Project Title: Capstone Project (Predicting Inventory)

Student: David Escolme (escolme@bartlebooth.co.uk)

Course: Udacity Machine Learning Nanodegree

Version: 1.0; **Date:** 05 November 2016

Definition

Project Overview

This project is based on a recent Kaggle competition[1]. The competition was set by Grupo Bimbo [2] of Mexico. Grupo Bimbo are a large bakery goods manufacturer who service over 1 million stores across Mexico. They sell a very wide variety of bakery-related goods and need to assess each week what demand for each product is likely to be across their client base. Demand, in their terms, is the difference between goods ordered by and goods returned from clients in any week.

Their current approach to assessing demand for their products is through expert but manual feedback from the multitude of direct delivery sales employees who interact with each store each week.

Each product sold by Grupo Bimbo has a limited shelf life before becoming unsaleable to the public. Hence, accurate prediction of demand for each product will translate into better utilisation of raw materials, lower wastage and higher profits for Grupo Bimbo.

The challenge set by Grupo Bimbo was to create a machine learning model that could predict inventory demand for its bakery products. That challenge forms the basis of this project.

Problem Statement

Given a set of 7 weeks' demand data, create a model that could predict the next 2 weeks demand for bakery products for each sales unit. Unit was defined as the combination of Channel, Route, Agency, Product and Client. Demand was defined as the difference between sales and returns for a given week.

However, the nature of the Kaggle competition and dataset meant that the test data set (weeks 10 and 11) did not have the predicted values associated with it, so that could not be used as

[1] https://www.kaggle.com/c/grupo-bimbo-inventory-demand
[2] http://www.grupobimbo.com/en/index.html

test data for this project. This led to an adjusted problem statement whereby given the first **6** weeks demand data, create a model to predict for the 7th week (week 9).

As the value of demand is a continuous value, the problem could be formed as a Supervised Regression problem.

The approach to problem solving taken was:

- Exploration: Understand the domain, any interesting or standout features of the data as well as topline summary statistics of the full data set
- Engineering:
    - The original training data set was too large to run on my local workstation, so i would have to reduce the dataset to a manageable number of rows for training and testing
    - Create additional features for each data row using a combination of the lookup categorical data and lagged numeric sales/returns data
    - Create an encoding strategy for each categorical feature so that provisional models which call solely for numerical features could be used
    - Identify and handle missing values and outliers
- Modelling:
    - Identify and select an appropriate supervised regression model for the problem
    - If the training model selected was sensitive to scaling, follow a sensible scaling approach for the numeric data
    - Trial the model with default parameters
    - Tune the selected best model's input parameters to identify the most optimum settings for the selected model

Metrics

As the problem was one of regression, I chose root mean squared as the performance metric for the model(s). This metric is the square root of the mean of the squared errors. The lower this metric is, the better the accuracy of the model. It penalises larger errors as it squares each error and is expressed in the units of the model (in this case units of demand for bakery goods).

$$rmse = \sqrt{1/n \sum_{i=1}^{n} (predicted_i - actual_i)^2}$$

In the justification section of this report, a comparison to the winning Kaggle model is made. Kaggle use root mean squared logarithmic error (RMSLE) as their metric.

$$rmse = \sqrt{1/n \sum_{i=1}^{n} (log(predicted_i + 1) - log(actual_i + 1))^2}$$

Analysis

Data Exploration

The dataset is formed of 1 main data table (train.csv) with 3 additional lookup tables:

Data Set Name	**Train.csv: 74,180,464 rows**		
Attribute	**Example**	**Unique**	**Anomalies / Engineering**
WeekNumber	3	7	As each row represents sales for a particular combination of attributes for a particular week, previous weeks' data will need to be engineered as input features for predicting a current week
AgencyId	1110	552	Unique ID linking to town_state, which provides an opportunity to engineer additional features
ChannelId	7	9	Describes the channel of sale but no domain information provided as to the exact meaning. Channel 1 accounts for ~50% of sales.
RouteId	3301	3603	Describes the route of sale but no domain information provided as to the exact meaning
ClientId	15766	880604	Describes the client and is likely to the outlet that sells the goods
ProductId	3509	1799	Product item sold. Links to a product table which has product name
Sales	3		Positively skewed distribution with 75% of values 7 units or fewer.
SalesPesos	25.14		Monetary value of sales
Returns	0		Positively skewed distribution with 99% of values 3 units or fewer. 1 significant outlier
ReturnsPesos	0.0		Monetary value of Returns
Demand	3		Demand = Max(Sales - Returns,0)

Summary Statistics

Feature	**Mean**	**Std**	**Min**	**25%**	**50%**	**75%**	**99%**	**Max**
Sales	7.31	21.97	0	2	3	7	65	7230
Returns	13.03	29.32	0	0	0	0	3	250,000
Sales $	68.54	338.98	0	16.76	30	56.10	563.17	647,360
Returns $	1.24	39.22	0	0	0	0	25.14	130,760
Demand	7.25	21.77	0	2	3	6	64	5000

In summary:

- An extreme outlier exists in the data for Returns, this needs to be removed to prevent it skewing the training model. Other outliers are probably due to differences in scale between smaller and larger clients and should not be removed
- The shape of the training data does not lend itself to predicting weekly demand. The data will need to undergo some kind of transposition so that previous weeks' sales data (for 'n' periods) can be used as features to predict current (and then future) weeks' demand
- There are too many training points for the workstation being used to build the model (8GB) and so a strategy to reduce the data will need to be created
- The categorical data, if used in the model, will need to be encoded to allow numeric algorithms to act on those features
- Scaling (where the model requires it) will be needed for the numeric attributes as (for example) Sales and Returns have very different ranges of values

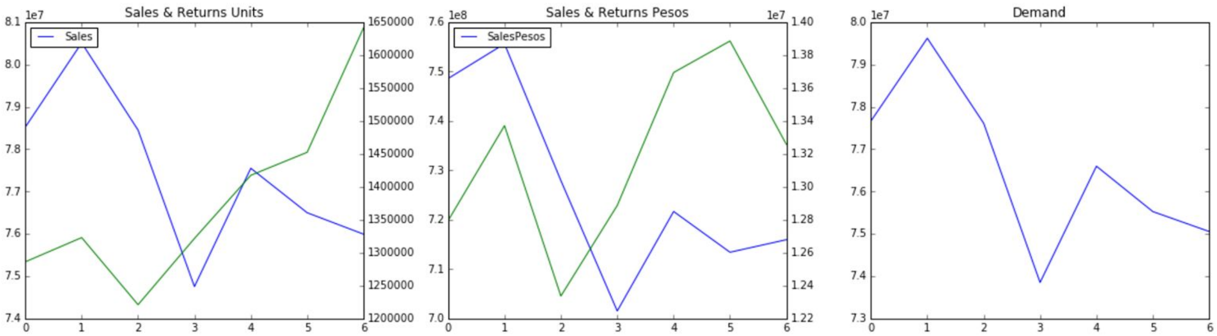
Data Set Name	**product.csv: 2592 rows, no duplicates**	
Attribute	**Example**	**Anomalies / Engineering**
ProductId	1	
ProductName	Capuccino Moka 6p 750g NES 9	The name can be split (by space) and parsed (regex) to surface additional features: **Capuccino** Moka **6p 750g NES** 9 Product Name, Number of Pieces, Weight, Supplier Brand

Data Set Name	**client.csv: 935,362 rows with 4862 duplicates**	
Attribute	**Example**	**Anomalies / Engineering**
ClientId	17123	
ClientName	OXXO XINANTECATL	There are 281,710 clients with the same name (NO IDENTIFCADO) The client name can be split and parsed to surface a potential aggregation: **OXXO** XINANTECATL

Data Set Name	**town_state.csv: 790 rows, with no duplicates**	
Attribute	**Example**	**Anomalies / Engineering**
AgencyId	1101	
Town	2004 AG. CUAUTITLAN	Town can extracted as an additional feature by joining to train on AgencyId There are 260 unique Town entries
State	ESTADO DE MÉXICO	State can extracted as an additional feature by joining to train on AgencyId There are 33 unique State entries

Exploratory Visualization

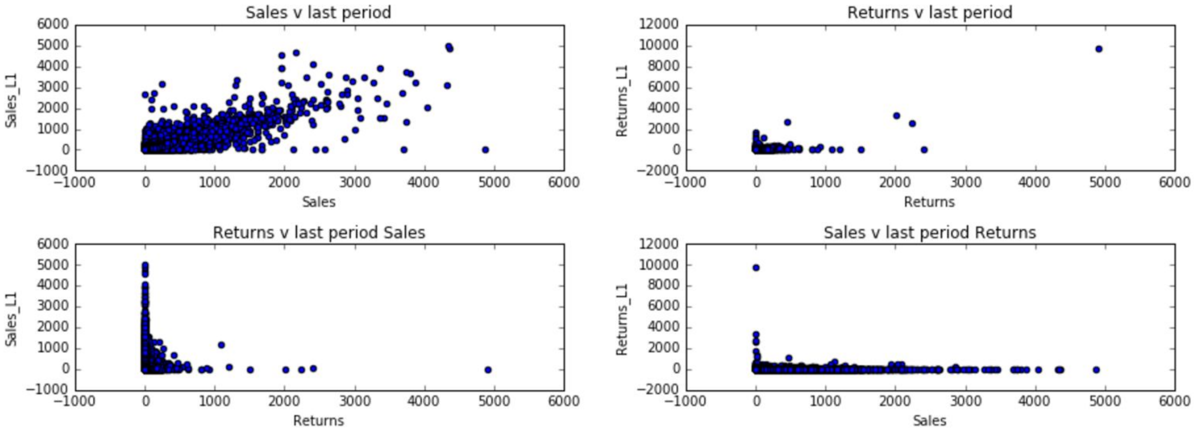
The first visualization is a grouping of all numeric attributes plotted against Week Number.



This shows that Returns are significantly fewer than Sales. It shows that the monetary values of Sales has a linear relationship with the units sold but the linear relationship between Returns and Returns (Pesos) is not evident in the final week - possibly due to the 1 significantly higher Return value highlighted in the previous section.

The values of Sales and Demand from week to week shows a 7% variance between min and max values. Returns shows a 25% variance, again due to the single anomalous value.

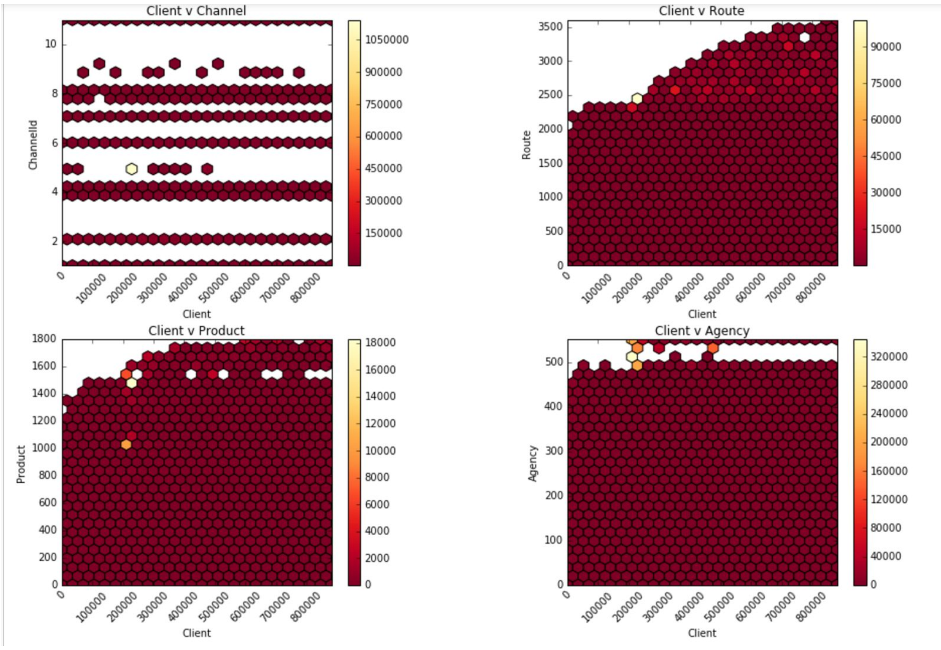
The second visualization indicates that previous weeks' transaction data would be a good input feature for predicting future week's demand. This is an important association as it leads directly to the need to engineer features from the current week data from previous weeks. The plot shows the previous week's demand data correlated to current week's. The correlation statistic for week 9 and week 8 sales is 0.88.



The final visualisation, a hexagonal bin chart, shows that demand between Clients and each of the other primary categorical attributes is generally low (many clients have low demand, clients are distributed across each of the other categorical attributes) but some client/attribute combinations show significantly higher demand. The attribute ids have been scaled to remove the numerical gaps that exist between Ids as those gaps do not represent any meaningful distance between the attributes.

The point of this visualisation is to show just how many individual categorical values exist in the data set. These will have to be translated using some form of encoding as although the Ids themselves are numeric, the number does not represent anything meaningful. As there are so many individual products, agencies, clients and routes, ideally, there would be some form of aggregation available either through features of each attribute or through statistical clustering.

This led to feature engineering in the modelling part of the project using the additional lookup tables provided.



Algorithms and Techniques

The problem of predicting demand, a continuous value, is one of supervised regression. From the wide selection of potential algorithms / techniques for supervised regression in sklearn, Gradient Boosting Regression Trees (GBRT)[3][4] was chosen as a candidate for the problem.

GBRT was chosen for this problem because:

- It’s a regression model and this problem requires a regression model for prediction
- Boosting (creating a single model by combining a series of weak models) has been shown to work well and provide good robustness with respect to bias and variance (under / overfitting)[5]

GBRT (like other boosting algorithms) works by fitting a series of weak learners (that is a model which is only slightly better than guessing) and then combining results of each model into a combined single model. The aim of each model is to minimize a loss function (eg. squared error). On each iteration, the next model is trained on the residuals of the last model. The theory being that with each successive iteration, the model becomes more accurate.

This is shown in mathematical terms by:

$$F(x) = \sum_{m=1}^{M} \gamma_m h_m(x)$$

Where F(x) is the final model for the training data set (x) and h(x) is the weak learner multiplied by a learning rate. At each step, the algorithm seeks a new h(x) which can minimise the chosen loss function L based on the previous step’s model:

$$F_m(x) = F_{m-1}(x) + \arg\min_{h} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) - h(x))$$

Benchmark

The benchmark chosen for the project was the root mean squared error (RMSE) using the previous 3 week’s average for demand as the prediction and comparing that to the actual values observed.

The thought process in choosing this benchmark was that a model should be expected to perform better than simply choosing a 3 week moving average.

[3] https://en.wikipedia.org/wiki/Gradient_boosting

[4] http://scikit-learn.org/stable/modules/ensemble.html#gradient-tree-boosting

[5] http://robjhyndman.com/papers/kaggle-competition.pdf

Methodology

Data Preprocessing

6 principle data preprocessing steps were followed.

Training data reduction

This was necessary due to memory limitations on the workstation being used. The goal was to create a reduced dataset which could be used for model training and testing.
The approach was to use a text file processing module to iterate through each line of the train file and which would select rows based on a particular channel Id (as the data was partitioned relatively evenly across each channel except Channel 1, which accounted for 50% of the transactions)

```
For each line in the training file, create a new line in the reduced file if the ChannelId == ‘X’
```

Creation of lagged sales, returns and demand data

This was necessary to create input features for prediction based on ‘n’ previous weeks’ transactions.
3 was chosen as ‘n to create the lagged sales, returns and demand data (eg. week 8 had input features from weeks 7,6 and 5). 3 was chosen as the lag period as that allowed for 3 weeks of data to train and test on (8,7,6) and 1 week to act as unseen data (9). The pseudo-code to create the lagged data was:

```
Create separate data-frames for each week
For each frame (week) starting from Min(Week)+3, join the previous 3 weeks’ data to the current week and tag each previous week as L1, L2, L3
Join each resulting set of weekly lagged demand data together using the common attribute keys to create the new training dataset
```

Some attribute combinations did not have data for every week, leading to NaN entries. As this indicated that the combination had no demand in that week, the NaN were updated to 0.

Engineering features from the agency and product lookup data

This step was introduced to allow for feature reduction as each of agency, product and client had very many unique items.

For Agency, a simple join was created from the training dataset to the town_state data set keyed on AgencyId to add in the Town and State features to the training data.

For Product, text processing of the product lookup file was done to engineer a higher order product name, weight, pieces and supplier code feature set, which could be added to the training data based on a join on ProductId. The psuedo-code was as follows.

```
For each product in the product table:
    Split the product name by ‘ ‘
    Use the first split as the new product name (resulting in an aggregated product name)
    If any of the splits are of the form %dg or %dml, capture that split as product weight
    If any of the splits are of the form %dp, capture that split as product pieces
    Capture the second to last split as supplier code
Join train to product on ProductId and append new Product Name, Weight, Pieces, Supplier Code attributes to train
```

A similar process was carried out on Client to extract an aggregated client name from the first part of the full client name. The pseudo code was:

```
For each client in the product table:
    Split the product name by ‘ ‘
    Use the first split as the new product name (resulting in an aggregated product name)
Join train to client on ClientId and append new Client Name attributes to train
```

Aggregation, encoding and scaling

Having engineered lag features and additional attributes, the sales, returns, demand data were then summarized by Town, State, Product Name, Weight, Pieces, Supplier Code. After experimentation with some models, RouteId, AgencyId, ClientId, ProductId and ClientName were dropped from the data set. It was necessary to drop the attributes as the workstation being used could not process an encoded model with these attributes due to available memory, so the aim in this step was to produce a data set which would enable a model to built, otherwise each of the Ids would need to have been encoded, leading to a very large number of dimensions.

Once the aggregated dataset was produced, each of the categorical features was one-hot-encoded, which resulted in a binary attribute for each Town, State, Weight, Pieces, ProductName, and Supplier Code. This took the dimension size to over 400 from under 30.

As Gradient Boosting with trees does not require scaling, this step was not performed. If an algorithm sensitive to scaling had been used, the training data for sales, returns, demand would have been scaled using a min-max scalar and that scalar would then have been used on the test and unseen data sets.

Implementation

Beyond the data pre-processing, the implementation of the basic model was straightforward. It can be described as:

- Create the dataset: Described above

- Split the dataset into a set of features and labels: this was achieved simply by taking the label column to another dataframe and dropping it from the original dataframe. The label column was the demand for the week being trained / predicted
- Split the dataset into a training and test data set: this was achieved using sklearn's train-test-split feature using 30% of the training data for testing. The split achieved is randomised and ensures that artificial patterns are not created in the training data compared to the test data and so helps with generalisation of the model.
- Instantiate the model: single line calls to sklearn to create the baseline regression model, using default hyper-parameters (the parameters which control how the model operates
- Fit the model with the training data using the .fit(X_train,y_train) method of the models
- Score each model using the test data set: using the results of .predict(X_test) method of the models as input to the mean_squared_error method and then taking the square root of the resulting error

The most complicated part was understanding how to create the original dataset and augment it with a reasonable set of features. The initial model was:

```
GradientBoostingRegressor(alpha=0.9, init=None, learning_rate=0.1, loss='ls',
             max_depth=4, max_features=None, max_leaf_nodes=None,
             min_samples_leaf=1, min_samples_split=2,
             min_weight_fraction_leaf=0.0, n_estimators=100,
             presort='auto', random_state=0, subsample=1.0, verbose=0,
             warm_start=False)
```

This shows that some key initial hyper-parameters were: Learning Rate = 0.1, Number of Trees = 100, max depth of each tree =4. The next stage was to refine the model by tuning the hyper-parameters.

Refinement

To refine the model, the GridSearchCV module in sklearn was used to trial combinations of hyper-parameters to realise the most optimum estimator settings for the model.

The hyper-parameter tuning was as follows:

- n_estimators (the number of trees in the ensemble): [50,100]
- max_depth (the number of levels in each tree): [4,6,8]
- learning_rate (the weight given to each successive iteration): [0.2,0.1,0.05]
- Min_samples_split (the minimum number of data points that can be used to split data): [10,50,100]

The hyper-parameters are all influential in controlling the balance of bias and variance in the model and trialling combinations of each of a number of different samples of the training data results in an optimal set of hyper-parameters which minimises error.

The best model selected by this refinement was:

```
GradientBoostingRegressor(alpha=0.9, init=None, learning_rate=0.2, loss='ls',
          max_depth=4, max_features=None, max_leaf_nodes=None,
          min_samples_leaf=1, min_samples_split=100,
          min_weight_fraction_leaf=0.0, n_estimators=50, presort='auto',
          random_state=42, subsample=1.0, verbose=0, warm_start=False)
```

Results

Model Evaluation and Validation

The model was evaluated in 2 ways:

- Using the randomly selected test data held out from the training set of weeks 5 through 8
- Testing the same model on week 9 data (unseen data)

The initial model resulted in errors of:

- Test Data: 255.12 (1146.33 benchmark) = 77.75% reduction in error
- Unseen Data: 457.65 (1206.79 benchmark) = 62.08% reduction in error

The results are good compared to the benchmark but it's obvious that the model loses accuracy when predicting for unseen data.

The optimal model from hyper-parameter tuning does improve marginally on the initial model for test data but the results for unseen data marginally worsen:

- Test Data: 253.68 = 0.56% improvement
- Unseen Data: 459.06 = 0.31% deterioration

Justification

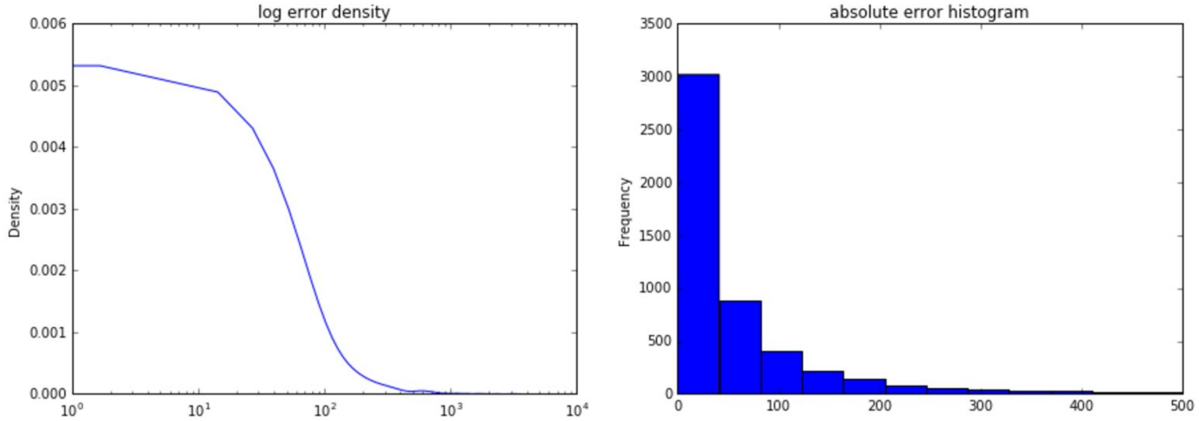
The model's results show a significant improvement (>60% reduction in error on unseen data) on the chosen benchmark (a 3 week average of demand) and from this, then, the model arrived could be said to be useful in predicting future week inventory.

However, the error is still quite large compared to the overall demand for products and would require further refinements to be of use. This is evidenced when comparing this model's results to the winning competition entry on Kaggle[6], where an root mean squared log error (RMSLE) of 0.426 won. This model's RMSLE is in the region of 2.4.

[6] https://www.kaggle.com/c/grupo-bimbo-inventory-demand/leaderboard/public

Conclusion

Free-Form Visualization



The chart above shows the logged error density for the test data set alongside the absolute error distribution. It shows that a large number of the prediction points are very close to the observed demand but that a few data points have a much larger error (which reflects also the test data set where a few data points have much higher demand than most of the rest of the test data).

Reflection

The project was very interesting in seeing how machine learning techniques could be applied to and refined on a forecasting problem. Each step of the process built on the previous one, adding insight into the problem domain:

- Exploration: gaining a reasonable understanding of the problem domain through data analysis and visualization to see how each attribute is distributed in of itself and with respect to the other attributes. This showed via a strong correlation how useful previous weeks' demand data would be to the model but also how difficult and many-dimensional the final model would need to be to capture the relationship demand has across the massive client base and product set that Grupo Bimbo operate within.
- Engineering: Identifying and then creating additional input features, this part concentrated on transposing the row-wise week by week demand data into lagged week data as input into current week prediction (eg. week 5,6,7 and input features for week 8) as well as parsing aspects of the agency, client and product lookup datasets to provide features that could be used to cluster each of these data elements so that instead of dealing with each individual agent, product, client an aggregated view of each could be created.

- Baseline Creation: The creation of a benchmark to measure the model's success. Using the average of the previous 3 weeks' demand.
- Model Creation: Taking the engineered data, holding out an entire week as unseen data and then using a randomised split on the remaining data to create a train and test set for model building.
- Model Refinement: Using grid search techniques to tune the model's hyper-parameters in order to find the optimal model.

The final model created, whilst better than the moving average benchmark, doesn't perform well compared to the top 1000 entries of the Kaggle competition from which the dataset was drawn.

Whilst, the model is plausible and does predict demand for the bakery products, the error is too large to be of much use in real-life. Thus, improvements would be needed to enable the model to be used.

Improvement

There are number of lines of attack to improve the model that's been created:

- More data: The current model is built with only 10% of the entire dataset due to memory constraints.
- More features: the lack of processing power compared to the size of the dataset meant that only a limited number of additional features could be engineered.
  - Client had to be removed from the dataset. Creating an engineered feature around client name would be interesting to see if certain client groups behave in specific ways.
  - RouteId was also excluded from the model and it would be interesting to explore any significant associations it has with other attributes to see a feature could be generated.
- Different algorithms: a further possible improvement would be to apply different models to the problem such as XGBoost or Neural Networks, which have been shown to perform well on time-series forecasting problems[7] compared to traditional decomposition methods.

To enable more features to be used, a bigger analysis environment would need to be created, probably using an amazon web services large memory ec2 instance. The primary motivation of creating a more scalable working environment would be to allow for more granular information to be retained in the dataset through the encoding of route and client information.

[7] https://www.researchgate.net/profile/Peter_Zhang6/publication/4871997_Neural_network_forecasting_for_seasonal_and_trend_time_series_Eur_J_Oper_Res/links/556343f708ae6f4dcc979689.pdf