UDACITY

Machine Learning Engineer Nanodegree

Capstone Project

Predicting Customer Churn in the Telco Industry

Manuel Seeger, 2019

Table of Contents

Definition **3**

Domain Background *3*

Personal Motivation 3

Problem Statement *3*

Evaluation Metrics *4*

Analysis **5**

Data Exploration *5*

Overview 5

Grouping 6

Target 6

Sample 6

Data quality 7

Churn rate across columns 7

Algorithms *11*

Benchmark *12*

Methodology **13**

Preprocessing *13*

Outlier removal 13

Feature selection 14

One-hot encoding 14

Resampling 14

Scaling 14

Summary 15

Implementation *15*

Keras 15

Global objectives 15

Refinement *15*

Feature selection 15

Hyperparameter tuning 16

Results **18**

Model Evaluation *18*

Learning Curves 18

Justification *19*

Metrics report 19

Cost modelling 20

Summary 21

Conclusion **22**

Free form visualization 22

Reflection *23*

Lessons Learned 23

Improvement *23*

Appendix **24**

A. References *24*

B. Keras model summary *24*

C. Global objectives *25*

Definition

Domain Background

Probably more than others, players in the telco industry suffer from "churn": The move of a customer from one provider to another. In developed markets, the cost of switching providers is increasingly low for the consumer. Reasons include regulation, like transferable phone numbers and an increasingly level playing field in terms of network coverage. At the same time, overall industry growth in terms of number of addressable consumers is slowing down due to high market saturation and maturation. Differentiators like network coverage or exclusive hardware offers are becoming smaller or disappear altogether.

It costs upward of 5 times as much to acquire a new customer than it costs to retain an existing one according to InvestP. Retaining their customer base by reducing churn is thus a very important part of a telco's customer relationship strategy. This study will seek to help predicting churn to support a churn prevention campaign.

While this study will work on data from the telecom industry, predicting and managing churn is a shared problem across most consumer-facing industry. The methodology applied in this project are expected to be transferable to similar problems cross industry, as long as suitable data exist.

Personal Motivation

As a consultant for customer relationship management software, smartly segmenting customers for targeted campaigns is a recurring problem to be solved. Where data-driven prediction and segmentation tools are missing, targeting is done by intuition of marketing professionals and/or based on naive heuristics. I hope to draw insights on how to improve campaign targeting from this project.

Problem Statement

Even if preventing churn is considered cheaper customer-by-customer than acquiring a new customer, it comes at a cost. A customer retention campaign might include costly incentives for customers to reconsider churning and extend their contracts. It is most important to target these incentives at the customers most likely to churn, and not "waste" a costly incentive on customers that would have stayed with the provider in the first place.

Predicting which customers are likely to churn, and which are likely to stay thus becomes a critical component of the customer base segmentation for a retention campaign. Neslin, Gupta, et al. estimate that an increase in lift[1] of churn prediction of just 0.1 can translate to an additional 100,000 USD of profit captured for a mid-size mobile carrier.
This project will develop different models for customer churn prediction on a customer dataset and select the model with the best performance on predicting if a customer is likely to churn.

In machine learning terms, predicting churn is a binary classification problem. There is one target variable, Churn, which can have 1 of 2 labels: Yes - the customer is leaving[2]; No - The customer is staying with the provider. Our machine learning problem solution will take information about the customer as input (such as financial, contractual, or personal), and output the probabilities that a customer belongs to the churner and the non-churner group.

Note: Churn can be understood as all customers leaving the company or separated in voluntary and involuntary churn. We are only concerned with voluntary churn, where the customer decides to leave (and will likely sign up with a competitor). Churn in this document thus means voluntary churn.

[1] Lift: Delta in probability, that a customer targeted by retention campaign is a churner compared to general customer population
[2] Churn: „Yes" actually states that the customer *will* be leaving 2 months after the observations from the dataset have been made.

Evaluation Metrics

We look at our model's performance in terms of the confusion matrix in Table 1.

	Predicted retention	Predicted churn
Loyal customer	true negative	false positive
Churning customer	false negative	true positive

Table 1 Confusion Matrix

Our model's predictions should have as few false negatives as possible, as those would be customers we likely lose because we don't target them with a retention campaign. We are not as much concerned with false positives, as a retention campaign is considered much less costly than a (re-)acquisition campaign. In other words, we can afford some false positives, as long as we miss as few as possible true churning customers.

In a real-world example, the distribution of churners and non-churners is highly skewed. Neslin, Gupta, et al. estimate around 2% of customers churn monthly for a typical US carrier. We are thus mostly concerned in keeping false negatives low, and in catching as many churning customers as possible.

In terms of performance metrics, we modelled this evaluation as a high-recall f-beta score. In the original approach to the project, an fβ-score with β = 2 was proposed as the performance metric. This however turned out to lean too much on recall, and caused models optimized against this metric to always predict churn (which was a winning strategy).
After adjustment, the primary metric used in the project is:

fβ-score with β = 1.25

In the evaluation section we review the models against additional metrics, like accuracy, and review the confusion matrix again.

Analysis

Data Exploration

Overview

The churn prediction was done on the cell2cell dataset of Telco customers. The dataset was originally donated by an (anonymous) US carrier and published by the Fuqua School of Business at Duke University as part of a machine learning competition. The descriptions have been taken from the appendix of V. Umayaparvathi, K. Iyakutt.

The dataset contains 56 feature columns, 1 target column, and one ID column.

Feature columns

Column	Description
Customer care	
DroppedCalls	Mean number of dropped voice calls
BlockedCalls	Mean number of blocked voice calls
CustomerCareCalls	Mean number of customer care calls
DroppedBlockedCalls	Mean number of dropped or blocked calls
RespondsToMailOffers	Customer responds to mail offers
OptOutMailings	Has chosen not to be solicited by mail
RetentionCalls	Number of calls previously made to retention team
RetentionOffersAccepted	Number of previous retention offers accepted
MadeCallToRetentionTeam	Customer has made call to retention team
Customer demography data	
UniqueSubs	Number of Unique Subs
ActiveSubs	Number of Active Subs
ServiceArea	Communications Service Area
Handsets	# Handsets Issued
HandsetModels	# Models Issued
CurrentEquipmentDays	Number of days the customer had the equipment
AgeHH1	Age of first household member
AgeHH2	Age of second household member
ChildrenInHH	Presence of children in HH
HandsetRefurbished	Handset is refurbished
HandsetWebCapable	Handset is web capable
TruckOwner	Subscriber owns a truck
RVOwner	Subscriber owns a recreational vehicle
Homeownership	Subscriber is a home owner
BuysViaMailOrder	Customer is known to buy via mail order
NonUSTravel	Has traveled to non-US country
OwnsComputer	Owns a personal computer
HasCreditCard	Possesses a credit card
NewCellphoneUser	Known to be a new cell phone user
NotNewCellphoneUser	Known not to be a new cell phone user
ReferralsMadeBySubscriber	Number of referrals made by subscriber
IncomeGroup	Income group the customer belongs to
OwnsMotorcycle	Owns a motorcycle
HandsetPrice	Handset price (0 = missing)
PrizmCode	Rural, suburban, town, or other
Occupation	Customer's professional occupation
MaritalStatus	Whether customer is married or not
Financial	
MonthlyRevenue	Mean monthly revenue
TotalRecurringCharge	Mean total recurring charge
PercChangeRevenues	% Change in revenues
MonthsInService	Months in Service
AdjustmentsToCreditRating	Number of adjustments made to customer credit rating (up or down)
CreditRating	Credit rating from lowest to highest
Customer usage pattern	
MonthlyMinutes	Mean monthly minutes of use
OverageMinutes	Mean overage minutes of use
RoamingCalls	Mean number of roaming calls
PercChangeMinutes	% Change in minutes of use
UnansweredCalls	Mean number of unanswered voice calls

ReceivedCalls	Mean number of calls received
OutboundCalls	Mean number of outbound voice calls
InboundCalls	Mean number of inbound voice calls
PeakCallsInOut	Mean number of in and out peak voice calls
OffPeakCallsInOut	Mean number of in and out off-peak voice calls
Value Added Services	
DirectorAssistedCalls	Mean number of director-assisted calls
ThreewayCalls	Mean number of three-way calls
CallForwardingCalls	Mean number of call-forwarding calls
CallWaitingCalls	Mean number of call-waiting calls

Table 2 Features and descriptions

Grouping

The columns in the dataset can be grouped into the following technical categories:

Binary columns: Yes/No columns, like OwnsMotorcycle
Categorical columns: Columns that contain a set of values from a category, like CreditRating
Continuous columns: Continuously distributed columns, like RoamingCalls (All customer usage and value-added service columns).
Continuous columns can be *measured* in discrete bins, like the column AgeHH1
Discrete columns: For example, number of handsets issued.

Target

The target column is Churn, which is given as Yes and No.

There are 51047 records in the dataset, of which 14711 are churners. This gives us an overall churn rate in dataset of 0.2882 churners to remaining customers.

We can see that the dataset is imbalanced, and we have more loyal customers than churners. Note: This data has seen some pre-processing / selection already. A real-world customer dataset would show a single digit monthly churn percentage per month typical for the Telco business (Neslin, Gupta, et al.).

Sample

Table 3 shows a sample of 5 randomly chosen records from the dataset.

	134	**47066**	**24098**	**42578**	**48560**
CustomerID	3000998	3371574	3189758	3338506	3381898
Churn	No	No	No	No	No
MonthlyRevenue	94.99	160.09	31.73	73.47	128.67
MonthlyMinutes	1204	2314	167	900	1083
TotalRecurringCharge	85	120	30	50	45
DirectorAssistedCalls	0	10.89	0	3.46	0.99
OverageMinutes	0	177	0	100	224
RoamingCalls	0.3	0	0.9	0	0.6
PercChangeMinutes	-784	-315	45	11	381
PercChangeRevenues	-6.6	-36.1	5.1	46.7	59.3
DroppedCalls	12.7	35.3	3	7.7	16.7
BlockedCalls	15.7	2.7	0	56	1.7
UnansweredCalls	45	274	15.3	58	107
CustomerCareCalls	0.3	10.7	0	2.3	0.7
ThreewayCalls	0	0.3	0	0.7	0
ReceivedCalls	440.5	435.2	12.2	173.9	268.6
OutboundCalls	39.3	68.7	4.7	51.7	158.7
InboundCalls	28.3	56.7	0	7	58
PeakCallsInOut	257.7	871.3	35.3	144.7	221
OffPeakCallsInOut	35.7	346	26	183	322.7
DroppedBlockedCalls	28.3	38	3	66	18.3
CallForwardingCalls	0	0	0	0	0
CallWaitingCalls	6.3	5.3	0.3	3.7	7.7
MonthsInService	51	8	16	9	7
UniqueSubs	5	1	1	1	1
ActiveSubs	5	1	1	1	1
ServiceArea	NOLKEN504	NYCTMR732	NYCKPT732	NCRKAN704	DETPON248
Handsets	10	2	2	1	1

HandsetModels	5	2	2	1	1
CurrentEquipmentDays	30	13	71	259	184
AgeHH1	0	38	28	54	58
AgeHH2	0	34	28	50	52
ChildrenInHH	No	Yes	Yes	Yes	Yes
HandsetRefurbished	Yes	No	No	No	No
HandsetWebCapable	Yes	Yes	Yes	Yes	Yes
TruckOwner	No	No	No	No	Yes
RVOwner	No	No	No	No	No
Homeownership	Unknown	Known	Known	Known	Known
BuysViaMailOrder	No	Yes	Yes	Yes	Yes
RespondsToMailOffers	No	Yes	Yes	Yes	Yes
OptOutMailings	No	No	No	No	No
NonUSTravel	No	No	No	No	No
OwnsComputer	No	No	No	Yes	No
HasCreditCard	No	Yes	Yes	Yes	No
RetentionCalls	0	0	0	0	0
RetentionOffersAccepted	0	0	0	0	0
NewCellphoneUser	Yes	No	No	No	Yes
NotNewCellphoneUser	No	Yes	No	No	No
ReferralsMadeBySubscriber	0	0	0	0	0
IncomeGroup	0	3	9	6	7
OwnsMotorcycle	No	No	No	No	No
AdjustmentsToCreditRating	1	0	0	0	0
HandsetPrice	80	150	150	Unknown	Unknown
MadeCallToRetentionTeam	No	No	No	No	No
CreditRating	2-High	5-Low	3-Good	3-Good	3-Good
PrizmCode	Other	Town	Suburban	Town	Town
Occupation	Other	Other	Professional	Professional	Other
MaritalStatus	Unknown	Yes	Yes	Yes	Yes

Table 3 Sample of 5 records from dataset

Data quality

Missing values

The following columns showed missing values in at least one record:

MonthlyRevenue, MonthlyMinutes, TotalRecurringCharge, DirectorAssistedCalls, OverageMinutes, RoamingCalls, PercChangeMinutes, PercChangeRevenues, ServiceArea, Handsets, HandsetModels, CurrentEquipmentDays, AgeHH1, AgeHH2

For the percentage changes, the missing values have been filled in with 0. For other continuous columns, the missing values have been filled in with the mean of the column.

Column AgeHH1 and AgeHH2 have been found to be of questionable quality and have been removed from the dataset.

Churn rate across columns

The columns of the dataset were analyzed for their differences in churn, to identify columns likely to contribute to classification and to explore potential of reduction of dataset dimension by removing columns.

Binary columns

For binary columns, a threshold of 0.05 has been defined. We discarded columns where the churn rate of the column differs from the churn rate of the whole dataset less than this threshold.

Example

Threshold = 0.05

Column	**Churn rate of column**	**Churn rate of dataset**	**Difference**	**Outcome**
TruckOwner	0.2836	0.2882	0.0046	Discard

HandsetRefurbished	0.3220	0.2882	0.0338	Discard
MadeCallToRetentionTeam	0.4504	0.2882	0.1622	Keep

Table 4 Examples of churn rates for binary columns

Categorical columns

For categorical columns, the churn-rate within the categories, per column, has been compared to determine likely impact on the classification. Based on this, discarding of those columns from the dataset has been considered.

The following chart shows churn rates across categories for the categorical columns of the dataset.

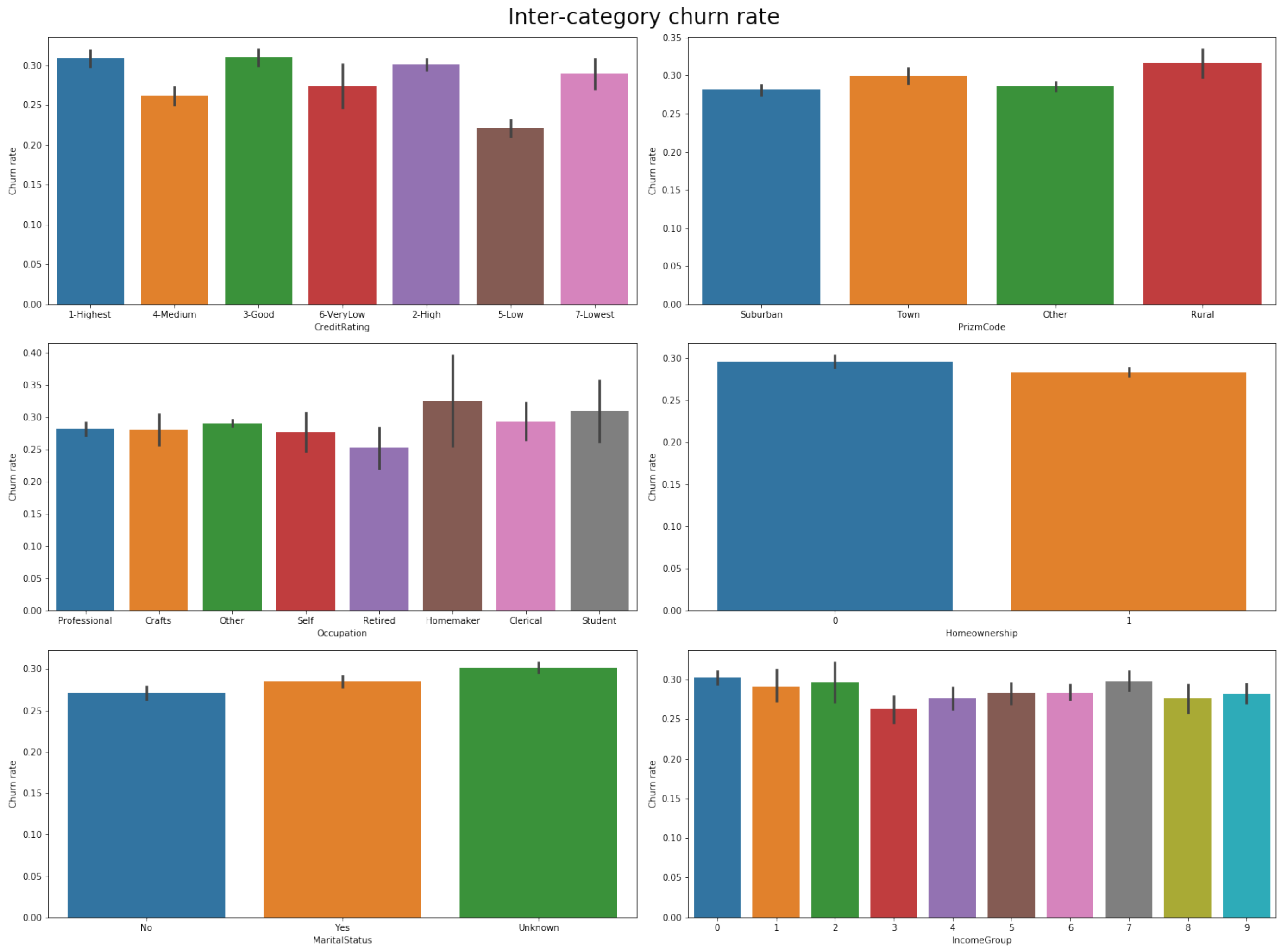

Figure 1 Churn rates of categorical variables

Observations on categorical columns:

- We can see differences in churn rates in credit rating, which is not surprising. High credit rating means a higher churn rate, which can be explained by customers with high credit rating having an easier time negotiating new, better contracts elsewhere. Interestingly, the lowest credit rating bin also has a high churn rate.
- We can see slight differences in location, with rural customers having a slightly higher churn rate, but the difference is very small.
- We can see differences in occupation, with students having a high churn rate, and retirees the lowest. This matches intuitive understanding of churn behavior.
- We see almost no difference in churn regarding home ownership
- We see a very small difference in churn marital status, and we have high number of unknowns.
- We see some difference in the income groups. Without knowing what the bins mean or in what order they were binned however, we cannot interpret these with certainty.

Based on these observations, MaritalStatus, Homeownership, and Prizmcode have been discarded from the data.

Service Area
It is reasonable that service area has some predictive power regarding churn. There might be bad coverage areas where competitor products have become more compelling.

However, there is a large number of service areas in the dataset, and encoding them would explode our dimensionality, adding over 700 dimensions, if we would encode service area into features.

We analyze churn rates of service areas similarly to the other categorical columns, but also consider size of each service area when making the decision on how to handle service area. The following chart shows number of subscribers per service area, their break down into churners and loyal customers, and their churn rates. The chart only displays service areas that have at least 100 subscribers.

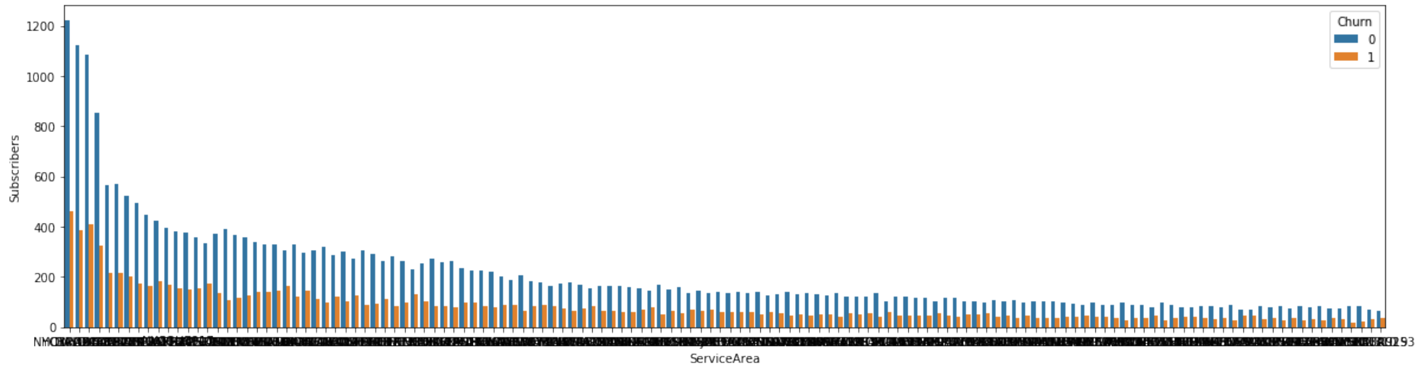

Figure 2 Subscribers per service area

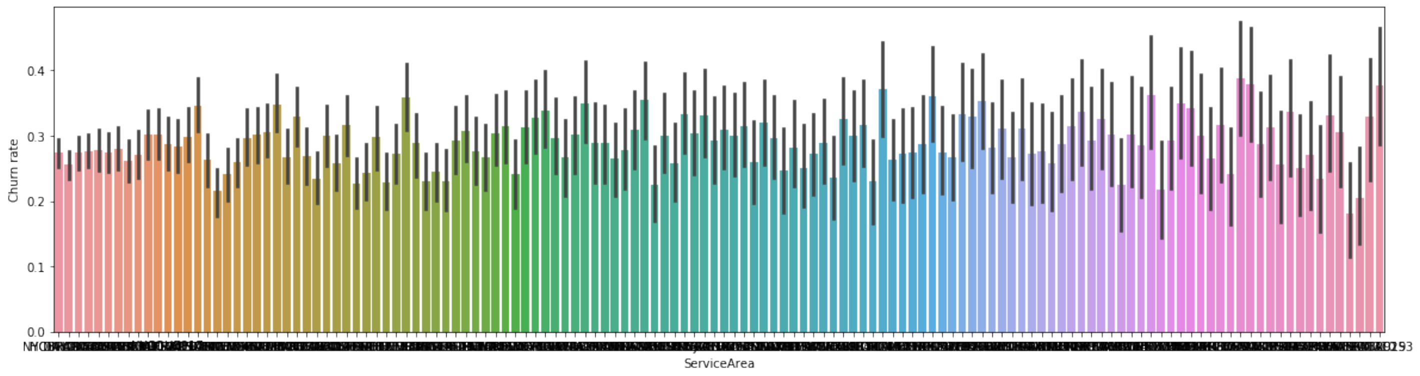

Figure 3 Churn rates per service area

The more subscribers a service area has, the more stable the churn rate, and the closer it is to overall churn rate of the dataset of 0.2882. Only where the areas are very small (just above the 100-subscriber threshold analyzed) do we see larger variations in churn rate. However, these small number of samples are not expected to generalize well.

With these observations of the data in mind, we chose to discard ServiceArea from the dataset.

Continuous and discrete columns

The majority of columns in the dataset are continuous or discrete columns. This includes all the customer usage pattern columns, financial information, value added services, as well as selected demography and customer care columns.

For customer usage pattern columns, we found that all columns are highly skewed to the right and most expose some degree of zero-inflation.

Examples

Four different kind of distributions were identified in the data for continuous and discrete variables. The following charts show pairs of examples for each kind of distribution.

We plot remainers and churners per automatically binned groups in histograms for each column.
What we look for are **clear separations** of the distribution of churners from remainers, **outliers**, and **normality**.

Example 1
We see a very clear separation of remainers from churners, and distributions approaching normality, with some right skew.

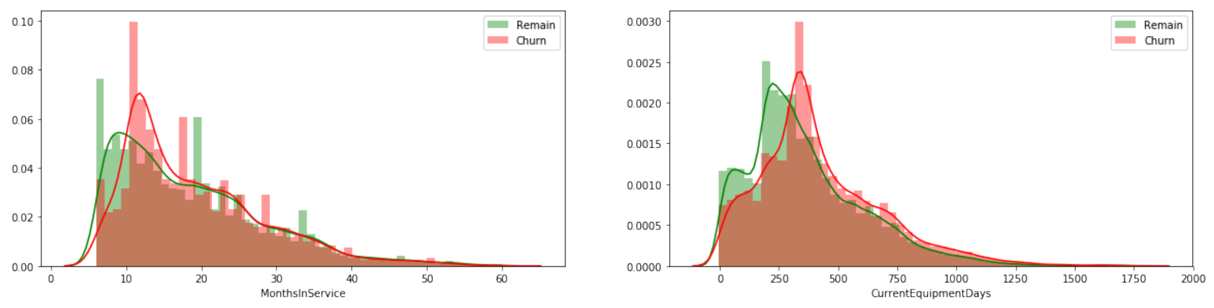

Figure 4 Numeric features with clear separation

Example 2
Some separation visible in plotting, but the separation is difficult to make out from just visual inspection. Right skewed and far outlying values are observable to the right.

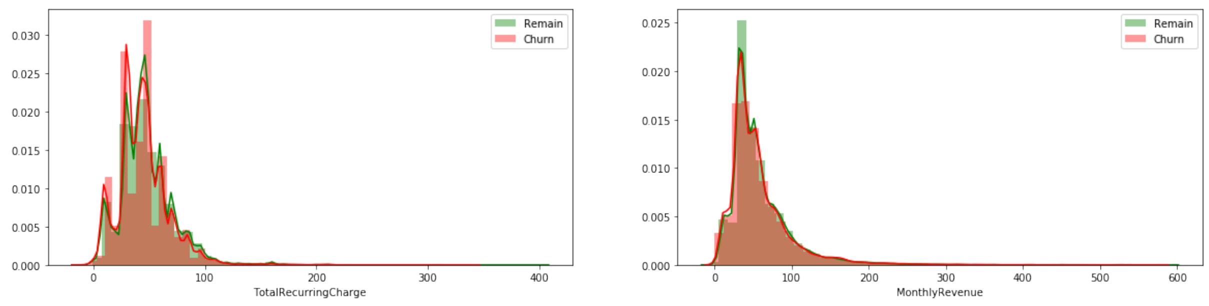

Figure 5 Continuous features with some separation visible

Example 3
Virtually no separation noticeable from visual inspection. Right skewed and far outlying values to the right.

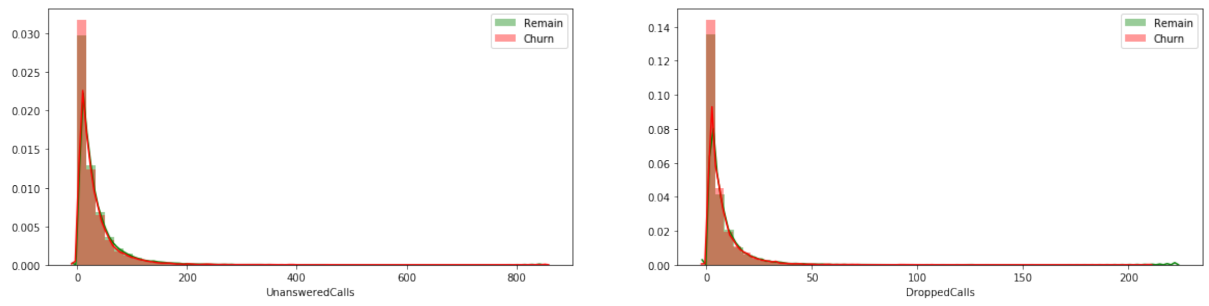

Figure 6 Numeric discrete features with high right skew

Example 4
Extremely thinly populated columns. Almost all values are near zero. Separation can’t be called from visual inspection. Right skewed and far outlying values to the right.

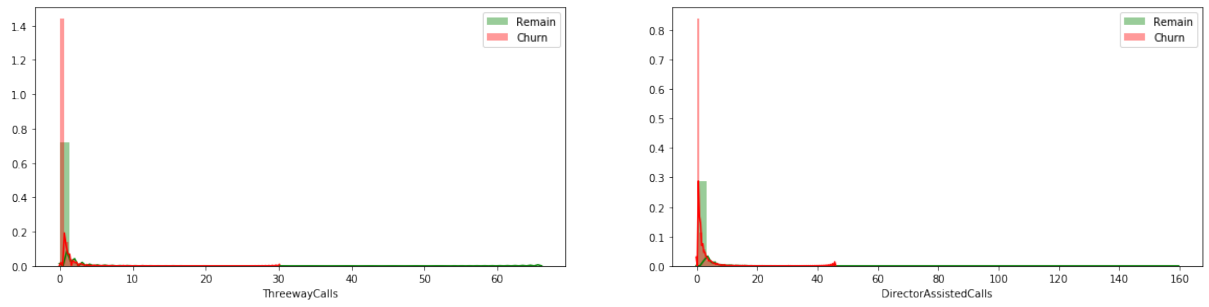

Figure 7 Highly zero-inflated features

Even though we cannot see a separation in a continuous and discrete distribution in our visualization does not mean that is not there. It might just not be visible to the naked eye, but even a slight separation can help in classifying.

For example, if we compare PercChangeMinutes with PercChangeMinutes log-transformed (on the absolute values of PercChangeMinutes), a separation of remainers and churners becomes more visible.

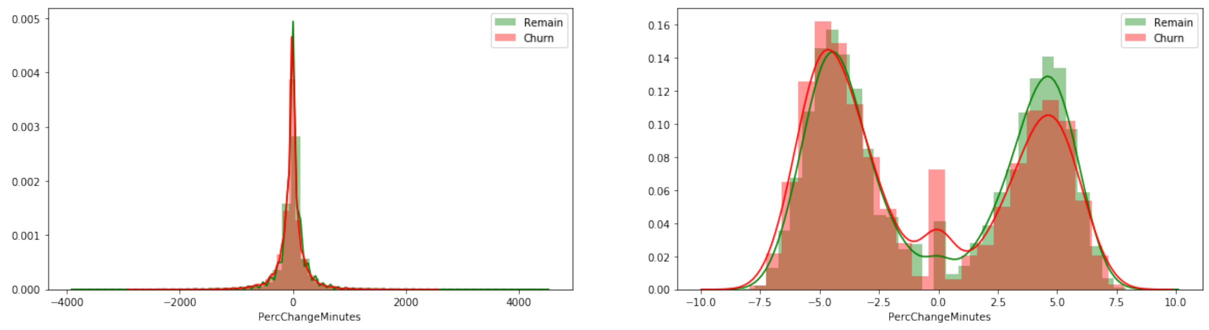

Figure 8 Separation made visible by log transformation

Note: This last transformation has not been made to the data but serves to illustrate that some of the differences in distribution of remain/churn are too small to make out in the histograms.

Algorithms

The following classification algorithms are proposed for possible solution models:

- Logistic Regression
- Decision Trees
    - Random Forest
    - Gradient boosted trees (XGBoost)
- Neural Network

Logistic Regression is relatively easy to fine tune and very fast. On the other hand, it can only learn linear hypothesis functions so it will have a hard time learning complex interactions. However, if the problem is simple enough to be solved by Logistic Regression there is no reason to spend effort and computation on more complex algorithms.

Decision Trees are easy to interpret as they output the features they make decision on. They are well suited for structured, tabular data like our dataset, and can handle large datasets well. Decision trees can learn non-linear interactions.
Decision trees are however prone to overfitting.

Looking at the analysis section of the project, overfitting is a concern with the large number of features and varying spreads across and within features.

Because of this, decision trees are usually not applied directly on a problem like ours, but as part of ensemble methods: We deploy two different ensemble methods, bagged trees in the form of Random Forest, and boosted trees in the form of XGB.

Random Forest combines the result of many decision trees, which have been learned on subsets of the dataset. These weak trees are aggregated and averaged into a strong learner by having the many small trees essentially vote on the classes to predict. The purpose of this is to combine many low biased, weak learners while reducing the overall variance. Random Forest are generally much more robust against overfitting, while keeping the advantages of pure decision trees, like suite for structured data and ability to learn complex interactions.

XGBoost is a boosting algorithm that can boost the performance of underlying decision trees. During boosting, the new weak learners are trained iteratively, with each learner correcting and improving on the predictions of the previous one. Because boosting is designed to reduce bias over variance, XGBoost is somewhat less robust against overfitting than Random Forest. On the other hand, it comes with a large array of hyperparameters to address potential overfitting.

Finally, XGBoost has an excellent reputation and track record for solving structured, tabular data. [3]

[3] https://www.kdnuggets.com/2015/12/harasymiv-lessons-kaggle-machine-learning.html

Neural networks model complex, non-linear interactions well. They can deal well with many input features like our dataset has them. They are very flexible and can be adapted extensively. And, due to their popularity, a wide selection of open source frameworks for implementation is available. We deploy a neural network classifier to have the option for deeper customization and fine tuning, should this be necessary.

Models based on all proposed algorithms will be trained and tested on the project dataset. In the evaluation section, the best performing model will be selected as the solution.

Benchmark

Each solution candidate model was benchmarked against a baseline model. For a baseline model, the data was split into churners and non-churners by a KNN-classifier with k=2. A KNN classifier is unlikely to capture complex interactions well compared to the candidate models but does give a good baseline for performance which the candidate models can improve on.

To keep benchmarking consistent, the KNN-classifier was run on the same training data as the solution models, after the data had been cleaned for obvious quality flaws (like missing values), but before the full pre-processing pipeline was built.

Without a data driven models for campaign selection available, customer service reps will often rely on simple heuristics or a schedule to determine which customers to target with a retention campaign. We modelled this in a second, naïve benchmark model which predicts churn for any customer within 3 months of their 1 or 2-year contract end dates.

The two benchmark models reported the following scores for our performance metric:

Benchmark	**Fbeta-score**
KNN-Classifier	0.1271
Naïve model	0.3159

Table 5 Benchmark performance on evaluation metric

The KNN classifier performs rather poorly with an F-score of ˜0.127. With no data preprocessing other than one-hot encoding, this is not unexpected.
The naive classifier performs much better than the KNN classifier with a F-score of ˜0.316. We kept both and refer to them in more detail in the model evaluation section. The naive classifier sets the benchmark to beat. It is also a more natural benchmark as it would show that a machine learning assisted campaign is more successful than a conventional, scheduled retention campaign.

Methodology

Pre-processing

Outlier removal

As examples 2 through 4 from the data visualization section show, a lot of columns are highly skewed to the right and show far outlying datapoints.

Extreme outliers

Extreme outliers were removed from the dataset. We identified extreme outliers by the Interquartile Range (IQR) of the *log-transformed* distributions of each column. We added 1.0 to the distribution before log-transforming, to safely handle zeroes in the data (done in one step using numpy log1p).

We then identified the number of points that were outside of 3x the IQR of this transformed distribution. Additionally, we plotted IQR and outliers in boxplots.

Example boxplots from the dataset are shown in Figure 9. Different outliers and their position relative to 3x IQR (the vertical lines dividing the boxplot from individually plotted data points) are visible.

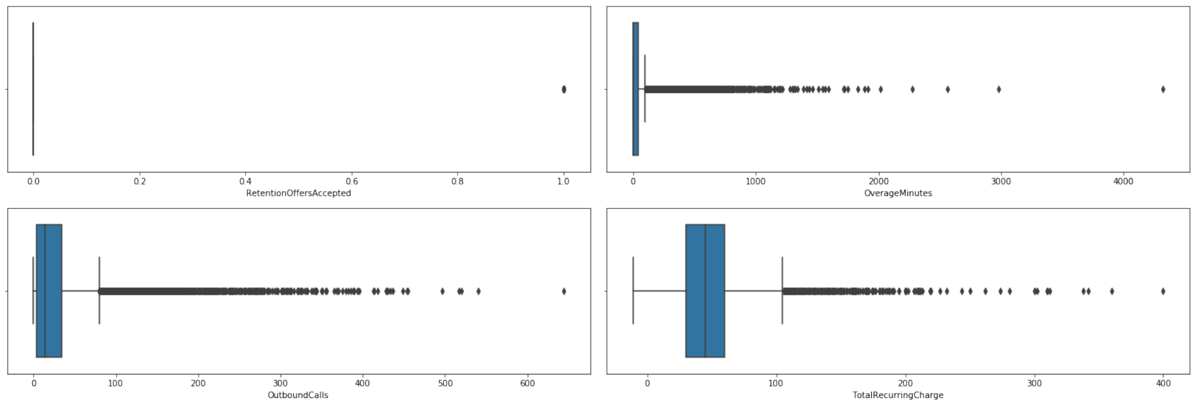

Figure 9 Boxplots showing different distributions of potential outliers

Where there are only a few extreme outliers (below 100), we removed the extremes. The cutoff points were identified based on the boxplots.

Feature engineering

Some of the columns with extreme outliers show the vast majority of records are either 0 or 1, and then a few outliers are greater than one. Examples of this are ReferralsMadeBySubscriber and RetentionCalls.

We clipped these columns at 1, essentially engineering a new binary feature from these nominally continuous columns, while removing outliers.

Log transformation

Removing the extreme outliers still leaves the high right-skew in the data. We applied log transformation on the highly skewed data to bring them closer to normality. As with identifying extreme outliers, log1p is used to safely handle zeroes in the data.

Below shows an example of OutboundCalls before (left) and after (right) log-transformation.

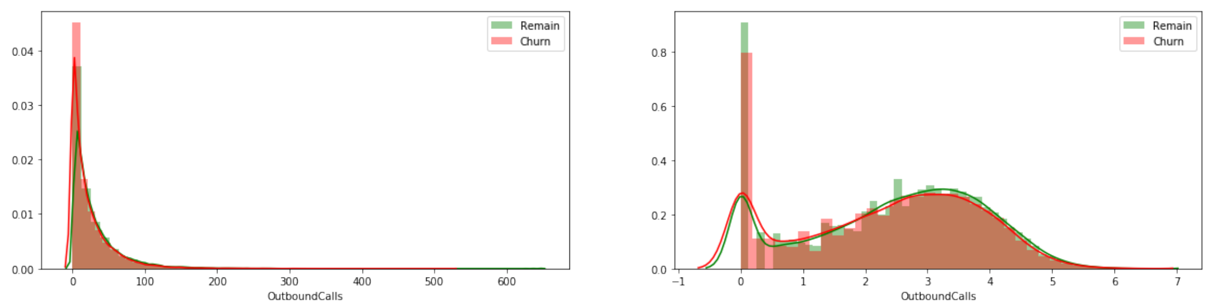

Figure 10 Log transformation revealing zero-inflation

The log transformation has reduced the long right tail of the outliers as expected. It also reveals the zero-inflated nature of the data.

Zero inflation

Some of the columns show an over-proportion of datapoint at zero. While such zero-inflated data is not uncommon, the literature seems to vary on how to handle it and is usually domain specific. Compare David I. Warton, Paul Allison, Chi Chang.

It is not certain if zero-inflation is a problem in our classification case. Methods for dealing with zero-inflation introduce their own error. A test classification run without all the zero-inflated columns did not yield any better results. For these reasons, zero-inflation was ultimately not handled in pre-processing.

Feature selection

During the first few iterations of the project, the classifiers showed signs of overfitting. Looking at the decision tree-based classifiers also showed that a lot of features were not considered in the classification.
Where the lack of importance of these features matches with our understanding of the data from domain knowledge, and where the visualization suggests poor impact on classification, features have been removed.

The following features were iteratively removed from the dataset based on these conditions:
AdjustmentsToCreditRating, CallForwardingCalls, ThreewayCalls, BlockedCalls, DirectorAssistedCalls, Handsets, CallWaitingCalls, ActiveSubs, UniqueSubs, IncomeGroup, Occupation, CreditRating, ReferralsMadeBySubscriber

One-hot encoding

All remaining categorical columns were transformed to binary columns per category value by applying one-hot encoding.

Re-sampling

As shown in the Analysis section, the data is imbalanced: we have far more remaining customers than churners. Before we can train our classifiers on the data, we considered re-sampling the data to have a balanced ratio of churners vs. remainers.

Since it is not obvious which sampling strategy is superior for our problem, the following strategies were tried on the data and classification algorithms:

- Over-sampling the data by generating new samples
- Under-sampling the data by randomly choosing a subset of the majority class (remain)
- A combination of over- and under-sampling
- No re-sampling at all

After a few iterations with different sampling strategies, **under-sampling** showed the best result across all tried classifiers.

Scaling

Some classifiers, such as Neural Networks, are very sensitive to different scales across features. The following scaling strategies (from scikit-learn) have been tried:

- StandardScaler
- Min-Max scaling
- RobustScaler

Of these, RobustScaler showed the best results. This can be explained by the remaining outliers and spread in the data, which RobustScaler is considered better at handling.

Summary

After pre-processing and feature selection, the final dataset used for training was reduced as follows compared to the initial dataset.

	Initial	**After pre-processing and feature selection**
Columns	56	24
Rows	51047	50695

Table 6 Dataset before and after pre-processing

Note: "After" column shows data *before* one-hot encoding was applied.

Implementation

KNN, Logistic Regression and Random Forest were implemented using the classifier libraries from scikit-learn. XGBoost was implemented with the xgb library, which provides an API compatible with scikit-learn.

Keras

A neural network classifier was built with Keras on a Tensorflow backend. Keras' sklearn API was used to make the model compatible with the other models in the project.

A model summary of the Keras classifier is included in Appendix B Keras model summary.

Making the Keras classifier work required a lot of manual fine tuning, as the classifier tended to always predict churn or always predict remain. The details are stated in the Refinement section. Even after fine tuning, the Keras model did not perform as expected. It only performed once global objectives were implemented.

Callback for early stopping based on Fbeta were tried for Keras, but ultimately were not successful. The resulting models predicted exclusively one class.

Global objectives

One approach implemented which improved the results of the Keras model somewhat was a custom loss function. Global objectives are an approach to optimize loss functions that approximate objectives such as precision, recall directly, instead of optimizing loss functions that relate to accuracy.

Elad ET. Eban, at al. show an implementation of directly optimizable global objectives in Scalable Learning of Non-Decomposable Objectives. The Tensorflow implementation of these objectives was adopted for this project and used as a loss function in Keras.

The loss functions from global_objectives had to be wrapped because the inputs of the loss functions did not receive the shape information when the Tensorflow graph was build. Since we deal with binary classification, we set the label shape to 1 and inferred the record shape. With these changes, the Keras model optimized directly on global objective loss functions.

With a combination of parameter tuning and global objectives, the Keras model no longer predicted one class exclusively across the whole test set.

Refinement

The project went through many iterations of refinement, with some of them starting over from the early data analysis steps.

Feature selection

At first, after some pre-processing and feature selection, none of the algorithms produced models that beat the naïve benchmark model. One reason was simply that too many features contributed little to a churn / remain decision. Figure 11 and Figure 12 show intermediate results on feature importance of RandomForest and XGBoost, respectively.

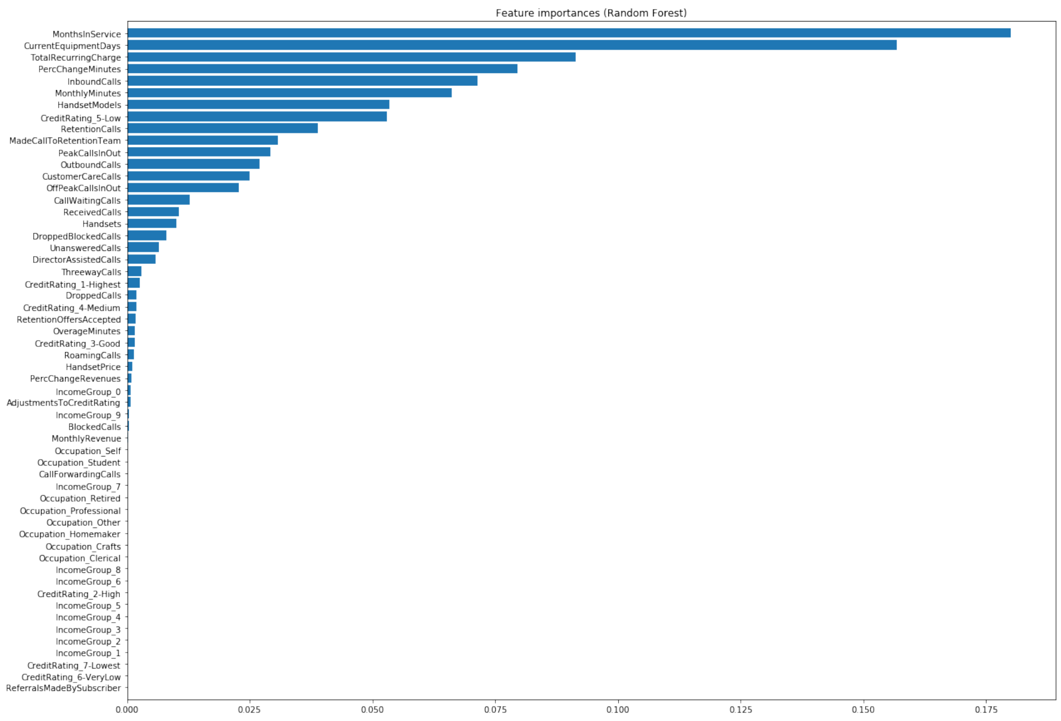

Figure 11 Random Forest feature importance (intermediate)

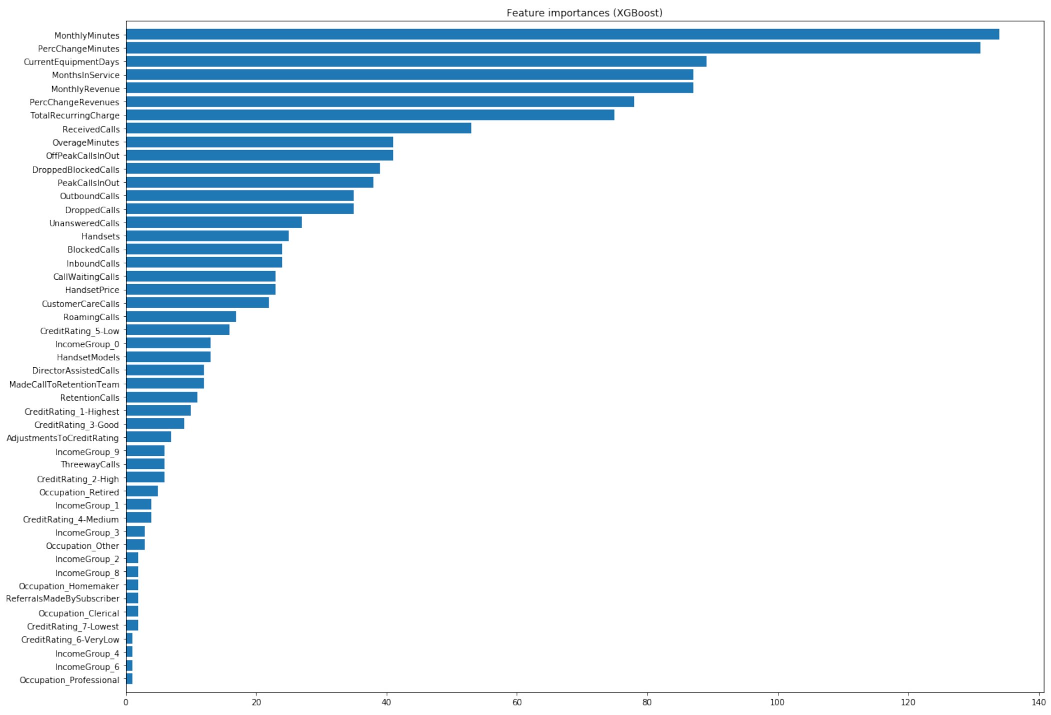

Figure 12 XGBoost feature importance (intermediate)

These intermediate results show that many features are considered unimportant by the algorithms or are not used at all. Based on these results, more aggressive feature selection was done which improved performance for the decision tree-based algorithms.

Hyperparameter tuning

The tuning process for the algorithms was deployed as follows:

- First pass with default settings to get a first impression at performance
- Based on that, definition of ranges for parameter search were defined
- Randomized search on 5-fold cross-validation was deployed for automatic tuning against the evaluation metric
- In further iterations, the ranges for parameter search were adjusted, and a new cross validation search was run

An exception to this process is the KerasClassifier: here, parameter tuning with randomized search proved infeasible. The search did not produce a model that had any predictive power.
Instead, model architecture and hyperparameters were adjusted manually based on intuition. The most impactful refinements to the Keras model were:

- Activation function ELU. This was the only activation function that produced a usable classification. All other activation functions tried resulted in a model which always predicted churn.
- L2-regularization with a rate of 0.001. Higher or lower (or no regularization) again made the model always predict churn or always predict remain.

Results

Model Evaluation

Parameters for the models were selected using random search over 5-fold cross validation, as described in section Refinement.

The final results of the parameter tuning for the solution candidate models from section Hyperparameter tuning are shown in Table 7.

Model	Parameter	Value
Logistic Regression	C	1.8727
	solver	saga
Random Forest	criterion	Entropy
	max_depth	3
	max_features	0.14
	min_samples_split	0.5
XGBoost	colsample_bytree	0.5
	eta	0.01
	gamma	10
	max_depth	5
	min_child_weight	1.48
	Subsample	0.74
Keras	activation	ELU
	dropout rate	0.4
	learning rate	0.0005
	L2 regularization rate	0.001

Table 7 Hyperparameters of potential solution models

A model summary of the Keras classifier is included in Appendix B Keras model summary.

Learning Curves

A look at the learning rates reveal an important difference between the solution models. The learning curves show the development of the evaluation metric **Fbeta** over training samples.

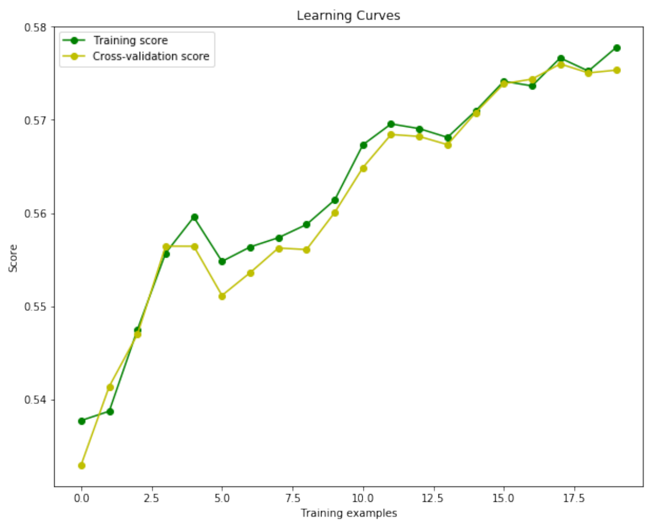

Figure 13 Learning curve: Logistic Regression

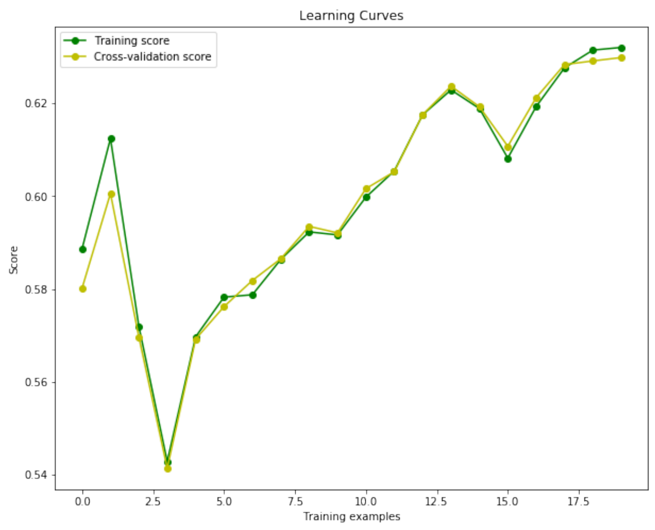

Figure 14 Learning curve: Random Forest

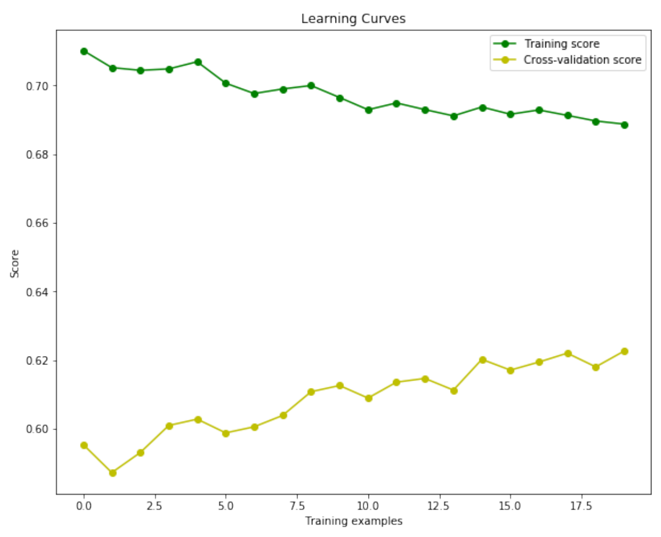

Figure 15 Learning curve: XGBoost

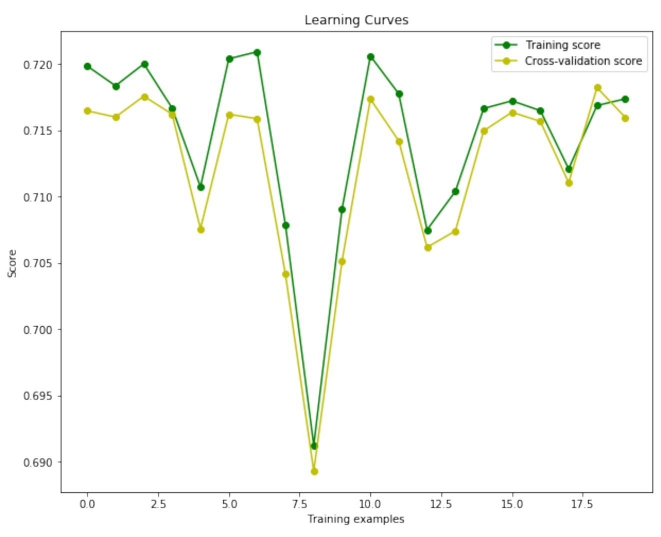

Figure 16 Learning curve: Keras

It looks like only XGBoost shows any kind of convergence of learning and cross-validation performance, if at high variance. We interpret these plots as follows:

Logistic Regression, Random Forest, and Keras seem to build models that are too simple. They appear to be learning very little generalizable information from the training set. Training and test set performance on the evaluation metric track each other after a few training samples almost exactly.

It is possible that there simply is no generalizable information to be learned from the data. The next section will explore the performance of the trained models on the test set, and judge which model qualifies for a solution.

Justification

Metrics report

For all implemented models, the following metrics have been collected from predictions done on the test set.

Classifier	Fbeta	Accuracy	TN	FP	FN	TP	Beta	Precision	Recall	F1
KNeighborsClassifier	0.1271	0.6855	8414	696	3318	334	1.25	0.0915	0.3243	0.1427
NaiveIntuitionClassifier	0.3159	0.6323	6965	2145	2548	1104	1.25	0.3023	0.3398	0.3200
LogisticRegression	0.4637	0.5950	5503	3519	1614	2038	1.25	0.5581	0.3667	0.4426
RandomForestClassifier	0.5035	0.5461	4394	4628	1125	2527	1.25	0.6919	0.3532	0.4677
XGBClassifier	0.5108	0.6120	5446	3576	1341	2311	1.25	0.6328	0.3926	0.4845
KerasClassifier	0.5194	0.3771	1387	7635	260	3392	1.25	0.9288	0.3076	0.4622

Table 8 Metrics report of models against test set

Some observations on the metrics:

- We can see that all our models beat the benchmark models in our primary evaluation metric Fbeta = 1.25.
- The Keras model has the highest Fbeta score, closely followed by XGboost, with both RandomForest and Logistic Regression trailing behind.
- The Keras model has the lowest number of false negatives. However, this comes at the cost of many false positives.
- XGBoost shows the highest recall and f1 scores.
- Of the solution candidates, XGBoost shows the highest accuracy

The summary metrics give a tendency: purely from the metric summary, it appears that the Keras and XGB classifiers performed the best.

A closer look however shows that out of the two, XGBoost is to be preferred: the Keras model over-predicts churners heavily. In fact, it only labels around 14% of the samples as remain. This can also be seen in the poor accuracy and recall of the Keras model.

We did however say that we can afford additional false positives, as long as we keep false negatives low. Which is what the Keras model is doing. The question is – how many can we afford? We answer this question in the next section.

Cost modelling

As part of the evaluation we created a simple model to calculate the cost of a retention campaign which is implemented based on the predictions of our different models.

Cost model terms:

TN	=	true negatives
FP	=	false positives
FN	=	false negatives
TP	=	true positives
α	=	cost of a retention campaign for a single customer, if successful
γ	=	cost of losing a customer
β	=	cost of contacting a single customer as part of a campaign
δ	=	success rate of a retention campaign

We can then define the total cost of the campaign as:

$$Cost = \beta * (FP + TP) + \alpha * FP + \alpha * \delta * TP + (1 - \delta) * \gamma * TP + \gamma * FN$$

To assign some plausible dollar value to a campaign based on the model predictions, we work with the following values:

α	=	\$50.00, we offer a \$50 rebate if the customer extends their contract
γ	=	\$500.00, cost of acquiring a new customer
β	=	\$2.00, cost of running a campaign, per contacted customer
δ	=	{10%, 30%, 50%}

We ran the cost model for different success rates of 10%, 30%, 50%. The results for the different classifiers are shown in Table 9.

Classifier	**Cost[0.1]**	**Cost[0.3]**	**Cost[0.5]**	**Delta**[4]
KNeighborsClassifier	\$ 1,847,830.00	\$ 1,817,770.00	\$ 1,787,710.00	-
NaiveIntuitionClassifier	\$ 1,890,068.00	\$ 1,790,708.00	\$ 1,691,348.00	-
LogisticRegression	\$ 1,921,354.00	\$ 1,737,934.00	\$ 1,554,514.00	\$ 52,774.00
RandomForestClassifier	\$ 1,957,995.00	\$ 1,730,565.00	\$ 1,503,135.00	\$ 60,143.00
XGBClassifier	\$ 1,912,579.00	\$ 1,704,589.00	\$ 1,496,599.00	\$ 86,119.00
KerasClassifier	\$ 2,077,164.00	\$ 1,771,884.00	\$ 1,466,604.00	\$ 18,824.00

Table 9 Cost simulation results

We can see that out of our candidate models, the XGB model gives the highest average cost savings over the best performing benchmark model. We further see that the Keras model yields the lowest savings over the benchmark model.

We thus select the XGB classifier as our solution model.

Caveats
This is a very simple cost model to assign some dollar value to the predictions made and would not be used for decisions in a real-world example. For a real-world model, α should be a function of the expected customer lifetime value (CLV) of each customer, and β will likely be a complex calculation that involves breaking down a lot of fixed costs in customer relationship management to the customer segment addressed. The success of the campaign should be measured on profitability of the campaign, also against CLV.

[4] Delta of mean simulated cost to lowest benchmark cost: How much is this classifier saving on average?

Summary

In isolation, the performance metrics do not look very impressive. Even the best model evaluated shows performance metrics such as accuracy, precision, recall, and F-score in the 0.4 to 0.6 range. Superficially, the results show good summary scores for the XGBoost model and the Keras model.

A closer look at a simulated financial impact of the models shows that the XGBoost model outperforms all other classifiers. The potential savings of **˜$86,000.00** are significant enough that we consider the project problem adequately solved by the XGBoost model.

We have to keep in mind that predicting customer churn is *hard*. The literature on churn in the telco industry measures predictive performance as lift: the difference in churn rate of a predicted sample of all customer, over the base churn rate of the whole customer population. In other words, how much more likely a model is to capture churning customers vs. random sampling from the customer population.

The following section, **Free form visualization**, expands on reasoning for selecting XGBoost as the solution model.

Conclusion

Free form visualization

In this conclusion, we analyze the precision-recall curve of the four models trained and tested on the data. A closer look at how the classifiers perform on precision-recall tells us a lot about how the classifiers compare, and about the problem in general.

Figure 17 and Table 10 show the precision-recall curve and the confusion matrix of our classification problem.

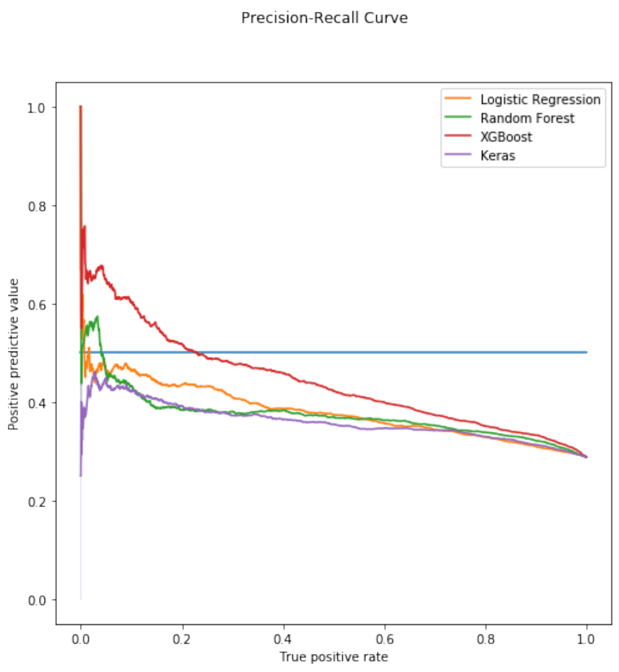

Figure 17 Precision-Recall Curve

	Predicted retention	Predicted churn
Loyal customer	true negative (TN)	false positive (FP)
Churning customer	false negative (FN)	true positive (TP)

Table 10 Confusion matrix

Precision and recall are calculated from the confusion matrix values as follows:

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

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

First, we consider what the horizontal (blue bar) in the precision-recall curve means. On a perfectly balanced dataset of a binary classification, this would give us the curve of random sampling. On average, for every additional true positive (and thus every reduction of false negatives), we also catch one false positive, and the precision term stays constant.

However, our problem statement is highly imbalanced. For our problem, a constant precision at 0.5 *would be great*! Consider that we can “afford” additional false positives, as long as we do not get additional false negatives. Targeting a loyal customer with a retention campaign is much less costly than losing a customer. Thus, a classifier that trades false negatives for false positives (the blue line) would be a fantastic solution for our problem.

Finally, the curve shows us much more distinctly how the best classifier, XGBoost, outperforms the others. Only XGBoost and RandomForest keep a precision of > 0.5 for any appreciable gain in the true positive rate (recall). As we decrease false negatives, XGBoost keeps the favorable ratio of true positives to false positives up much longer than RandomForest.
Once all lines have dipped under the blue line (and are now getting more than one false positive for every true positive), XGBoost only slowly approaches the other algorithm’s curves. The sweet spot between 0.4 and 0.6 recall is nicely visible in the graph: while all other classifier’s curves have converged, XGBoost still maintains distance and better precision.

Reflection

The project successfully delivered a classifier for churn prediction on a real-world customer dataset of telco customers. After considerable analysis and pre-processing of the input data, four different classifiers were trained on the data. The classifiers were tested on the hold-out data and compared against a benchmark model and each other.

Of the four classifiers, an XGBoost based model performed best, and was selected as the solution model. A simulation of customer retention campaigns showed average savings of ~$85,000.00 for the solution model compared to the benchmark model.

Lessons Learned

80% of machines learning is data pre-processing. This is a common saying, and it surely was true for this project. Easily 80% of the implementation effort for this project was spent on data analysis and preparation.

Computation requirements should not be underestimated. On comparatively straightforward, structured data such as the project dataset, computation time adds up as visualization, cross-validation, parameter search, and training rounds are being added. Even with state of the art single-machine hardware, computation became a bottleneck towards the end of the project.

The window for a **good bias-variance tradeoff** can be very small. Most of the models appear to have a bias which is too high: the models are too simple to capture important edge cases, and thus the models only perform marginally better than the benchmark. The solution algorithm, XGBoost, builds a more complex model, and captures more of the churners. However, this comes at the price of a higher variance, and the model appears to be slightly too complex, as shown by the learning curve. Still, it performs better, so it also appears to make a better bias-variance tradeoff.
Getting this sweet spot right was incredibly difficult, and many tuning iterations were needed to get the solution model to an acceptable performance.

Summary metrics can be misleading and must be interpreted in the context of the problem statement. None of the reported metrics by themselves told us enough about how the models truly compare. Only when interpreted on the problem statement and applied to a cost simulation did it become obvious how good or bad the different models are at solving the problem.

Finally, while the cell2cell dataset can still be used to practice machine learning on, a more up-to-date, quality dataset from the telco industry from the smartphone era would be more interesting to analyze.

Improvement

An interesting area of potential improvement are the global objectives proposed by Elad ET. Eban, at al. These were tried on the KerasClassifier during this project. Refer also to the Refinement section.
Ultimately, the neural net classifier did not show convincing performance. It did improve performance slightly though when given the global objective to maximize recall at a given precision.

Potential ways this novel approach could be explored further:

- Fine tuning of the target precision while maximizing recall. Making target precision a hyperparameter and tuning it could find a more appropriate target.
- Fine tuning of neural net hyperparameters with global objective loss functions. Much of the neural net hyperparameters were tuned with a conventional logloss minimizing function. It is well imaginable that the neural net would respond differently to the global objective loss function and require different hyperparameter tuning.
- A similar approach could be tried on another gradient optimizer, like XGBoost. Prince Grover suggests something along these lines. However, it is unclear if XGBoost provides the access to learning metrics that a global objective loss function will need the same way Tensorflow does. Likely this would conflict with XGBoost-native optimizations.

Appendix

A. References

InvestP
Customer Acquisition Vs.Retention Costs – Statistics And Trends
https://www.invespcro.com/blog/customer-acquisition-retention/

Cell2cell Dataset
Kaggle
https://www.kaggle.com/jpacse/datasets-for-churn-telecom

V. Umayaparvathi, K. Iyakutti
A Survey on Customer Churn Prediction in Telecom Industry: Datasets, Methods and Metrics
https://www.irjet.net/archives/V3/i4/IRJET-V3I4213.pdf

Neslin, Gupta, et al.
Defection Detection: Improving Predictive Accuracy of Customer Churn Models
https://www.semanticscholar.org/paper/Defection-Detection-%3A-Improving-Predictive-Accuracy-Neslin/0cebc1e74803e483033a709e8f50bb8832e6f3cc

Elad ET. Eban, at al.
Scalable Learning of Non-Decomposable Objectives
https://arxiv.org/abs/1608.04802

David I. Warton
Many zeros does not mean zero inflation: comparing the goodness-of-fit of parametric models to multivariate abundance data
https://onlinelibrary.wiley.com/doi/abs/10.1002/env.702

PAUL ALLISON
Do We Really Need Zero-Inflated Models?
https://statisticalhorizons.com/zero-inflated-models

Chi Chang
Classification Accuracy of Unsupervised Learning Methods with Discrete and Mixture Distributed Indicators: a Monte Carlo Simulation Study
https://ww2.amstat.org/meetings/jsm/2018/onlineprogram/AbstractDetails.cfm?abstractid=330911

Prince Grover
Custom Loss Functions for Gradient Boosting
https://towardsdatascience.com/custom-loss-functions-for-gradient-boosting-f79c1b40466d

B. Keras model summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_607 (Dense)            (None, 400)               9600      
_________________________________________________________________
activation_405 (Activation)  (None, 400)               0         
_________________________________________________________________
dropout_405 (Dropout)        (None, 400)               0         
_________________________________________________________________
dense_608 (Dense)            (None, 400)               160400    
_________________________________________________________________
activation_406 (Activation)  (None, 400)               0         
_________________________________________________________________
dropout_406 (Dropout)        (None, 400)               0         
_________________________________________________________________
dense_609 (Dense)            (None, 1)                 401       
=================================================================
Total params: 170,401
Trainable params: 170,401
```

```
Non-trainable params: 0
_________________________________________________________________
```

C. Global objectives

Code for global objective loss functions has been taken from Tensorflow research module https://github.com/tensorflow/models/tree/master/research/global_objectives

A copy of the module is provided with the project.