Bryan Pritsker

Data Mining

12/5/23

**Final Report**

*Introduction*

The focus of this final project is to analyze a company’s customer data and predict customer churn behavior. Customer churn is defined as the act of a customer unsubscribing from a company’s services or discontinuing business with the firm. The project’s dataset, downloaded from the “Kaggle” website, presents data on one particular telecom business. The dataset provides information about their customers like the customer’s tenure with the company (the number of months the consumer has used the firm), the customer’s monthly payments and total payments, and the type of internet and phone service the customer uses. Additionally, the dataset also supplies more distinctive types of information about their customers, such as their gender, the number of dependents they have, and if they are a senior citizen. The final column of the dataset is titled “Churn” and is composed of binary results that either say “No” or “Yes”, which indicate whether the customer churned from the business. Throughout the project, all the features in the dataset are examined and scrutinized to detect patterns on customer churn behavior.

*Data Preprocessing Part 1: Missing Values and Imbalanced Data*

Before building robust machine learning models that can predict customer churn behavior, it is first necessary to understand the data that is being provided. At the first glance while reviewing data types and missing columns in the DataFrame, it seemed like there were no missing values in the dataset. However, after doing a little more research, the column “TotalCharges” had 11 missing values that were unaccounted for in the data. These missing values were of the type NMAR (“not missing at random”), because the column “Tenure” was correspondingly equal to zero for each one of these missing instances. It can be concluded that these customers had just signed a contract with this telecom company and did not yet make any payments to the business. So, we decided to not delete these missing instances from the dataset, but instead replace them with a zero in the “TotalCharges” column. Note that these 11 instances will provide a valuable insight into the churn behavior of the firm’s customers, since for all 11 of these instances the output column “Churn” had an output of “No”.

Furthermore, while examining the label column and the other input features, exploratory data analysis uncovered a significant imbalance in the dataset. Specifically, one pie chart which compared the proportion of customers who churned to those who did not churn, revealed that 73% of all customers in the dataset were categorized as non-churners. Such a heavy imbalance, if not properly addressed, could produce misleading results in the machine learning stage of the project. It is crucial to point out that imbalanced datasets are biased towards the majority class, since the majority class has more instances to train on. Consequently, prediction models that ignore imbalances in the data produce poor performances on the minority class and unreliable results for the total accuracy scores. To produce the most favorable and reliable results, certain categorical models will need to be tuned with imbalanced data techniques before they can make their final predictions.

*Data Preprocessing Part 2: Data Visualization*

Exploratory data analysis also highlighted useful correlations between features in the dataset. For example, one bar chart showed that the average monthly charges of customers who churned was $13 higher than the customers who did not churn. Moreover, a separate bar chart emphasized that the average monthly charges for new customers (customers who just started their contract) was $27 cheaper than the monthly charges for existing customers (customers who have stayed with the company for more than one year). These results not only suggest that certain input features are correlated with the output column, but they also indicate that specific input features have strong correlations with one another. To further support this claim, note in one bar graph how the correlation between customer contract (input feature) and churn rate (the label column) is not particularly significant. However, if we view the bar chart which combines the features “tenure” and “contract type”, then we see that customers with month-to-month contracts rarely last more than 18 months with the telecom business. This profound result suggests that we should not ignore the interdependence between features in the dataset. For this reason, when we will perform feature selection to optimize certain categorical models, we will implement the wrapper method over the filter method. The wrapper method will be more effective in this case because it will consider the interactions between input features and assess their performance with respect to a machine learning algorithm.

*Data Preprocessing Part 3: Transforming Categorical Features*

One last preprocessing step before we can implement machine learning algorithms is to transform the categorical variables in the dataset into numerical values. Considering that our dataset has over 15 categorical columns, we decided to first apply label encoding to all the non-numerical features. We utilized label encoding because it is a swift and effective tool for encoding categorical labels. It converts categorical labels into a value between 0 and the “number of classes - 1”, where n is the number of distinct labels. If a label repeats, the label encoder assigns the same value as what was assigned earlier. After implementing the label encoder however, our dataset encountered one more key problem. Some of the categorical features that got converted now had a “meaningful order” between their classes, when originally, these columns had nominal variables (values with no meaningful order). To combat this issue, we employed another method known as “one-hot encoding”. Unlike label encoding, one-hot encoding retains nominal variables by representing each unique class with its own binary vector. Simply, one-hot encoding takes a feature which has categorical data (which has been label encoded) and splits this column into multiple columns. Each column then gets composed of 0’s and 1’s where only the position corresponding to the category is marked with a 1, and all other positions are set to 0. Since this procedure is far slower than label encoding and greatly increases the dimensionality of our data, we decided to only exercise one-hot encoding on four key columns: “OnlineSecurity”, “PaymentMethod”, “Contract”, and “InternetService”. All of these features are composed of nominal classes, and it was far more useful to preserve this relationship for the machine learning algorithms that we next implemented.

*K-Nearest Neighbors Algorithm: Standardizing Numeric Attributes*

Accordingly, the first machine learning algorithm that we exercised was the K-Nearest Neighbors algorithm. KNN is a simple to implement and easily interpretable algorithm. When making predictions, KNN identifies the k-nearest neighbors (for some positive integer k) to a given test data point and then it assigns the test point a class based on the majority of its neighbors. To find these nearest neighbors, the model uses a distance metric, such as Euclidean distance, to measure the similarity between data points. When the features have different scales though, this can unevenly impact the distance calculations. The features with larger scales dominate the distance metric, and this could lead to inaccurate and biased predictions. This was especially true in our dataset, because the column “TotalCharges” had significantly larger values than the column “tenure”. Thus, to enhance the effectiveness of the KNN algorithm, we rescaled the original, numerical features in the dataset by using “Standard Scaler” normalization. Standard Scaler is a type of normalization which subtracts the mean of the column from each data point, and then divides the result by the standard deviation of the column. We applied Standard Scaler on the columns: “tenure”, “MonthlyCharges”, and “TotalCharges” in order to ensure that these three numerical features held equal weightage in the model.

*K-Nearest Neighbors Algorithm: Goals for the Model*

Next, we implemented a regular KNN (with k = 11) without using any data mining techniques, to observe the key factors that were necessary to improve the model’s performance. With a train test split of 0.3, the regular KNN’s total accuracy was 76.15%. As stated earlier though, this result was very misleading. When viewing the confusion matrix results of the regular KNN model, this algorithm made 274 false positive predictions out of 566 total churners in the test data. That earns a “specificity” score of 51.6% (True Negative / `True Negative + False Positive`). This means that 48.4% of the time, if the model viewed a churner, it incorrectly predicted that this customer did not churn. Clearly, the regular KNN performed terribly on the minority class, and its results were highly biased towards the majority (non-churners) class. Therefore, the goal for our optimized model was to reduce the number of false positives in the KNN algorithm while sustaining strong results in the “recall” (True Positive / `True Positive + False Negative`) score.

*K-Nearest Neighbors Algorithm: Feature Selection*

Consequently, we outlined three specific tasks to optimize the performance of the KNN model: feature selection, imbalanced data techniques, and cross validation. Feature selection was necessary for the KNN model because this algorithm’s major weakness is “The Curse of Dimensionality”. As the number of features in a KNN model increases, the number of dimensions in the distance calculation also goes up. Hence, this causes the distance calculations between data points to become less meaningful and causes the model to run much slower (because greater dimensions requires more computational energy). Therefore, to best implement feature selection for the KNN model, we utilized a form of the wrapper method known as “sequential backward elimination”. During the process of sequential backward elimination, we fitted all the features to a KNN model (with k = 11), calculated the performance of these features based on the “Roc AUC” score (with tol = 0.02), and then greedily removed the least relevant features (one at a time) that did not improve the performance of the algorithm. This process stopped once removing any extra feature became detrimental to the model’s performance (would cause the Roc AUC score to drop by 0.02). We utilized Roc AUC scoring over regular model accuracy because the ROC curve is more informative than accuracy for imbalanced data. The ROC AUC score is designed to evaluate how efficiently the model is performing. The higher the AUC score, the better the model is at distinguishing between the positive (majority) and negative (minority) classes. In our case, the wrapper method dropped five columns in the selected features for the algorithm, indicating that the KNN model could be improved by cutting out these features.

*K-Nearest Neighbors Algorithm: Imbalanced Data Techniques*

Secondly, after selecting the best features to use in the KNN model, our next task was to correct the imbalance in the training data. We utilized an imbalance data technique known as “balanced bagging with oversampling”. This technique allowed us to combine the principles of ensemble learning and oversampling in one method, all while utilizing K-Nearest Neighbors as the base estimator algorithm. During the balanced bagging procedure, we set the number of bags in the classifier equal to the default of 10 and the “sampling strategy = auto”. Then in each bag, by adding the hyperparameter “oversampler” to this classifier, the balanced bagging classifier drew samples from the majority class that were slightly greater than the number of minority samples in the training data. It then randomly oversampled some of the minority instances (made copies) until the number of minority instances equaled the number of majority instances in each bag. Finally, the classifier decided the class label for each instance by taking a majority vote among the 10 KNN classifiers. We decided to combine the techniques of oversampling and balanced bagging because we wanted the KNN model to reduce the number of false positive mistakes it was making, but still be able to train on a substantial amount of majority class instances within each respective bag. This ensured that the KNN model would elevate its performance on the minority class and still maintain a strong “recall” score.

*K-Nearest Neighbors Algorithm: Cross Validation and Implementation of Model*

Finally, to increase the test accuracy of the KNN algorithm, the last technique we applied was cross validation to choose the best k-value for the model. We performed cross validation by using the “Stratified K-Fold” algorithm provided by Sci-Kit Learn packages. Stratified K-Fold is a type of cross validation that preserves the ratio of the majority class to the minority class when splitting the training data into folds. We specifically applied the Stratified K-Fold algorithm within the balanced bagging KNN classifier. Note that for each fold in the cross-validation process (we set the number of folds = 5), the training set fold was used to train a balanced bagging ensemble of KNN models. This process was repeated for values of k that ranged from 7 to 20, and each result for k was stored in its own respective vector. We did not consider any k’s less than 7 because such values of k that were too small would have a higher chance of overfitting to the training data. After the cross validation process finished, we determined the best k value by taking the average of the results in each vector, and comparing which k value gave the highest average score on the five folds. In our cross validation technique, the best performing k value was k = 8. Once we acquired this optimal k value, we retrained the balanced bagging classifier with k = 8, and finally evaluated the performance of this model on the test data. As expected, the total accuracy of this KNN model dropped slightly to 72.23%. However, the model made significant improvements on the number of false positive mistakes that it made. As shown in the confusion matrix, after applying the imbalanced data techniques the KNN model only made 100 false positive predictions out of 479 total churners in the test data. That amounts to a 79% specificity score, which is an almost 30% higher result compared with how the regular KNN model performed. The imbalanced data techniques clearly improved the KNN model’s predictions on the minority class. Moreover, the model successfully maintained a 90% precision score and a 70% recall score, implying that the model successfully maintained its performance on the majority class, while greatly improving its accuracy in identifying minority class instances.

*Decision Tree Algorithm*

The next algorithm that we applied to the churn dataset was the Decision Tree. A Decision Tree uses information gain and entropy to recursively split the dataset into subsets based on the values of input features. The Decision Tree is constructed in a top-down manner, where the most informative features (the ones with the highest information gain or lowest entropy) are used to split the data early in the tree. One of the biggest flaws of this algorithm is that it is prone to overfitting, especially if the dataset contains many irrelevant features. Thus, to combat this issue we implemented a strategy known as “reduced error pruning”. Firstly, we split the dataset by making the test size equal to 35%. Then, we cut this test size into two equal parts, where 17.5% of the data was used only for X-test and y-test, and the other half was used to create a validation set. From here, we created a Decision Tree Classifier with balanced class weights, and let the tree grow out completely. Once the full Decision Tree was formed, this is where we applied the pruning stage. We applied a “for loop” iterating through all the possible max depths for this tree, and correspondingly tested this model on the validation set for each max depth. The max depth that resulted in the best validation accuracy was noted and then used to prune all the nodes in the tree that were lower than this max depth. Accordingly, the reduced error pruning method greatly simplified the tree, and it created a classifier that better generalized to the test data. After testing the pruned Decision Tree on the test data, we observed that this algorithm had a training accuracy of 77.2% and a test accuracy of 75.8%. These results showed that the model was not overfitting to the training data and its overall accuracy turned out to be better than the previously utilized KNN model. Furthermore, in the classification report the Decision Tree showed a specificity score of 78.5% and a recall score of 74.7%! Hence, this algorithm performed compatible on both the majority class and the minority class instances, despite the imbalance in the number of churners in the dataset.

*Random Forest Algorithm*

The last model that we utilized was the Random Forest algorithm. A Random Forest is an ensemble learning method that builds multiple Decision Trees during the training process, and then merges their predictions to improve overall accuracy and stability. Each tree is trained on a random subset of the data and makes independent predictions. The final prediction is determined by a majority vote among the individual trees, and this should result in a more reliable and generalized model. Consequently, the Random Forest algorithm does not technically require any pruning, since it already reduces variance by building multiple trees using random subsets of data and random subsets of attributes. However, for our Random Forest model, when we did not set a max depth for the algorithm, the model ended up having a 99.8% training accuracy and a 78.8% test accuracy. This clearly indicated that the Random Forest was overfitting to the training data. Also, in the classification report, the algorithm showed a strong bias towards the majority class; its specificity score was a horrible 45.3%. Thus, this simple Random Forest was not reliable for our imbalanced dataset.

Accordingly, we ran the Random Forest algorithm again, but this time calculated the training accuracy for each possible max depth in the individual Decision Trees. After printing out the training accuracy for each max depth, we picked the max depth that had the closest training accuracy to 82.5% (we wanted a training accuracy that fell in the range between 80% and 85%). We figured that this specific max depth would allow the model to better generalize to new test instances, not overfit to the training data, but still have a strong overall accuracy on the training points. The max depth that fit this criteria was 9. When utilized on the testing data, the tuned Random Forest algorithm with a max depth of 9 got an overall testing accuracy of 77.6%; this was just 1% lower than the simple Random Forest model from earlier. Furthermore, the correlation matrix confirmed that this Random Forest no longer remained biased towards the majority class. The specificity score went up from the mediocre 45.3% in the earlier model to 77.2% in this tuned Random Forest model! Also, the algorithm preserved a high recall score, getting a 77.8% accuracy in predictions on the number of non-churners (TP / `TP + FN`).

*Conclusion: Best Model*

Hence, when considering the nature of the imbalanced dataset, the Random Forest algorithm had the highest overall performance compared with the other two models. It had the most compatible specificity and recall scores, and both metrics got an accuracy that was over 75%. For these reasons, it is safe to conclude that the Random Forest model performed the most reliably on our imbalanced churn dataset. It could ultimately help the telecom business predict and prevent certain customers from churning in the future. This model could be crucial for the company, especially considering how in the telecom industry, retaining existing customers is more cost-effective than acquiring new ones.

Works Cited

II, Rupak (Bob) Roy -. “Handle Imbalanced Dataset with Ease.” *Medium*, Medium, 20 Mar. 2023, bobrupakroy.medium.com/handle-imbalanced-dataset-with-ease-8cd577192b40.

Shaikh, Rahil. “Choosing the Right Encoding Method-Label VS OneHot Encoder.” *Medium*, Towards Data Science, 9 Nov. 2018, towardsdatascience.com/choosing-the-right-encoding-method-label-vs-onehot-encoder-a4434493149b.