Testing Project

May 5, 2020

Team # 4

Jenna Beutler

Sydney Bookstaver

Nick Cottrell

Cassidy Stearns

Nay Vichitpunt

Contents

Summary

Model metrics / Economic Impact Summary

MODEL	Sensitivity	Specificity	Precision	Accuracy	Cost
Logistic Regression (No PCA)	1.5%	99.2%	1.6%	98.3%	**$141,405**
Logistic Regression (w/PCA)	4%	98.6%	2.4%	97.8%	**$130,935**
Random Forest (No PCA)	41.1%	100%	100%	99.5%	**$2,855**
Random Forest (w/ PCA)	20.57%	100%	100%	99.3%	**$9,455**
Gradient Boost (No PCA)	6.6%	97.7%	2.5%	96.9%	**$357,815**
Gradient Boost (w/ PCA)	10.5%	92.3%	1.2%	91.6%	**$1,149,775**
Ada Boost (No PCA)	2.8%	99.0%	2.4%	98.2%	**$161,775**
Ada Boost (w/ PCA)	1.9%	99.2%	2.1%	98.4%	**$131,735**

Recommendations

Overall, there is only one model that saves Seagate money: a Random Forest without principal component analysis. However, this model is also unable to detect any true or false positives. This model could perhaps be improved upon with continued tuning.

Despite the significant amount of time and effort put into data cleaning in an attempt to balance the data, including using SMOTE and a combination of over and under sampling, the models as a whole are still not able to detect anomalies very well.

Data

Summary

The original dataset includes almost two million rows and 165 columns of Seagate's vendor data, ranging from test dates to several different motor parameters. All variables are masked. The target column specifies whether or not a particular drive passed the first of many tests before going out to sale.

Cleaning approach

The data cleaning portion of the project was the most labor and time intensive. All columns that were 50% or more NA's were dropped from the dataset. Binary variables (such as target, run type, etc) were transformed from string characters to 1s and 0s. We transformed NULL values to NAs, which are easier to deal with in R. We then looked into the distributions of each variable to

determine whether or not they should be considered numeric or categorical, and changed the types accordingly, and extracted months and days as a numeric version of a date (since many models cannot handle datetime objects). We then went on to binning the categorical variables (bins contained 20% of the original number of levels) so we could create dummy columns for each level within the variable to ease modeling efforts.

We created a correlation matrix, and of the columns that were strongly correlated (~0.8 and higher), we dropped the variables with the most amount of levels (if categorical). Of the ~73 variables left, we performed a knn imputation for any columns still missing data with k=9.

In order to further prepare our data for modeling, we decided to do a combination of oversampling and undersampling through SMOTE on our 'target' class so that we could model on a more balanced dataset.

Principal Component Analysis

Description

Principal component analysis was done on the balanced (smote) training data. The purpose of doing a PCA was to reduce the dimensions of the data. We had a total of 70 columns to deal with, and PCA helped us reduce the number of dimensions to make the machine learning computations more efficient and more accurate by splitting the data into 48 principal components. We then used the principal components to run each of the models to hopefully improve the performance of each model.

Parameters

The number of principal components (48) was chosen by setting the parameters to include the minimum number of principal components such that 95% of the variance was retained. We also scaled the data such that the standard deviation was 1 and mean 0. This new set of data consisting of the 48 principal components helped our models run faster and more accurately.

Learning

This data proved to be the most challenging data that I have worked with as a data analyst. I found this challenging yet rewarding. The hardest part was working with the sheer size of the data, not necessarily the technicality of the cleaning and model building. For completing PCA and running everyone's model, I found myself doing a lot of waiting on computing, but it also gave me the ability to really see and organize how we all as a group were going about building models. So much so, that I was able to catch a lot of mistakes with regard to consistency and methodology amongst the five of us. One thing that I am glad we did, was to run each model separately with and without PCA. We saw that PCA was successful in improving the model in two cases (logistic regression and ADA) and unsuccessful when applied to Gradient Boosting and Random forest. Using PCA was a great idea, but the performance seems to be dependent on

the model, so I am glad we ran models with and without applying PCA so that we could compare and contrast.

Model 1: Logistic Regression

Description

We ran two separate logistic regression binary classifier models: one before doing a PCA, and one after to compare results. Logistic Regression classifiers are based on the concept of probability, meaning that predictions that come from these models are a prediction of how likely a data point is to belong to either class of the target variable. Any predictions will be a number between 0 and 1, corresponding to the predicted probability that a point belongs in a given class.

Parameters

Parameters used within the Logistic Regression model in Python are as follows:

- `Random_state = 0`
- `C = 1.0`
- `Class_weight = None`
- `dual=False`
- `Fit_intercept = True`
- `intercept_scaling=1`
- `l1ratio=None`
- `penalty='12'`
- `Solver = 'lbfgs'`
- `tol=0.0001`
- `verbose=0`
- `warm_start=False`

After stepwise variable selection on several models in R, we found the most statistically significant variables to be:

drive_interface	Hkpinp2_parameter1	dpg_2KH113
run_type	Hkpinp3_parameter1	dpg_2KH133
mtrprm2_parameter3	Hkpinp4_parameter1	dpg_2KH203
Mtrprm3_parameter1	Hkpinp4_parameter3	dpg_2KH213
Mtrprm4_parameter1	hkpinp4_parameter5	dpg_2KH233
Mtrprm6_parameter1	hkpinp4_parameter7	dpg_2KK103
Mtprm8_parameter1	Hkpinp4_parameter11	dpg_2L2103
Mtrprm8_parameter2	hkpinp4_parameter13	dpg_2P2113
Mtrprm9_parameter1	Hkpinp4_parameter15	dpg_2RP102
Mtrprm9_parameter2	Hkpinp4_parameter18	dpg_2RV103
Mtrprm9_parameter3	Drive_part_number_1	dpg_2RW103
Mtrprm11_parameter1	Drive_part_number_2	mtrprm_ship_month
Mtrprm12_parameter1	Drive_part_number_3	mtrprm_ramp_date_code_K

Mtrprm_motor_line_num Mtrprm20_parameter1 Mtrprm21_parameter2	Drive_part_number_4 Motor_lot_1 Motor_lot_3 Motor_lot_4 motor_lot_5	

Note that these significant variables are the ones included in the 'Before PCA model' only.

Confusion matrices

Before PCA:

After running several logistic regression models to select variables that were statistically significant, our test set gave us the following confusion matrix. This model struggles to find true positives (i.e. drives that are predicted to fail and fail). Given our full dataset only includes about 0.8% failure, the model predicts about the same ratio in our test set, however has a hard time distinguishing what exactly makes a drive fail or pass (given the higher numbers of false positives and false negatives). We will discuss the economic impact of this model in the next section.

True/Predicted	Positive	Negative	Total
TRUE	50	370,546	370,596
FALSE	3,138	3,227	6,365
Total	3,188	373,773	376,961

After PCA:

True/Predicted	Positive	Negative	Total
TRUE	273	355,667	355,940
FALSE	2,940	2,940	5,880
Total	3,213	358,607	361,820

Economic impact model

The following chart breaks down the costs for each section of the confusion matrix from **before a PCA.** It shows us that the total cost to Seagate if this model was implemented would be 141,405, so we don't recommend deploying this model as it will cost far more than it saves (which is 250 dollars). If Seagate were to retest every predicted failure, it would cost over 100 thousand dollars, with little to no payoff.

Before PCA:

True/Predicted	Positive	Negative	Total
TRUE	-$250	$0	-$250
FALSE	$125,520	$16,135	$141,655
Total	$125,270	$16,135	$141,405

After adjusting for PCA, we see that the model does improve at predicting true positives, and saving a lot more money than the model without PCA, but it also predicts more false negatives and false positives making the model unsuitable for use.

After PCA:

True/Predicted	Positive	Negative	Total
TRUE	-$1,365	$0	-$1,365
FALSE	$117,600	$14,700	$132,300
Total	$116,235	$14,700	$130,935

Data prep

Apart from dropping variables and scaling before modeling, there wasn't much data prep that went into this specific model farther than what we performed on the original dataset. The PCA performed reduced data dimensions from 70 variables to 48 principal components influenced by those variables. We wanted to see if there was a difference in the model performance between a model with principal components, and a model without.

Learning

The data presented to us is far more complicated than a regression should be able to effectively handle. We believe a model that handles imbalanced data or does more sophisticated operations on the data would be more likely to garner better results for the company. Another interesting thing to note is that the PCA in fact made our model better, and would've ended up costing the company around ten thousand less dollars if implemented over the non-PCA model, which is insignificant given the scale of the project/costs.

Model 2: Random Forest

Description

The random forest is a model made up of many decision trees. It creates many classification trees and a bootstrap sample technique is used to train each tree from the set of training data. This method only searches for a random subset of variables in order to obtain a split at each node. In this case, it is used for classification in which the input vector is fed to each tree in the Random

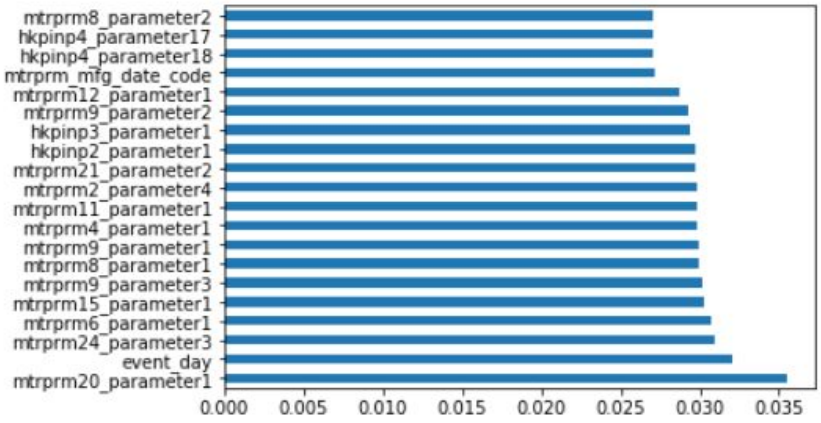
Forest and each tree votes for a class. Finally, the Random Forest chooses the class with the highest number of votes. To improve the performance, one of the good methods is Hyperparameter tuning which is a parameter of the model that is set prior to the start of the learning process.

Parameters

The target variable is 'Target' column (Binary classifier). The parameters used within the Random Forest model by using RandomizedSearchCV function in Python with tuning hyperparameters included :

- n_estimators : 600
- min_samples_split: 2
- min_samples_leaf : 2
- max_features : auto
- max_depth : 70
- bootstrap : False

Moreover, to run the model with PCA, we used x2_test from PCA and y_test with cross validation = 5. After doing hyperparameters, it can make our classification model more accurate, which lead to more accurate predictions overall. The variable importance by random forest are shown as the below. As we can see, those below 3 variables have the most scores which are mtrpm20_parameter1, event_day and mtrpm24_parameter3.



Confusion matrix

A random forest with PCA and without PCA gave us exactly the same total and true negative value. Also, true negative and positive predictive are at the same rate at 100%. However, you can see that the true positive from the model before PCA gave us the better true positive rate at 41%. In contrast, the result after PCA gave us only 20.5% since the false negative number is slightly more than the one without PCA. Therefore, a random forest before PCA would be the best choice as shown below.

Before PCA:

True/Predicted	Positive	Negative	Total
TRUE	1,321	368,439	369,760
FALSE	0	1,892	1,892
Total	1,321	370,331	371,652

After PCA:

True/Predicted	Positive	Negative	Total
TRUE	661	368,439	369,100
FALSE	0	2,552	2,552
Total	661	370,991	371,652

Economic impact model

The total cost that we derived from a random forest after doing PCA is $9,455 with no true negative and false positive. On the other hand, the total cost without PCA is $2,855 which is much less than the model with PCA by around $6,000. If we compare with other models, Random forest performed the best and saved the most money for Seagate. Therefore, we do recommend Seagate to deploy random forest without PCA.

Before PCA:

True/Predicted	Positive	Negative	Total
TRUE	-$6,605	$0	-$6,605
FALSE	$0	$9,460	$9,460
Total	-$6,605	$9,460	$2,855

After PCA:

I was surprised with the PCA results for the random forest, I knew that random forest would be the best model, and I expected the PCA to improve the results for the model when that is not the case here with a difference of about $6,000. I am unsure as to why this may be the case.

True/Predicted	Positive	Negative	Total
TRUE	-$3,305	$0	-$3,305
FALSE	$0	$12,760	$12,760
Total	-$3,305	$12,760	$9,455

Data prep

Beyond the initial data preparation, only little was required to prepare the data for a Random Forest classifier. We dropped 'event_date', 'mtrprm_ship_date','Unnamed: 0' since we already extracted them as day and month in mtrprm_ship_day, mtrprm_ship_month, event_day column. Moreover, “dpg_2T7223” is dropped due to null values. Moreover, we used the PCA to reduce the feature space by eliminating features.

Learning

From this project, I have learned many new things because this dataset has tons of columns and rows with real business problems. First, I can learn how to apply what we learnt in the class to real-world situations in both Python which I am not familiar with at first. In terms of coding skills, I can learn a lot of new syntax and coding techniques such as sckitlearning. Moreover, one of the challenging things is to handle overfitting and imbalanced data. Therefore, I need to explore and learn how to deal with it which makes me need to integrate and adapt all the knowledge that I have learnt from the business analytics course to the project.

Model 3: AdaBoost

Description

Adaptive Boosting classification models are known to be particularly useful for naturally unbalanced datasets, this encouraged us to use this as one of our models. An adaptive boosting model is a variation of a decision tree. It uses a boosting algorithm, meaning it uses a

classification problem to convert a group of weak classifiers into a strong classifier. Adaboost has a unique technique for creating weak learners. One unique quality to AdaBoost is that it begins by randomly generating a set of weak learners before beginning the learning process. This allows the models to develop a weight for the learners that can then be used to develop a strong learning. Another distinctive aspect of AdaBoost is that at each iteration, the sample distribution changes by modifying the weights. The weight of each learner is learned by whether or not it predicts a sample correctly. For example, if a learner predicts incorrectly, the weight of the learning is reduced. This process is continued and applied to each learner. The higher a learner performs the more it contributes to the strong learner.

Parameters

After Tuning the AdaBoostClassifier using GridSearchCV() the suggested parameters for this model included:

- N_estimators : 800
- Learning_rate : .1
- Random_state : 0

Confusion matrix

Before PCA:

The confusion matrix before including PCA is shown below. The model seems to poorly predict true positives here in comparison to other models who have shown to be more successful in doing so. One other issue here is that there are more false positives than we had hoped for, this is a problem as they tend to cost the company more money. This is consistent with the confusion matrix that includes the PCA. However, including the PCA lowered the number of false positives which in turn lowered the overall cost. Unfortunately, neither of these would financially benefit the company.

True/Predicted	Positive	Negative	Total
TRUE	89	364,774	364,863
FALSE	3,665	3,124	6,789
Total	**3,754**	**367,898**	**371,652**

After PCA:

True/Predicted	Positive	Negative	Total
TRUE	61	365532	365,593
FALSE	2,907	3,152	6,059
Total	**2,968**	**368,684**	**371,652**

Economic impact model

As mentioned above, the model does not prove to financially benefit the company. This biggest issue here is the significant number of false positives which is very costly. The total cost would be $161,775, therefore is not recommended for implementation.

Before PCA:

True/Predicted	Positive	Negative	Total
TRUE	-$445	$0	-$445
FALSE	$146,600	$15,620	$162,220
Total	**$146,155**	**$15,620**	**$161,775**

After PCA:

We can see here that the PCA did improve the overall performance of the ADA model, and reduced the overall cost of deployment. However, if this model was deployed, Seagate would be spending a significant amount on false positives. ADA with PCA detects less failures overall. There are less true and false positives predicted than the model without PCA.

True/Predicted	Positive	Negative	Total
TRUE	-$305	$0	-$305
FALSE	$116,280	$15,760	$132,040
Total	**$115,975**	**$15,760**	**$131,735**

Data prep

Apart from the initial data preparation little to no data preparation was needed to run this model. The columns mtrprm_ship_day, mtrprm_ship_month, event_day column were removed considering they were not likely to help to model performance. Additionally, GridSearchCV was implemented to tune the model and suggest best performance metrics given the dataset.

Learning

The initial dataset provided by Seagate required a significant amount of data cleaning which allowed us to gain experience with ways to clean unbalanced datasets, for example using smote in R. Additionally, the data columns provided very little information about what the data actually was, this required us to do further data inspection in order to figure out how which columns to keep and how to format them. Because of this our decisions relied highly on analytics. When it came time to run the models, my research led me to an Adaptive Boost model, something I was unfamiliar with. I found the model technique to be very interesting and seemed like it would apply well to our dataset. Based on the model performance it wasn't the best, but certaining more tuning could improve this model.

Model 4: Gradient Boosting

Description

For the fifth and final model, we selected a gradient boosting model (GBM), sometimes otherwise referred to gradient boosting machines. A GBM is yet another form of decision tree modelling and can be applied to both regression and classification problems. Gradient boosting begins the same as Random Forest and AdaBoosting: with an ensemble of weak learners. Weak learners are defined as classifiers that are not highly correlated with the true classification - meaning that they are only slightly better than random guessing. The goal of GBM is to take those weak learners and transform them into strong learners. As the name suggests, this is accomplished through a method known as boosting. Boosting iteratively builds upon each previous classification and adjusts the weights of observations accordingly; essentially, it is a model that learns from its past mistakes. Each time the model iterates, it calculates the loss using a loss function and attempts to minimize it in the next iteration. This is the important distinction between a GBM and other forms of modeling, such as AdaBoosting. The loss can then be used to calculate the gradient, the slope (aka severity) of the error. Although GBMs are incredibly powerful models, they are also prone to overfitting.

As with the previous three models, the GBM was trained and tested on a dataset pre-PCA and post-PCA to assess the effectiveness of PCA on the modeling process. The model created with post-PCA data performed significantly worse. Although this was not the initial hope, it is also not entirely unexpected. PCA reduces the dimensionality of data and its performance is usually entirely dependent on the data itself. Without having much information on what each of the variables in the dataset truly were, there was no way to know if PCA was appropriate for the data or not. After observing its effect on the GB models, it is conclusive that PCA is not appropriate for this portion of the analysis. The impact of PCA on the gradient boosted models can be seen below under the Confusion Matrices and Economic Impact subheadings.

Parameters

To create a GBM, we simply call on the `GradientBoostingClassifier()` function in the scikit learn library in Python. The function takes many parameters, but the ones used for this model include:

- `n_estimators`
- `learning_rate`
- `Max_features`
- `max_depth`
- `random_state`

The `n_estimators` parameter specifies the number of boosting stages to take place. Typically a larger number helps to decrease the risk of overfitting, but it increases the time it takes for the model to run.

Overall, two models were created: one with static parameters outside of `learning_rate` (model 1) and one that used hyperparameter tuning (model 2). The inspiration for model 1 came from preliminary research. Various examples observed online used the same values for the parameters listed above and tried several different values for `learning_rate` to see how the model improved. For model 1, values 0.05, 0.075, 0.1, 0.25, 0.5, 0.75, and 1 were all used for `learning_rate` with `n_estimators=20`, `max_features=2`, and `max_depth=2` held constant. `random_state` was set to 0 for each model. This parameter simply makes all results reproducible. The resulting area under the curve for model 1 is 0.67 with an accuracy of 97.7%.

Model 2 tried various different parameters for each of the parameters above with the exception of `random_state`. The best parameters are `n_estimator=120`, `max_features=20`, `max_depth=4`, and `learning_rate=1`. While these parameters helped the area under the curve to increase to 0.73, the accuracy dropped to 96.9%. The loss in accuracy ended up increasing the number of false positives and, in turn, caused the cost to quadruple.

Models 1 and 2 were repeated with data that had PCA applied to it. Both models performed worse than the pre-PCA data, particularly with flagging false positives, and so they are not recommended for deployment. Because model 1 (pre-PCA) performed the best in terms of total cost, it is used as the baseline model for the GB analysis and is shown below alongside the post-PCA results.

Confusion matrices

Before PCA:

True/Predicted	Positive	Negative	Total
TRUE	218	365,100	365,318
FALSE	8,584	3,109	11,693
Total	**8,802**	**368,209**	**377,011**

After PCA:

True/Predicted	Positive	Negative	Total
TRUE	337	340,012	340,349
FALSE	28,427	2,876	31,303
Total	**28,764**	**342,888**	**371,652**

Economic impact model

Before PCA:

True/Predicted	Positive	Negative	Total
TRUE	-$1,090	$0	-$1,090
FALSE	$343,360	$15,545	$358,905
Total	**$342,270**	**$15,545**	**$357,815**

After PCA:

We can see that adding PCA to the model increases the overall likelihood of predicting a failure, and because predicting a false positive is so expensive, the overall cost of deployment is the highest of all of our models. This model would be good if false positives had no cost associated with it. However, in this case predicting false positives is more expensive than the money gained from predicting true positives. Therefore this model should not be considered for deployment, and the GB without PCA will provide better results (in this case) than GB with PCA.

True/Predicted	Positive	Negative	Total
TRUE	-$1,685	$0	-$1,685
FALSE	$1,137,080	$14,380	$1,151,460
Total	**$1,135,395**	**$14,380**	**$1,149,775**

Data prep

Outside of scaling and removing the columns already specified in the sections above, no specific data preparation was performed for this model. To scale the data, the `MinMaxScaler()` function in Python was used to scale the training, validation and test data.

Learning

Overall, this project gave me an opportunity to apply general data science concepts to a real world problem. The majority of my experiences in this program were formulaic; I knew which model and data cleaning methods to apply because it was directly told to us by a professor. We rarely had the opportunity to take everything we had learned and try to figure out how to apply them. Additionally, I learned about several new modeling techniques, including AdaBoosting and gradient boosting, and data preparation methods such as SMOTE. Lastly, I also learned the general process of tuning hyperparameters for a predictive model in Python and was able to implement it for the gradient boosted model. The overall performance of this model could most likely be improved with continued tuning.