UDACITY MACHINE LEARNING NANODEGREE

CAPSTONE PROJECT

Predict Backorders

by

Hui-yu Yang

August 2017

Contents

Chapter 1

Introduction

1.1 Project Overview

Backorders happen when supply outstrips the demand, which is a problem for businesses that may result in lower customer satisfaction or even loss of loyal customers. Multiple factors such as incompetent management, miscommunication, or lack of data to accurately forecast the demand [1]. An accurate prediction of backorders can prevent competitor taking advantages of the situation, or potentially losing loyal customers.

The dataset on Kaggle contained the historical data for 8 weeks prior to the week we were looking to predict [2]. It was taken as weekly snapshots at the start of each week with 22 features below:

- sku - Random ID for the product
- national_inv - Current inventory level for the part
- lead_time - Transit time for product (if available)
- in_transit_qty - Amount of product in transit from source
- forecast_3_month - Forecast sales for the next 3 months
- forecast_6_month - Forecast sales for the next 6 months
- forecast_9_month - Forecast sales for the next 9 months
- sales_1_month - Sales quantity for the prior 1 month time period
- sales_3_month - Sales quantity for the prior 3 month time period
- sales_6_month - Sales quantity for the prior 6 month time period
- sales_9_month - Sales quantity for the prior 9 month time period
- min_bank - Minimum recommend amount to stock
- potential_issue - Source issue for part identified
- pieces_past_due - Parts overdue from source
- perf_6_month_avg - Source performance for prior 6 month period
- perf_12_month_avg - Source performance for prior 12 month period
- local_bo_qty - Amount of stock orders overdue
- deck_risk - Part risk flag
- oe_constraint - Part risk flag
- ppap_risk - Part risk flag
- stop_auto_buy - Part risk flag
- rev_stop - Part risk flag
- went_on_backorder - [Target variable] Product actually went on backorder.

1.2 Problem Statement

My main objective is to build a predictive model using a supervised machine learning technique to identify parts at risk of backorder before the event occurs. After a model is selected based on the performance metrics (XGBoost was selected), parameter tuning will be performed. In the end, the features that are most important for backorder prediction will also be presented. For this project, the following tasks will be performed:

1. Download and preprocess the Kaggle- "Can you predict backorders" data
2. Data exploration and visualization
3. Compare several supervised machine learning techniques to acquire the best model
4. Parameter tuning for the selected model
5. Model evaluation

1.3 Metrics

Even though accuracy is the most common metric for binary classifiers, it is not a good metric for imbalanced data as it is misleading due to the accuracy paradox. Accuracy only reflects the underlying distribution when the data is imbalanced, not how good the prediction is [3]. Due to the highly imbalanced dataset with 1:143 ratio, we will use the metrics below instead:

- **confusion matrix**: a 2 by 2 table showing a breakdown of correct and incorrect predictions. A confusion matrix shows the number of true positive (TP), false negative (FN), false positive (FP), and true negative (TN).

- **precision**: measure of classifier's exactness

$$Precision = \frac{TP}{TP + FP}$$

- **recall**: measure of classifier's completeness

$$Recall = \frac{TP}{TP + FN}$$

- **F1 score**: weighted average of precision and recall

$$F_1 = \frac{2TP}{2TP + FP + FN}$$

- **Cohen's kappa**: classification accuracy normalized by the imbalance of the data

- **ROC curve**: graphical examination of sensitivity vs 1-specificity. It exhibits the balance threasholds of sensitivity and specificity, which can be used for model selection.

Chapter 2

Analysis

2.1 Data Exploration

Data files Kaggle_Training_Dataset_v2.csv and Kaggle_Test_Dataset_v2.csv can be downloaded at https://www.kaggle.com/tiredgeek/predict-bo-trial. The training set has 1687860 samples and the test set has 242076 samples. Both of them have 22 features and 1 target variable. The data sample showed that there are missing data, text data, and some values such as -99 that may signal missing values (Table 2.1). The sample showed 22 features including target variable since one feature was used as the index.

Table 2.1: Sample Data from Training Set

sku	national_inv	lead_time	in_transit_qty	...	went_on_backorder
1026827	0.0	NaN	0.0	...	No
1043384	2.0	9.0	0.0	...	No
1043696	2.0	NaN	0.0	...	No

Training set and testing set has similar backorders rate, and the training set and testing set are both imbalanced with a ratio of 1:143 and 1:86, respectively (Table 2.2).

Table 2.2: Characteristics of Training and Test Set

	Training Set	Test Set
Sample Size	1687860	242075
Backorder Rate (%)	0.669	1.110

We can see that national_inv, perf_6_month_avg and perf_12_month_avg contains negative values, or -99, highlighted in Table 2.3. We can assume these are missing values and we can impute these in the preprocessing step. All of the categorical features are binary, and we should convert them into dummy variables for analysis purpose (Table 2.4).

We will examine whether there are any abnormalities in the data that needs to be addressed. Both performance metrics (perf_6_month_avg, perf_12_month_avg) are skewed to the left, so we will impute them with their median in the preprocessing step (Fig. 1).

As for missing values, the only feature with missing value is lead_time, and about 6% of lead_time is missing.

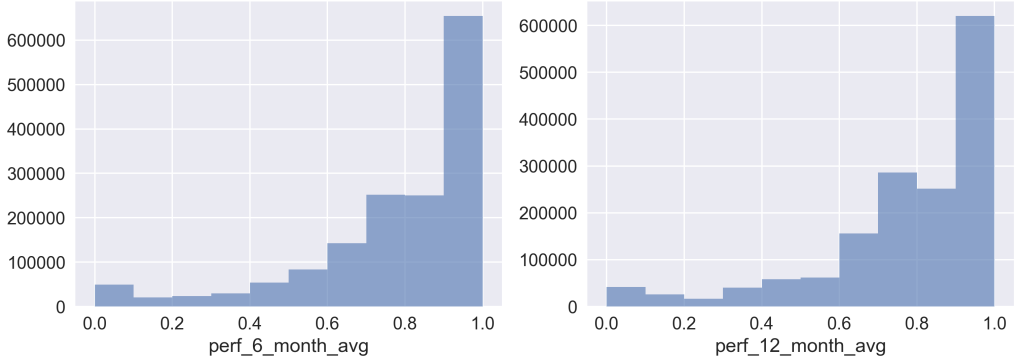
TABLE 2.3: Summary Statistics on Numerical Features

	N	Mean	St. Dev.	Min	Max
national_inv	1687860	496.11	29615.23	-27256.00	12334400
lead_time	1586967	7.87	7.06	0.00	52.00
in_transit_qty	1687860	44.05	1342.74	0.00	489408.00
forecast_3_month	1687860	178.12	5026.55	0.00	1427612.00
forecast_6_month	1687860	344.99	9795.15	0.00	2461360.00
forecast_9_month	1687860	506.36	14378.92	0.00	3777304
sales_1_month	1687860	55.93	1928.20	0.00	741774.00
sales_3_month	1687860	175.03	5192.38	0.00	1105478
sales_6_month	1687860	341.73	9613.17	0.00	2146625
sales_9_month	1687860	525.27	14838.61	0.00	3205172
min_bank	1687860	52.77	1254.98	0.00	313319.00
pieces_past_due	1687860	2.04	236.02	0.00	146496.00
perf_6_month_avg	1687860	-6.87	26.56	-99.00	1.00
perf_12_month_avg	1687860	-6.44	25.84	-99.00	1.00
local_bo_qty	1687860	0.63	33.72	0.00	12530.00

TABLE 2.4: Summary Statistics on Categorical Features

	N (%)
potential_issue	
Yes	907 (<1)
No	1686953 (99)
deck_risk	
Yes	387483 (23)
No	1300377 (77)
oe_constraint	
Yes	245 (<1)
No	1687615 (99)
ppap_risk	
Yes	203834 (12)
No	1484026 (88)
stop_auto_buy	
Yes	61086 (4)
No	1626774 (96)
rev_stop	
Yes	731 (<1)
No	1687129 (99)

Figure 2.1: Histograms of Perf_6_month_avg and Perf_12_month_avg



2.1.1 Outlier Detection

We performed outlier detection to avoid any particular data point (influential point) influencing our analysis results. Even though there are several methods for determining outliers in a sample, we will use a popular method called **Tukey Fences** [4]. Tukey used the quartiles to define "fences," and the values that fall outside the inner fences are considered to be outliers. The fences are defined as:

$$UpperInnerFence : Q_3 + 1.5 \times IQR$$
$$LowerInnerFence : Q_1 - 1.5 \times IQR$$

where Q_1 is the first quartile (25^{th} percentile), Q_3 is the third quartile (75^{th} percentile), and IQR are the interquartile range, or Q_3 - Q_1. The weakness of this fence-rule is that the length of the two whiskers are identical, or if the data has a symmetric distribution. However, we have transformed the data and so this should not be problem.

However, if we remove outliers that are outlying for more than 1 feature, then we would lose about 46% of data. It is too much data, so we will simply use a machine learning technique that's insensitive to outliers in the analysis step.

2.1.2 Exploratory Visualization

The boxplot of numerical features showed that most numerical features are greater than 0 except national_inv (Fig. 2.2). The variance of national_inv is much larger than the other variables as well. The scatterplot matrix showed that the forecasting metrics are linearly correlated, so are the sales metrics (Fig. 2.3). That makes sense as those variables are essentially snapshot of the same variable at a different time. All of the forecasting and sales variables are skewed to the right and nearly all show a linear trend (Fig. 2.4).

Figure 2.2: Boxplot of Numerical Features

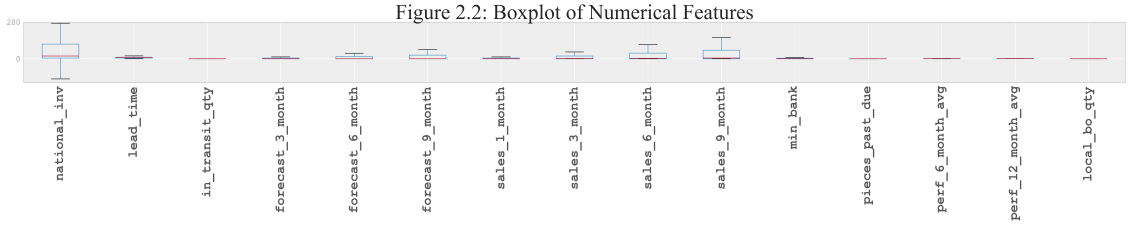
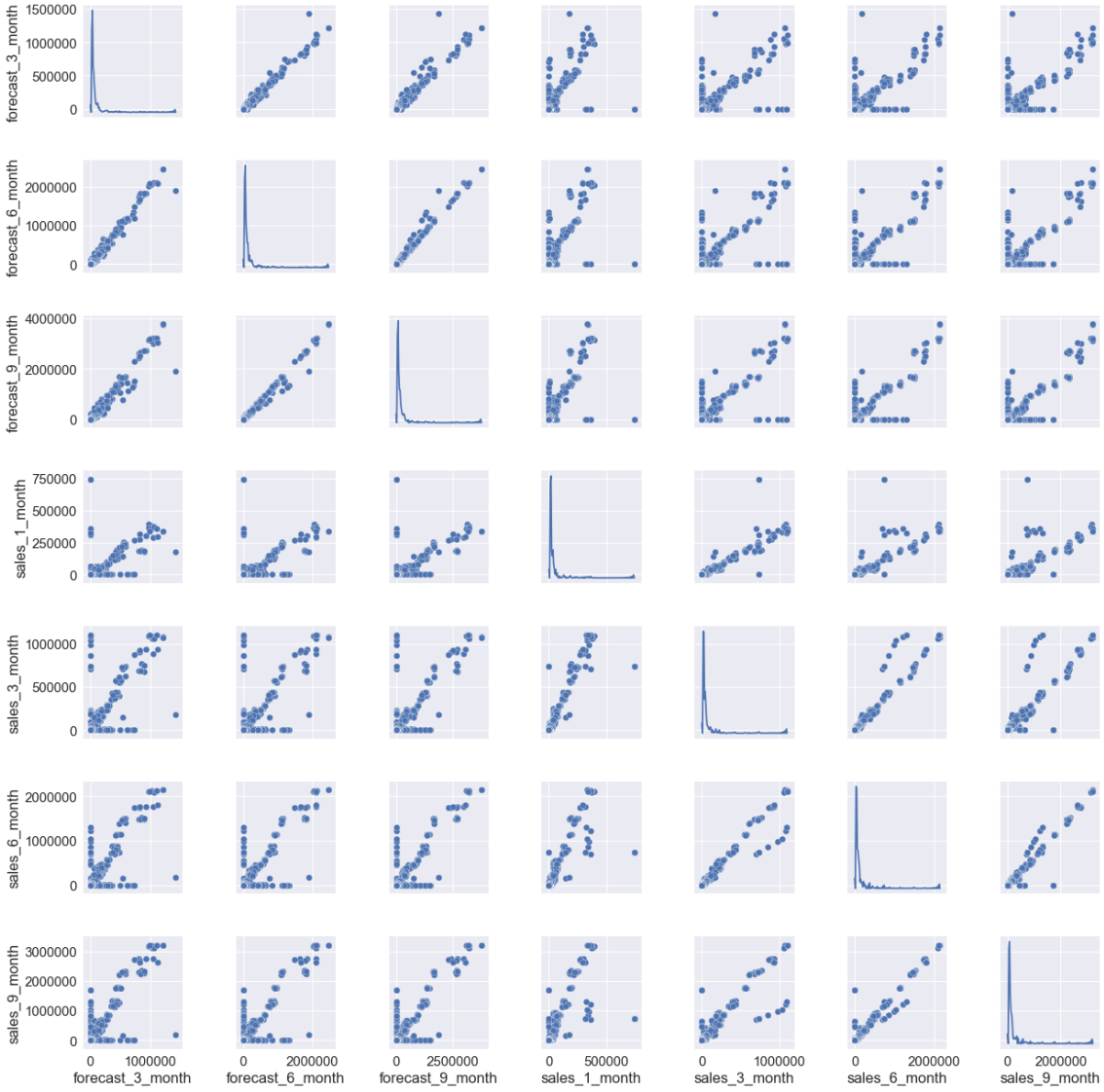


Figure 2.3: Scatterplot Matrix of features

Figure 2.4: Scatterplot of Features of Interest



The heatmap of features also showed the forecasting and sales features are correlated (Fig. 2.5). However, on top of that, it also showed that perf_6_month_avg and perf_12_month_avg are highly correlated with a correlation of 0.97. In addition, variables in_transit_ty, forecast_3_month, forecast_6_month, forecast_9_month, sales_1_month, sales_3_month, sales_6_month, sales_9_month, and min_bank are highly correlated as well. This strong correlation showed multicollinearity of these variables and these variables give almost the same information (Fig. 2.6).

Figure 2.5: Heatmap of Features

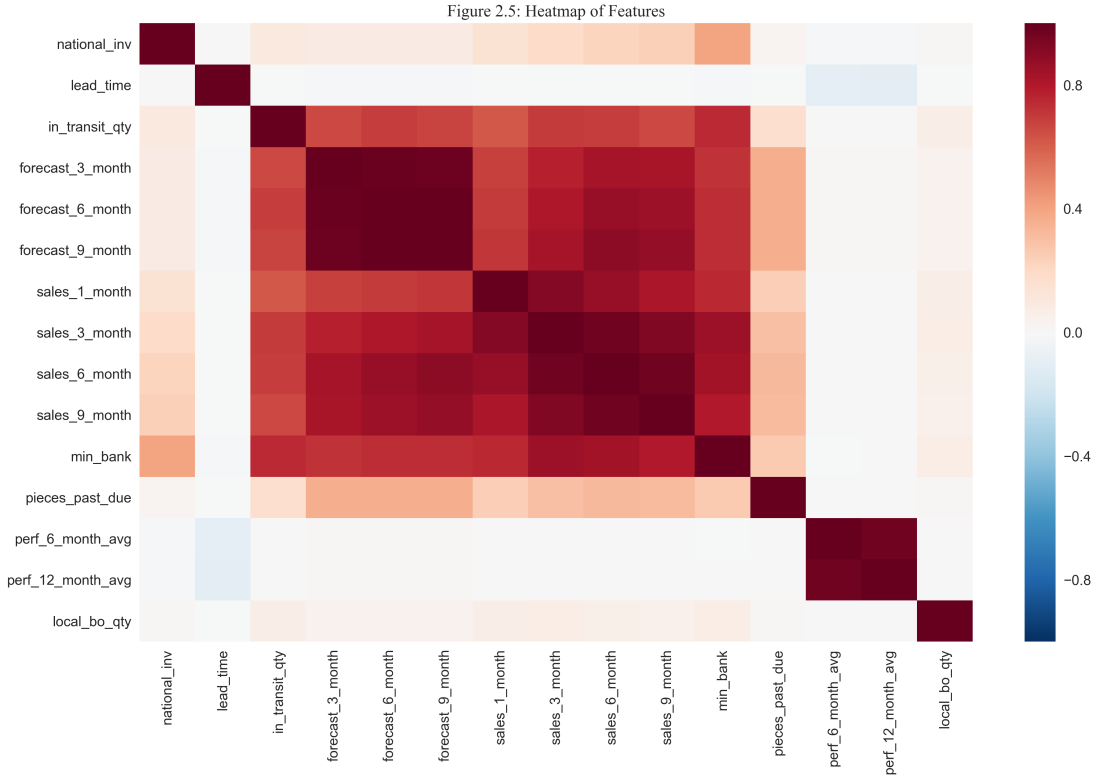
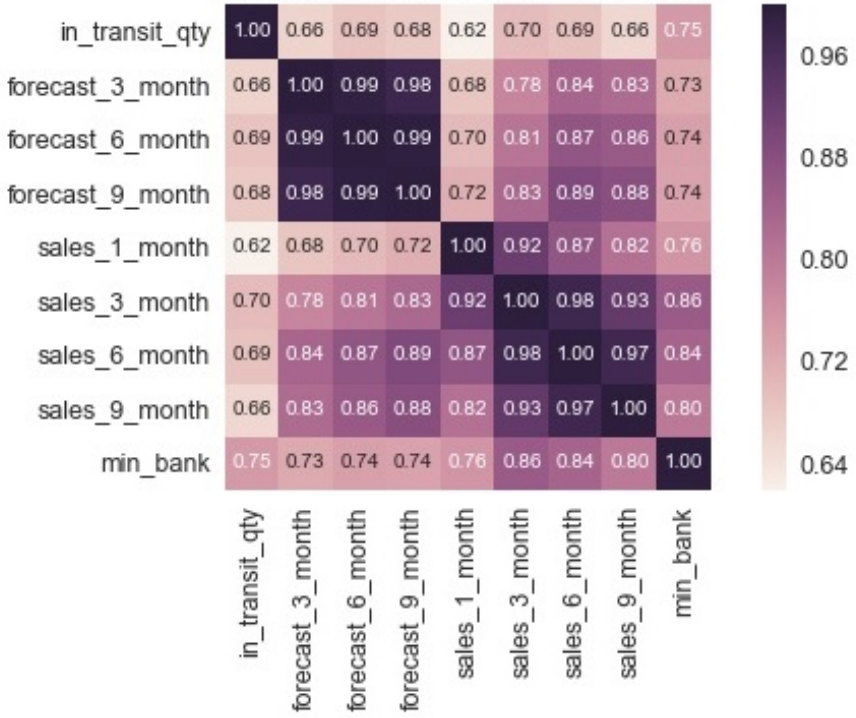


Figure 2.6: Heatmap of Specific Features



2.2 Algorithms and Techniques

The supervised machine learning algorithms that were applied for this binary classification problem are: Decision Tree, Bagging, Random Forest, Adaptive Boosting (AdaBoost), and Gradient Boosting. All of these methods are either ensemble methods or good at dealing with class imbalance, which are suitable for our binary classification problem.

2.2.1 Methods Considered

Decision Tree is a non-parametric method that has no assumptions about the space distribution and the classifier structure. It splits the data into two or more homogeneous sets based on the most significant splitter in features. It is one of the fastest way to identify the most significant features and the relation between the features. It requires less data cleaning compares to other techniques as its not influenced by outliers and missing values to a fair degree. It can also handle both numerical and categorical features like we have in the data. This method is based on 'Gini index' or 'entropy,' which doesn't rely on accuracy and it is perfect for our imbalanced data. The splitting rules can force both classes to be addressed.

Bagging combines the result of multiple classifiers modeled on different sub-samples of the same data set to reduce the variance of our predictions. First it creates multiple datasets from the original data with replacement, then it build multiple classifiers on each data set. Lastly, the predictions of all the classifiers are combined using their mean, median, or mode value depends on the problem.

Random Forest is an ensemble bagging method that is formed by a group of weak models (trees) to form a powerful model. It undertakes dimensional reduction models, treats missing values and outliers. To classify a new object based on attributes, each tree will come up with their classification and the classification having the most votes from the trees will be the classification for the new object. Random Forest has the power of handle large data set with higher dimensionality. On top of that, it outputs Importance of features. It has methods for balancing errors in data sets where classes are imbalanced like ours.

Gradient Boosting and AdaBoost are both boosting methods. Boosting technique combines weak learner to form a strong learner. First the base learning machine learning techniques with a different distribution were applied to generate a weak prediction rule. Boosting focuses more on cases with misclassification from the preceding weak learners. After several iterations, the boosting algorithm combines them into a single strong prediction rule until the limit of base learning algorithm is reached or higher accuracy is achieved.

Gradient boosting is a method that allows a system to learn based on regression analysis and classification. It is a combination of Gradient Descent and Boosting. The result is that the machine might be able to predict outcomes based on history. Adaboost is short for adaptive boosting, which describes an algorithm for using other methods to produce a statistically weighted outcome, which the machine can then use to boost classification of use cases.

2.2.2 XGBoost

The final model selected was XGBoost, short for "Extreme Gradient Boosting." XGBoost is software library that implements the gradient boosting decision tree algorithm. It offers several advanced features, including parallelization, distributed computing, out-of-core computing, and cache optimization [5].

In order to understand XGBoost, we will start with the mathetmatics behind decision trees first. If we have a model with K decision trees, then the model is

$$\sum_{k=1}^{K} f_k$$

where each f_k is the prediction from a decision tree [6]. The prediction at the t^{th} step is defined as

$$\hat{y_i}^{(t)} = \sum_{k=1}^{t} f_k(x_i).$$

To train the model, we need to optimize a loss function that controls the predictive power, while using an regularization term to control the model complexity to prevent overfitting. Therefore, we have the objective of the model as:

$$Objective function = L + Omega$$

where for binary classification,

$$L = -\frac{1}{N}\sum_{i=1}^{N}(y_i log(p_i) + (1 - y_i) log(1 - p_i))$$

and

$$\Omega = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

where T is the number of leaves, and w_j [2] is the score on the j^{th} leaf.

In XGBoost, gradient descent is used to optimize the objective by taking the partial derivative of the objective function with respect to y and yˆat each iteration to minimize the objective function. Rewrite the objective function for a iterative algorithm, we have

$$Obj^{(t)} = \sum_{i=1}^{N} L(y_i, \hat{y_i}^{(t)} + \sum_{i=1}^{t} \Omega(f_i) = \sum_{i=1}^{N} L(y_i, \hat{y_i}^{(t-1)} + f_t(x_i)) + \sum_{i=1}^{t} \Omega(f_i).$$

To optimize the objective function by gradient descent, we will need to calculate the gradient. Or we can improve its performance by considering both the first and second order gradient. We will calculate the second order Taylor approximation since we do not have derivative for every objective function.

$$Obj^{(t)} \simeq \sum_{i=1}^{N} [L(y_i, \hat{y}^{(t-1)} + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \sum_{i=1}^{t} \Omega(f_i)$$

where

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

$$h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}).$$

Removing the constant terms, the objective function at step t becomes

$$Obj^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t).$$

We can see that this function only depends on g_i and h_i, which means users can define the customized objective functions with XGBoost.

2.3 Benchmark

The benchmark predictor is a logistic regression, which is the most natural algorithm to use for binary classifications. This benchmark predictor has a poor performance other than AUC (Table 2.5).

TABLE 2.5: Benchmark Predictor Performance

	Benchmark Predictor
Precision	0.107
Recall	0.001
F1 score	0.003
Kappa	0.002
AUC	0.919

Chapter 3

Algorithms and Techniques

3.1 Data Preprocessing

The following steps were performed during data preprocessing:

- Converted text "Yes" and "No" into 1 and 0 values for the risk factors, namely, deck_risk, oe_constraint, ppap_risk, stop_auto_buy, and rev_stop. The target variable was converted as well.
- Set the -99 values for performance avg metrics (perf_6_month_avg, perf_12_month_avg) to missing, then imputed them with median because they are both skewed.
- Normalized the numerical variables before applying methods like Principal component analysis (PCA), which projects the data onto directions which maximize the variance.

3.2 Implementation

To compare the six supervised machine learning techniques, the necessary packages were first loaded. We then do a training-validation set split with 20% of data for the validation set. Test set was untouched for evaluating model performance in the end. We then created the classifiers, trained them, then obtained predictions on the validation set. Comparing their performance on various metrics including ROC curves, f1 scores, kappa scores, precision, recall, and confusion matrices, we concluded that XGBoost is the best technique to use for our final model. The code snippet is shown in Appendix A.

After we chose the specific machine learning technique, we went through parameter tuning by first fix several parameters and improved them one at a time with grid-search and cross-validation.

During the coding process, the Cohen kappa score function kept returning values outside of the (-1, 1) range. Therefore, kappa score was recalculated based on a function written based on the definition.

3.3 Refinement

The model selected was XGBoost, short for Scalable and flexible gradient boosting. With XGBoost package, we performed parameter tuning with GridSearchCV. We will first fix learning rate and several estimators for tuning tree-based parameters. The initial estimates will be tuned later as well. The initial values are listed below:

- max_depth = 5 : maximum depth of a tree, typically between 3 and 10.
- min_child_weight = 1 : A smaller value is chosen due to our imbalanced outcome. It's also used to control over-fitting.
- gamma = 0 : gamma specifies the minimum loss reduction required to make a split. A smaller value like 0.1-0.2 can also be chosen for starting.
- subsample, colsample_bytree = 0.8 : the fraction of observations to be randomly samples for each tree. Lower values make the algorithm more conservative and prevents overfitting but too small values might lead to under-fitting. It's typically set between 0.5 and 1.
- scale_pos_weight = 1: a value greater than 0 should be used in case of high class imbalance as it helps in faster convergence.

A detailed result from parameter tuning is shown in Table 3.1.

Table 3.1: Characteristics of Training and Test Set

	Definition	Values Tested	Optimized Value
Max_depth	Maximum depth of a tree	(3,5,7,9,11-24)	23
Min_child_weight	Minimum sum of instance weight (hessian) needed in a child node	(1,3)	1
Gamma	Minimum loss reduction required to make a further partition on a leaf node of the tree	(0,0.1,0.2,0.3)	0
Subsample	Subsample ratio of the training instance	(0.6, 0.7, 0.8, 0.9, 1.0)	1.00
Colsample_bytree	Subsample ratio of columns when constructing each tree	(0.6, 0.7, 0.8, 0.9)	0.8
Reg_alpha	L1 regularization term on weights	(0, 0.00001, 0.001, 0.005, 0.01, 0.05, 0.1, 1, 100)	0.01
Learning_rate	Step size shrinkage used in update to prevent overfitting	(0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 1)	0.2

The final model is an XGBoost model with tuned parameters. The improvement from the unoptimized model is significant (Table 3.2). The models were tuned based on cross validation on the training set.

Table 3.2: Parameter Tuning Results

	Unoptimized Predictor	Optimized Predictor
Precision	0.811	0.995
Recall	0.126	0.943
F1 score	0.218	0.968
Kappa	0.994	0.999
AUC	0.973	0.999

Chapter 4

Results

4.1 Model Selection

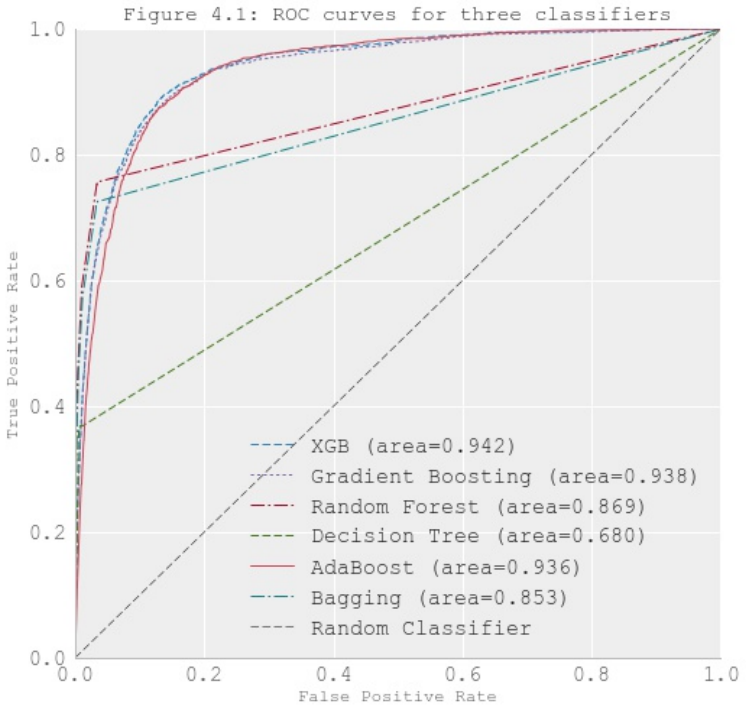
With ROC curves (Fig. 4.1), we can see that boosting techniques (XGB, Gradient Boosting, AdaBoost) perform the best among all. For precision/recall, XGB is the best if we take an average of precision and recall (Table 4.1). For F1 score and Kappa score, decision tree is the best.

At this stage, we will no longer consider AdaBoost as Gradient Boosting is better than it in terms of all four metrics. We will also not consider Decision Tree as it has the worst performance for ROC curve. We will also not consider Gradient Boosting as it is fundamentally the same as XGBoost, but XGBoost is more memory-efficient, can be parallelized, much faster, and is superior than Gradient Boosting in terms of performance[http://xgboost.readthedocs.io/en/latest/model.html]. Therefore, we will compare the following methods: Bagging, Random Forest, XGBoost.

Since false negative is more important than false positive here (it's better to prepare for a backorder that may not happen instead of missing a backorder that actually will happen), we have gathered the number of false negative and XGBoost has the least amount of false negatives. Combine all criteria, we have decided to use XGB for predicting backorder.

Table 4.1: Model Performance of Supervised Machine Learning techniques

	Decision Tree	Bagging	Random Forest	Gradient Boosting	XGB	AdaBoost
Precision	0.327	0.774	0.814	0.579	1.000	0.227
Recall	0.327	0.201	0.182	0.015	0.0004	0.012
F1 score	0.327	0.319	0.298	0.029	0.0009	0.023
Kappa	0.322	0.317	0.296	0.028	0.0009	0.022
AUC	0.680	0.853	0.869	0.938	0.942	0.936



4.2 Model Evaluation and Validation

4.2.1 Benchmark Predictor vs. Optimized Predictor

Comparing the performance of benchmark predictor and the optimized predictor, we see a substantial improvement in terms of all of the metrics (Table 4.2).

TABLE 4.2: Model Performance on Training Set

	Benchmark Predictor	Optimized Predictor
Precision	0.107	0.995
Recall	0.001	0.943
F1 score	0.003	0.968
Kappa	0.002	0.999
AUC	0.919	0.999

4.2.2 Sensitivity Analysis

Sensitivity analysis was performed to examine the robustness of the final model, we have tested its performance by checking its performance under four different random states. The final model is not sensitive to different random states (Table 4.3). We have also tested the model performance on the test data, and we can see the performance is worse than that on the training data as expected. However, most of the performance on the test data is acceptable (Table 4.4).

TABLE 4.3: Sensitivity Analysis - Random States

	Random State 100	Random State 48	Random State 293	Random State 137
Precision	0.995	0.996	0.995	0.996
Recall	0.943	0.943	0.942	0.943
F1 score	0.968	0.968	0.968	0.968
Kappa	0.999	0.999	0.999	0.999
AUC	0.999	0.999	0.999	0.999

TABLE 4.4: Model Performance on Training and Test Set

	Training Set	Test Set
Precision	0.995	0.999
Recall	0.943	0.536
F1 score	0.968	0.698
Kappa	0.999	1.000
AUC	0.999	0.975

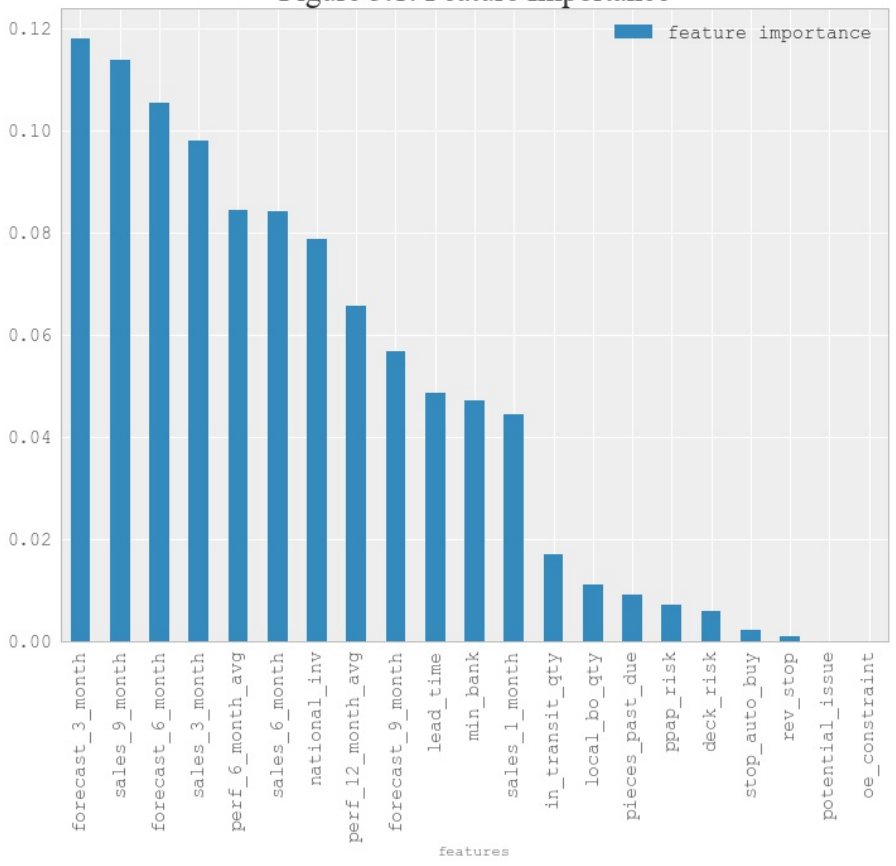
Chapter 5

Conclusion

5.1 Feature Importance

Other than figuring out a good model for predicting backorders, we are also interested in obtaining several good features for such prediction. Figure 5.1 below shows the feature importance of the features used.

Figure 5.1: Feature Importance



The features that are most relevant to predicting backorders are: forecast_3_month, national_inv, sales_9_month, and sales_3_month (Fig. 5.1). Most of the binary risk factors have little or no importance at all, such as oe_constraint. It showed that even though risk factors may signal any abnormality in the process of supply chain, it is more important to rely on the actual sales data and the forecast data. On the other hand, National_inv is a good reference, but it is not as important as the actual sales or the forecast data.

We can see from Figure 5.1 that the importance of forecast data decreases as the number of months increases. It makes sense as forecasting is complex and not always accurate, especially for days further in the future. On the other hand, the actual sales data did not exhibit such trend. Sales_3_month was actually more than important than Sales_6_month, but less important than Sales_9_month. It does not make much sense, but the assumption is that there are more backorders occurring in the last three months, so Sales_9_month is the most important among the three features.

TABLE 5.1: Model Performance with top 10 most important features

	Training Set	Test Set
Precision	0.996	1.000
Recall	0.470	0.497
F1 score	0.638	0.664
Kappa	0.996	1.000
AUC	0.977	0.972

Comparing Table 5.1 with Table 4.4, we can see that even though the performance on training set gets worse, the performance on the test set is actually better. Surprisingly, using only the top 10 most important features improved the performance of the model on the test set.

TABLE 5.2: Model Performance with top 5 most important features

	Training Set	Test Set
Precision	0.995	0.998
Recall	0.136	0.161
F1 score	0.240	0.277
Kappa	0.994	1.000
AUC	0.957	0.954

We have tried to reduce the model even further with only the top 5 most important features, but the performance was much worse than expected. Therefore, it is better to train the model with top 10 most important features, but not the top 5 only.

5.2 Conclusion

With sufficient information, we can predict product backorders to save resources such as time and money. During our data analysis process, we were able to use a XGBoost model to predict backorders with high kappa score, which reflects the classification accuracy normalized by the data imbalance. We have also learned that the most relevant features for predicting backorders are the forecast sales for the next 3 months, current inventory level, and sales quantity for the prior 9 month time period. With the top 10 most important features, we could acquire a more parsimonious model with similar performance.

5.3 Reflection

This project is more difficult than a classic binary classification problem because the target variable is imbalanced, hence using accuracy as one of the metrics is invalid. In addition, training the models take a long time due to the large size of the training data. Another challenge with the project is that there are a huge amount of outliers based on Tukey's fences. Unable to remove any of the outliers, techniques that are insensitive to outliers were necessary.

Data preprocessing did not take too much time, but the selection of a well-performing supervised learning algorithm required more thinking as we were evaluating the techniques based on multiple metrics, and none of them is superior than others in terms of all metrics. XGBoost was nevertheless chosen to be the final model due to its efficiency and parallel computing capability. With some research, XGBoost in fact has an easy implementation and its cross validation function xgb.cv was used to perform parameter tuning. Parameter tuning actually took a lot of time due to the huge sample size of the training set.

5.4 Improvement

During outlier detection, we found a huge amount of outliers based on Tukey's fences. It is a conservative method, and there are in fact more statistics for finding outlying or influential points, such as the PRESS statistic. If an alternative method was applied, we would not be restricted to methods that are insensitive to outliers.

There are also several ways to combat with imbalanced data, such as sampling our training set to add copies of instances from the under-represented class (the products that went on backorder) to perform over-sampling. Another way is to generate synthetic samples using attributes from instances in the under-represented class. The problem would be slightly more complicated, but the evaluation would be simpler as accuracy would be a valid metric to use for model evaluation.

Appendix A

A.1 Model Selection Code Snippet

```
# III - Model Selection.ipynb
from sklearn import cross_validation
from sklearn import metrics
from sklearn.grid_search import GridSearchCV
from xgboost import XGBClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier

features = train_df.drop('went_on_backorder', axis=1,
    inplace=False).columns.values.tolist()

X_train, X_validation, y_train, y_validation =
    cross_validation.train_test_split(train_df[features],train_df['went_on_backorder'],

    test_size=0.2, random_state=0)
#create classifiers
xgbclf = XGBClassifier(random_state=100)
gbclf = GradientBoostingClassifier(random_state=100)
rfclf = RandomForestClassifier(random_state=100)
dtclf = DecisionTreeClassifier(random_state=100)
abclf = AdaBoostClassifier(random_state=100)
bclf = BaggingClassifier(random_state=100)

# train classifiers
xgbclf.fit(X_train, y_train)
gbclf.fit(X_train, y_train)
rfclf.fit(X_train, y_train)
dtclf.fit(X_train, y_train)
abclf.fit(X_train, y_train)
```

```
bclf.fit(X_train, y_train)

# obtain predictions
xgbpreds = xgbclf.predict(X_validation)
gbpreds = gbclf.predict(X_validation)
rfpreds = rfclf.predict(X_validation)
dtpreds = dtclf.predict(X_validation)
abpreds = abclf.predict(X_validation)
bpreds = bclf.predict(X_validation)
```

Bibliography

[1] Backorder. September 2014. URL http://www.businesspundit.com/encyclopedia/general-business/backorder/.

[2] Can you predict product backorders? March 2017. URL https://www.kaggle.com/tiredgeek/predict-bo-trial.

[3] Jason Brownlee. 8 tactics to combat imbalanced classes in your machine learning dataset. August 2015. URL http://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/.

[4] Hoaglin DC. John W. Tukey and data analysis. 18(3):311–318, 2003.

[5] Jason Brownlee. A gentle introduction to xgboost for applied machine learning. August 2016. URL http://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/.

[6] Introduction to boosted trees. 2015. URL http://xgboost.readthedocs.io/en/latest/model.html.