Capstone Project Report - Analyzing Customer Churn for a Telecommunications Company

Rafael Aviles - Udacity Machine Learning Nanodegree

Project Overview

Many companies in the consumer market and enterprise sectors have to deal with churn. A churned user is a user that stops using a product, subscription or membership. Especially with the growth of SaaS companies, churn has become a hot topic in recent years. Accurately predicting if a user will churn, is crucial to keep Monthly Recurring Revenue (MRR) steadily growing. An accurate prediction would allow businesses to address the issue and take action before customers actually leave. This can be done in different ways such as, discounts, individual customer support and other different measures. I personally work for a SaaS company and churn is a huge deal for us. If companies succeed in accurately predicting churn, the business implications can have a substantial effect on the success of the company. I'm deeply fascinated by this problem and with this project, I would like to be part of the solution for my company to overcome it.

The dataset consists of customers from a telecommunications company, where the goal is to predict if the customers will churn or not. The data is taken from an IBM's Watson blogpost, where IBM demonstrates the prediction features of their Artificial Intelligence technology, Watson. It consists of 7043 customers, from which 1869 of them did indeed churn and 5174 remained as paying customers.

This is a classic classification problem, where customers are classified as churners or retained customers. To address this problem, I've used Support Vector Machines, a machine learning algorithm used to classify samples into different groups by learning a hyperplane to separate the classes and assign future data points into one of them.

Project Statement

The goal of this project is to create a classification model, that accurately predicts if a user will churn or not. This is a binary classification problem and the following procedure was taken to achieve this.

- Explore dataset and prepare data for model
- Clean data according to exploration
- Explore relationship between features and target variable
- Select model

- Justify selection of model
- Run model and obtain predictions
- Compare model to a baseline using chosen metric
- Analyze Results and write conclusion

Metrics

The most important metrics for this report are *Recall and Precision (see definition in image below).* False Negatives are the errors we want to avoid, because the ultimate goal is to reduce churn by taking action and accurately predict customers that will churn. Thus, the ones we miss and upon no action would be taken, are the ones that would hurt us the most. On the other hand, we could predict that all users will churn, take action with all of them and probably reduced actual churn rate. Nevertheless, this is not efficient and would be too expensive to take measures with all customers, that's why precision is also important. Consequently the f1 score was taken as final metric, as this takes into account both precision and recall: `F1 = 2 * (precision * recall) / (precision + recall).`

		Prediction	
		churn	*not churn*
Actual	*churn*	*TP*	*FN*
	not churn	*FP*	*TN*

Confusion matrix

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{TN + FP}$$

$$Accurracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Data Exploration

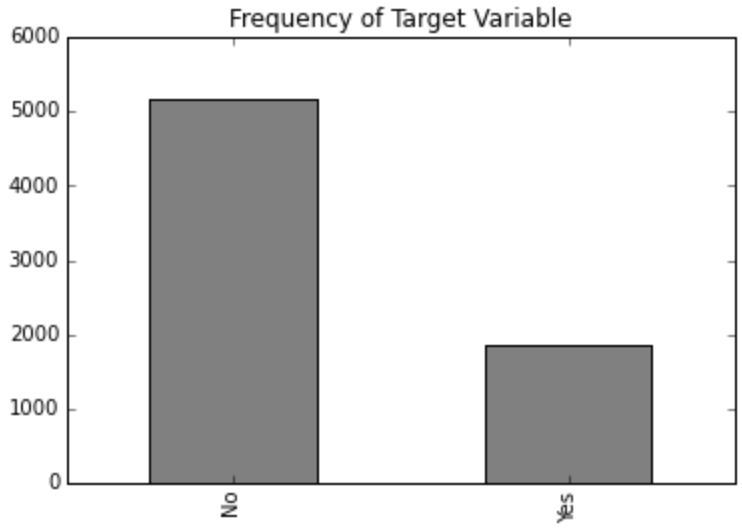
The data set was extracted from one of IBM's Watson blogpost [7]. There are 7043 data points or customers in total. The 21 columns consist of 1 being the predicted variable 'Churn' with the values 'yes' or 'no', another as a specific identifier for each customer and other 19 columns with potential predictive power. The number of customers who churn is 1869 and 5174 customers who did not churn. Overall, only 11 values were missing from the variable *Total Charges*, being so few I decided to substitute them with the mode of that column, which is 20.20. For the three numerical variables (*tenure*, *Monthly Charges* and *Total Charges*) no outliers outside of +/- 1.5 IQR were found.

Table 1: Dataset variables

Feature	Description	Values
customerID	Customer Unique ID	Unique String
gender	Gender of the customer	Female, Male
SeniorCitizen	Indicates if customer is a senior	0,1 (1 being yes)
Partner	Indicates if customer has a partner	No, Yes
Dependents	Indicates if customer has children	No, Yes
tenure	Months of current contract	from 0 to 72
PhoneService	Indicates if customer has phone service	No, Yes
MultipleLines	Indicates if customer has more than one line	No, No phone lines, Yes
InternetService	Type of internet service	DSL, Fiber Optic, No
OnlineSecurity	Indicates if user has signed up for online security	No, No internet service, Yes
OnlineBackup	Indicates if user has signed up for online backup	No, No internet service, Yes
DeviceProtection	Indicates if user has signed up for device protection	No, No internet service, Yes
TechSupport	Indicates if user has signed up for tech support	No, No internet service, Yes
StreamingTV	Indicates if user has signed up for streaming TV	No, No internet service, Yes
StreamingMovies	Indicates if user has signed up for streaming movies	No, No internet service, Yes
Contract	Type of contract	Month-to-month, One year, Two year
PaperlessBilling	Indicates if customer uses paperless billing	No, Yes
PaymentMethod	Type of payment method	Bank transfer (automatic), Credit card (automatic), Electronic check, Mailed check
MonthlyCharges	Amount paid per month	From 18.25 to 118.25
TotalCharges	Total amount paid to date	From 18.80 to 8,684.80
Churn	Specifies if customer churned or not	No, Yes

Exploratory Visualization

The following plot shows the distribution of the target variable, *Churn.* It can be observed that the dataset is imbalanced with 5174 customers who did not churn and 1869 who did churn.



Moreover it has been decided to visualize the top 5 variables where a high correlation with the target variable can be observed. All the other charts can be found in the Python Notebook if necessary. Categorical variables have been plotted using Seaborn's Factorplot and numerical variables have been plotted against the Churn variable using Seaborn's Distplot.

Target variable by Techsupport

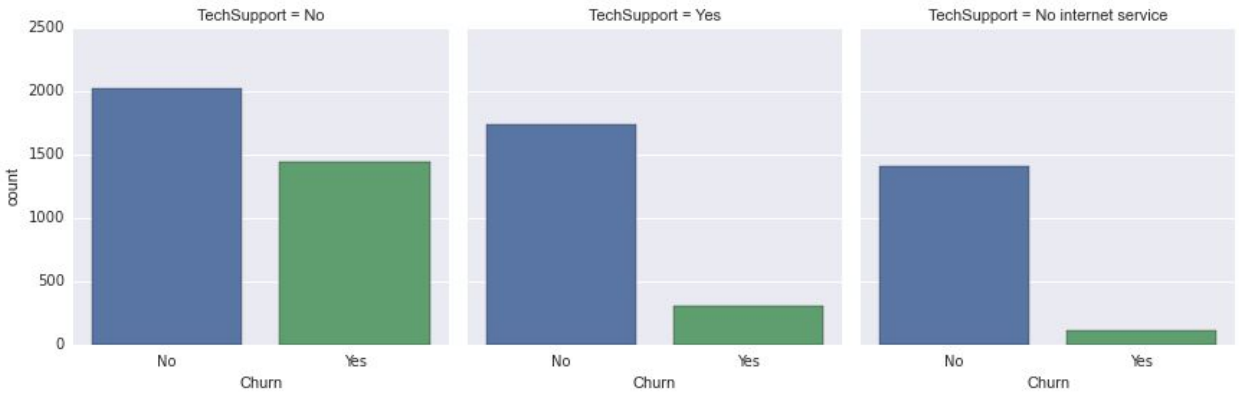
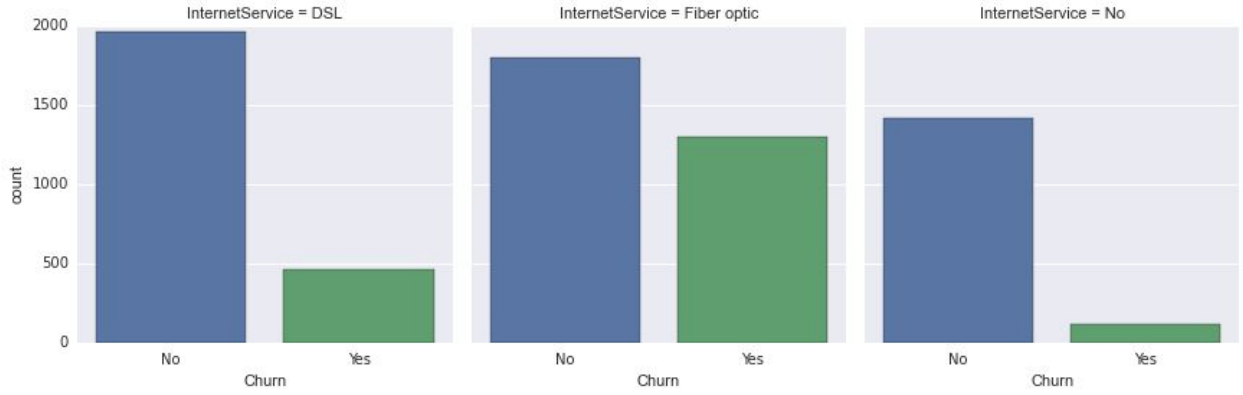


Image 2

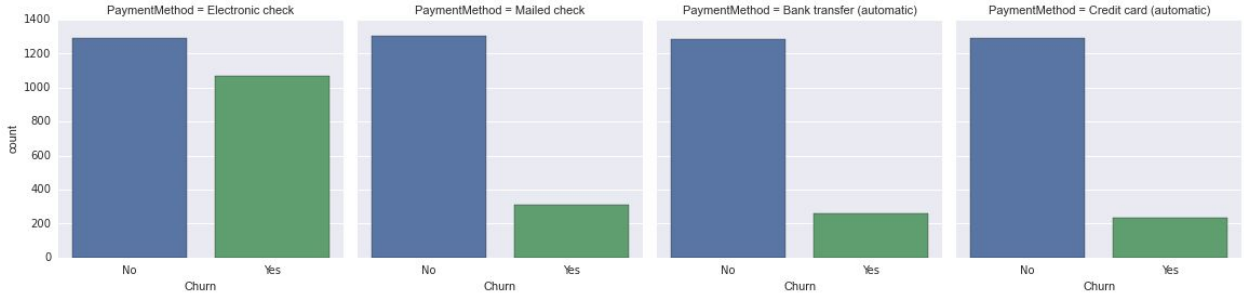
It can be observed that there is a substantial difference between the amount of customers who churned vs the one who did not churn, where TechSupport was given and where it wasn't. I looks like customers who didn't have TechSupport are much more likely to churn.

Target Variable by InternetService



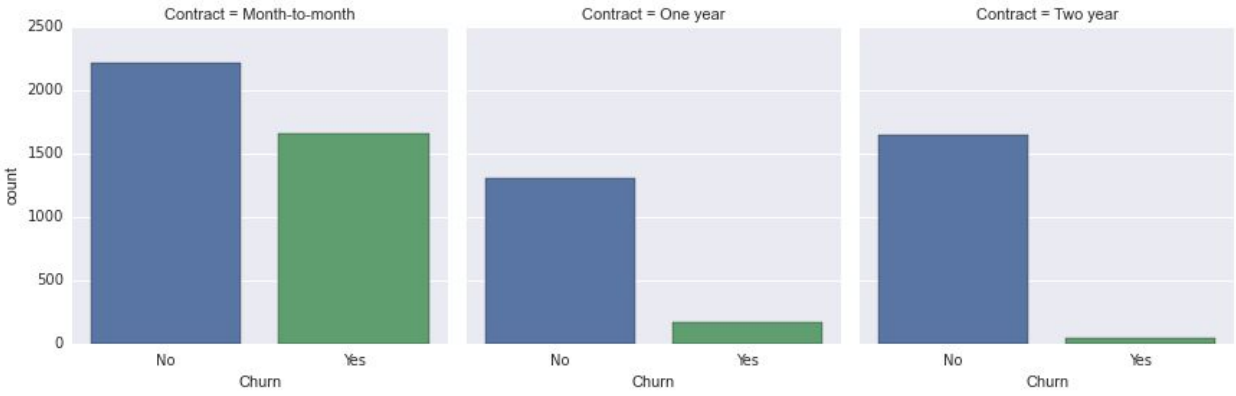
Customers who had 'Fiber Optic' as the InternetService are also more likely to churn than customers, who were using DSL as InternetService.

Target Variable by PaymentMethod



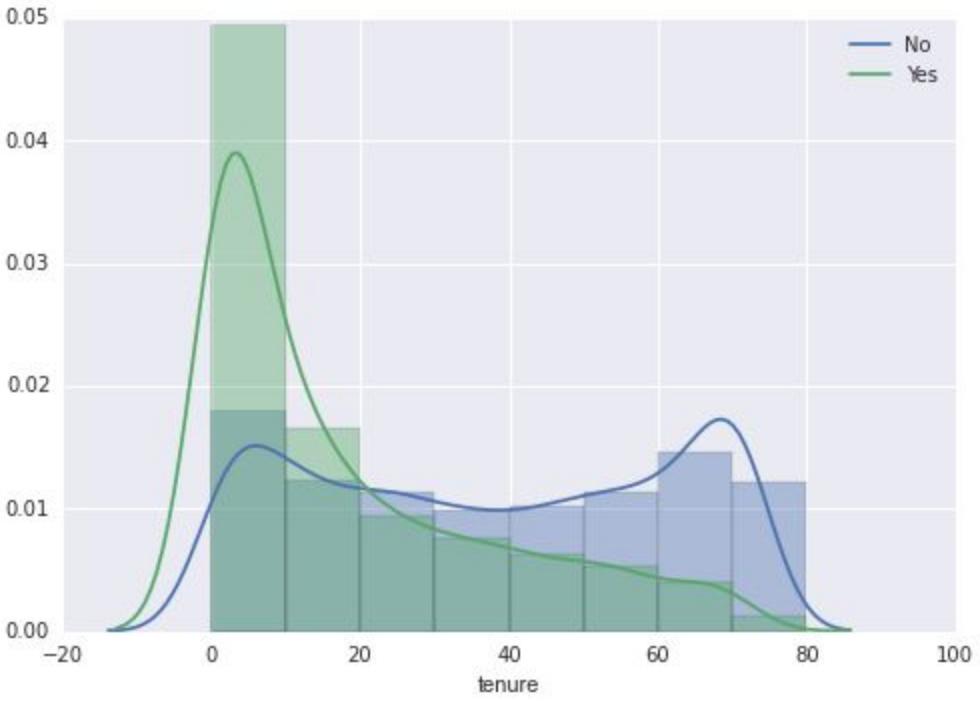
From this variable the segment of customers, who were using Electronic check were more likely to churn than all the other payment methods.

Target Variable by Contract Type



Customers with Month-to-Month contracts seem to be also more likely to churn than customers with yearly contracts.

Density Estimation of Tenure by Target Variable



It can be observed that the longer customers have been on the contract, the less likely they will churn. On the contrary there seems to be no or at least a weak correlation of customers, who didn’t churn and an increase in the tenure variable.

Algorithms and Techniques

This is a classical problem supervised machine learning, more specifically, a classification problem. Hence, we need algorithms and models that are adequate to such problem.

The algorithm chosen for this problem is Support Vector Machines (SVM). SVM work well when there is a clear margin of separation between the classes. Moreover this separation doesn't necessarily have to be linear, as SVM supports kernels, so you can model even non-linear relations. Finally they maximize the margin, which makes them more robust and less likely to high have variance.

They don't work well where there is a lot of noise in the data and when the target classes are very overlapping. Moreover, if the datasets are large, their computation is cubic as a function of the size of the dataset, which makes it computationally expensive. This data set has very little noise and the samples are less than 10K, which according to Sklearn documentation, can be a good threshold to pick SVM[1]. More specifically, I have decided to use Linear SVM, as it scales better and would allow me to test more values using *gridSearch*, when training the model as it would be more efficient.

The following parameters can be tuned to optimize the estimator are:

- C
- tol

Other Algorithms

Other suitable algorithms that could be used for classification problems are:

1. **Ensemble Methods:** These algorithms seem to perform well**.** From 2 of the most known algorithms, Boosting requires a large training sample and our dataset [8] with 7043 samples is not as large. Adaboost for instance is quite sensitive to noisy data [9] and even though this example, our dataset is relatively clean, I wouldn't expect to have such a clan data set in a real SaaS example, like our company. At least in our case, data we gather from Intercom, our own backend and other sources is relatively noisy, so the model would be harder to generalize.
2. **Decision Trees:** Decision trees (DTs) are easy to train and easy to interpret. In other words DTs don't perform well when there is not much data or features, so they tend to overfit and fail to generalize on new data. My goal for this project is to take it as a basis for the company I work for (and hopefully others), which will have probably different data and DTs' tendency to overfit, has make me prefer SVM.
3. **Naive Bayes:** Doesn't need much data to be trained, which is the case of this project[2]. Moreover it's easy and fast to train. Nevertheless, some of the variables seem to correlate and be dependent from each other. For instance, there are 6 variables that depend if the customer has internet service at all. Thus, the conditional independence

[1] http://scikit-learn.org/stable/tutorial/machine_learning_map/

[2] Dataset is not really massive <10.000 samples

might not hold and I preferably would like it to do so, in order to use this technique in this project.

Benchmark

I'll provide two benchmark models. First, a random model, where the target variable will be randomly predicted. Here a recall of 0.5 is expected. Secondly I've taken one of the models from IBM's Watson's examples from where the data was extracted. Below the image describing the top decision rules form the model described there, which has precision of 86%. After analyzing the benchmark, an f1 score of 0.104 and a recall of 0.05.

In my opinion the precision will be the hardest to surpass, but I am very confident that the f1 score will be higher, as I expect my model to have a more balanced relationship between recall and precision. The recall for Watson is relatively low and thus, the f1 score is also low.

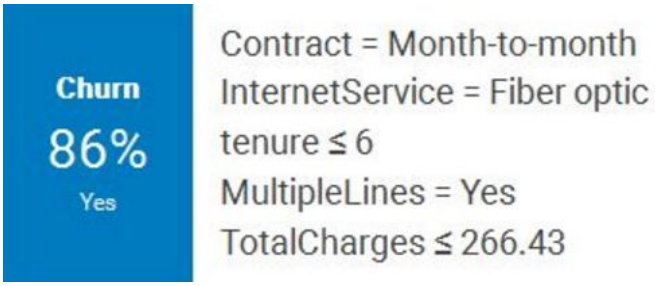


Image 8: IBM Watson's Decision Tree

Data Preprocessing

For the variables *Partner, Dependents, PhoneService, PaperlessBilling* the value 'Yes' was replaced by a 1 and the value 'No' was replaced by a 0.

For the variable *gender,* the value 'Female' was replaced with a 1 and the value 'Male' with a 0.

Numerical variables *tenure, MonthlyCharges, TotalCharges* were normalized using min-max method. This step was taken as it is normally suggested that SVM should have normalization as a preprocess step [2].

For all categorical variables, dummy variables were created. From these, for the internet related variables *OnlineSecurity, OnlineBackup, DeviceProtection, TechSupport, StreamingTV, StreamingMovies,* there was one dummy for customers who did not have internet service at all, which contains the same information included in the dummy *InternetService_No*. Thus, for these 6 variables the final dummy ***variable**_No_Internet_Service* was dropped.

Implementation

For the implementation of the algorithm the package Scikit-learn has been used, more specifically the LinearSVC class [4]. The following steps were taken:

1. Split test and train sets using *train_test_split,* with different number of training samples (1000, 2000, 3000, 4000, 5000 and 6000). On top of that, the parameter *random_state* was always set to 40, so results could be reproducible.
2. Define parameters of model and possible values for each:
    - **C:** The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. Values used: 1, 5, 10, 1000
    - **Tol:** The tolerance error allows the algorithm to finish in less number of steps even if the optimal hyperplane has not been computed yet. Values used 0.0001, 0.001, 0.01, 0.1
3. Initialize classifier with a *random_state* of 40. This allows to always have the same randomized sample and make the results reproducible.
4. Define metric the parameter *scoring* for *gridSearch,* using *make_scorer* from *Sklearn.metrics,* using parameters pos_*label = 1* and *average = binary*, to get the results from the perspective of the class 'Yes' (for instance a True Positive would be a customer who churned and where the model predicted she would churn too).
5. Train model using *gridSearch*, with the parameter *cv = 10*, to determine the number of folds to use in the splitting strategy.
6. Finally, the model was trained 6 times using the number of training samples specified above. In the table below you can see the results for the testing sample, split by the number of training samples.

Table 2: Testing and Training score of the model (before refinements)

Model Nr.	*Testing Sample*	*Training Sample*
Training sample = 1000	*0.551*	*0.534*
Training sample = 2000	*0.598*	*0.605*
Training sample = 3000	*0.591*	*0.614*
Training sample = 4000	*0.589*	*0.607*
Training sample = 5000	*0.598*	*0.596*
Training sample = 6000	*0.555*	*0.602*

The final model contained the following parameter values:

{'loss': 'squared_hinge', **'C': 1**, 'verbose': 0, 'intercept_scaling': 1, 'fit_intercept': True, 'max_iter': 1000, 'penalty': 'l2', 'multi_class': 'ovr', 'random_state': 40, 'dual': True, **'tol': 0.0001**}

Refinements

As part of the refinements, the following actions were taken

1. Used parameter *stratify* in *train_test_split* to make sure that the distribution of values in the test and training data sets are the same. This is very important, as we are dealing with an unbalanced dataset in this case.
2. Introduced different values for the *class_weight* (as suggested in [1]) parameter that gives more importance to a certain class, in this case the churning users with target variable of value 1. Again this is done due to the unbalanced classes.

Table 3: F1 score of test (validation) sets for model comparison after refinement using *class_weight and stratify*

Model Nr.	*1*	*2*	*3*	*4*
Stratified Dataset	*No*	*Yes*	*No*	*Yes*
Class weight	*No*	*No*	*Yes*	*Yes*
Training sample = 1000	*0.551*	*0.568*	*0.623*	*0.628*
Training sample = 2000	*0.598*	*0.573*	*0.627*	*0.625*
Training sample = 3000	*0.591*	*0.587*	*0.625*	*0.630*
Training sample = 4000	*0.589*	*0.586*	*0.624*	*0.635*
Training sample = 5000	*0.598*	*0.608*	*0.623*	*0.634*
Training sample = 6000	*0.555*	*0.589*	*0.590*	*0.632*
Average	***0.580***	***0.585***	***0.619***	***0.631***

The final model contained the following parameter values:

{'loss': 'squared_hinge', **'C': 1**, 'verbose': 0, 'intercept_scaling': 1, 'fit_intercept': True, 'max_iter': 1000, 'penalty': 'l2', 'multi_class': 'ovr', 'random_state': 40, 'dual': True, **'tol': 0.0001, class_weight: {1:2}**}

Model Evaluation and Validation

Firstly the model's parameters were chosen after using Gridsearch. For C, 1 returned the best value and for tol, 0.0001 did so. During refinement the parameter *class_weight* was added to the *gridSearch* and the value {1:2} returned the most optimal score in combination with the previous C=1 and tol=0.0001 values. Moreover, when the dataset was also stratified, the best f1 score was achieved. This can be observed in Table 3. The final model looked the following:

- Stratified test and training sets with a distribution of 1 churned customer for every 2.76 not churned customers.
- Model parameters: {'loss': 'squared_hinge', 'C': 1, 'verbose': 0, 'intercept_scaling': 1, 'fit_intercept': True, 'max_iter': 1000, 'penalty': 'l2', 'multi_class': 'ovr', 'random_state': 40, 'dual': True, 'tol': 0.0001, 'class_weight': {1: 2}}.
- Training set: 4000 samples, as it seems from all 4 models, that with more training points the model starts overfitting.

In order to test the robustness of the model, different distributions of the target classes, were tested. The model using a total of 4000 sample leaves out 3043 samples to use as test set. From these 3043 samples, different distributions were extracted and the f1 score calculated, along with recall and precision as seen in Table 4. The code for this section can be found in the Python Notebook under the section Oversampling (for Class 'Yes' > 26%) and section Undersampling (for Class 'Yes' < 26%).

Table 4: Robustness test for the model using different class distributions

Relative Frequency of Class 'Yes'	F1 Score	Recall	Precision
75%	0.819	0.754	0.897
50%	0.756	0.754	0.760
26% (actual)	**0.635**	**0.754**	**0.550**
10%	0.400	0.754	0.272
5%	0.262	0.795	0.157

With an increase in the number of customers who churned, the f1 score improves, as the precision goes up. As the number of customers who churned decreases, so does the f1 score, dragged by a fall in the precision. This tells us that the model is very robust, when it comes to predicting the customers who will churn, but not so much when it comes to customers who won't churn.

Justification

The model developed by IBM Watson is a decision tree and was replicated in the Python Notebook for this work. It has the following scores:

Table 5: Scores for IBM Watson's Decision tree

Model	F1 Score	Recall	Precision
IBM Watson Decision Tree[3]	0.104	0.055	0.858

As seen in table 5, this model has a substantially lower f1 score than ours, roughly 6 times lower. On one side it has a very high precision and on the other, a very low recall. After looking more closely, this model predict only 120 customers churning, when in fact 1869 did so. The precision comes from 103 customers who did in fact churn from these 120. In other words, this model was good a finding a specific group of users who might churn, but not so good at predicting more of these. In this case I would argue that, the model developed in this work, does a better job for the telecommunications company, because it would indeed reduce the amount of users churning. Our model only misses around 25% of churning customers, whereas Watson's model misses roughly 95% of them.

Free-Form Visualization

Below a visualization of both, the final model and the benchmark model from IBM Watson. With these, it is easier to observe how unbalanced the prediction from the benchmark model is, where 6923 out of 7043 samples are predicted as 'Not Churn'. On the other side our final mode, looks more balanced, having a ratio of 1.95 'Not Churn' for every 'Churn', much more closer to the 2.76 of the real classes than the 58 ratio of the benchmark model.

Table 6: Confusion matrix from the final model

		Prediction		
		Churn	Not Churn	Total
Actual	Churn	1187	682	1869
	Not Churn	971	4203	5174
		2158	4885	7043

[3] As seen in Image 8

Table 7: Confusion matrix from IBM Watson's model

		Prediction		
		Churn	Not Churn	
Actual	Churn	103	1766	1869
	Not Churn	17	5157	5174
		120	6923	7043

Reflection

The process of this project can be summarized in the following steps:

1. Define problem: Predict churn of customers for a telecommunications company
2. Load, clean and preprocess data
3. Visualize important traits from the data
4. Define baseline model (benchmark), as a decision tree from IBM's Watson
5. Split data in different test and split tests with different sizes
6. Train the model using training data
7. Refine model by using different techniques for unbalanced data, more specifically using the parameter *class_weight* and stratifying the training and testing samples
8. Retrain model with refinements and GridSearch to find the best combination of parameters
9. Analyze and compare the results with the baseline

Overall this problem has the very interesting feature of being an unbalanced dataset. This presented the highest challenge, because the distribution of the classes seems to drastically affect the precision, as seen in table 4. So it was a difficult task deciding, if the model was suitable at the expense of precision, which at the end I did find suitable. The reason for this, is that the data set is not so small and I assume a good representation of the entire population, thus future unseen data having roughly the same distribution as our data and thus returning a similar precision. Moreover the robustness of the recall is so strong and the model seems to do well in predicting the users who will indeed churn, that the model seems to be very good at reducing the number of customers that matter the most: the ones who will churn.

Improvements

Something that wasn't carried out at all was Feature Selection. I didn't apply any methods to reduce dimensionality nor did I remove features with low correlation with the target variable. Including this process might return different results and would be interesting to observe the effect in my opinion. Even though some research [6] suggests that Feature Selection might not have any effects on the accuracy, it still would be interesting to see if there is an effect in robustness, precision and recall.

Another important improvement, could be using different kernels to train the data. In our case the computational cost was too high for the hardware we were using and it took too long to train models that had not a linear kernel. A comparison of the results using different kernels would give more insights on the quality of this model.

Sources

[1] http://scikit-learn.org/stable/modules/svm.html#svm-classification
[2] http://www.hpl.hp.com/techreports/2009/HPL-2009-31R1.pdf
[3] http://cs229.stanford.edu/notes/cs229-notes3.pdf
[4] http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC
[5] https://www.quora.com/What-is-the-computational-complexity-of-an-SVM
[6] http://www.ida.liu.se/~jospe50/ecml2006.pdf
[7|https://www.ibm.com/communities/analytics/watson-analytics-blog/predictive-insights-in-the-telco-customer-churn-data-set/
[8] http://www.slideshare.net/marinasantini1/lecture06-ml4-ltmarinasantini2013
[9] http://www.cbcb.umd.edu/~hcorrada/PracticalML/pdf/lectures/EnsembleMethods.pdf