**Machine Learning Models & Analysis**

**Data Pre-Processing**

As was the case with our exploratory data analysis, we need to take some further steps to get our data in the right format before we begin testing different machine learning models. Pre-processing steps vary by the algorithm being used, so I will mention all pre-processing steps that were necessary before beginning my modeling.

**Encode Labels:** A few of the machine learning algorithms I used in my analysis require converting categorical variables to integers used to represent the different categories. To do this, I used the LabelEncoder method within the sklearn preprocessing library. LabelEncoder also has methods that make it easy to convert our integer labels back into their labeled categories, so that we can interpret the results of our models easily as needed.

**Split Data:** To ensure that my analysis adheres to the best practices of machine learning, I perform a random split of my data into a training set (where the models will be fit and hyperparameters tuned) and a test set, which will be used for validating the robustness and forecasting accuracy of my models. In this study, I used a 70%/30% split, where 30% represents the amount of data left over to test the results of various models. In addition, to ensure that my results were not fit to one random split, I used 5-fold cross-validation with 2 repeats and made sure to stratify my data, so that all folds contained a proportion of positive and negative classes that was representative of the proportion for the whole dataset; in this study, that means that each fold contained the percentage of customers who churned and who did not churn at a rate close to that of our entire dataset.

**Scaling:** Since some machine learning models are sensitive to the values of our independent variables, they require scaling the data so that all variables can be compared on a uniform scale. For my study I went with a scale of mean 0 and standard deviation 1, which can be achieved by applying the StandardScaler method (also found in the sklearn preprocessing library).

**Modeling Pipeline and Hyperparameter Tuning**

After pre-processing the data, there are some additional steps to be taken before we can begin analyzing output from our models appropriately. First, we need to set up the appropriate pipeline to feed into our model. Common steps handled to feed into the pipeline include imputing our data for missing values (as a reminder, missing values were already handled during EDA though), scaling values as required by model type, and of course, specifying the model to be used. As was mentioned in the data pre-processing section above, I used stratified, repeated 5-fold cross-validation and this is also instrumental in making sure that all hyperparameters for my models are not unduly influenced by one split of the data. Hyperparameter tuning will be discussed further on a model-specific level, but having hyperparameters tuned over our cross-validated data gives us confidence that our results can be compared across models when we are deciding upon a best model(s) upon completion of this analysis.

**Models**

**k-Nearest Neighbors (kNN)**

To set up our kNN model, we hypertuned parameters for the number of neighbors, the weights to be given to each neighbor (chosen either uniformly or by distance), and the definition of distance to be used. The optimal parameters for the model used 19 neighbors, a uniform weight for neighbors and a Manhattan definition of distance.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Class | Precision | Recall | F1-score | ROC AUC | PR AUC | Accuracy |
| kNN | 0 | 0.81 | 0.89 | 0.85 | 0.79 | 0.57 | 0.77 |
|  | 1 | 0.59 | 0.42 | 0.49 |  |  |  |

As we can see from the charts and table above, the model has 77% accuracy, but most of the predictive power comes from the negative class (class 0, or those customers who did not churn) and struggles more with the positive class of customers who did churn. The Receiver Operating Curve (ROC) shows us how when you change the threshold for classification (default is 0.5), the true positive rate (TPR) and false positive rate (FPR) also change. The values for the threshold vary between [0,1], with 0 being all classifications being labeled in the negative class and 1 representing all classifications being labeled in the positive class (no randomness involved at either extreme). Therefore, a perfect classification model would be two perpendicular lines that intersect in the upper left corner and have an ROC area under the curve (AUC) of 1.0, as this would represent a TPR of 100% and a FPR of 0%. The blue dotted line has an AUC of 0.5 and signifies what the ROC curve would like if all our classifications were left to chance, so anything above that line means our model is performing better than classifying by chance. The Precision Recall (PR) curve is another graphic for evaluating model performance and similarly looks at the relationship between precision and recall as you vary the classification threshold from [0,1]. The perfect model would again be two perpendicular lines, but this time intersecting in the upper right hand corner, which would represent 100% for both precision and recall. Again, we can look at the PR AUC as a top line metric for comparing models.

Since our data is imbalanced (~73% of our customers will not churn), it is important to also use the PR AUC score. Since the formulas for precision and recall (and therefore the PR curve) avoid the use of the value for true negatives, it accounts better for changes in the distributions of the positive and negative class. Whereas the ROC curve only looks at true and false positive rates, the precision component of the PR curve tells us how right we actually are the model predicts the positive class.

**Logistic Regression**

Since logistic regression is designed to handle binary classification problems, this was a logical model to fit the data to. First, all data was normalized, again using the StandardScaler method in scikit-learn. In order to optimize the logistic regression algorithm, values for the hyperparameter C were tested using GridSearchCV and the optimal value was found to be 10.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Class | Precision | Recall | F1-score | ROC AUC | PR AUC | Accuracy |
| Logistic Regression | 0 | 0.84 | 0.89 | 0.87 | 0.83 | 0.61 | 0.80 |
|  | 1 | 0.64 | 0.55 | 0.59 |  |  |  |

After tuning the model and scaling our data, the model was fit and predictions on customer churn were generated. Compared to the kNN model, there were modest improvements across the board as the ROC AUC score improved to 0.83 from 0.79, the PR AUC score improved to 0.61 from 0.57 and overall accuracy improved from 0.77 to 0.80. The corresponding ROC and PR curves were also shown above for graphical visualization of these results.

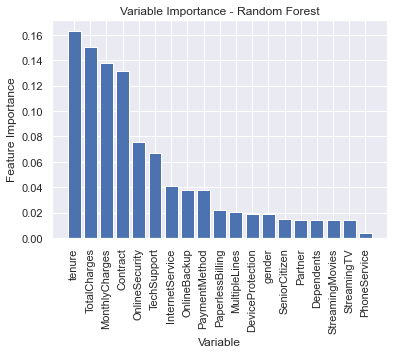
**Random Forest**

Random forests are one of the more robust and popular machine learning algorithms in use today and was the next choice in our model selection and evaluation process. Random forests build upon the notion of a decision tree by introducing randomness via selection of a random subset of the feature (independent) variables. While decision trees on their own are not great predictors, a collection of trees significantly increases predictive power and lowers the test error of prediction.

In contrast to the other two models we have looked at thus far, the random forest has several parameters that can (and should be) tested. The parameters tuned for this classification problem were the number of trees in the forest, the maximum number of features considered before splitting a node, the maximum number of levels in each decision tree (or depth), the minimum number of data points placed in a node before it is split, the minimum number of data points allowed in each leaf node and whether or not the method for sampling data points would be with or without replacement (where with replacement is referred to as bootstrap sampling). Random forests are equipped to deal with raw, or unscaled data, so this pre-processing step was skipped for this method. Since the total combinations of all parameters to be tested for the random forest model was quite large, RandomSearchCV was used initially; this method in scikit-learn looks at a random subset of the hyperparameter combination set and therefore does not test all combinations of hyperparameters. This allows for quicker computation time to get an idea of where the optimal values for our parameters lie. After this was performed, I then used GridSearchCV to test a neighborhood of values around those found from the RandomSearchCV method. The final hypertuned parameters for the model were n\_estimators = 200, min\_samples\_split = 4, min\_samples\_leaf = 4, max\_depth = 36, max\_features = ‘sqrt’, and bootstrap=True.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Class | Precision | Recall | F1-score | ROC AUC | PR AUC | Accuracy |
| Random Forest | 0 | 0.83 | 0.91 | 0.87 | 0.83 | 0.63 | 0.79 |
|  | 1 | 0.65 | 0.49 | 0.56 |  |  |  |

While the random forest model did not show improvements across the board, there were some noteworthy takeaways. With the logistic regression model being our best model so far, it serves as a useful model for comparison. The ROC AUC score of 0.83 was equivalent to that of the logistic regression model, but the PR AUC score of 0.63 was an improvement over our prior best score of 0.61. There was a small reduction in accuracy from 0.80 to 0.79, but given that we are working with an imbalanced dataset, it seems that the increase in PR AUC score makes the random forest a contender for our best model.



One other takeaway from our random forest model is the ease with which feature importance can be performed. A model’s predictive power can be thought of as how well of a job it does in explaining the variance of our dependent variable, which again for this analysis is customer churn. From the figure above, we can see that our 4 most important variables are tenure, TotalCharges, MonthlyCharges and Contract. Given the high correlation between TotalCharges and MonthlyCharges, we could simply drop one of these variables if we wanted to reduce the dimensionality of future models further. Contract type shows up as our most important categorical variable, which supports our findings from our EDA, where it seemed as if contract type had a large influence on customer churn. Surprisingly, payment method comes in lower on the importance totem pole, but would serve as a useful cutoff for where we might stop including features if we were to look at a reduced dimensionality model.

From the chart above, future analysis could be performed using a subset of features including: tenure, total or monthly charges, contract type, whether a customer had online security, whether a customer used tech support, if a client opted to purchase internet service and what their payment method was for their monthly bill, totaling 7 features, from our initial set of 19 features.

**Support Vector Machine (SVM)**

Support vector machines are one of the most widely used algorithms for classification problems, so this was one of the machine learning models used in my analysis. Since SVMs are not scale invariant, it is imperative that we scale the data in our pre-processing steps. In addition, we tune the hyperparameters C and gamma, as these are two of the more important hyperparameters for establishing a well-tuned SVM model. The tuning process in scikit-learn has the model use a value of 10 for C and 0.01 for gamma.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Class | Precision | Recall | F1-score | ROC AUC | PR AUC | Accuracy |
| SVM | 0 | 0.83 | 0.91 | 0.86 | 0.80 | 0.60 | 0.79 |
|  | 1 | 0.64 | 0.47 | 0.54 |  |  |  |

With our optimal hyperparameters in place, we again fit the model and analyze our set of predictions. Unfortunately, the SVM model scores lower or equal to our best two models (the logistic regression and random forest models). Overall prediction accuracy of 79% was on par with the results of our best models, but the ROC AUC score of 0.80 and the PR AUC score of 0.60 show that this model did struggle with prediction in a relative sense.

**AdaBoost**

Next we take a look at another powerful ensemble method called AdaBoost, which is one of the most popular and successful algorithms for boosting. Like random forests, AdaBoost also gets its roots from decision trees and looks to build upon the relatively weak predictive power of decision trees to produce a more powerful prediction model. Boosting methods usually are composed of weaker decision trees (fewer nodes and leaves), but are important because they learn from each slice of the data made to use predictions in the prior iteration. In particular, boosting methods seek to improve performance in subsequent iterations by honing in on areas of weak performance in previous iterations. There is no scaling of data needed for the AdaBoost model, so we focus on tuning two hyperparameters of interest for this model, the number of estimators or how many trees will be developed, and the learning rate, which indicates how each tree contributes to the overall results. Again using GridSearchCV, we obtain optimal parameters of 300 for the number of estimators and 0.1 for the learning rate.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Class | Precision | Recall | F1-score | ROC AUC | PR AUC | Accuracy |
| AdaBoost | 0 | 0.84 | 0.89 | 0.86 | 0.84 | 0.64 | 0.79 |
|  | 1 | 0.64 | 0.51 | 0.57 |  |  |  |

Upon fitting the model to our training data and viewing the performance on our test class, we see that the AdaBoost model has produced our best model yet. With overall accuracy of 0.79 (just 0.01 off our best model) and the best model scores in both ROC AUC and PR AUC at 0.84 and 0.64, respectively, it’s safe to say that we’ve found the best model thus far in our analysis.

**Ensemble Model**

After tuning and analyzing 5 models, we decide to see if we can improve our results by using another ensemble method, a voting classifier within scikit-learn, where we combine our previous models into one model. With the VotingClassifier class in scikit-learn, we specify the voting parameter to be “soft” when we are inputting well-tuned models into the voting classifier, which the above 5 models certainly qualify as. Since all models being inputted have already been appropriately pre-processed, these steps are not needed for this algorithm.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Class | Precision | Recall | F1-score | ROC AUC | PR AUC | Accuracy |
| Voting Classifier | 0 | 0.82 | 0.91 | 0.87 | 0.83 | 0.63 | 0.79 |
|  | 1 | 0.66 | 0.46 | 0.54 |  |  |  |

The results from this model are a little underwhelming, as it performs close to as well as our AdaBoost model, but does not show any improvements in top-line metrics and performs a little bit poorer in our positive class (customers who churn); this is made a little bit more evident when you take a look at the F1-scores for class 1 and see that AdaBoost has an F1-score of 0.57 and the Voting Classifier has an F1-score of only 0.54.

**Model Comparisons**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Class | Precision | Recall | F1-score | ROC AUC | PR AUC | Accuracy |
| kNN | 0 | 0.81 | 0.89 | 0.85 | 0.79 | 0.57 | 0.77 |
|  | 1 | 0.59 | 0.42 | 0.49 |  |  |  |
| Logistic Regression | 0 | 0.84 | 0.89 | 0.87 | 0.83 | 0.61 | 0.80 |
|  | 1 | 0.64 | 0.55 | 0.59 |  |  |  |
| Random Forest | 0 | 0.83 | 0.91 | 0.87 | 0.83 | 0.63 | 0.79 |
|  | 1 | 0.65 | 0.49 | 0.56 |  |  |  |
| SVM | 0 | 0.83 | 0.91 | 0.86 | 0.80 | 0.60 | 0.79 |
|  | 1 | 0.64 | 0.47 | 0.54 |  |  |  |
| AdaBoost | 0 | 0.84 | 0.89 | 0.86 | 0.84 | 0.64 | 0.79 |
|  | 1 | 0.64 | 0.51 | 0.57 |  |  |  |
| Voting Classifier | 0 | 0.82 | 0.91 | 0.87 | 0.83 | 0.63 | 0.79 |
|  | 1 | 0.66 | 0.46 | 0.54 |  |  |  |

All model results are displayed in a single table upon completion of our analysis. While the AdaBoost model seems to be the best model with all factors considered, the logistic regression model could also be used for predictions in practice. With a bit more accuracy (.01) and a higher F1-score, it is still a very useful model.