IBM Data Science Capstone Project

Predict Product Backorder in Supply Chain Management

By Kartik Dube

1. BUSINESS PROBLEM

Back orders in supply chain are defined as ordered by customers in the form of products or services which the company isn't able to fulfill due to lack of availability. In short when the demand is very high and the supply is not available. Due to high competition and high requirements back orders might affect the relationship between the company and the customers. Also, A company cannot keep their products in an inventory because this process is expensive. Due to this reason all the companies in the Supply Chain industry want to predict whether their service or product will face backorders or not. The question here is, "Given the sales, transit and forecast data about a product, can we predict whether a product will face backorders?"

2. DATA COLLECTION AND CLEANING

2.1 Collection

The data was collected from Data.World[1] as a CSV file. The dataset contains the historical data for some weeks prior to the week we are trying to predict. In this dataset we'll use the feature "went_on_backorder" for our prediction. The dataset gives a lot of information like inventory storage, sales and forecast, potential issues and many more, we'll use these features to make our predictions. The dataset has 1687861 rows and 23 columns. The features of the dataset are as follows:

- 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 recommended 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 : Product actually went on backorder

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1687861 entries, 0 to 1687860
Data columns (total 23 columns):
 #   Column             Non-Null Count    Dtype
---  ------             --------------    -----
 0   sku                1687861 non-null  object
 1   national_inv       1687860 non-null  float64
 2   lead_time          1586967 non-null  float64
 3   in_transit_qty     1687860 non-null  float64
 4   forecast_3_month   1687860 non-null  float64
 5   forecast_6_month   1687860 non-null  float64
 6   forecast_9_month   1687860 non-null  float64
 7   sales_1_month      1687860 non-null  float64
 8   sales_3_month      1687860 non-null  float64
 9   sales_6_month      1687860 non-null  float64
 10  sales_9_month      1687860 non-null  float64
 11  min_bank           1687860 non-null  float64
 12  potential_issue    1687860 non-null  object
 13  pieces_past_due    1687860 non-null  float64
 14  perf_6_month_avg   1687860 non-null  float64
 15  perf_12_month_avg  1687860 non-null  float64
 16  local_bo_qty       1687860 non-null  float64
 17  deck_risk          1687860 non-null  object
 18  oe_constraint      1687860 non-null  object
 19  ppap_risk          1687860 non-null  object
 20  stop_auto_buy      1687860 non-null  object
 21  rev_stop           1687860 non-null  object
 22  went_on_backorder  1687860 non-null  object
dtypes: float64(15), object(8)
memory usage: 296.2+ MB
```

Fig 1. Data Description

2.2 Cleaning

We first plot the heatmap to get an overview of all the missing values in the dataset.

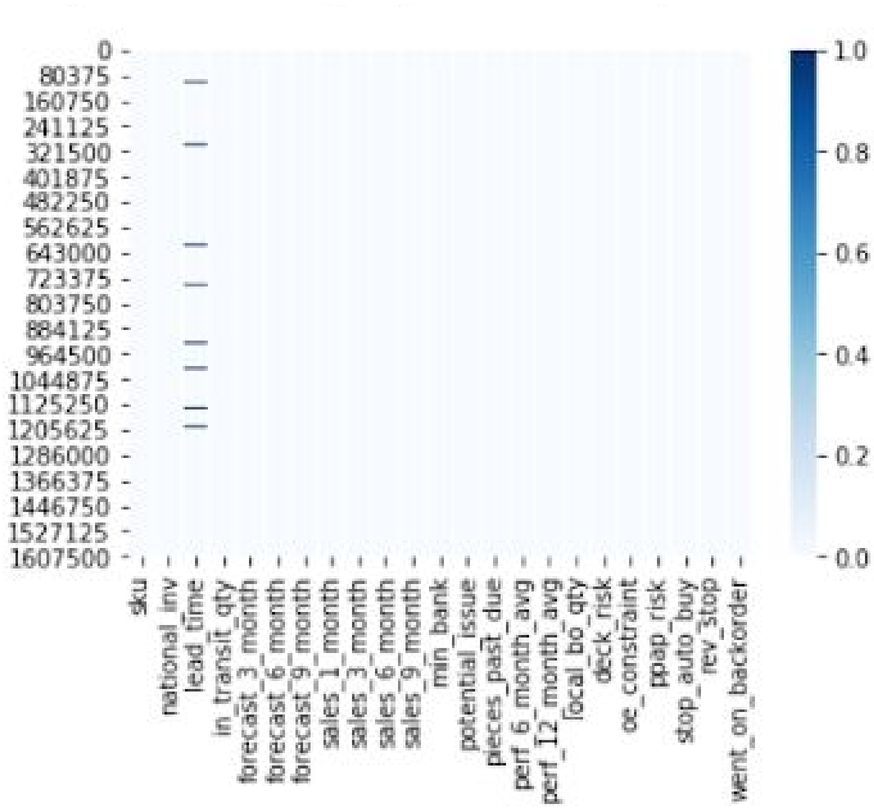

Fig 2. Heatmap of Missing values

We'll first separate the categorical variables and numeric variables to fill the missing values. Then add mean for the numeric data and "No" in the categorical variables since these variables are related to risk and the majority of the products are not related to risks or damages.

Three numeric predictors have missing data:

- *lead_time* (6% missing)
- *perf_6_month_avg* (7.7% missing)
- *perf_12_month_avg* (7.3% missing)

```
['sku',
 'potential_issue',
 'deck_risk',
 'oe_constraint',
 'ppap_risk',
 'stop_auto_buy',
 'rev_stop',
 'went_on_backorder']
```

Fig 3. Categorical features

We will remove the “sku” column since it only denotes product id and can be neglected. The column “went_on_backorder” will be used as target value.

	national_inv	lead_time	in_transit_qty	forecast_3_month	forecast_6_month	forecast_9_month	sales_1_month	sales_3_month	sales_6_month	sales_9_month
0	0.0	7.872267	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	2.0	9.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	2.0	7.872267	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	7.0	8.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	8.0	7.872267	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0

5 rows × 22 columns

Fig 3. Dataframe after cleaning

3. Exploration

Several predictors are skewed or have huge outliers, Part quantities (stock, sales etc.) can be on very different scales. Descriptively, backordered parts are on average associated with lower inventory, lower sales forecasts, worse sales history, more frequent potential risk flags.

Several predictors are highly correlated, especially the sales and forecast variables which are related and have overlap (e.g. 3 month sales history and 6 month sales history)

The dataset is very unbalanced, products which did not go into backorders are considerably larger than the products which did not. We can solve this problem using the ‘Smote’ function in the ‘imblearn’ library in Python.

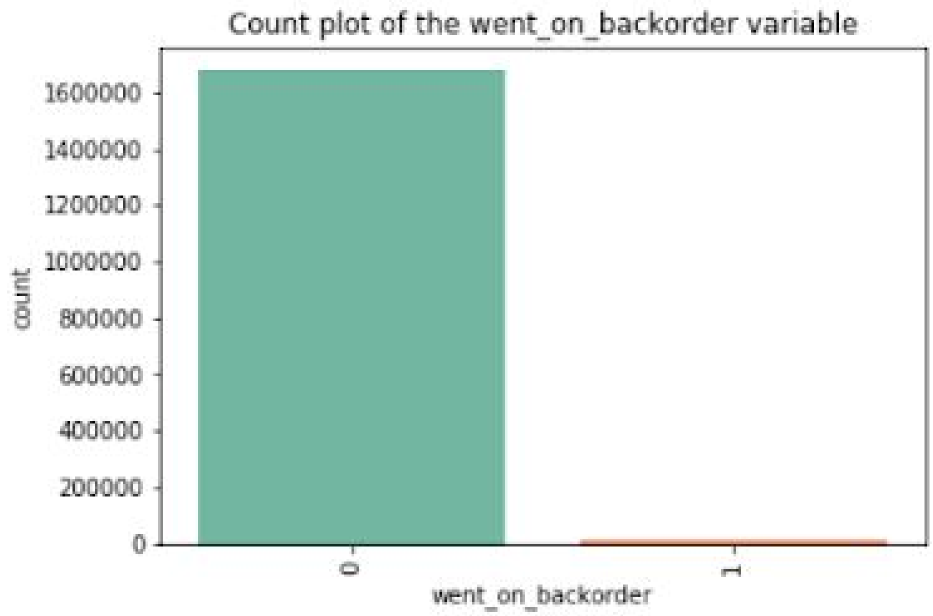

Fig 4. Count x went on backorder original values

```
Original dataset shape Counter({0: 1676568, 1: 11293})
Resampled dataset shape Counter({0: 1123345, 1: 1123345})
```

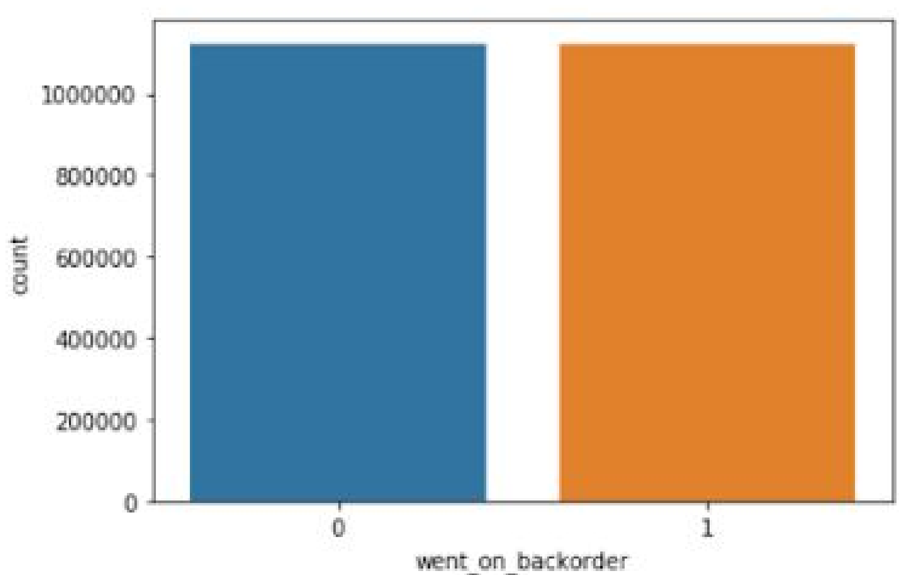

Fig 5. After balancing the training set.

4. Modeling

After balancing the training set, we'll use three classification algorithms for out dataset:

1. Decision Tree
2. Random Forest
3. XGBoost

4.1 Decision Tree

```
Train Result:
===============================================
accuracy score: 0.9996

Classification Report:
        Precision: 0.996489750070205
        Recall Score: 0.9436245180162213
        F1 score: 0.9693368845181999

Confusion Matrix:
 [[1123320      25]
 [    424    7097]]
```

Fig 6. Train Result for Decision Tree

```
Test Result:
===============================================
accuracy score: 0.9917557608237058

Classification Report:
        Precision: 0.3931735278791037
        Recall Score: 0.40005302226935313
        F1 score: 0.3965834428383706

Confusion Matrix:
 [[550894   2329]
 [  2263   1509]]
```

Fig 7. Test result for Decision Tree

4.2 Random Forest

```
Train Result:
================================================
accuracy score: 0.9996

Classification Report:
        Precision: 0.9946830838113894
        Recall Score: 0.9452200505251961
        F1 score: 0.9693209708208345

Confusion Matrix:
 [[1123307      38]
 [    412    7109]]
```

Fig 8. Train result for Random Forest

```
Test Result:
================================================
accuracy score: 0.9946426808140109

Classification Report:
        Precision: 0.8947895791583166
        Recall Score: 0.23674443266171794
        F1 score: 0.3744234800838574

Confusion Matrix:
 [[553118    105]
 [  2879    893]]
```

Fig 9. Test result for Random Forest

4.3 XGBoost

```
Train Result:
================================================
accuracy score: 0.9943

Classification Report:
        Precision: 0.9060205580029369
        Recall Score: 0.1640739263395825
        F1 score: 0.2778340650681076

Confusion Matrix:
 [[1123217     128]
 [   6287    1234]]
```

Fig 10. Train result for XGBoost

```
Test Result:
================================================
accuracy score: 0.993545723031625

Classification Report:
        Precision: 0.6698656429942419
        Recall Score: 0.09252386002120891
        F1 score: 0.16259026321919404

Confusion Matrix:
 [[553051    172]
 [  3423    349]]
```

Fig 11. Test result for XGBoost

5. Evaluation

After getting a very high accuracy value on the test sets, we want to verify which algorithm fits the best and plot the Receiver Operating Characteristics curve.

We will first calculate the AucRoc values for all three algorithms and graphically verify which algorithm works best for the dataset. From the calculation we find that, Decision Tree has the highest value and works outstandingly well.

```
roc_auc_score for Random forest:  0.9668019010737251
roc_auc_score for Decision Tree:  0.7147441424659958
roc_auc_score for XGBoost:  (array([0.00000000e+00, 0.00000000e+00, 1.80758934e-06, ...,
       9.99958425e-01, 9.99963848e-01, 1.00000000e+00]), array([0.00000000e+00, 2.65111347e-04, 2.65111347e-04, ...,
       1.00000000e+00, 1.00000000e+00, 1.00000000e+00]), array([1.9609549e+00, 9.6095490e-01, 9.5079678e-01, ..., 1.5
144546e-07,
       1.4942790e-07, 7.2262750e-08], dtype=float32))
```

Fig 12. AUCROC values for all three algorithms

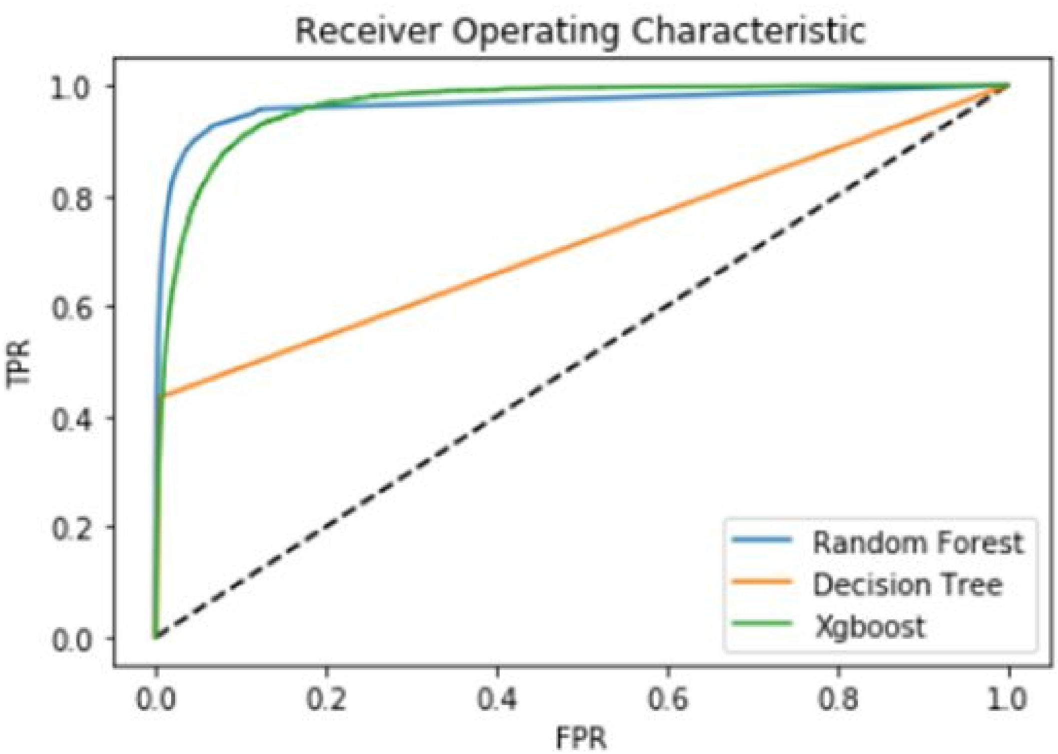

Fig 13. ROC Curve for all three algorithms

6. Result

From the accuracy, precision, recall and f1 scores with ROC curve we conclude that Random Forest Classifier works best on the dataset and performs outstanding on the test set.

7. Discussion

Even though the dataset provides the majority of information about a product, there is still some scope to add more features for better insights. These features can include the products life span, inventory space for the product, average time required in transit, etc.

8. Conclusion

We achieved our goal to predict whether or not a product will be backordered with test accuracy of 0.991 and AUCROC value at 0.966 using Random Forest Classifier. We first cleaned the data, managed null values and categorical variables. Then we balanced the data to avoid biases using smote function. Working with Decision Tree, Random Forest and XGBoost we found out the best model.

9. REFERENCES

1. https://data.world/josephf/can-you-predict-product-back-order