shows a clear improvement. Given that this metric uses a log 10 base scale the difference is significant. This results performs much better than average.

Conclusion

Free-Form Visualization

The figure below shows an exploration of the solution space for the Random Forest algorithm as grid search explored all possible parameter combinations. It can be seen that as the number of estimators increases log loss decreases. However, the improvement is asymptotic and as there are computational limitations it isn't necessary for the number of estimators to be extremely large. Log loss is also quite high when the maximum features settings is set to "None", while the "log2", "sqrt", and "auto" settings show a clear improvement. Sqrt and Auto show exactly the same values as would be expected, given that they have the same meaning in the scikit learn documentation. The last three settings also lead to about the same log loss values. At lower number of estimator levels selecting "balanced" for the class weight settings has a clear advantage over selecting "None". However this advantage disappears at higher number of estimator levels.



Reflection

In this project Walmart transactional receipt data was analyzed to classify each receipt (or trip) as a particular trip type among the 38 trip types Walmart uses. To do this the data was explored, features were created, data was cleaned, data was aggregated, and four different machine learning classification models were used to analyze the data while employing grid search and cross-validation. The best algorithm to emerge from this process was Random Forest.

Learning about the log loss metric was enjoyable, as this is the first time I've used it. It's advantage is its consideration of probabilities of different class membership as opposed to simply looking at the final predicted classification.

The most difficult part of this process was improving my Python coding knowledge. After this project, I am much more confident about my abilities in Python. In the future I will most likely use R to munge data and then go over to scikit learn for modeling. Some of the algorithms have a setting which allows use of each core in the CPU. This is something that R does not offer, and drastically reduces computer run time for smaller datasets.

The Random Forest algorithm is a classification methodology which is broadly applicable to many classification problems. I was a little surprised that ADA Boost didn't provide better results, as it has received a lot of attention lately for generating winning submissions in Kaggle competitions. However, each algorithm has it's strengths and weaknesses. This is why it's important to evaluate a dataset with multiple classification algorithms.

Improvement

One possible improvement would be to use stacking (ensemble learning). Most winners of Kaggle competitions use this. It is a way to use the input of more than one classification algorithm (base learner) to contribute to the final classification decision. Essentially take the Decision tree, Random Forest, and Ada Boost models and take aspects of each of them to create a better model.