Telco Churn Prediction

Nick Brewers†
Machine Learning
University of St. Thomas
St. Paul Minnesota USA
Brew9896@stthomas.edu

Emily Bosacker
Machine Learning
University of St. Thomas
St. Paul Minnesota USA
ebosacker@stthomas.edu

Matt Ackerman
Machine Learning
University of St. Thomas
St. Paul Minnesota USA
matt.ackerman@stthomas.edu

Roshine Jeyachander
Machine Learning
University of St. Thomas
St. Paul Minnesota USA
jeya2694@stthomas.edu

Introduction

After examining a number of datasets and comparing the pros and cons of these different data sets we decided that Telco customer churn would allow for us to create a fascinating prediction model. With over 7,000 instances this data set would provide plenty of data for us to analyze. There are nineteen different independent variables, 3 numeric and 16 categorical, for us to examine with one very important dependent variable. The independent variables range from seemingly important variables such as longevity of the client and contract type to seemingly unimportant variables such as payment method or whether or not they had a partner. This data set also included one very important dependent variable, as the last column, and that is customer churn. In this context churn is best described as clients who do not renew their contract with the telco company. It was immediately evident to us that it would be interesting to see based on these variables how accurately we could predict whether a customer was going to stay with their telecom provider or if they were about to "churn" and switch to a new provider.

The dataset was originally given to IBM from an anonymous telecommunications company. IBM used this data as a way to help train individuals on their AI platform, IBM Watson. As a result of this they made the data set publicly available for anyone to use for training purposes in AI. There are certain questions that need to be looked at when examining any data set: How was it collected, if there are any data quality issues, are there biases in the data or how it was collected and how might this impact the conclusions of our analysis.

For the first question of, how was it collected, we do not have a ton of detail about the collection process that was used by the telecommunications company. What we do know is that they were collecting the data on their own customers based on information they had direct access to in their internal database. This tells us that this should be a very accurate data set.

The second question is if there are any data quality issues, while we do not perceive anything to be a major data quality issue there is one small issue with the data set. The "total charges" column of the data has five instances where there is no input. We presume that there is a logical reason for this such as the customer cancelled after a free trial period, however we do not know. Due to this, we had to accomodate for that when analyzing the data.

The third question gets a bit more complicated, are there any biases in how the data was collected or with the data itself. We unfortunately probably do not know for sure if there is any bias but with how straight forward the questions are we are presumming there are no biases with the data and we moved forward as if there are not.

We also have an imbalanced data because luckily for this telecom company the majority of their customers renew and do not churn. We overcame this by doing both undersampling and oversampling. By strategically adjusting the number of instances that contain customers who have churned and those who have not churned we were able to balance the data and create more accurate models.

With the large amount of categorical variables that this dataset contains we had to do a lot of one hot encoding to make all of the categorical data work for our models.

With this in mind we are interested in looking at what variables are driving the churn rate. This telco company has a retention rate of roughly 73.5% and therefore a churn rate of 26.5%. This rate seems abnormally high and investigating this further to understand what variables drive

this rate would be critical to understanding how the company can improve their customer churn.

Other Work on This Data Set

We were originally introduced to this dataset via a data science project that we found on Kaggle[1]. As mentioned this was published by IBM for use in data science so it was not surprising to find other people had used this data set for their own data science projects. It is a rather dry and straight forward look at the data. There was not much text around why they chose this data set or what they were looking to accomplish by analyzing the data set. Most of this had to be inferred by the code. Luckily they did do a strong job of laying out their code and it was easy to understand their workflow and how they arrived at their conclusions.

To start their project they walked through how they cleansed the data and prepared it for their needs and purposes. They converted everything that had a yes or no answer to 1's and 0's. They checked for null values and replaced all null values with 0's. Why they did not replace it with a mean or another method is not clear in their documentation. This group decided it would be best to balance the data set with an even number of churned and unchurned clients. They ended up with 1,869 clients in both categories.

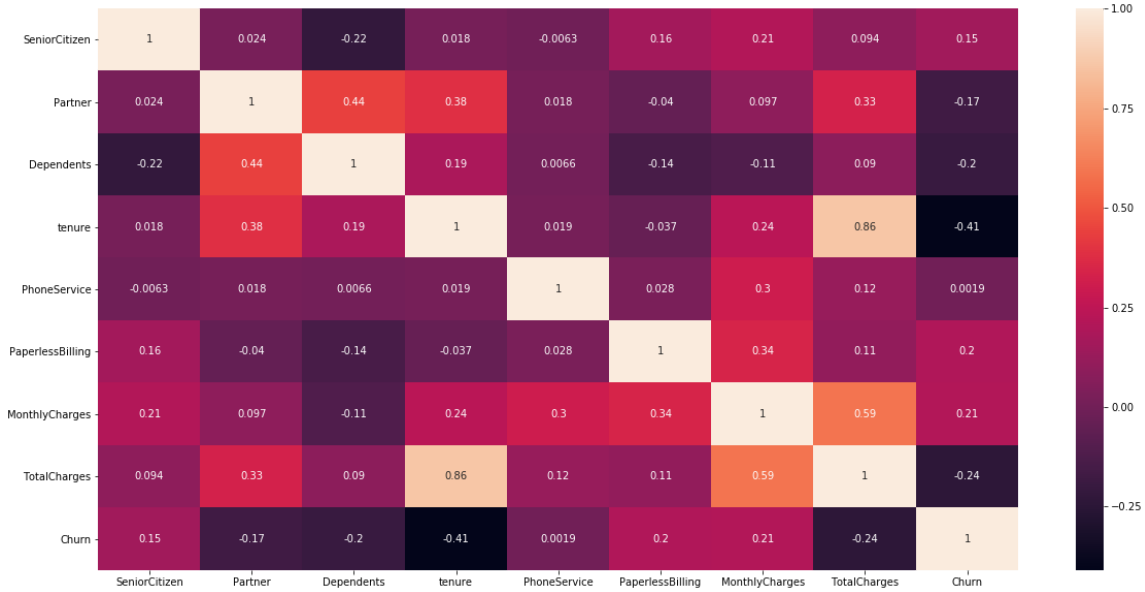
At this point in their project they started to do a bit of data analysis of their current data set that is now reduced to 3,738, down from over 7,000 instances. One of the first things they did was look at the correlation of the independent variables to the dependent variable churn. They produced a great visual using matplotlib (see below). This visual elegantly shows how much impact the independent variables have on the Churn variable and it is color coded to represent the amount of impact it has. After this they finished up their data preparation by one hot encoding the categorical variables.

Now that they have the data prepared for their needs they started applying machine learning techniques. They ran their train test split at 30%. They decided to apply the use of Kneighbors and Decision Tree to the data. The frustrating part is they do not document why they went with their test split, although 30% is a fairly normal split, or why they decided to use Kneighbors and Decision Tree. For kneighbors they decided to go with the hyper perimeter 5 and for the decision tree they also set the hyper perimeter of max depth at 5. Once it was complete they analyzed their data by looking at it in a confusion matrix. The confusion matrix model accuracy is 74% or 836 out of 1121. They made their model fairly accurate, although they probably could have gotten it more accurate. This group did not choose to elaborate as to why they decided not to try other methods of machine learning or why they went with the hyper perimeters they did. Had they tweaked those hyper perimeters or tried other methods they probably could have gotten a more accurate model.

After all of this was complete they did go one step further and apply all of this to their data set. They took a look at each individual in the complete data set of 7,042 instances and determined what the likelihood was of that person churning from the customer base. It would be interesting to dissect those numbers because they did have a fair amount of individuals that have a zero percent chance of churn which seems highly unlikely.

Data Preparation

The correct



preparation of our data was crucial to ensure our models and analysis would be based off data that abide by our expectations. We handled the initial loading of our dataset by utilizing python libraries io and pandas. The first (io) was used to convert our data, uploaded to google colab in a file object, to an in-memory text stream object. After decoding the stream to utf-8, we used the later library (pandas) to replace all empty string values in our data to null. This was critical so as to avoid misrepresenting null data as empty strings, which could potentially be included and operated upon as viable values.

We then changed the data type of the column containing the now null values, TotalCharges, (no other columns had missing values) to float64 in order to overwrite their missing values with imputed ones. There is a helpful impute class provided by sklearn we used to speed up this process. We passed it arguments of the missing value to identify as well as the strategy for replacing it, which we specify to use columns MonthlyCharges and TotalCharges to impute the missing values in TotalCharges.

The next step in data preparation was one hot encoding. Having 15 categorical columns is not ideal on a dataset of 21 columns, including the dependent. However, for the most part they only had a couple categories, with the most having 3. We then deleted the extra dummy column for each that was now one hot encoded. The final fix for categorical columns came in the form of encoding our class label, which is represented by the Churn column.

Machine Learning and Statistical Techniques

1. Sampling

With preparation out of the way, we can focus on the analysis and model creation, beginning with sampling techniques. Having a relatively large distribution inequality between class values in our dataset meant we needed to apply undersampling and oversampling to our original X and y data splits in order to adjust the class distribution in a way to combat this inequality to allow for more reliable predictions down the road. One cannot be certain that fixing this inequality via under or over sampling will increase accuracy of our models, but it was something that we considered, tested, and reviewed, in order to increase our confidence in the models.

Our regularly sampled class label counts were No: 5174 and Yes: 1869. Undersampling, then, logically resulted in equal distributions with both No and Yes at 1869 counts. Oversampling, with 5174 counts for both.

To test these sampling techniques and their viability on this specific dataset, we split the dataset thrice (once for each sampling technique: regular, undersampled, and oversampled) using a standard, double split, holdout technique. With each split we kept a 70/30 train-test split with a sklearn random state argument of 0, to ensure the splits contain the same data for all 3 sampling techniques. We did not conduct K-fold cross validation at this point, which is retrospectively discussed given new insight in section 4.

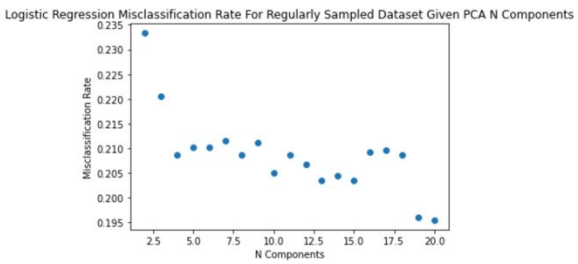
2. Dimension reduction

Next, we performed PCA dimension reduction to reduce the number of input variables in an attempt to make the predictive modeling task less challenging to our models. To determine the best number of PCA n components, we ran PCA with each n component ranging from 2 to 20 and ran a Logistic Regression model on each. Then, based off the misclassification rates, we determined the best n components. The loop between n components and misclassification rate capture was performed 3 times, once for each sampling technique.

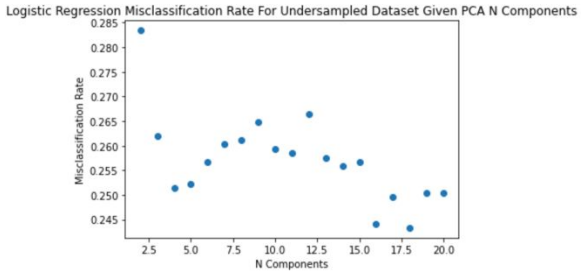
Although, on a more complex dataset than ours, only running one model (in our case Logistic Regression) to test each n component might not be sufficient to determine the optimum number of components, for our relatively simplistic dataset we deemed it as such for multiple reasons including lack of worries regarding computation, complexity and the overfitting that comes with datasets that have closely related values of quantity of dimensions in comparison to record counts.

We could have also tested n components post-model selection. That is, we could have determined the model architecture and hyperparameter (if one exists) as well as the sampling technique combination that has the best classification rate on our data, and then conducted dimension reduction techniques to determine which n components or technique was optimum. However, that then would potentially miss model and sampling technique combinations that might have operated better to our supposed top model given different initial dimensions. These complications were considered, discussed, and eventually led to our decision to handle dimension reduction on each sampling technique first.

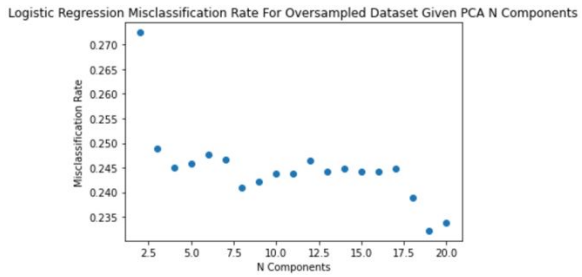
PCA dimension reduction led to similar results, regardless of sampling techniques. We were able to not only determine the best n components, but also the sampling technique that produced it. Our scatterplot containing the best overall misclassification rate (.20) given a specific n component (20) for any of the sampling techniques is shown below. As stated in the scatterplot's title, these results are from the regularly sampled dataset. We utilitzed matplotlipb's pyplot class to create the plots.



Finding that our best misclassification rate came from our regularly sampled dataset and with the max n components, we were a little surprised. However, the trend apparent in the above scatterplot is followed relatively well when we compare PCA reduction using our other 2 sampling techniques, again using Logistic Regression to calculate misclassification rates. Each result (for the most part) with higher n components outperforming lower n components.



Our undersampled dataset is the primary trend deviator of the 3 sampling techniques in relation to their n component performance against the logistic regression model. We found its best misclassification rate was .24 and occurred at an n components of 18. Below, we have our PCA component analysis for our final sampling technique, oversampling.

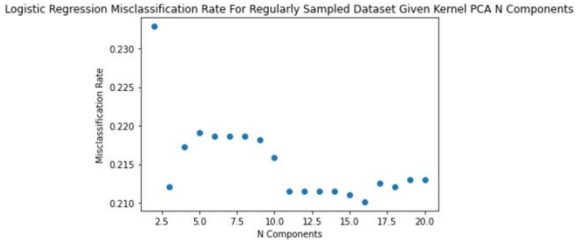


This trend is very close to the result from our regularly sampled dataset and has a best misclassification rate of .23 occurring at an n components of 19. Given the close results from all 3 sampling techniques, we are confident that, if choosing PCA dimension reduction on this dataset, a high amount of n components is ideal.

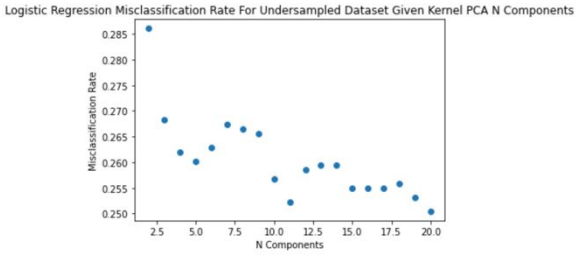
We then move on to our second dimension reduction technique, Kernel PCA. This reduction technique was handled in a similar way, with n components looping between 2 and 20 on each of the 3 data samples, with logistic regression being the model we test the components against to determine misclassification rates. We quickly faced the challenge of Kernel PCA being more computationally taxing than its not kernel counterpart.

We worked around this problem by shifting to a Google Colab notebook with a GPU runtime. Although fixing the issue at hand, this shift to Colab caused other problems, such as importing the data file and running the python package imbalanced-learn. These problems were resolved by changing our import to match Colab's expected process, importing sys and using it to perform system executables to install the imbalanced-learn package and upgrade the scikit-learn package using pip

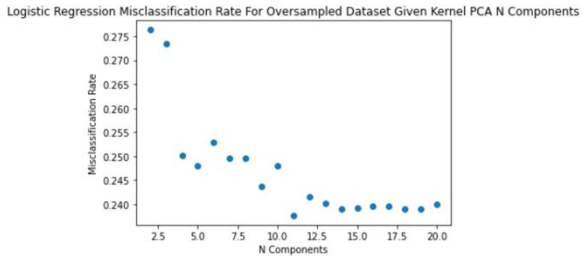
The misclassification results from performing Kernel PCA varied between each sampling dataset more than they did when performing regular PCA. What didn't change, however, is that the regularly sampled dataset resulted in our best misclassification rate. Note, when running Kernel PCA, we used a Gaussian radius basis function (RBF) by supplying it as the hyperparameter. Below, you can see the misclassification rates given n components for the regularly sampled dataset, as determined by running logistic regression using each n components, just like before with standard PCA. The optimum n component is 16 which resulted in a misclassification rate of .21.



Much like with regular PCA, Kernel PCA's worst optimum n component of the 3 sampling techniques resided with undersampling. Seen below, the best n component was 20 with a misclassification rate .25.



The oversampled dataset hovered right around a misclassification rate of 24 between n components 11-20, with the optimum component number being at 11 with a rate of .24, as visualized below.



With PCA dimension reduction, (Both Kernel and non) we've found our best misclassification rates came from the regularly sampled dataset and our full n component quantity of 20.

Our final dimension reduction technique, LDA, was less demanding as the number of components will always be n - 1, where n represents the number of class variable instances, which is 2. To stay consistent with our analysis of the other dimension reduction techniques, we test LDA against a Logistic Regression model, using all 3 sampling techniques. Due to there only being 3 misclassification rates we're going to compare, there was no need to produce a scatterplot to make visualization easier. The results were as follows:

LDA - Regularly sampled: .20
LDA - Undersampled: .25
LDA - Oversampled: .23

These classification rates will also be included as we discuss our Logistic Regression model results toward at the beginning of section 3. Note: the rates moving forward will be by classification accuracy (1 - misclassification rate). LDA holds true to our findings with PCA reduction techniques. That is, the regularly sampled dataset, post-dimension reduction, has better optimum classification performance than its 2 counterparts. However, given that these initial dimension reduction tests were reliant on one specific model, (Logistic Regression) we will be continuing to use all 3 samples throughout the creation of our classification models.

3. Classification Models

Due to having 3 datasets due to different sampling techniques, we will have to iterate through each dataset and compare the results in order to avoid missing practicable models. We will also be creating a model instance for each dimension reduction technique; None, PCA, LDA, and Kernel PCA. There will be 12 model instances per type.

To promote consistency as well as increase our confidence in our models with unseen data, we use K-Fold cross validation on each model. We utilize sklearn's model_selection class cross_val_score to implement the validation. We provide it our model, the training data, and the number of splits and iterations we want it to conduct, which we selected 10 for. The resulting accuracies for each K-Fold validation are captured and the mean is taken from them. That value was then compared against the value of the other models to gain insights.

The performance statistics presented for each model moving forward will be the confusion matrix and classification rates (for holdout Train-Test split as well as K-fold split) for the *optimum* combination of sampling method and dimension reduction technique for that specific model.

The optimum combination was determined by the lowest K-fold cross validation mean. However, we understand that, depending on the context the model is expected to perform in, that is not always the determining factor for which model is ideal. There are settings in which confusion matrix results might be equally as crucial, if not more. For example, had this dataset been relating to the healthcare sector and the dependent variable been whether a patient had cancer or not, the false negative rate might be far more indicative of the model's quality due to the consequence of that specific incorrect result. Given the nature of our dataset, focusing on the mean K-fold cross validation classification rate was deemed adequate. Although there are instance where the 2 split holdout splitting method outperforms K-fold, we still abide by K-fold as it is likely a more accurate representation of how the model will perform on unseen data. Due to close results, we will be listing classification rates to thousandths decimal place.

<u>Logistic Regression</u>
Sampling: Regular
Dimension Reduction: None
Classification Rate: .8041

1395	416
254	299

Confusion Matrix:

<u>Kernel SVM</u>
Sampling: Regular
Dimension Reduction: LDA
Classification Rate: .8012

1419	141
279	274

Confusion Matrix:

<u>k-NN</u>
Sampling: Regular
Dimension Reduction: LDA
Classification Rate: .8015

1388	172

283	270

Confusion Matrix:

Decision Tree
Sampling: Oversampled
Dimension Reduction: LDA
Classification Rate: .8411

1200	347
121	1437

Confusion Matrix:

Random Forest
Sampling: Oversampled
Dimension Reduction: PCA
Classification Rate: .8682

1285	262
125	1433

Confusion Matrix:

4. Interpretation & Conclusion

The creation of these models provided us insight we didn't have prior. For one, had we utilized K-fold cross validation prior to examining the viability of different dimension reduction techniques, we would have potentially found different technique performance. In other words, when testing the misclassification rates of various n components in PCA and Kernel PCA against a logistic regression model, we might have found different model performances. In hindsight, this is one (potential) place of fault within our models. They might be not using the optimum number of n components for PCA and Kernel PCA during model creation. Although given the competitive nature of the outcomes of PCA related dimension reduction vs LDA, there is a decent likelihood it would not have made much of a difference. However, it is still important to note, and was discussed amongst our team, especially given LDA's being used in 3 of the top 5 models.

A note on something handled externally to our submitted notebook was the evaluation of the n neighbors hyperparameter for our k-NN models. Although missed in our final notebook submission, the process was standard trial and error. We looped through n values, ran our model against the regularly sampled dataset, (at this point we had come to the conclusion that excluding the other two sampling techniques in this situation would suffice) and compared the results. This led us to our decision of using 25 neighbors.

As far as the performance indicators of our models are concerned, wow. Decision Tree and Random Forest are head and shoulder better than the others. Their strongest results came when oversampling to fix our class imbalance. Both of their performances dropped significantly from oversampling to undersampling. Especially in regard to our Decision Tree models, whose classification rates dropped almost 20%, regardless of dimension reduction techniques.

Logistic Regression, Kernel SVM, and k-NN all with very similar classification rates that are relatively low compared to our top two performers.

Random Forest with oversampling and PCA dimension reduction resulted in our optimum model. With a K-fold cross validation mean classification rate of .8682, and with a confusion matrix without any extreme abnormalities and similar to its nearest competitor, (Decision Tree with oversampling and LDA) we feel confident prescribing it as the most viable model for predicting customer churn.

REFERENCES

[1]https://www.kaggle.com/danwheble/churn-prediction-telco-customer-churn/execution