Automobile Spare Parts Demand Forecasting Using Machine Learning

Oguzhan Akan, David Shin
California State University Northridge Computer Science

***Abstract*—This research utilizes an automotive distributor company's dataset, containing 5,000 spare parts, to test which of the three models - Gradient Boosting, Random Forest, and K-Nearest Neighbors will have better results in forecasting demand of automobile spare parts. The preliminary results showed that Random Forest model was the best model with a RMSE of 6.61. In the final results, the hyperparameters of the three models were optimized and evaluated against the Support Vector Machine and Multi-layer Perceptron Regressor. The research suggests that the Gradient Boosting algorithm performed the best against the other models with a RMSE score of 6.56. Main conclusions of the study are (i) Gradient Boosting is a useful algorithm for predicting intermittent demand in a data that has a decreasing (or increasing) trend, (ii) Hyperparameter optimization is a crucial step of modeling as it affects the outcomes significantly, and (iii) The difference between Gradient Boosting and Multilayer Perceptron algorithms are quite close to each other and thus requires future research with a more extended dataset.**

***Index Terms*—Machine Learning, Gradient Boosting, K Nearest Neighbors, Random Forest, Demand Forecasting, Spare Parts, Automotive, Multilayer Perceptron, Support Vector Machines**

I. Introduction

Automobile companies have various spare part products in their warehouses to cater towards customers' needs. The range of spare parts allow the industries to have modularity in their vehicles parts for repair and maintenance, however, it is difficult to predict when and what parts customers need. In order to address this issue, demand forecasting using machine learning helps industries recognize patterns to predict future sales. This allows their warehouses to be filled with parts customers will most likely need, reduce the shelf life of unnecessary parts and increase production in factories of parts that are highly requested. In return, production costs are lower and sales are higher and faster.

In this paper, we conduct a research on an extensive, 3-years of spare part order history data that are done by the dealers. The aim is to predict future trends in this very intermittent demand, and compare our Gradient Boosting approach with previous research - which use Support Vector Machines and BPNNs (BP Neural Network). We do not compare the model results with statistical approaches like Croston's method or Markov bootstrapping as our scope is only interested in Machine Learning area, and it has been mostly proved that Machine Learning approaches outperform statistical models in the recent studies [1].

As mentioned previously, in order to prevent shortage of products and to reduce wasted resources, existing works have used a wide range of tools to improve the performance of demand forecasting models. Also, common challenges for demand forecasting spare parts are the intermittent data and stochastic characteristics [1]–[4].

In our research, our objective is to conclude which of the three models: Gradient Boosting (GB), K-Nearest Neighbors (KNN) and Random Forest (RF) will perform better for demand forecasting automobile spare parts. The main language of the study is Python and the tools are used in the study are Pandas, NumPy, Plotly, and Sci-kit learn [5]–[7]. Our research compares three popular models and analyze them comprehensively for future researchers to have a direct comparison of these models in demand forecasting automotive spare parts.

A. Related Work

1) Logistic Regression and Support Vector Machine Hybrid (LRSVM): The authors designed the LRSVM to "synthetically evaluate autocorrelation of demand time series and relationship of explanatory variables with demand of spare part." By implementing a hybrid model, LRSVM was able to perform better than the Croston's method, Markov bootstrapping method, IFM method and the SVM method. The hybrid method allowed for the SVM forecast results of nonzero occurrences to be replaced with explanatory variables to formulate a more accurate prediction [8].

2) Grey's Forecasting Model (GFM) and BP Neural Network (BPNN) Hybrid: In this research, a hybrid model using the Grey's Forecasting model and BP Neural Network. The GFM was used to produce new data series from the original data with reduced noise to be used for forecasting. The BPNN then uses the new data series to generate prediction results. By combining these two models, the authors were able to produce higher forecasting precision than the models executed separately [9].

3) Fractional Order Discrete Grey Model (FOGM): Qiu et al proposed a derivation of the Grey's model, FOGM, which "... can effectively deal with the problem with "small sample" and "poor information", and can predict the data sequence with very few data samples." The genetic algorithm is used to estimate fractional order and generation coefficient for the FOGM to minimize the MAE of the predicted value. With this modification, the researchers are able to produce a model that is closer to the actual value than a single gray model [10].

4) Support Vector Machine (SVM): In this research, the SVM was used to make predictions of spare parts for the Canadian Armed Forces. In comparison to the ARIMA, Naive, Croston and SES, the researchers concluded that the SVM had a lower average A-MAPE than the other models that were compared. They also found that the Asymmetric Error, which is designed to "... highlight the costs incurred when over-forecasting, and the loss of operational availability when under-forecasting (orders go unfilled)," was lower for the SVM for intermittent demand series than any other models [11].

5) Gradient Boosting Model (GB) and Deep Neural Network (DNN): Authors of this experiment compared the GB and DNN models in retail for forecasting univariates sale time series. They proposed either of the two models can be relied on when there is a lack of causal factors or historical data. In this research, the GB performed better than the DNN and mentioned that the GB may have the tendency to overfit the data while the DNN were vulnerable to vanishing and exploding gradients [12].

B. Hypothesis

We have chosen the Gradient Boosting and Random Forest methods for this experiment because they are ensemble models. In our research, we found that hybrid models produced more effective results, which led us to believe that ensemble models would produce more positive results than using a single model [8]–[10]. To further verify this, we have selected the K-Nearest Neighbors method as a comparison method. We also believe that the Gradient Boosting model will outperform the other models because it utilizes multiple dependent trees and adjusts the weights of the trees, which is what we predict to be the most efficient when it comes to predicting seasonal products. With this knowledge, we predict the Gradient Boosting method will outperform the other models.

II. Dataset

The dataset used in this research is obtained from Dogus Automotive, a private automobile distributor based in Turkey. The company distributes cars of many brands including Volkswagen, Seat, Audi, Bentley, and Lamborghini and their spare parts. Dogus made its spare part catalog and order data public for their data competition in 2020, so our dataset is publicly available.

A. Properties of the Dataset

The dataset is made up of two tabular-format tables which can be joined together using **part ID** column. Here are the details of the tables:

- **catalog**: The catalog table consists of information related to each automobile part. There are 5,000 parts in this table and it has 5 descriptive features for each part: *part_definition_id, part_product_class_id, common_part_catalog_id, preferred_supplier_id, part_family_id*. Each descriptive feature refers to some ID number sourced in other tables that are not available. Therefore, we decide to use the ID numbers scratch in the feature selection phase. In order to decide the approach we want to go by with each feature, we first look how many unique values exist in each of them:

feature	**# of Unique Values**
part_definition_id	1918
part_product_class_id	446
common_part_catalog_id	29
preferred_supplier_id	5
part_family_id	225

Table I: Number of unique values exist in the data for each descriptive feature of a *part*.

- **orders**: The orders table consists of information about orders having 206,306 rows of data. A sample order information can be as simple as the related *part_id, order_quantity, date*, and *firm_id*. Therefore, we do not have any descriptive features in this table yet. Still, we can create features from scratch using Time Series Analysis as the history of each parts' orders matters.

III. Methodology

For our research we selected three algorithms with one being a base estimator and other two being different kinds of ensemble Machine Learning algorithms. We found there were not many researches using these algorithms in this field. The three algorithms we will be experimenting in this project are K-Nearest Neighbors, Random Forest and Gradient Boosting. To implement the research, we used Jupyter Notebooks in Python 3 with the data manipulation and modeling tools such as Sklearn, Pandas, and Numpy. For data visualization we made use of Seaborn and Matplotlib.

A. Algorithms

1) K-Nearest Neighbors: K-Nearest Neighbors (KNN) is a classification technique that uses a given set of "neighbors" to classify new data. The K represents the number of nodes the algorithm will use to classify a new data and it determines the new data classification by finding the distance of the new node's neighbors [13].

2) Random Forest: Random Forest is an ensemble learning method that utilizes decision trees to make predictions by taking either the mode of the decision trees or the mean prediction of each tree [14].

3) Gradient Boosting: Gradient Boosting is a machine learning algorithm for regression and classification problems. It produces predictions, usually by using decision trees, and minimizes error by modifying the weights of the trees that produce the best outcomes [15].

B. Validation Strategy

We decided to move forward with Time Series Cross Validation (TSCV) method in this research as it fit the ML goal's better in terms of seasonality. In TSCV, we select K folds just as we do in the K-fold Cross Validation, but we separate the folds according to a Time Series split instead of a random and/or shuffled split [16]. The intuition behind this choice is that we see a gradually decreasing trend in orders table (Figure 1).

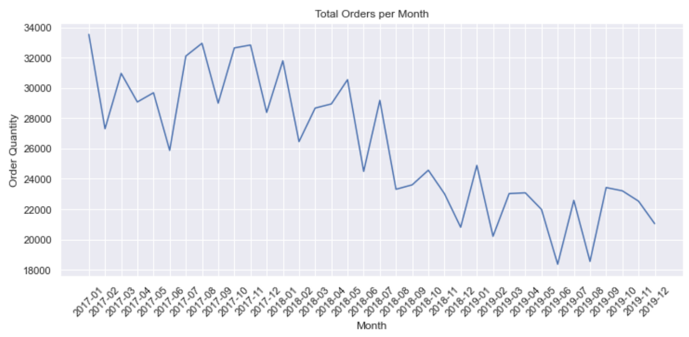
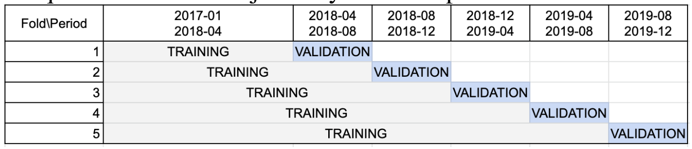


Figure 1. Decreasing trend in total order quantities with respect to each month.

Now that we have defined our validation strategy, we can select the time ranges we want build the models on. We selected 5 Folds with each fold's validation period being 5 months (Figure 2), based on the fact that 5 months of order data consisting of 25,000 rows is an adequate number to objectively evaluate performance metrics.



Fold\Period	2017-01 2018-04	2018-04 2018-08	2018-08 2018-12	2018-12 2019-04	2019-04 2019-08	2019-08 2019-12
1	TRAINING	VALIDATION				
2	TRAINING		VALIDATION			
3	TRAINING			VALIDATION		
4	TRAINING				VALIDATION	
5	TRAINING					VALIDATION

Figure 2. Time Series Cross Validation (TSCV) with 5 folds used in this research.

C. Preprocessing

1) Handling Outliers: Before applying bins to the data, a clamp transformation has been applied to it as the data had outliers. We cut the data into bins while setting the maximum quantity to 72. That is because after observing the frequencies, values greater than 72 can be considered as outliers. Since the lowest value of an order quantity can be minimum zero, no transformation is applied to the lower bound. After applying clamp transformation to the target feature, *order_quantity*, it is seen that the target is much skewed to the left (positively skewed) with a skewness value of 4.51 (Figure 3). We did not apply any normalization to this feature as it is the target feature we want to predict, thus cannot be transformed to any other scale.

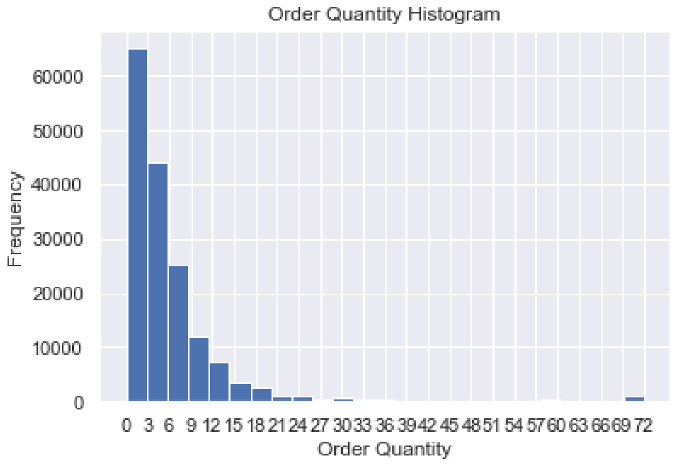


Figure 3. order_quantity_bin histogram graphic that shows the skewness in the data.

2) Data Imputation: Since the *orders* table has the rows at which some order exists, it does not give an information about the days where there is no order, which may be an insightful detail when building the model. Therefore, we inserted zero orders to the days where there is no order for a specific *part_id*. After this imputation, the dataset's initial size of 206,306 rows increased to 237,805.

3) Handling Duplicate Rows: Since the aim of the project is to predict the monthly orders for each *part_id*, we need to have exactly 1 row for each part in the training data. Therefore, the raw data, where there may be more than one order for one part in some month, is grouped by the *part_id* and returned their sum or *order_quantity*. Ultimately, our training data consisted of 167,059 rows.

4) Time Series Analysis: "Time series analysis is a method to establish a stochastic model for time series data based on its property, and utilizes the stochastic model to predict the long term trend" [17]. In order to predict the order quantities for a given month, it is a good practice to generate time series features in addition to the descriptive features of *part_id*s. We generate lag-N features where N represents some number of months. For example, *order_quantity_lag3* represents the order quantity 3 months before. Then we take the sum, mean, and standard deviations of lag-N features for $1 \leq N \leq 6$ to test which lag best explains the target feature.

5) Clustering Categorical Features: The descriptive features of *part_ids* are in ID format, meaning that they cannot be used for modeling purposes as a lot of Sklearn's algorithms cannot handle categorical features. Thus, we need a way to convert these categorical features into continuous features. We use Binary Encoding, which encodes information to a bit, for the features that have a few number of categories [18]. For the rest, we will apply a K-means clustering with K=5. K-means clustering is an unsupervised learning algorithm that groups data points into K clusters by calculating their Euclidean distance [19]. The algorithm first randomly creates K center points which are called centroids, then in every iteration, it assigns the data points to the centroid which has minimum distance to it.

The reason why we applied K-means clustering to some of our categorical features is avoiding the possible overfitting during modeling. As it can be seen in Table I, there are too many distinct values exist for *part_definition_id, part_product_class_id, and part_family_id* that can cause the model to overfit.

Therefore, we used normalized order quantity averages and standard deviations of each unique value in each feature to assign them clusters. Figure 4 shows before and after clustering of the three features.

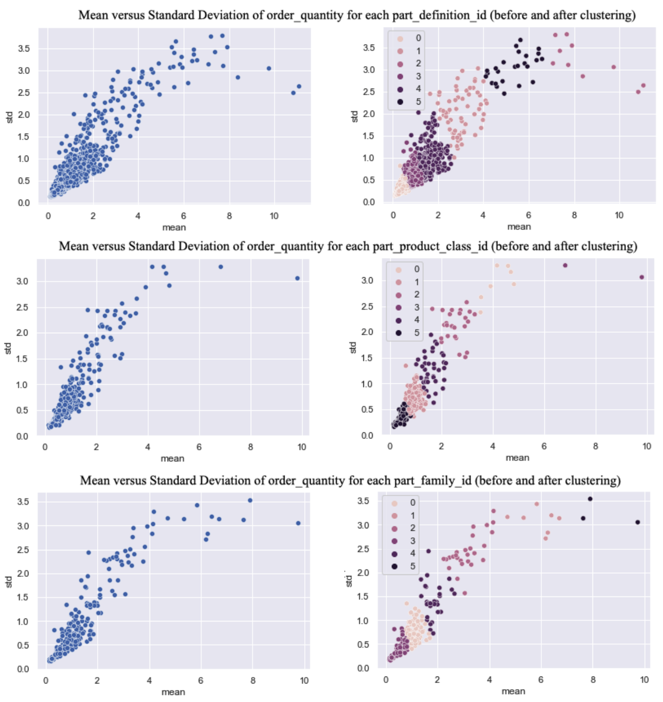


Figure 4. Scatterplot of three descriptive features, mean versus standard deviation of order_quantity, before and after K-means clustering.

6) Binary Encoding Categorical Features: As we clustered three of five descriptive features of *part*s, we need to transform the remaining two features, *common_part_catalog_id* and *preferred_supplier_id*, into either a continuous variable or a binary variable in order to be able to use them for modeling. We chose to convert *common_part_catalog_id* with a default transformation as it still have a lot of unique values. *part_family_id*, however, can be transformed to a binary variable using binary encoding. Binary encoding is a categorical feature transformation method that creates N new binary features where N is the number of unique values for that categorical feature [18]. Thus, in our case, we create 5 new features for *preferred_supplier_id* and specify them with a suffix *_binenc[N]*.

7) Handling Null Values: When we create new features, particularly in Time Series Analysis, some values have null value due to non-existent previous data. Therefore, we need to fill those null values in order to build the models. As a common approach, we first tried to fill the nulls with their category means. After model trials, however, we decided to continue with filling with category maximums because we see a decreasing trend in order quantities. The intuition behind is that the lag-N features bring the order quantities N months ago, and they can be even higher than the category maximum for this specific case.

8) Final Training Data: The training data consists of 167,059 rows with the target feature, *order_quantity_bin*, and descriptive features:

- time_diff_mean
- time_diff_std
- order_quantity_lag[N] where $1 \leq N \leq 6$
- order_quantity_lag[N]_mean where N in (3, 6)
- order_quantity_lag[N]_std where N in (3, 6)
- pdid_cluster
- pfid_cluster
- pcid_cluster
- psid_binenc[N] where $1 \leq N \leq 5$
- psid
- cpcid

D. Feature Selection

Before building the models we want to eliminate some of the features we have if they are highly correlated with each other. But how do we decide which one to drop if we have two correlated features? In order to build better models, we need to know which one is describing the outcome better than the other. Therefore, we first use a SVM algorithm from Sklearn's Linear Support Vector Regression module in order to get coefficients of each feature. After applying the SVM, we see the most and least correlated features with the target feature (Figure 5).

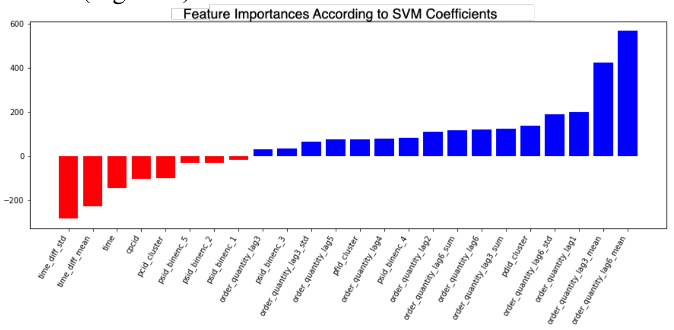


Figure 5. Bar chart of feature importance according to Support Vector Machine Regressor coefficients.

Now that at least we know which feature performs better than the other, we can build a correlation matrix and determine a maximum threshold that can two features be correlated. We selected 0.90 as the threshold and starting from the most important feature, we scanned the rest of the features and excluded them when necessary. Figure 6 shows the correlation matrix where darker colors represent higher correlation.

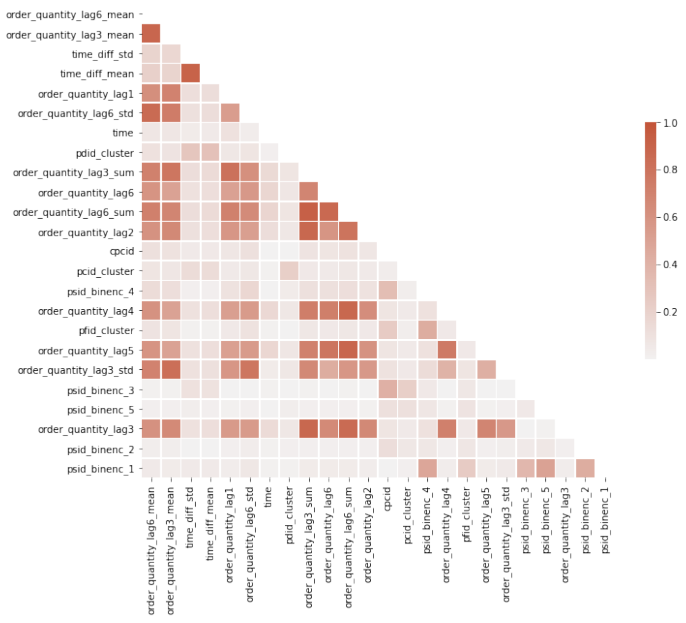


Figure 6. Correlation matrix of features ordered from most important to least important.

After running the scan, we drop the following features due to high correlation with other features: *time_diff_mean, order_quantity_lag6_sum*

IV. Preliminary Results

After inserting missing rows, correcting the data, clustering categorical features, filling null values, dropping correlated features, deciding on the modeling algorithms and validation strategy, we finally build the KNN, RF, and GB models. We use Sklearn's KNeighborsRegressor, RandomForestRegressor, and GradientBoostingRegressor modules respectively to build our models. There are many parameters used in each model, but in this research we will focus only on some of them for the sake of simplicity.

For evaluating performances of each type of model, we decided on considering Mean Absolute Error (MAE) and Mean Squared Error (MSE) as these metrics are commonly used, and MAE ranges is in the target's scale, thus making it easier to interpret the results.

A. K-Nearest Neighbor Model (KNN)

In the KNN model we set *n_neighbors* parameter to *5* in order to limit computing power. The remaining parameters are in their default settings. For example, the *weights* parameter is set to 'uniform', meaning that all the points in each neighborhood are weighted equally.

After running the KNN model, we obtained a RMSE of *7.2032* from our validation data. Table II represents the metrics details against each fold.

	fold_1	fold_2	fold_3	fold_4	fold_5	avg
rmse_train	6.140292	6.250312	6.369166	6.520116	6.706615	6.397300
mae_train	3.507478	3.590461	3.700427	3.818290	3.961042	3.715540
rmse_valid	7.050073	6.896217	7.043624	7.370869	7.655564	7.203269
mae_valid	3.919632	3.882288	3.899062	4.235225	4.576852	4.102612

Table II: Training and validation results of KNN model.

B. Random Forest Model (RF)

RF model can have many parameters regarding tree creation or ensembling, and here are our settings different from the Sklearn's default parameters:

- n_estimators=100
- criterion='mse'
- max_depth=7
- min_samples_split=100
- min_samples_leaf=40
- max_leaf_nodes=45

After running the RF model, we obtained a RMSE of *6.6147* from our validation data. Table III represents the metrics details against each fold.

	fold_1	fold_2	fold_3	fold_4	fold_5	avg
rmse_train	6.790433	6.874281	6.997079	7.130805	7.274396	7.013399
mae_train	3.984479	4.057571	4.166363	4.291245	4.444998	4.188931
rmse_valid	6.463199	6.427443	6.501119	6.714780	6.966765	6.614661
mae_valid	3.617103	3.634180	3.724027	3.931205	4.197993	3.820902

Table III: Training and validation results of RF model.

C. Gradient Boosting Model (GB)

In GB model we set the parameters as the same as they are being used in RF model. Since the two algorithms are both based on decision trees, they have similar parameters in creating the trees. They differ in the way they ensemble, and we let them be in their default for now.

After running the GB model, we obtained a RMSE of *6.8383* from our validation data. Table IV represents the metrics details against each fold.

	fold_1	fold_2	fold_3	fold_4	fold_5	avg
rmse_train	7.135785	7.252168	7.420062	7.586390	7.740793	7.427040
mae_train	4.188768	4.305372	4.468736	4.626195	4.780832	4.473981
rmse_valid	6.655619	6.638982	6.658236	6.937305	7.301183	6.838265
mae_valid	3.787618	3.867394	3.992326	4.289923	4.555360	4.098524

Table IV: Training and validation results of GB model.

V. Final Results

In this section we investigate what may cause a model would perform better or worse from one another. We ran the three models, KNN, RF, and GB, with different parameter sets in order to find out which model is actually performing the best. In following subsections, we share the effects of changing parameters in our performance metric, RMSE.

A. KNN with increasing number of neighbors

After running the KNN model with 2, 3, 4, ..., 10, 20 neighbors, we end up with an optimal RMSE of *6.77* when the neighbors n=20, which is significantly better than it was when neighbors n=5 (7.20). Figure 7 depicts how training and validation scores differ when the number of neighbors increase. It can be seen that when $n \leq 5$, there is an obvious overfitting problem. Lastly, although n=20 neighbors perform the best in validation dataset, we should prefer to finalize our model with n=10 neighbors since the difference in RMSE between training and validation sets is minimum, and the overall RMSE is still good enough.

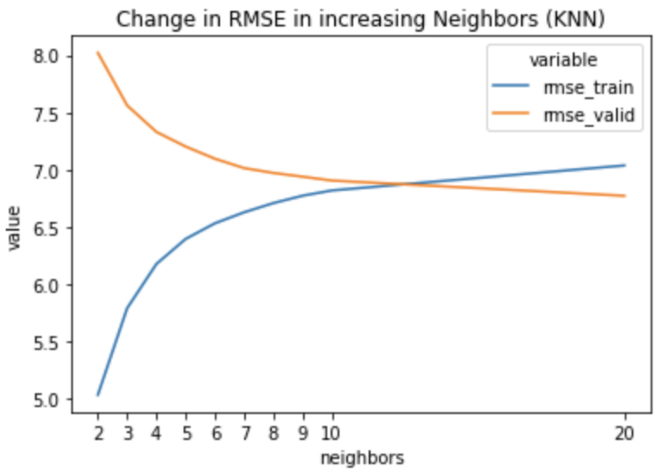


Figure 7. The trend in KNN model's performance in increasing number of neighbors.

B. *RF with increasing number of estimators*

The RF model's performance changes extremely little in increasing number of estimators. Unlike the KNN model, we observe distance between training and validation MAE's which pushes us to question the necessity of using multiple trees instead of just one tree. Indeed, the RF is an ensemble model that uses averaging among its trees, but if 10 trees perform the same with 100 trees, then it would only be wasting time to use the algorithm. Figure 8 shows how RMSE's stay still while we increase the number of estimators. Ultimately, the RF model still performs with an optimum RMSE of 6.61, which is greater than the KNN model.

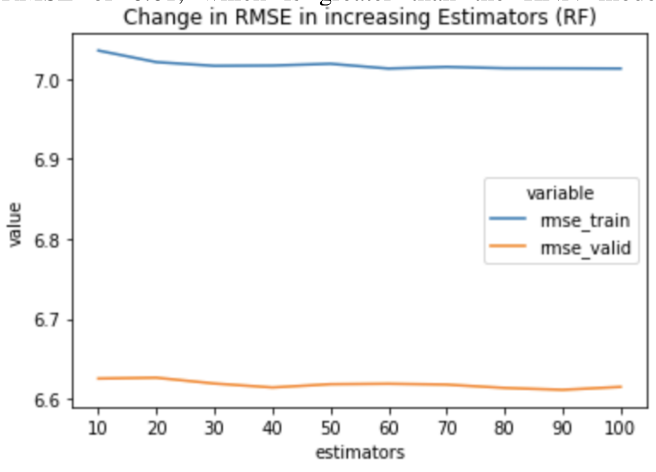


Figure 8. The trend in RF model's performance in increasing number of estimators.

C. *GB with increasing number of estimators*

As we argued in our initial hypothesis, the GB model performed the best among the three models with an optimum RMSE of 6.56 when the number of estimators n=1000. Since the ensembling method of GB is different than RF, that is, GB calculates its errors in every other estimator while RF is a vote-based algorithm, we were already expecting a decrease in the error when we increase the number of estimators. Due to our limited computing power, we did not investigate further the parameter changes effects, however, we have significant difference between GB and RF to conclude that GB is the best-performing model among all three algorithms. Figure 9 shows how the change in estimators change model scores. It is also important to note that increasing number of estimators may not always be beneficial, as for example, n=800 have higher score than n=700 in the GB model.

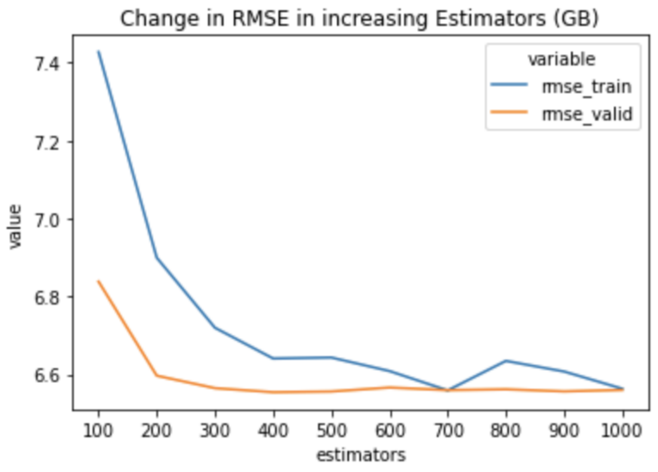


Figure 9. The trend in GB model's performance in increasing number of estimators.

VI. Discussion

A. *Model Comparison*

1) Initial Performance: Contrary to KNN, RF and GB models performed better in validation sets as opposed to training sets. This occurred because of the decreasing trend in the data. Although we divided the training and validation sets using time split, we needed to include more features that are related to seasonality. Nonetheless, both RF and GB performed better than KNN in validation sets.

As we draw the line plots of each model's predictions and the actual outcome, we see a very similar trend in RF and GB, while KNN's trend is closer to the actual outcome (Figure 10). Although KNN's trend is similar to actual outcomes, we must do the comparison using the actual MAE's of the models.

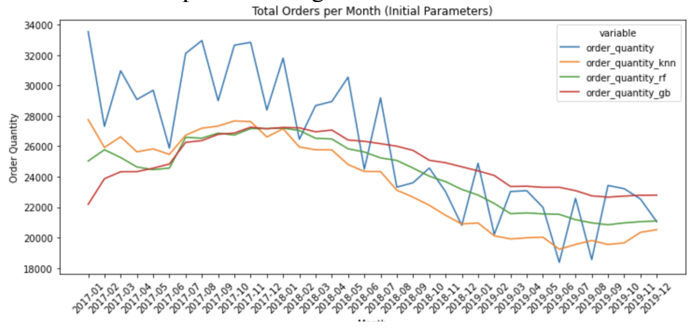


Figure 10. Initial prediction results and actual outcome for KNN, RF, GB models.

Our initial hypothesis suggested that GB would perform better than RF but when their RMSEs compared, RF seemed to perform better. This initially proved that RF is a better intermittent demand forecasting algorithm, although the preliminary results lack the usage of hyperparameter optimization.

2) Final Performance: After running each algorithm with different parameter sets, we have ended up with a best RMSE of 6.56 using GB, which supports our hypothesis. As we draw the actual versus predicted order quantities graph, we can see that the prediction trend is more similar the actual outcome, which shows the importance of the parameters in Machine Learning algorithms (Figure 11).

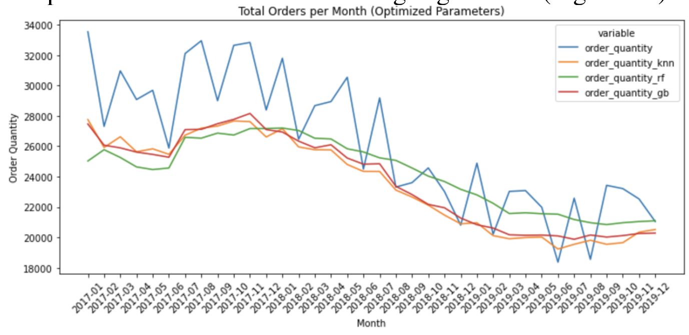


Figure 11. Final prediction results and actual outcome for KNN, RF, GB models.

It is easy to notice that how GB got close to actual outcome as opposed to our initial results. Although KNN predicted very similar with GB, GB was slightly more successful in catching jumps and downs. Lastly, RF seemed to perform the same as before.

3) Efficiency: It is well-known that ensemble models are expected to work slower than the base estimators. As we calculated the time spent to fit models, we saw that KNN model fitted in *5.053* seconds, while RF fitted in *58.620* seconds, and GB fitted in *103.104* seconds. Nonetheless, we can increase or decrease fitting times of each model by changing parameter. The question we are interested in is ”How can we shorten the duration without losing validation performance?”

B. Novelty of the Reserach

1) Machine Learning Algorithms: After a careful literature review, it is observed that there are few previous researches done on Gradient Boosting Regression on demand forecasting. This research proves that Gradient Boosting is also an applicable algorithm for this subject area.

2) Data Preprocessing Steps: We generally observed Time Series Analyses in the previous research. In addition to Time Series, we applied (i) clustering the categorical features, (ii) inserting zero orders for the days that there were no orders, and (iii) Binary Encoding of the rest of categorical features.

C. Comparison with Previous Work

1) Comparison with SVM: Our trial with SVM model has shown that the GB model still performs better, although we lacked of the exact steps taken in the actual paper. After running the SVM model, we obtained the results presented in Table V. With 7.01 RMSE, the SVM algorithm performs very similar to our RF model, and worse than our GB model. Therefore, we can conclude that the GB can be an alternative algorithm in this field.

	fold_1	**fold_2**	**fold_3**	**fold_4**	**fold_5**	**avg**
rmse_train	7.416550	7.546724	7.743978	7.944773	8.200194	7.770444
mae_train	4.219605	4.327490	4.498295	4.663637	4.869179	4.515641
rmse_valid	6.768481	6.803502	6.787505	7.190507	7.494893	7.008978
mae_valid	3.610522	3.639753	3.691825	3.952011	4.202671	3.819357

Table V: Training and validation results of SVM model.

2) Comparison with BPNN: We imitated the BPNN in Python using Sci-kit Learn’s MLPRegressor module with a maximum number of iterations from 200, 300, 400, ..., 1000. Ultimately obtained a RMSE score of 6.81 in the same dataset, which is very close to our GB model’s score (6.56). The difference is not significant as opposed to it was in regards to SVM, thus a further research on difference between BPNN and GB can be conducted to conclude one algorithm precedes the other. The results obtained from BPNN model shared in Table VI.

	fold_1	**fold_2**	**fold_3**	**fold_4**	**fold_5**	**avg**
rmse_train	6.739252	6.930863	7.035137	7.304298	7.409231	7.083756
mae_train	3.852870	4.035582	4.188096	4.338786	4.359530	4.154973
rmse_valid	6.593811	6.693984	6.602706	6.942484	7.218114	6.810220
mae_valid	3.568023	3.637440	3.628372	3.926752	4.183883	3.788894

Table VI: Training and validation results of BPNN model.

VII. Conclusion

The computing power was a primary obstacle of our research. As we ran 100 different models with local computers, and it took nearly 24 hours to run all of them. Therefore, we recommend future researchers to direct towards cloud computing resources for obtaining better results.

Other challenges of this project include newly added products, missing parameters, outliers, computation power and the time restriction [20]. As a project is demand forecasting, newly added products may not have enough historical data to make an accurate prediction. Some of the missing parameters are price and size. This is a limitation to our project because these parameters may have strong correlations to sale orders. The price of an item may be correlated to sale orders because if the price of a product is expensive, the sale orders may be less and vis versa. The size may affect sale orders because if the object is too small or too big, it may be ordered more or less respectively. The computation power can be expensive as hyperparameters are increased. And finally, the time restriction is a limitation to our project because we may not have as much time as we would like to train and test our models for improved performance.

Another recommendation for future research would be optimizing parameters with Bayesian hyperparameter optimization in the GB model. Since the primary goal of this research is to evaluate if Gradient Boosting is a good alternative in intermittent demand forecasting, a Grid Search within the parameter space is adequate to have a conclusion. Still, there is an area of improvement in building the best GB model.

Ultimately, based on the research conducted, we can now accept the hypothesis that suggests GB performs better in terms of RMSE than RF and KNN. Furthermore, after comparing with previous work that was done with SVM, GB still stands as a good alternative for predicting demand in spare parts industry. Lastly, we still wonder how GB would differ from the Neural Networks (NN). In this research, we compared our results with BPNN, though the difference is too small to have a conclusion. Besides, there are specific NNs that might perform well in demand forecasting, such as Recursive NNs. As a last word, we suggest performing another research just aiming to exploit the performances between NNs and GBs.

References

[1] L. Shenstone and R. J. Hyndman, "Stochastic models underlying croston ' s method for intermittent demand forecasting stochastic models underlying croston ' s method for intermittent demand forecasting," *Business*, 2003.

[2] M. R. Amin-Naseri and B. Rostami Tabar, "Neural network approach to lumpy demand forecasting for spare parts in process industries," *Proceedings of the International Conference on Computer and Communication Engineering 2008, ICCCE08: Global Links for Human Development*, pp. 1378–1382, 2008.

[3] I. RuiRui Xing, Xianliang Shi, Member, "A BP-SVM combined model for intermittent spare parts demand prediction," pp. 1085–1090, 2019.

[4] Q. Xu, N. Wang, and H. Shi, "A Review of Croston's method for intermittent demand forecasting," *Proceedings - 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2012*, no. 61100009, pp. 1456–1460, 2012.

[5] W. McKinney and P. D. Team, "Pandas - powerful python data analysis toolkit," *Pandas - Powerful Python Data Analysis Toolkit*, 2015.

[6] S. V. D. Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: A structure for efficient numerical computation," *Computing in Science and Engineering*, vol. 13, 2011.

[7] B. Ghaddar and J. Naoum-Sawaya, "High dimensional data classification and feature selection using support vector machines," *European Journal of Operational Research*, vol. 265, 2018.

[8] Z. Hua and B. Zhang, "A hybrid support vector machines and logistic regression approach for forecasting intermittent demand of spare parts," *Applied Mathematics and Computation*, vol. 181, no. 2, pp. 1035–1048, 2006.

[9] H. Song, C. Zhang, G. Liu, and W. Zhao, "Equipment spare parts demand forecasting model based on grey neural network," in *2012 International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering*, 2012, pp. 1274–1277.

[10] Q. Qiu, C. Qin, J. Shi, and H. Zhou, "Research on demand forecast of aircraft spare parts based on fractional order discrete grey model," in *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*, 2019, pp. 2212–2216.

[11] A. Boukhtouta and P. Jentsch, "Support vector machine for demand forecasting of canadian armed forces spare parts," in *2018 6th International Symposium on Computational and Business Intelligence (ISCBI)*, 2018, pp. 59–64.

[12] K. Wanchoo, "Retail Demand Forecasting: a Comparison between Deep Neural Network and Gradient Boosting Method for Univariate Time Series," pp. 5–9, 2019.

[13] S. Khan, Z. A. Khan, Z. Noshad, S. Javaid, and N. Javaid, "Short Term Load and Price Forecasting using Tuned Parameters for K-Nearest Neighbors," *ITT 2019 - Information Technology Trends: Emerging Technologies Blockchain and IoT*, pp. 89–93, 2019.

[14] A. Lahouar and J. Ben Hadj Slama, "Random forests model for one day ahead load forecasting," *2015 6th International Renewable Energy Congress, IREC 2015*, 2015.

[15] M. Gumus and M. S. Kiran, "Crude oil price forecasting using XGBoost," *2nd International Conference on Computer Science and Engineering, UBMK 2017*, pp. 1100–1103, 2017.

[16] R. Medar, V. S. Rajpurohit, and B. Rashmi, "Impact of Training and Testing Data Splits on Accuracy of Time Series Forecasting in Machine Learning," *2017 International Conference on Computing, Communication, Control and Automation, ICCUBEA 2017*, no. April 2020, pp. 1–6, 2018.

[17] Tingting Huang, L. Wang, and T. Jiang, "Prognostics of products using time series analysis based on degradation data," in *2010 Prognostics and System Health Management Conference*, 2010, pp. 1–5.

[18] M. D. Todd, *Sensor data acquisition systems and architectures*. Woodhead Publishing Limited, 2014, vol. 1. [Online]. Available: http://dx.doi.org/10.1533/9780857099136.23

[19] S. Nuchprayoon, "Electricity load classification using K-means clustering algorithm," *IET Conference Publications*, vol. 2014, no. CP649, 2014.

[20] B. M. Pavlyshenko, "Machine-learning models for sales time series forecasting," *Data*, vol. 4, no. 1, pp. 1–11, 2019.