Sequential User Retention Modelling

HELDER MARTINS

Master in Machine Learning
Date: July 6, 2017
Supervisor: Hedvig Kjellström (KTH) and Sahar Asadi (Spotify)
Examiner: Patric Jensfelt
Swedish title: Detta är den svenska översättningen av titeln
School of Computer Science and Communication

Abstract

English abstract goes here.

Sammanfattning

Träutensilierna i ett tryckeri äro ingalunda en faktor där trevnadens ordningens och ekonomiens upprätthållande, och dock är det icke sällan som sorgliga erfarenheter göras ordningens och ekon och miens därmed upprätthållande. Träutensilierna i ett tryckeri äro ingalunda en oviktig faktor, för trevnadens ordningens och och dock är det icke sällan.

Contents

Contents iii

1 Introduction 1
- 1.1 Streaming services . . . 2
- 1.2 Spotify and the growing competition . . . 3
- 1.3 Churn prediction . . . 4
- 1.4 User retention . . . 4
- 1.5 Research Question . . . 4
- 1.6 Ethical and Social Aspects . . . 5
- 1.7 Outline . . . 5

2 Background 6
- 2.1 Definitions . . . 6
- 2.2 Time Series and Aggregated Data . . . 7
- 2.3 Predictor Models . . . 8
  - 2.3.1 Logistic Regression . . . 8
  - 2.3.2 Decision Trees . . . 9
  - 2.3.3 Random Forests . . . 11
  - 2.3.4 Recurrent Neural Networks . . . 12
  - 2.3.5 Long Short-Term Memory . . . 16
- 2.4 Dimensionality Reduction . . . 17
  - 2.4.1 Principal Component Analysis . . . 17

3 Related Work 19
- 3.1 Churn Prediction . . . 20
- 3.2 Sequential Modelling for User Behaviour Data . . . 22
- 3.3 Evaluating and Training Churn Prediction Models . . . 24

4 Dataset creation and exploration 26
- 4.1 Infrastructure and Data Sources . . . 26
- 4.2 Dataset creation . . . 26
  - 4.2.1 Sampling . . . 27
  - 4.2.2 Parsing . . . 28
  - 4.2.3 Feature Extraction . . . 29
  - 4.2.4 Data Pre-processing . . . 31
- 4.3 Data Exploration . . . 33
  - 4.3.1 User Distribution . . . 33

4.3.2 User Behavior through Time . . . 36
4.3.3 Feature Correlation to Churn . . . 36
4.3.4 Dimensionality Reduction . . . 37
4.4 Software and libraries . . . 37

5 Methodology **40**
5.1 Handling the Class Imbalance Problem . . . 40
5.2 Cross-Validation . . . 41
5.3 Hyperparameter search . . . 42
5.4 Model training . . . 43
5.4.1 The LSTM recurrent network . . . 43
5.4.2 Training Procedure . . . 44
5.5 Evaluation Metrics . . . 45
5.5.1 Confusion Matrix . . . 45
5.5.2 Classification Accuracy . . . 46
5.5.3 Precision, recall and other metrics . . . 46
5.5.4 Receiver Operating Characteristic . . . 47
5.5.5 Precision-Recall curves . . . 48
5.6 Statistical significance . . . 48
5.7 Software and Libraries . . . 49

6 Results and Discussion **50**
6.1 LSTM vs. baseline models . . . 50
6.2 Experimenting on different window sizes . . . 51
6.2.1 Observation Window . . . 52
6.2.2 Prediction Window . . . 53
6.3 Effect of Class Distribution . . . 53
6.4 Experimenting with Dimensionality Reduction . . . 53

7 Conclusions and Future Work **56**

Bibliography **57**

A Additional Results **62**
A.1 Precision-recall curves . . . 62
A.2 ROC curves . . . 62
A.3 Metrics tables . . . 62

Chapter 1

Introduction

This chapter will be dedicated to introduce the topic of this thesis project, the motivations on why it is relevant to both academia and to the industry, the current context where this project is being developed on, the hypothesis that shall be explored, the methodology used to verify it, and the contributions of this work.

The rise of streaming services has in recent years revolutionized the way customers get access to digital content. The traditional model of media ownership, even though it still represents a significant share of revenue, is continuously losing ground to a new right-of-access model where users get access to content ether by paying a monthly subscription or by being exposed to advertisement. The customer base of said services is steadily growing, and with it the amount data collected tracking how they interact with the provided digital media. This information is of high-value to any provider who expects improve the user experience, increase its total number of clients and also avoid losing the current ones to the competition.

One application of a data set based on historical user behaviour that has received attention from the industry is to predict the clients that are more likely to leave the service in the near future and anticipate it by performing a set of actions with the goal of avoiding it from happening. It is predicted that there is a set of features which are highly correlated with their desire of abandoning the application, and thus a model could be trained to leverage this information and classify users based on the probability of that event occurring. Such models are called *churn predictors*, and will be a central subject on this thesis project. Another interest will be to evaluate which set of features among the available ones actually correlates with churn and extract from it actionable insights that can be suggested to the provider as a form of improvement to their application, a task made harder by the extremely large number of attributes that is commonly captured.

Predicting whether a user is going to leave the service provider in the near future is a subject that has received some attention in the academia in the past few years **To do** add ref (1). Most of them share however a common trait regarding the way the data is represented before being used for model training: the data is split into *time windows* spanning different time ranges (usually one after the other), where an older time window used for model training and a time window closer to the present date to verify whether the observed user is going to churn or not. The common approach for dealing with the user data that is by nature sequential is by aggregating the feature values into a single representation for each user, normally through a mean function. Even though this approach simplifies the task of training the predictor models, latent temporal factors hidden in the sequence that could prove useful for increasing the performance of models are inherently lost in this process. One of the

hypothesis that is going to be explored in this project is if using the raw sequential data and also a predictor model that knows how to leverage this property, a significant performance gain can be achieved when comparing to a state-of-the-art baseline model.

The user behaviour data and the required computational resources were provided by *Spotify*, a well known music streaming service that joined this project in a tutoring partnership with the student and the university. Creating a suitable data set for training our proposed models is also part of the project, and it was performed by first exploring the data at our disposal and identifying the features that highly correlates with churn. While the available data is mostly related to the music domain, the methodology of this project and the conclusions reached can also be extended to other domains like video streaming with minor modifications.

1.1 Streaming services

The last decade has witnessed an overwhelming increase in popularity in all kinds of streaming services around the globe. Applications like Netflix and Spotify have changed the industry by offering a legal and affordable way of accessing content through a subscription-like contract instead of the classical pay-by-content commonly used by traditional media providers. This change in business model has proven to be successful by the sheer amount of customers that the most popular service providers have nowadays. Netflix for instance has reported to its shareholders a user base of 93 million worldwide at the end of 2016 [1], while Spotify recently reached the amazing feat of 50 millions paying subscribers, 20 millions of that only in one year [2]. Even though bureaucratic challenges like the dispute between the service providers and the labels regarding the value of licenses are still on debate, the adoption of this new model by consumers indicates no interest of going back to the old ways of getting access to digital content, like through physical or downloadable pay-by-content media.

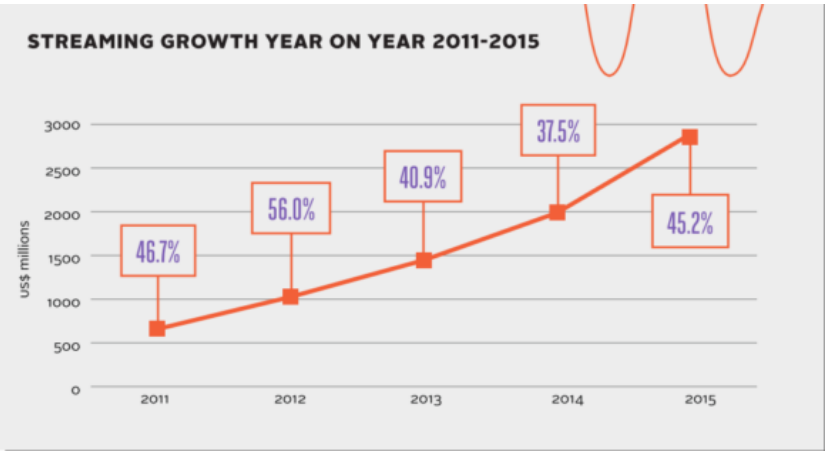

Figure 1.1: IFPI's streaming growth report

Music streaming follows this trend of growth on the latest economical reports. Over the five year period to 2015, the observed revenue from the industry grew four times to US $2.89 billion as can be seen in Figure 1.1 from the IFPI organisation [3]. The increase in the number of paying subscribers is also of notice, seeing an increase from 8 to 68 millions people over the same period. This new model also helped bring some countries where licensed music

market was losing ground to piracy like China and Mexico, and it corresponds to around 20% of the industry's revenue in the top five markets in the world.

1.2 Spotify and the growing competition

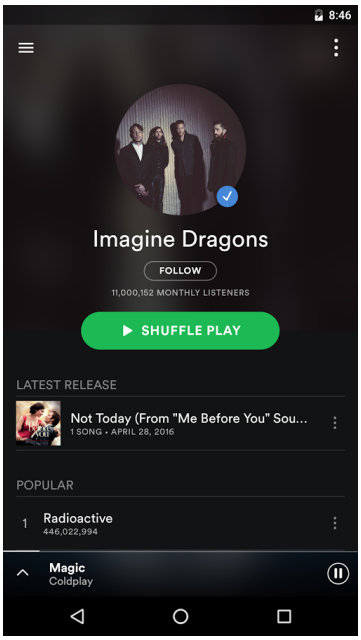

Figure 1.2: A screenshot of Spotify's Android app

Spotify is the top music streaming service globally in terms of number of users, totalling 100 millions of which 50 millions are Premium paid subscriptions [2]. The content is delivered through applications available on several different operating systems like Windows, macOS and Linux, as well as devices like iOS and Android tablets and smartphones, an example is shown in Figure 1.2. Users may access over 30 millions songs available in its catalog through two methods: a free membership where the client is exposed to advertisement, and a Premium paid subscription with additional features like offline downloads, increased music quality and unlimited number of song skips. Spotify also is well known for its music recommendation subsystems like Discover Weekly and Daily Mix that provides playlists tailored to each user based on their listening history. The service is available in most of Europe, as well as in Australia, New Zealand, most of the Americas and several Asian countries.

The promise of growth in the area however has attracted well known brands to the dispute like Apple and Google, and their growth, while still not up to par with the top players, has been steady and may be significant in the following years [3]. The market is becoming saturated with competition, and any contender to the top position in the music streaming area must strive to keep its current user base at all costs. The loss of a customer, be it for the competition or not, is called in the industry as *churn* and is commonly seen as a metric that should be minimized in a customer relations framework (CRM) of a service provider. For that goal, an important step is to identify prematurely the candidate churning customers and to leverage actions that may avoid them from leaving, and this is commonly achieved by making use of the wealth of historical information that digital providers have about their users.

1.3 Churn prediction

Churn prediction, the task of identifying the most probable churning customers of a service provider, is an established topic in the literature and has seen a considerable amount of research specially on the mobile telecom industry [4] [5] [6]. Several classical techniques for predicting churn like decision trees and logistic regression have been used on recent work [7], however most of them make use of user behaviour data at a single point in time, normally the most recent one when the dataset was created. The user is commonly represented by summarizing or aggregating his behavioural feature values over a fixed time window, thus losing any latent sequential information that might be contained in the data set. This is a simplification that eases the burden of training a predictor model since the customer data is commonly high-dimensional and difficult to work with, nonetheless latent attributes, when used properly, have been shown to improve significantly a predictor's performance for industries like mobile telecom [8].

Intuitively one can think that latent factors hidden in the temporal axis of the user behaviour data could yield a better prediction accuracy when comparing to a model which leverage a single and static point in time. For instance, a user that is gradually reducing its consumption of the service over time can be easily thought of a prospect churner. While this intuition is trivial to come up with, we are also interested in learning strong correlations between temporal aspects of the data and the churn rate which are not as clear to the eye. Our hypothesis is that making use of the properties hidden on user behaviour data over time, a churn rate predictor could improve its accuracy greatly when compared to methods which are static on time. **To do** Add this paragraph somewhere else maybe? It seems strange here (2)

1.4 User retention

A churn rate predictor which can identify possible churners accurately is just a trigger on the user retention process that can span several different stages. One of these stages is to identify what actions can be taken by the service provider as to avoid the user abandoning the service. An important problem to solve beforehand is to identify which features of the data have a strong statistical correlation with the churn rate. With that information, the providers could for example set up automated actions as to influence the value on these features. Learning which are the most important features is also of relevance for the task of choosing what data to use for training the predictor models, since commonly service providers have a vast amount of data about the user but only a fraction of that is of importance for predicting churn: training models with a full dataset will commonly introduce error on the system and take a large amount of computing resources to train.

1.5 Research Question

The goal of this project will be to research different models for churn prediction that can make use of the temporal aspect of user behaviour data at its fullest. Our hypothesis is that latent factors hidden on usage patterns may increase the accuracy of state-of-art predictors which considers only the state of the data at a single point in time. We shall experiment on different models that are known to perform well on time series data and compare their accuracy using different evaluation methods. Detecting possible churners, while important, does not improve user retention without an associated action performed by the service provider.

To derive insights on what can done to improve this metric, a deep data exploration and feature analysis is also a goal of this project. Features shall be correlated to their influence on the churn rate, and feature selection techniques shall be researched and implemented in an attempt to reduce bias on the churn classifiers. **To do** Improve the research question section, add some bullet points (3)

1.6 Ethical and Social Aspects

This thesis... **To do** Write ethics section (4)

1.7 Outline

This thesis is organized as follows... **To do** Write the Outline (5)

Chapter 2

Background

For a better understanding of this degree project, some concepts are required to be properly introduced. Some of them like the ones present in the Definitions section are mainly application-specific, while others pertains to our choice of methodology and evaluation metrics.

2.1 Definitions

Every service provider has a different definition of what churn is. Some of them use the explicit cancellation of a contract as a indicator of user abandonment, which has the advantage of being straightforward to calculate and having a clear correlation to revenue metrics, while others prefer to explore the user activity instead, which has the benefit of better leveraging his satisfaction level and working with the assumption that a satisfied customer has a lower chance of leaving the service for the competition.

In this degree project, the *user activity* will be used as indication of churn. Even though Spotify has a paid Premium version that could be used as a source of churn labels, a considerable fraction of Free users who generates revenue through advertisement would be excluded from this study. Thus their engagement will be measured instead disregarding for labeling purposes whether they are paying customers or not, however they may be split into different clusters for visualization purposes.

To build up the exact definition of what churn is for Spotify, some concepts need to firstly be introduced, which follows.

Definition 1 *The action of a user listening to a song through any of the provider's supported platforms is called a* stream*, also sometimes called* playback*. A stream is initiated when the user starts listening to a song (either by explicitly selecting it through the application or passively by being the next song in a list), and the stream ends when the user listen to a song until it finishes, when he actively skips it by the press of a button, or other technical reasons.*

The playback is the basic measurement of activity of the user on the service. It incorporates the user music streaming behavior, and it is also source of key metrics for the company's financial health. Since Spotify is mainly interested in evaluating the user based on its main feature of content streaming, other user interactions (eg. following another user or artist) are out of the scope for this project.

Definition 2 *A user is considered to be* active *during a time period if he possesses one or more streams of at least 30 seconds in length, and* inactive *otherwise.*

The definition of an active user will be important during the sampling phase: there is no interesting information that can be gathered from customers that churned a long time before the first day of behaviour observation. More details about the sampling process will be thoroughly explored in subsection 4.2.1.

Before we arrive in the definition of churn, a brief introduction to time windows will be given to understand further concepts in this section. These definitions will be thoroughly explored on section 2.2.

Definition 3 *The* observation window *is the time span used to observe user behavior and to train the predictor models. It is always followed by the* prediction window *which is exclusively used to predict whether a user in the observation window is going to churn in the future or not. The last few time steps of the observation period is called* activity window*, and is mainly used for fetching user samples.*

Definition 4 *A user is defined to have* churned *if he is active during the activity window and inactive during the prediction window.*

Note that as mentioned before this definition is not concerned whether the user is a paying customer or not, but only if he is actually making use of the service. For example, a user may be subscribed to a Premium paid account and still be considered to have churned if no activity was registered in the prediction window. However, it should be noted that empirical observations by the service provider suggest that there is a significant behavioral difference between the two classes of users that will be taken into consideration when the experiments are performed.

2.2 Time Series and Aggregated Data

In this section we introduce some concepts that explores the sequential nature of user data. Consider $\mathbf{x}_{nt}$ as the D-dimensional vector representing the features of user $n = 1, ..., N$ at time step $t = t_0, ..., t_{\alpha+\beta}$, with t being an equally spaced and discrete time interval. Now consider that the intervals are split into two sequential and non-overlapping time windows: an *observation window* where $t = t_0, ..., t_\alpha$ which is going to be used for observing user behavior, and a *prediction window* where $t = t_{\alpha+1}, ..., t_{\alpha+\beta}$ used solely for predicting whether the user is going to churn in the future or not, with α and β being the size of each of the respective windows. A third *activity window* where $t = t_{\alpha-\gamma}, ..., t_\alpha$ overlaps with the last γ time steps of the observation window and it is used for sampling active users. The time windows can be better seen in Figure 2.1.

Being $\mathbf{x}_n$ a sampled user who is active in the activity window, y_n is the variable specifying if customer n has churned between time steps $t_{\alpha+1}$ and $t_{\alpha+\beta}$ of the prediction window, such as

$$y_n = \begin{cases} 0 \text{ if user } n \text{ is active between } t_{\alpha+1} \text{ and } t_{\alpha+\beta} \\ 1 \text{ otherwise} \end{cases} \tag{2.1}$$

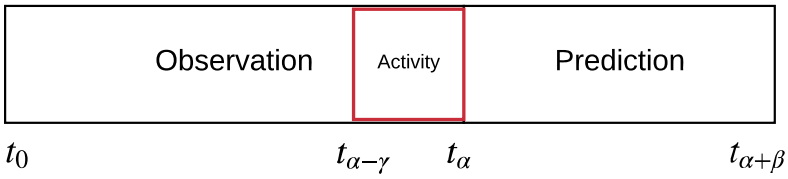

Figure 2.1: The time windows

which assigns one label per customer in our dataset. This definition follows business metrics originated from the service provider, where the counting of active users over a predetermined time span is one of the key metrics indicating business health. For instance, choosing a β sized at the number of time periods over 30 days would yield whether the customer will be a monthly active user the following month after being observed for α time steps.

The literature on churn prediction commonly describes the user features as a single point over the whole observation period, ignoring the time component of the data by transforming it through an aggregation function like the mean. As to train our non-sequential predictor models, we need a similar representation of the data which summarizes all information that we posses of a user over time into a single D-dimensional vector. Following the works by [8], we define a *single period training data (SPTD)* as the dataset consisting of one observation per customer who is active during the sampling period. In this approach, the dataset is

$$\text{SPTD}_{t_0 \to t_\alpha} = [\mathbf{X}_{t_0 \to t_\alpha}, \mathbf{y}] \tag{2.2}$$

where $\mathbf{X}_{t_0 \to t_\alpha}$ is a $N \times D$ matrix consisting of the constructed features $f(\mathbf{X})$, $f()$ is a feature aggregation function that maps all α user vectors over the observation window to a single one with the same number of D features, and $\mathbf{X}$ as the $N \times \alpha \times D$ matrix containing information about all users over the whole observation period. $\mathbf{y}$ is a $1 \times N$ column vector containing churn labels for all samples.

While the simple format of SPTD can be used for training most types of predictor models, it is unsuitable when the goal is to leverage the properties of the time component of the data since it is destroyed after the aggregation function. Thus, we define a *multiple period training data (MPTD)* as

$$\text{MPTD} = \begin{bmatrix} \mathbf{X}_{t_0} & \\ \mathbf{X}_{t_1} & \text{y} \\ \ldots & \\ \mathbf{X}_{t_\alpha} & \end{bmatrix} = [\mathbf{X}, \mathbf{y}] \tag{2.3}$$

which contains a single representation of a user at each time step during the observation period.

2.3 Predictor Models

2.3.1 Logistic Regression

Logistic regression is a linear regression model used to predict the probability of occurrence of a categorical variable, like the churning and non-churning labels. First developed by

mathematician David Cox [9], this technique has been used in wide range of domains like medical fields, marketing and economics **To do** add refs (6), and is as of today one of the most used models for predicting the churn rate of customers on the telecom industry [7]. Despite its name, logistic regression is used mostly as a classification algorithm, since its output is a probability score that together with a threshold can map the input to one of the supplied labels (churn and no-churn, for example).

The premise of logistic regression is that the output variable y can be modeled as a linear probability function dependent on a set of input feature values $\mathbf{x} = [x_1, x_2, ..., x_n]$ through a set of equations that follows:

$$p(y = 1|\mathbf{x}) = f(t) \tag{2.4}$$

$$f(t) = \frac{1}{(1 + e^{-t})} \tag{2.5}$$

$$t = w_0 + w_1 x_1 + ... + w_n x_n = \mathbf{w}^T \mathbf{x} \tag{2.6}$$

where y is the true output variable that can take the values 0 or 1 in a binary classification problem. t is a linear combination of the input $\boldsymbol{x}$ with a trainable weight vector $\mathbf{w}$. $f(t)$ is the logistic function that gives name to the model. This function has the property of "squashing" any real input to a value between 0 and 1, and thus it can be interpreted as a probability. **To do** Add logistic function (7)

The likelihood of the logistic regression function can be written as follows:

$$p(\mathbf{y}|\mathbf{w}) = \prod_{i=1}^{n} p(y_i = 1|\mathbf{x})^{y_i} (1 - p(y_i = 1|\mathbf{x}))^{1-y_i} \tag{2.7}$$

An error function can be usually be defined by taking the negative logarithm of the likelihood function, which follows:

$$E(\mathbf{w}) = -\ln p(\mathbf{y}|\mathbf{w}) = -\sum_{i=0}^{n} y_i \ln p(y_i = 1|\mathbf{w}) + (1 - y_i) \ln(1 - p(y_i = 1|\mathbf{w})) \tag{2.8}$$

Thus, we can pose logistic regression as an optimization problem where the goal is to minimize the error function $E(\mathbf{w})$

$$\min_{w} E(\mathbf{w}) + R(\mathbf{w}) \tag{2.9}$$

where $R(\mathbf{w})$ is a regularization term, commonly either L1 or L2 norms.

2.3.2 Decision Trees

Decision trees are a class of learning algorithms where the prediction is accomplished by dividing the input space into a series of cuboid regions and then assigning a label (for example "retained" and "churned" for a classification problem) into each one of them. Although simple in nature, it is a widely used method in the in academia, also present in several different papers on churn prediction [10][4] [11] [6]. While there are several different frameworks for implementing a decision tree, this project will focus on the *classification and regression trees (CART)* popularized by Breiman *et al.* [12]. Figure 2.2 shows an example a decision tree splitting a 2-dimensional input space.

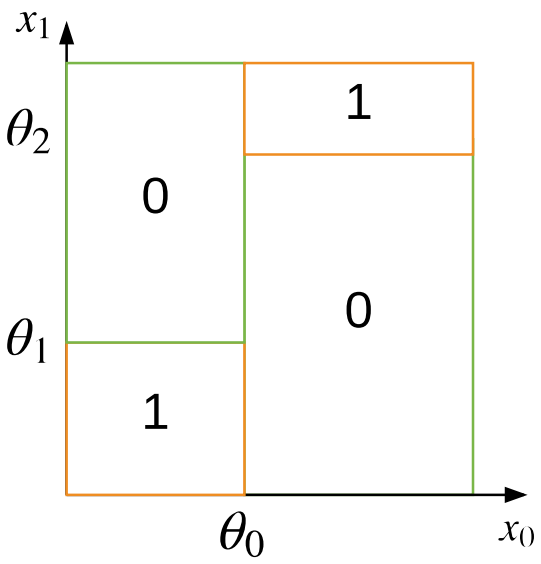

Figure 2.2: Visualization of a 2-dimensional input space being partitioned into 4 regions by a decision tree with its assigned binary labels in each region

In this example, the first partition divides the input space into two regions by verifying whether the value for attribute x_0 is smaller or greater than θ_0, being θ_0 a parameter learned by the model during training. These two regions are independent after splitting, and thus can be further divided efficiently in parallel. For instance, the region $x_0 <= \theta_0$ is partitioned by comparing if the value of x_1 is smaller or greater than θ_1, and since there are no more partitions in this subregion the labels can be assigned. The process is similarly repeated for the region $x_0 > \theta_0$, where θ_2 was chosen as the splitting point for attribute x_1.

After the model is trained, it can be easily visualized as a binary tree where each partition corresponds to a node in the tree, and where the leaves contains the identifying labels, which can be seen in Figure 2.3. Whenever a new input vector **x** is fed to the model, traversing the tree starting from its root will yield a class label in one of its leaves by sequentially navigating through the set of if-else clauses contained at each one of the nodes.

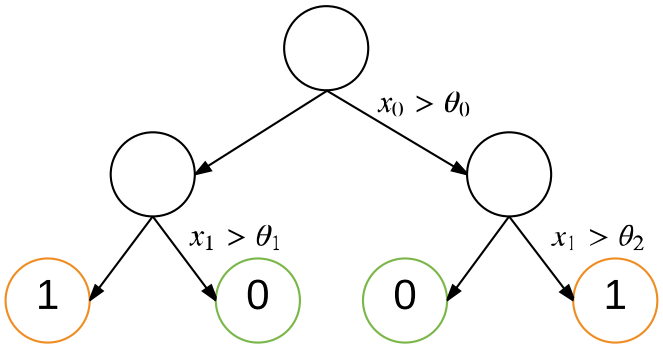

Figure 2.3: Resulting binary tree corresponding to the partitioning performed in Figure 2.2

The learning process of a decision tree corresponds to constructing a binary tree that minimizes the misclassification rate of its training data. That includes finding the optimal

number of splits which will define the structure of the tree, and also the values for the splitting coefficients θ_i. To understand how such a tree can be learned, consider a classification problem where $\mathbf{x}$ is a D-dimensional vector and y its corresponding class label. The training data is composed of N user vectors forming the $N \times D$ matrix $\mathbf{X} = \{\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_N, \}$. A tree can be grown by starting at its root and adding nodes sequentially, greedily selecting an attribute and a value to partition the data that minimizes a predefined error function $H(\mathbf{X})$, which can be performed through an exhaustive search. The set of leaf nodes indexed by τ represent regions $\mathcal{R}_\tau$ where the prediction labels are assigned.

When decision trees are used for classification, a popular cost function is the *Gini coefficient*, a metric commonly used to measure the statistical dispersion of a population. Let N_τ be the total number of data points assigned to region $\mathcal{R}_\tau$, the proportion $p_{\tau k}$ of data points in region $\mathcal{R}_\tau$ assigned to class $k = 0, ..., K-1$ such as

$$p_{\tau k} = \frac{1}{N_\tau} \sum_{\mathbf{x}_n \in \mathcal{R}_\tau} \mathcal{I}(y_n = k) \tag{2.10}$$

where $\mathcal{I}$ is the indicator function. The Gini coefficient can be defined as

$$H(\mathbf{X}) = \sum_{k=0}^{K-1} p_{\tau k}(1 - p_{\tau k}) \tag{2.11}$$

which is minimized whether $p_{\tau k} = 0$ or $p_{\tau k} = 1$. The Gini coefficient favors grouping high proportions of data points that belong to a single class into each region. The splits are chosen to be the ones that minimize $H(\mathbf{X})$ at each node of the tree, which is why decision trees are considered "greedy" algorithms. The tree can be grown until the reduction of the associated error falls below a predetermined threshold. Empirical observations have shown that pruning back some nodes of the tree after the expansion has finished can improve the performance of the predictor.

The decision tree algorithm posses a key property which is one of the reasons why it is so popular: the model can be easily interpreted. The ability to visualize the decision process of the predictor model as a binary tree linked together by a simple set of rules is of great value, and is sometimes deemed crucial in fields like medical science for example. However its largest drawback lies in the fact that decision trees are commonly tightly fitted to the data which they were trained on, and small changes in the input may yield radically different models. This problem, which is known in machine learning as high *variance*, is one the reasons for the creation of the ensemble method in the following section.

2.3.3 Random Forests

Random forest is an algorithm mainly used for classification and regression which utilizes an ensemble method for prediction: by training several different and independent classifiers, the predicted class or value can be interpreted as the resulting majority vote from all learners, through a mode or mean function for classification and regression, respectively.

Popularized by Breiman [13], random forests have been used in several different domains, specially for the task of predicting user churn [14] [15]. The main principle behind the method is that a group of "weak" learners can achieve a better performance when compared to a single "strong" learner, which can also be interpreted as an implementation of the "wisdow of the crowd".

In this context, a "weak" learner is a decision tree trained with a subset of the samples and features of the original data. No single tree has access to all input variables, however they predict by casting votes on a class, and the final prediction is the class label which has the majority of the votes.

Each one of the trees is grown as follows:

1. For a dataset of size N, sample N examples from the original dataset with replacement, also known as *bootstraping*.
2. For a dataset containing M features, select $m << M$ which are the number of features selected at random for building each tree.
3. Grow every tree to its maximum extent, without any pruning.

Suppose that we generate M bootstrap data sets and use each one to train a decision tree $y_m(\mathbf{x})$ where $m = 1, ..., M$. For a classification problem, $y_m(\mathbf{x})$ outputs the most likely class label k, that is the highest $p(k|\mathbf{x})$, for every $k = 1, ..., K$. The final prediction of the ensemble can be defined as the majority vote of every classifier which is given by

$$y_{RF}(\mathbf{x}) = \arg\max_{k \in K} \{y_m(\mathbf{x})\} \tag{2.12}$$

2.3.4 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a class of artificial neural networks well suited for dealing with data which is sequential in nature. Traditional feed-forward neural networks assume that all input data points are independent of each other, however this can lead to spurious results if this assumption does not hold. For instance, predicting the next word in a text document is tightly correlated with the words that appeared before it.

Unlike feed-forward neural networks, RNNs allows connections between hidden units, more specifically between the units that are contiguous in time. Updates in a hidden unit in a time step depends on the values in the hidden unit immediately before, and so on until the start of the sequence. While this seems to be a trivial modification over the classical multilayer perceptron, an important property is obtained in the process: RNNs can in theory map the entire history of previous inputs to each output.

The memory structure that holds this historical data, hereby called *state*, is the main engine that gives RNNs its predictive power. Consider $\mathbf{h}_t$ and $\mathbf{x}_t$ as the values of the state and input at time step t, respectively. The value of state $\mathbf{h}_t$ depends both on the current input $\mathbf{x}_t$ and also the previous state $\mathbf{h}_{t-1}$ as follows:

$$\mathbf{h}_t = \sigma(\mathbf{W}^{xh}\mathbf{x}_t + \mathbf{W}^{hh}\mathbf{h}_{t-1} + \mathbf{b}_h) \tag{2.13}$$

where $\mathbf{W}^{xh}$ is the weight matrix connecting the input to the hidden units, $\mathbf{W}^{hh}$ the weights connecting states time steps t and $t-1$, b_h the bias vector for hidden unit h, and $\sigma()$ the activation function, commonly the sigmoid or the hyperbolic tangent. Values for state h_{t-1} can be calculated in a similar fashion, until the initial state h_0 which is only dependent on the input. This recursive dependence means that an RNN can be "unrolled" through time as to better visualize the interactions between units of the network, which can be seen in Figure 2.4.

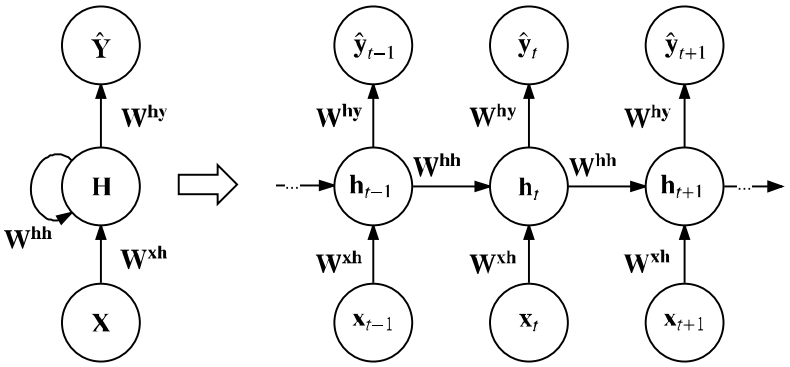

Figure 2.4: A recurrent neural network unfolded through time

In a classification problem, the predicted output $\hat{\mathbf{y}}_t$ is a K-dimensional vector with the normalized probabilities of input $\mathbf{x}_t$ belonging to the classes $k = 1, ..., K$, which can be calculated through the following equation:

$$\hat{\mathbf{y}}_t = \phi(\mathbf{W}^{hy}\mathbf{h}_t + b_y) \tag{2.14}$$

where $\mathbf{W}^{hy}$ are the weights connecting the hidden layer to the output, $\phi()$ the softmax function that normalizes its input into probability values, and b_y the bias vector for the output layer.

Loss function

To calculate how effective our prediction is comparing to true labels of the training data, a loss function $\mathcal{L}()$ that computes the distance between true values and the prediction is needed. A common function for a discrete classification problem is *cross entropy* which calculates the amount of information needed to distinguish one distribution from the other, here the predictions $\hat{\mathbf{y}}$ and the true labels $\mathbf{y}$. The equation follows:

$$\mathcal{L}(\hat{\mathbf{Y}}, \mathbf{Y}) = -\sum_k \mathbf{y}_k \log \hat{\mathbf{y}}_k \tag{2.15}$$

Initializers

Any neural network requires a method to initialize its weight matrices $\mathbf{W}$. A common practice is set it to small random numbers centered around zero so the weights can independently learn patterns from the data, which is a better approach when compared to naively setting all values to zero.

Glorot and Bengio[16] proposed a method to initialize the weights which has widely adopted in academia, and it goes by the name of *Xavier initialization*. In this method, the values are sampled from a uniform distribution such as

$$\mathbf{W} = U[-\frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}, \frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}] \tag{2.16}$$

where $n_{\text{in}}, n_{\text{out}}$ is the size of input and output layers, respectively.

Another method to initialize the weights was proposed by Saxe *et al.*[17] by choosing $\mathbf{W}$ to be a random orthogonal matrix such as

$$\mathbf{W}^T\mathbf{W} = \mathbf{I} \tag{2.17}$$

where $\mathbf{I}$ is the identity matrix. This is called *orthogonal initialization*, and it is a commonly used method for initializing the weights of the recurrent state in a recurrent neural network.

Regularizers

A machine learning model is judged to be of good quality if it can generalize well to unseen data. When a model shows good performance during training but fails to achieve the same using test data, the model may have fit all its parameters perfectly to the training set instead of finding a function that explains the data well, a problem which is called *overfitting*.

A common approach to avoid this complication is to add a term in the loss function that penalizes large values of parameters, which is called *weight decay*. This method favors simplicity by avoiding models that overly complex, which can be considered an application of Occam's razor which states that "among competing hypotheses, the one with the fewest assumptions should be selected". A L_2-norm is a regularization that penalizes the parameters by adding the square of the magnitude of the weights to the loss function that we aim to minimize[18], as follows:

$$\mathcal{L}_r(\hat{\mathbf{Y}}, \mathbf{Y}) = \mathcal{L}(\hat{\mathbf{Y}}, \mathbf{Y}) + \lambda \sum_i |w_i|^2 \tag{2.18}$$

where i iterates over all available parameters and λ controls the fraction of the regularization term that will be added to the loss function.

Another common approach is *dropout*[19], which randomly drop hidden units along their connections during training. This technique acts as a regularization method by removing the units with a probability p, avoiding the weights of co-adapting too much. Recurrent neural networks however requires a special implementation of dropout as for it be effective, a topic fully explored by Zaremba and Sutskever[20].

Optimizers

The learning process in an RNN involves finding the set of parameters $\mathbf{W}$ that minimizes the loss function $\mathcal{L}()$. This is accomplished by an *optimizer*, an algorithm that iteratively applies small updates to the weights in the direction where loss function is minimal. A classic optimization algorithm is *gradient descent*, which updates the parameters by computing the gradient of the cost function w.r.t the weights for the entire dataset and moving it in the opposite direction, such as

$$\mathbf{W} = \mathbf{W} - \eta \nabla_W \mathcal{L}(\mathbf{W}) \tag{2.19}$$

where η is the learning rate that controls how much of the gradient will be applied to the weights and $\nabla_W \mathcal{L}$ the gradients. Reading the whole dataset to apply one update to the weights is commonly not feasible due to its computing resources requirements. A more popular approach is to apply updates by looking at a subset of the training samples instead, a process called *mini-batch gradient descent*.

$$\mathbf{W} = \mathbf{W} - \eta \nabla_W \mathcal{L}(\mathbf{W}; \mathbf{x}^{(i:i+n)}, \mathbf{y}^{(i:i+n)}) \tag{2.20}$$

While mini-batch gradient descent is suitable when the dataset is too large to fit into the memory of a single machine, it has its own set of challenges to solve. First, choosing a proper value for the learning rate might be difficult: low values slows down greatly the convergence process, while large values makes the loss function fluctuate around the local minimum and sometimes even diverging from it. Also, a single learning rate regulates the updates of all parameters, not taking into consideration how sparse the data is and how frequent the features are. It is intuitive to think that weights applied to features that rarely ever are different than zero could have a larger update than the ones that are present in every sample.

To address these problems, new optimizers have been recently introduced. *Adagrad* [21] is a gradient-based optimization algorithm that adapts the learning rate dynamically based on the frequency of the parameters. Instead of having a single global learning rate, Adagrad instead has a different learning rate for every parameter of the model, making it suitable for datasets that are sparse in nature. Adagrad updates the learning rate for every weight w_i at time step t by looking into the previous gradients that have been calculated for it, as follows

$$w_{i,t} = w_{i,t-1} - \frac{\eta}{\sqrt{\mathbf{G}_{ii,t-1} + \epsilon}} \nabla_w \mathcal{L}(w_{i,t-1}) \tag{2.21}$$

where $\mathbf{G}_{t-1}$ is a diagonal matrix containing the sum of squares of the gradients for parameter w_i up until time step $t-1$, and ϵ a smoothing parameter.

Another optimization algorithm that aims to smartly adjust the learning rate based on the parameters is called *Adadelta*[22], which is an extension of Adagrad that more efficiently makes use of the stored past gradients. Instead of storing every gradient into matrix $\mathbf{G}$, a fixed sized time window is set, which is used alongside a decaying average function over the past squared gradients in the window, such as

$$\begin{aligned} \mathbf{E}[g^2]_t &= \gamma \mathbf{E}[g^2]_{t-1} + (1-\gamma) g_t^2 \\ w_{i,t} &= w_{i,t-1} - \frac{\eta}{\sqrt{\mathbf{E}[g^2]_t + \epsilon}} \nabla_w \mathcal{L}(w_{i,t-1}) \end{aligned} \tag{2.22}$$

where the running average $\mathbf{E}[g^2]_t$ depends on a fraction of the previous average and the current gradients. A similar unpublished approach that was developed independently and presented in a Coursera lecture by Hinton is *RMSProp*[23], which will also be used in this project.

But how are the gradients calculated in a recurrent neural network? This is accomplished through the *backpropagation through time* (BPTT) algorithm introduced by Werbos [24]. The method is an extension of the standard backpropagation algorithm of feed-forward neural networks, where the gradient calculated at time step t is propagated back to the previous time steps until the start of the sequence t_0. The reader is invited to refer to Werbos' paper for details of its implementation.

Vanishing and exploding gradients

Even though theoretically RNNs can be used to map input and output sequences of an arbitrary size, in practice learning from long-range dependencies poses a challenge that is difficult to overcome. This is due to a problem called *vanishing and exploding gradients* explored by Bengio *et al.* [25] and extended by Hochreiter *et al.* [26]. Neural networks learn by updating the parameters by a proportion of the gradient of the error function w.r.t the current

weights, and this process can be made difficult either if the gradients go to zero (no learning at all) or explode to large numbers (missing the local optima at every turn). In RNNs and also other deep architectures, the top layers may fail to learn in an adequate pace since its update is backpropagated through the bottom layers by multiplying the intermediate gradients through the *chain rule*. If those gradients are consistently between -1 and 1, which is the case of the most common activation functions, through the chained multiplication of the derivatives the weight update of previous layers will quickly approach zero. Other activation functions like the Rectified Linear Unit $max(0, x)$ may instead make the gradients increase rapidly due to its constant derivative of 1 w.r.t x for every value greater than 0.

Some solutions have been proposed to tackle this complication. A well-known variant of the learning algorithm of RNNs is the *truncated backpropagation through time* (TBPTT) which aims to solve the exploding gradient problem for continuously running networks [27]. In it, one can set a maximum number of time steps alongside which error can be propagated. Using a small cutoff value would alleviate the problem of gradients increasing too fast, however at the cost of learning long-range dependencies. A more modern approach however carefully design the nodes as to avoid the vanishing and exploding gradients problem while also being able to learn from information hidden long back in the input sequence, which shall be explored in the following section.

2.3.5 Long Short-Term Memory

The Long short-term memory network is an extension of a RNN that can extract long-range dependencies from the sequential input without suffering from the negative consequence of the vanishing and exploding gradients common in deep architectures. Firstly proposed by Hochreiter and Schmidhuber [28] and further extended by Gers and Schmidhuber [29], this technique has recently been applied successfully for domains like speech recognition [30] and video classification [31].

The main property that differentiates LSTMs with its classic counterpart is its *memory cell* which controls the flow of information that goes from the input $\mathbf{x}_t$ to the hidden states $\mathbf{h}_t$. This is accomplished by using a unique *cell state* variable $\mathbf{s}_t$ which contains the global information that the network currently has about the input sequence, which can be seen as its "memory". A visualization of the memory cell and its traversing state variable can be seen in Figure 2.5.

The memory cell structure of a LSTM is responsible for updating the cell state at each time step. This is accomplished by a set of so called *gates* that controls how the information will get in and out of the cell state. A modern LSTM architecture is composed of three of these gates, hereby called *input*, *output* and *forget gates*, alongside the *input node* which is a transformation over the raw input sequence. A LSTM memory cell at time step t is fully described by the following set of equations:

$$\begin{aligned}
\mathbf{g}_t &= \sigma_g(\mathbf{W}^{xg}\mathbf{x}_t + \mathbf{W}^{gh}\mathbf{h}_{t-1} + \mathbf{b}_g) \\
\mathbf{i}_t &= \sigma_i(\mathbf{W}^{xi}\mathbf{x}_t + \mathbf{W}^{ih}\mathbf{h}_{t-1} + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma_f(\mathbf{W}^{xf}\mathbf{x}_t + \mathbf{W}^{fh}\mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{o}_t &= \sigma_o(\mathbf{W}^{xo}\mathbf{x}_t + \mathbf{W}^{oh}\mathbf{h}_{t-1} + \mathbf{b}_o) \\
\mathbf{s}_t &= \mathbf{g}_t \odot \mathbf{i}_t + \mathbf{s}_{t-1} \odot \mathbf{f}_t \\
\mathbf{h}_t &= \sigma_h(\mathbf{s}_t) \odot \mathbf{o}_t
\end{aligned} \tag{2.23}$$

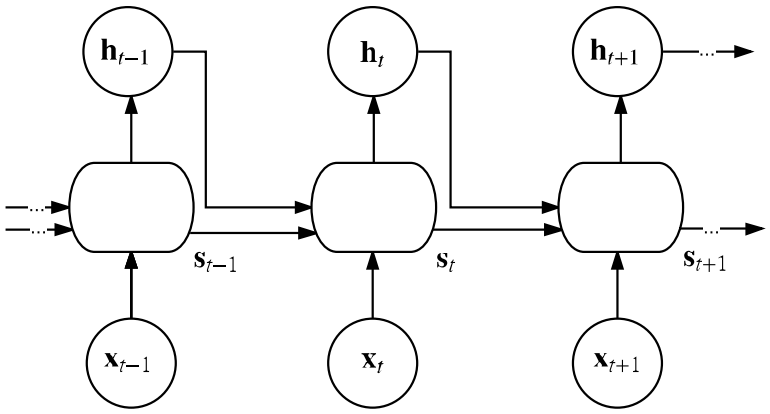

Figure 2.5: A LSTM network unfolded through time, with the oval shaped structure as the memory cell

where $\mathbf{i}_t, \mathbf{o}_t, \mathbf{f}_t$ are the input, output and forget gates, respectively. $\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{s}_{t-1}$ are the input vector at the current time, and the hidden units and cell state from the previous time step, which are the inputs to the memory cell. $\mathbf{W}, \mathbf{b}$ are the weight matrices and the bias vectors for each gate and node. $\mathbf{g}_t$ is the input node. σ are the activation functions: modern architectures use σ_g, σ_h as the hyperbolic tangent function, whereas $\sigma_i, \sigma_f, \sigma_o$ as the sigmoid [32]. The output of the memory cell are the updated cell state $\mathbf{s}_t$ and hidden units $\mathbf{h}_t$.

One can intuitively think of LSTM gates as controllers whether the activation for a given unit is let in and out of the cell state at each time step, which can be learned during network training. During the forward pass, if the input gate is zero, no activation can get into the cell state. Similarly, the output gate learns when it should let the activation leak out of it. The forget gate being equal to zero erases all the memory previously stored on the cell state, basically forgetting what was learned in the past. During the backward pass phase, the gradient may propagate several time steps in the past without exploding or vanishing due to its constant error carousel. Through this angle the memory cell learns as to when it should let the error in and out of the cell state.

Networks that makes use of the memory cell structure of an LSTM have been shown to outperform classical recurrent networks by being able to learn long-term dependencies from the sequence in a more efficient way. Positive results were obtained both in artificial datasets carefully designed to test the ability of the network learning long-term dependencies [25] and more recently in demanding sequence processing tasks [30][33].

2.4 Dimensionality Reduction

2.4.1 Principal Component Analysis

Principal component analysis (PCA) is a widely used method for applications like data compression, dimensionality reduction and data visualization. Initially invented by Pearson[34] and later independently developed and named by Hotelling[35], PCA strives to find an orthogonal projection of the original data into a lower dimensional space (also known as the principal subspace) where the variance of the projected data is maximized.

Consider a dataset composed of N samples, each one being D-dimensional vector $\mathbf{x}_n$,

and also a D-dimensional unit vector $\mathbf{u}$ where the samples are going to be projected on. The goal of PCA is to find a projection into a M-dimensional space where $M < D$ and where the variance of the projected observations are maximized, which can better be visualized in Figure 2.6.

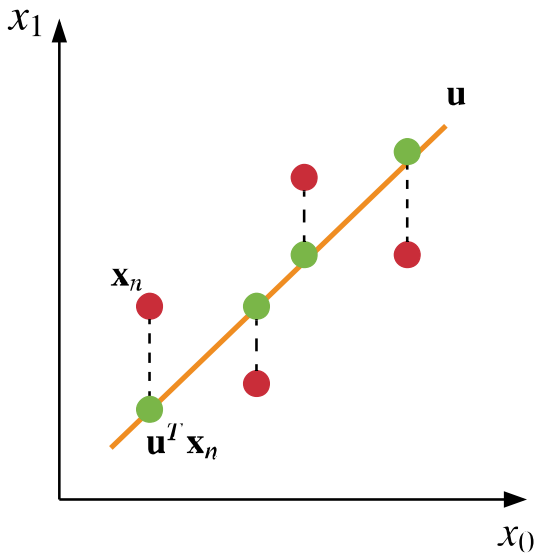

Figure 2.6: An example of PCA reducing the dimensionality of the data

After projecting the observations into the unit vector previously defined and obtaining $\mathbf{u}^T\mathbf{x}_n$, the variance of the projected data can be calculated with the following set of equations:

$$\begin{aligned} \mu &= \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n \\ \Sigma &= \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n-\mu)(\mathbf{x}_n-\mu)^T \\ \mathbf{V} &= \frac{1}{N}\sum_{n=1}^{N}(\mathbf{u}^T\mathbf{x}_n-\mathbf{u}^T\mu) = \mathbf{u}^T\Sigma\mathbf{u} \end{aligned} \tag{2.24}$$

where μ and Σ are the mean and the covariance matrix of the sampled data, respectively. Maximizing the value of the variance $\mathbf{V}$ with respect to $\mathbf{u}$ is an optimization problem constrained by $\mathbf{u}^T\mathbf{u} = 1$. Hotelling[35] shows that by introducing a Lagrange multiplier λ, this can be turned into an unconstrained optimization where

$$\mathbf{u}\Sigma\mathbf{u} = \lambda_1 \tag{2.25}$$

and so the variance will be maximized by picking the eigenvector corresponding the largest eigenvalue λ_1, which is known as the first *principal component*. The following $M-1$ principal components can then be chosen sequentially by picking the vector which is orthogonal to all other principal components already considered, which is equivalent to choosing the eigenvectors corresponding to the largest $M-1$ eigenvalues $\lambda_2, ..., \lambda_M$.

Chapter 3

Related Work

There is considerable amount of work done on the churn prediction realm. The most common topic that is addressed by the literature is evaluating different types of predictor models trained on a domain-specific dataset, like the telecom and financial industries for example. Understanding what techniques are commonly used on the field is deemed crucial for the success of any churn rate predictor model.

The use of data represented as a time-series for predicting churn, although of high interest to this project, is a rather under researched topic as of today, the user features being commonly aggregated over a time window on the whole observation period. Recently though some interest has been sparked on the literature about trying to leverage temporal aspects of the user behaviour by representing the data as a sequential input, and some examples similar to the task at hand, while not abundant, can nevertheless be found.

Other research papers focus on answering how different properties inherent to the churn prediction problem affect the performance of classifiers, like for example how to deal with unbalanced data sets, how the evaluation procedure should be performed and how far back in the user history should models be trained on as to obtain a proper balance between accuracy and processor usage.

This chapter will introduce the literature on the fields of interest for this project, and will focus mainly on the following topics:

- Identifying the state-of-the-art for training, modeling and evaluating churn predictor classifiers, the algorithms being used, the methodology for the experiments and the observed results.
- Researching current techniques for creating models that leverage the properties of time-series data, applied for churn prediction (if any) or similar domains where we can map our problem to. It shall also be studied common features of sequential data as to better understand how to best exploit its properties as to improve the accuracy of our models.
- Understanding prevalent characteristics on data sets used for churn prediction as to improve the performance of models and also to avoid common pitfalls when training and evaluating their accuracy.

3.1 Churn Prediction

Identifying which users are most likely to churn is an important step in any customer relationship management (CRM) framework interested in improving user retention. It has been shown that several economical values are directly correlated with the customer lifetime metric that can be crucial in a competitive market, like how the costs of acquiring new customers surpass those of retaining existing ones and how loyal users usually spend more and can bring in new customers [8]. Thus, churn rate prediction is an established topic in the literature and has seen studies being performed on a wide range of domains like the telecommunication industry [5][6][4], social multiplayer games [36][37] and community-based question answering services (CQA) [10][38].

The very first aspect of the study that any researcher has to take into consideration is the formal definition of what churn is, which depends on the domain that the data set belongs to. [39] divides the definition into three types: *active* where the contract is officially terminated, *hidden* when the customer is inactive (not using the service) for a significant amount of time, or *partial* when the customer is not using the service at its fullest, opting for a competitor instead. Subscription-based services like mobile companies [5][4] and massive multiplayer games [36] commonly utilizes the active churning definition caused by the explicit termination of the agreement. Social games [37][40] and CQA services [10][38] however usually model the user interest by measuring its level of activity and engagement since no contract is formally established, opting for the hidden churn definition instead.

Even though Spotify has a contract-based Premium service, this project is interested in the clients who are inactive for some time, whether they are paying customers or not. Delving into the hidden churn definition, one could ask what is considered a "significant amount of time" as to judge a user to have churned or not, and different domains have distinct approaches depending on the way the interaction between user and service takes place. For example, for a mobile network like the one studied in [6] this definition can span months of inactivity since customers on that market are commonly more loyal, even though this paper focus on pre-paid accounts with "soft" contracts. On the other hand, rapid-changing markets like free-to-use services as social games and CQA sites prefer to define the churn period as small as possible (commonly days or weeks), since the absence of an agreement makes it easy for customers to switch for the competition, having the service the obligation to act faster if any sign of loss of users is identified [6][40]. On [38] the focus is set on new users so the the churn period is explicitly set to be their first week of activity. [10] however focus on both new and experienced users by considering the first days and posts (questions and answers) as parameters on the models trained, evaluating each independently.

Current literature for prediction of churn rate uses a wide range of machine learning technologies, such as decision trees [10][4] [11] [6], logistic regression [8], neural networks [37] and its convolutional variant [41], random forests [38] and support-vector machines [42]. The number of techniques is vast, however it can be mentioned that current research suffers from a difficulty to reach insights that can generalize well for other areas, since the applications that the data sets belong to are domain-specific, greatly influencing how the users interact with the service. Interpretability is a key component in any application since the end goal is always not only to detect the users most likely to defect but also the reasoning behind their departure, as to plan reasonable actions to increase retention. With that in mind, even though some models like weighted random forests [43] and neural networks [37] offer a higher accuracy for some domains, the difficulty in getting key drivers behind the con-

sumer's behaviour in some "black-box" type of models have led to researchers opting for a more interpretable technique like logistic regression which is commonly a close second place in terms of performance [37] [38]. For that reason, decision trees and logistic regression are as of today the most popular techniques for churn prediction, yet a consensus of the best model cannot be reached by researchers as stated by [7].

The telecom industry is by far the most well researched area for predicting churn, and a proper review would be incomplete without mentioning some examples. In [4], two popular models for predicting user churn were empirically compared: decision trees and logistic regression. By making use of the data set provided by a mobile operator, two models were created by independently training and selecting their best performing versions. Evaluation was executed by comparing the AUC of their ROC curves, as also their Lift score and overall accuracy. The conclusion is that decision trees consistently perform better when compared to logistic regression models, and thus should be a preferred choice. In [5], churn classification was performed on users in a rather unique way. The Gentle AdaBoost boosting algorithm was used alongside a logistic regression base learner as to train a model to do the separation between the churning and non-churning classes. However, one further step was performed where the users were split into two clusters based on the weights assigned by the boosting algorithm. One of the clusters was identified to have a considerably larger churn rate than the other, and thus another logistic model was trained using the clusters as labels, and its performance evaluated using ROC curves. The idea behind this approach is to use a model to mark users who have a high risk of leaving the service (and be the focus of user retention actions), instead of labelling them as churners and non-churners directly. On [6], billions of call detail records from a mobile phone operator was the focus of a churn prediction study and feature analysis. The initial data set was filtered to the calls made by 100.000 subscribers during a 6 month period. A brute-force approach to feature engineering was then performed to create 12.914 out of the initial 10 features by combining every feature from each of the manually split 8 different groups. Feature selection is then performed in two distinct ways: first, an individual Student's t-test score is computed for each individual feature to evaluate how well it can differentiate between the churners and non-churners sets. Second, a tree-based method was used as to estimate the the accuracy of a joint classifier by adding features one at a time. Features are then ordered by their statistical correlation with churn, and the top 100 features were selected to train several different classifiers. Evaluation was performed mainly by comparing the AUC of each model, where AdaBoost using a decision tree classifier performed the best, followed closely by a logistic regression model.

The rapid increase in Internet popularity has spawned a plethora of new services in the last decade, and thus a wider range of domains like CQA sites and gaming applications are being researched for churn prediction. In [38], an explorative study was made on the Yahoo! answers website as to discover an efficient churn predictor model and also the features that correlates with the user leaving the service, however differently from other works this paper focused on new users with less than a week of activity on the service. Features were grouped into Question, Answer and Gratification categories, and were used for training several different classifiers. For this dataset, random forests performed the best with logistic regression as close second. Features were also ordered by the amount of information gain that they provide, and the number of questions and answers are the top features on that regard (inversely correlating with churn), followed by the period of time the user is active and gratification features for answers given and questions made. In [10], the younger CQA service Stack Overflow was the focus of a study on user behaviour characterization and churn rate pre-

diction. An extensive data exploration was made as to correlate features to the chance of a user leaving a service, and with those insights classical modelling techniques were used, where the best performing one was a decision tree. The approach used for extracting and categorizing features (temporal, frequency, gratitude, etc) and the insights that follow the study (like the importance of temporal features for predicting churn) is of high value and can be mapped to concepts on a different domain like a music streaming service with minor modifications. In [37], players responsible for generating the most profit on two casual social games were the focus of a study on churn analysis and prediction. Formal definitions for active players and churning were made, and the problem was defined as a binary classification task where users were labelled as leaving the service or not in the following week of when the models were trained. Four different models were then trained, and a neural network classifier obtained the best area under the curve (AUC) score, with logistic regression as a close second. The formal definitions of "churn" and "activity" can be applied to a music streaming service in a similar way, as the evaluation of performance using a series of receiver operating characteristic (ROC) curves. In [36], the activity log of a subscription-based multiplayer online game was the source of an experiment focused on comparing two different methods for predicting the users with a higher tendency to leave the service, a theory-driven and data-driven approaches. For the modelling, an ensemble method was used with several different classifiers, where the choice of model was made based on whether the cluster of the data set (found with K-means) that will be used to train the classifier has a significant proportion of churners or not. Even though the performance of the data-driven model was superior, the difference was negligible when taking into consideration the complexity of the models and also its interpretability. Moreover, if marketing resources are constrained the theory-driven model was considerably superior for the first 40% of users on the lift chart.

3.2 Sequential Modelling for User Behaviour Data

Even though there is currently a wealth of research on churn prediction techniques, most of them treat the user behavioral features as static points in time by aggregating or summarizing their values over a time window [44]. It is a simplification that eases the hassle of training models with a data set which is commonly high-dimensional and difficult to work with, however information that could be used to improve the predictors' performance is lost in this process [8]. For example, there is no information as to when over the time window being analysed a user decides to churn, it could be that all defectors are grouped into a period where an external event (like the end of an offer) has taken place. Environmental variables, that is those that varies through time but are common to all users, are also omitted on static models, leaving it to the experts the task of manually integrating these features on their analyses. Newer models like [10] attempt to integrate some temporal data by manually engineering features related to time (like the duration of an event, for example), however since each user is represented as a sample in a single point in time the sequence of data transformations is inherently lost in this process.

It is natural to think that the human behaviour evolves through time. Our opinions and desires changes in accordance to the stimuli that we receive from the world around us, and summarizing our persona by looking at a single point in our life is an oversimplification that does not make justice to the path that leaded us to where we are. Dealing with temporal data is by no means trivial however, and to leverage fully its properties some careful considerations about this type of data must be done. On [45], an overview of the challenges on creating

models that make use of the time component on data sets are presented to the reader. Also, since hand-crafted features are generally domain-specific and difficult to create, a review of the current research on unsupervised feature learning applied for time-series data is the focus of this paper. From the challenges, it is worth of mention the uncertainty that there is enough information to understand the underlying process, its non-stationary characteristics like mean, variance and frequency, and its common high dimensionality and noise. The right representation is deemed then crucial for any model to be successful in its goal. Several different techniques currently being used for unsupervised feature learning are then presented in details to the reader, where it is worth of quick mention the conditional Restricted Boltzmann Machine (cRBM), Recurrent Neural Networks (RNN, with is Long Short-Term Memory extension), the Gated RBM, the Time-Delay Neural Networks, and the Space-Time Deep Belief Network. Finally, several classical time-series problems are reviewed alongside the best models currently being used for each domain.

The hypothesis that sequential patterns hidden on temporal data can improve the accuracy of predictor models is not an uncharted territory however, and has being recently explored by researchers on similar domains as churn prediction that could be mapped to with minor modifications. Sparked by positive results of recurrent neural networks on domains like speech recognition [30] and video classification [31], some papers attempt to recreate similar deep architectures on their own prediction problems. On [46], a technique was presented as to predict the next event and its timestamp on a running case of a business process (a help desk process, for example). Three different LSTM architectures were experimented on, one for each of the problems being tackled: estimating the next activity and its timestamp, all the remaining activities in a use case and the remaining time of a process. This technique could be used for churn prediction by interpreting the user interaction with the service as actions, and "churn" could be an event that may or may not exist in the process chain. The LSTM architecture was also present on [44] alongside a quantile regression (QR) model as to predict which of the past buyers of a store chain will return after acquiring a product on sale. The hypothesis is that a performance gain can be achieved by combining models that are know to perform well on both temporal and aggregate features, respectively. It was empirically shown that a mixture of experts approach between these two techniques can reach a significant mean-square error improvement when comparing to any of the models when evaluated independently.

Deep architectures like LSTMs could theoretically be applied in a similar way for the task of predicting which users are more likely to abandon a service, however the lack of research using techniques like these suggests that this mapping may not be as easy as it seems, nevertheless some examples can still be found in the literature. Inspired by results obtained on image recognition [47], a convolutional neural network (CNN) model was used as a churn predictor on the work by [41]. On it, users were represented as 2-dimensional images where columns are the tracked features and rows are days organized in sequences. Two different CNN models were used, and both outperformed a baseline decision tree model. On [8] the researchers propose a new data set generation framework that can better leverage the temporal aspect of user behaviour data for churn prediction models. The hypothesis being tested is that by boosting the data with features on different time periods instead of only the last one, a significant performance gain can be achieved. The main framework of this study is called Multiple Period Training Data (MPTD), which basically consists of aggregating to each user their features at each predefined time step along the time range of the data set, alongside their churn labels at each point in time and also environment variables which varies through

time but are common for all users. This framework was tested against a classical static data set, demonstrating a significant improvement (by *p-value* comparison) on AUC and TDL for both logistic regression and decision tree models for predicting if the user is going to churn on the next test period or nor. Another experiment was focused on predicting churn several periods on the future, and for that a common survival analysis method called Cox regression was used as a benchmark for comparison against logistic regression models trained on MPTD. It was empirically concluded that using several independently trained binary classifiers, each trained on a different parameter to the "churn within δ periods" (WδC) feature, a significant performance gain can be achieved when comparing their respective AUC and TDL.

3.3 Evaluating and Training Churn Prediction Models

Data sets from service industries often have a common characteristic that may be problematic while training models, which is the uneven distribution between the churning and non-churning classes. It is a common scenario for the number of non-churners to heavily outweigh the number of churning samples on real data sets. Class imbalance is thus a recurrent concern in almost all domains, however research in this area lacks the proper attention. [43] aims to solve this mystery by focusing entirely on how the uneven class distribution affects the performance of several different classifiers. The performance boost estimated to be received by using a cost-sensitive learning algorithm (where false negatives are assigned a greater cost than false positives) is also evaluated, and it has taken the form of a weighted random forest model which was compared to other classical baseline models like logistic regression and random forests. Under-sampling, where fewer samples from the majority class are incorporated in the training data as to artificially change the distribution between the labels, is proved to significantly improve the performance of the underlying models, however the exact ratio between churners and non-churners is confirmed to be case-dependant, not necessarily being the even distribution the perfect choice.

To deal with the problems brought by the unbalanced distributions mentioned above, proper evaluation metrics must be used as to avoid the predictors to be biased towards the majority class. It is a well know fact that the accuracy of a classifier is not an appropriate method when comparing the performance of different models [48], and so no paper reviewed used this metric by itself. The overwhelming majority of the literature abides by the use of AUC as a proper metric for predicting churn rate, [11] [8] [6] [5] to name a few. [43] delves into this topic and defends the use of AUC as a proper metric, while adding the cumulative gains chart as to graphically represent the percentage of customers has to be targeted to reach a percentage of churners. The F-score is also sometimes added as a metric of interest, normally in conjunction with AUC [38] [6]. In respect to graphical representations, ROC curves and lift charts dominates the literature as appropriate methods, and it is present in works like [5] and [43].

Another question that can be made is how far in the user history models must be trained on as to achieve the best trade-off between accuracy and computational burden. It is intuitive to imagine that user actions far in the past exerts little influence on predicting whether the user is going to churn or not in the near future, however the exact threshold that should be used is not trivial to find. [11] attempts to answer this question by experimenting with three different models (logistic regression, decision trees with and without bagging) on a newspaper data set consisting of 16 years of user history. The models were evaluated for

their performance by training with data ranging on this interval with a 1-year gap between each measurement. It has been concluded that after year 5 no gain in performance (measured by AUC) is statistically relevant enough to warrant the increase in computational power needed for the training. However, the researchers also question whether this result can be extended for other domains or not, leaving that for future work.

Current data sets from service providers have no lack of user historical information stored, however the probability that all available features have a significant correlation to churn rate is pretty slim. Using too many features may lead to undesirable effects like over-fitting and falling into the "curse of dimensionality" problem, reducing the classifiers performance as cited by [49]. It is a common step to pre process the data in a way as to mitigate this risk, but still there is no single method in the literature that dominates the scene. Several papers do a manual subset selection of features based on theories, testing each using baseline classifiers and selecting the subset with the best performance [10][37]. Others works like [36] and [38] use information gain as a metric to classify the features on their expected reduction in entropy. One step further is taken by works like [5] and [6] where full decision trees are built specifically for feature selection. Another more exotic method was used by [41] with an autoencoder to discover which features influenced the churn rate the most.

Churn prediction has been explored in different kind of applications on recent literature, however music streaming services are as of today a quite under-researched area, even though there is a prominent feature that makes it significantly different from other more researched and similar areas like video streaming, which would be the absence of the constant attention factor that video demands. Studying the user behaviour in each application can possibly help optimizing parameters that are difficult to tune without expert knowledge on how the interaction between costumer and application occurs. The work by [50] aims to fill that gap by analysing patterns of usage of Spotify's Premium subscribers on features like session and playback arrival patterns, user behavior on single and multiple devices and favorite times of day for streaming. The main contributions of the study can be summarized as follows: First, daily patterns can be observed on features like session arrival, playback arrivals and session length. Second, there is a high probability of users to continue on the same device for consecutive sessions. Third, users have their unique times of day when they prefer to stream music on the platform. Fourth, a session length can be used as a good indicator of the next session length and also downtime.

Chapter 4

Dataset creation and exploration

In God we trust: all others bring data.

William Edwards Deming

Creating a dataset that is suitable for predicting churn is far from being a trivial task. Spotify has at their disposal an enormous amount of data that tracks every user interaction with the application, reaching terabytes in size every single day. This chapter will be dedicated to delving into details on how the dataset used for training our models was generated, how the sampling was performed, which features were chosen and engineered, and also exploring it in details.

4.1 Infrastructure and Data Sources

Every time a user streams content on Spotify, be it through the mobile application, desktop, web or any other platform, it generates a log entry with information regarding details on what has occurred in this interaction. This information can be the feature of the application used on this stream, its length in miliseconds, the context in the app where this stream originated from, to cite a few. Through a complex data pipeline, all the log data from different clients is aggregated into a massive dataset hosted in a cloud provider, and is externalized through Google's fully-managed query service called BigQuery [51].

After processing and cleaning procedures, a set of tables, named by the provider as *ReportingEndContent*, are generated one for each day, and it serves as the main source of data for this project. The tables have an average of 2.8 billions of rows with 38 columns, occupying up to 1.7 TB of storage in disk each, one of the largest datasets currently stored in Google Cloud platform. These tables are further joined with other tables as to gather a more detailed information about the stream, which are capped to a limit of 90 days due to privacy rules established by the provider.

4.2 Dataset creation

Creating a dataset that realistically represents the target population of this is by no mean an easy feat to accomplish. While it is desirable to train models with all the data that is available to us, doing so is an engineering feat by itself that could span the entirety of a project. Thus,

we need to make smart decisions as to create a dataset that is representative of the audience while also avoiding scalability pitfalls that eventually occurs when working with big data.

In this section, the *data pipeline* used for building the data used for model training and evaluation will be presented, which can be seen in Figure 4.1. Each subsection corresponds to a sequential stage in the pipeline, while the source are the raw stream information provided by the table *ReportingEndContent* and the output is the dataset in a format suitable for model training.

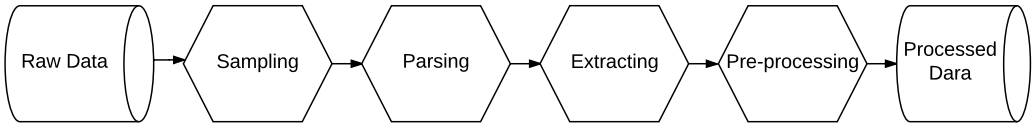

Figure 4.1: Dataset creation pipeline

4.2.1 Sampling

Each day hundreds of terabytes of streaming information are collected through the provider's data pipelines, aggregating customer usage of the application over several different platforms and countries. Since it is out of the scope of this project to train models with the data in its entirety, one of the challenges will be to understand the information that is available to us and to choose a subset of samples that are representative of the target population that we intend to focus on in this study. This task can be extremely daunting by itself, so this project will make use of previous private studies performed by the service provider as a guidance on which features to choose.

The main rule applied for sampling users is by filtering those who have at least one stream of significant length during the activity window described in section 2.2. The minimum length of the stream is set to be 30 seconds, which follows rules of royalty payments from the service provider. By applying this filter, we make sure that the users were active right before the period of prediction. Thus, users marked as churning did so between the period of observation and prediction, and we guarantee that we are not choosing samples who churned a long time ago.

Selecting users that created accounts any time during the observation period might skew our models towards a level of inactivity that does not corresponds to reality. For example, if a new account was created right by the end of the observation window, we might erroneously assume that the user was inactive for all days before the registration date. To avoid this, only users that registered into the service any time before the observation starts are sampled into our dataset.

Spotify is a service provided to countries all around the world, and we must be aware of the influence that seasonality plays into user behaviour if we intend to avoid overbiasing our samples. To limit the influence of day-night cycles in the learning process, users were sampled from three countries which share a similar timezone between each other and encompass a large share of Spotify's userbase, that is Brazil, Mexico and United States. **To do** Review the paragraph about countries (8)

The Spotify service is provided through a wide range of platforms like through Apple's iPhone or a desktop PC. To minimize the influence that different platforms may have on predicting churn, and also due to higher reliability on collected metrics, the dataset shall

be filtered on streams performed on the Android platform, be it on a smartphone or tablet device. **To do** Check Sahar: This is only for sampling, so will this be true? (9)

Even after all the previously mentioned filters, we are still left with a large amount of users which is more than any single machine could handle. While randomly selecting samples is a common performed method, for reproducibilty reasons we instead choose to deterministically select samples by applying a hash function over the user ID followed by a modulo operator over the desired fraction of users. Through this method we guarantee that the same users will be sampled, as long as the fraction does not change.

Through the application of all aforementioned rules, a grand total if *518.233 users* were sampled, and their streams were gathered through the course of 86 days (the standard size of the time windows), summing up to *1.8 billion* playbacks. This was only made possible by making use of the vast amount of computational resources available for this project, and also through well-known distributed frameworks like Spark and BigQuery.

4.2.2 Parsing

A playback may originate from several different contexts in Spotify. For example, a user may stream a song from a playlist manually created by an editorial team or personalized to his tastes based on his listening history (eg. 'Discover Weekly'), from the top charts of a genre, from an artist or album page on the app, and so on. These are called *play contexts* by Spotify, and they provide a great deal of information about the playback. Previous studies from the provider indicates that there is a non-zero relationship between the most commonly used context that a user's streams originates from and its probability to churn, and to leverage it we classify streams into 11 different contexts, called by the provider as *set types*. A sample of set types available follows:

- `personalized_playlist` A stream from a playlist personalized to the user's taste, like Discover Weekly and Year in Music.
- `user_playlist` A playback from a playlist manually created by the user.
- `user_collection` A stream from a set of artists, tracks, albums that were marked as favorite by the user.
- `catalog` A playback from a page manually searched by a user, like an specific artist or album.
- `editorial_playlist` A playback from a playlist created manually by Spotify's editors.

While the play context contains information that we would like to use in our models, its raw format is unsuitable for feature creation: it is basically a string value with hardcoded rules which indicates which set type the stream belongs to. A parsing method is available though, but due to the sheer amount of data available in our dataset it is unfeasible to iterate over the streams and apply the parser sequentially over the streams. Thus, we instead make use of a bath-processing programming model called Apache Beam **To do** add ref (10) executed on top of the managed service Google Cloud Dataflow **To do** add ref (11). By using a table in BigQuery with the sampled data as a source, we create a pipeline that maps every play context string to its correspondent set type, saving the results back into a BigQuery table. An example of a successfully executed parsing job can be seen in Figure 4.2.

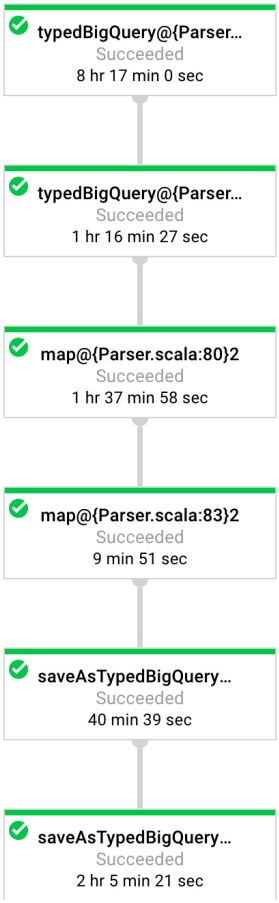

Figure 4.2: A parsing job in Cloud Dataflow

4.2.3 Feature Extraction

After filtering the group of streams from the users that we are interested on and parsing the play context information, the process of extracting and engineering the features can be started. The main goal of this stage is to create the sequential dataset that is going to be used for training the temporal models, while also calculating the binary labels that will indicate whether the users are going to churn or not. Before introducing the resulting features however, understanding how the time range was divided into different windows is utmost importance as to better comprehend how these features are generated.

Time windows

In order to predict whether a user is going to churn in the future or not and generate the churn binary labels, the time sequence is split into two time windows. The *observation window* is the time span where the user behavior is tracked, data which will be used for training the predictor models. It is sequentially followed by a *prediction window* that is exclusively used for generating the churn label. The windows can be better visualized in Figure 2.1. In order to make sure that selected users did not churn a long time before being observed, the last few days of the observation window are used for sampling users who are active, a time span called *activity window*.

The size of each window is a parameter that shall be experimented on, but a standard

length of *56 days* for the observation window, *30 days* for the prediction window and *3 days* for the activity window were chosen by the provider as values that are suitable for the task at hand. This encompass user information gathered from March 5th until May 29th, 2017. Unless otherwise stated, there are going to be the values in use for our experiments.

The user is considered to have churned if he has at least one active stream during the activity window, but no stream during the prediction window, which follows the definition of "churn" from Definition 4. An important factor that should be mentioned is that every user will have a single label for the whole sequence, and not a different label per point in time. This can be considered a simplification that reduces the complexity of our predictor models, however it fails to capture users that change between "churned" and "retained" states often: a user that churned may return to the service given more time. Changing the labeling method will be briefly explored in Future Works. **To do** Add this Future Work (12)

Our source dataset is sequential by nature. Every interaction is logged and stored in tables representing all streams of users in any given day. However, taking this data as it is is unsuitable to be used for training since commonly temporal models requires that (a) the time step between events to be constant and (b) it possess the same number of samples in every time step. Since there is no control as to when any given user will interact with the application, some data transformations must be performed.

First, a constantly-sized *time step* was chosen as to split the observations equally across time. To capture the routine of users between every day-night cycles, a time step of 8 hours was deemed to be a suitable value. The features of users are then aggregated over this time range through a mean function.

Second, the features of users that have no streams in any single time step are set to zero as to keep the number of samples constant through time. This processing will result in a dataset which is quite sparse, since rarely ever a user streams content in all 8 hours time spans over the course of 56 days of observation. An example of this processing can be seen in Figure 4.3.

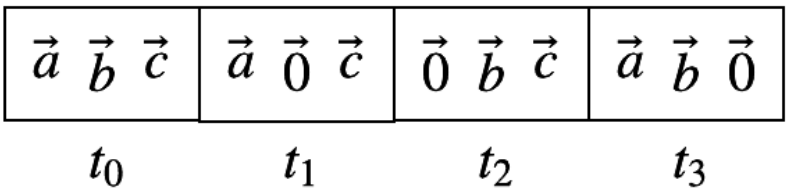

Figure 4.3: Zero-filling 4 time steps for 3 different users

After applying the previously mentioned transformations, the number of rows of the dataset is severely reduced to *87 million*, mainly due to the 8 hours aggregation procedure.
To do improve the zero-filling figure (13)

Features Description

No dataset would be fully described without a detailed explanation of the features within. This section will be dedicated to exposing the features extracted and engineered from the data sources described in previous sections.

The most basic feature of every stream is its duration in seconds, hereby called `sec_played`, which calculates how long the user playback lasted. Higher values may indicate that users are listening to a song in its entirety, while smaller values may indicate that the user skipped

a song or an error occurred, for example. They are not normalized by the duration of the song however, so it may indicate that the song is just shorter in length. The `skip_ratio` indicates the percentage of streams the user skipped comparing to all of his streams, commonly by pressing the "Forward" button on the player. Inter-arrival time (hereby called `iat`) is the time between streams: lower values indicates an user that is constantly using the app, and higher values a user that streams a song every now and then. `total_streams` is simply the count of streams the user had in the current time period, and `sum_sec_played` is an accumulating sum of the feature `sec_played` since the beginning of the observation period.

After parsing the play context from each stream, we engineer new features by combining them with three of the aforementioned basic ones, which are called `sec_played_<set_type>`, `sum_sec_played_<set_type>` and `total_streams_<set_type>`. Also, the number of unique set types is calculated and aggregated in the feature `unique_top_types`: a previous internal study by the service provider indicates that there is a non-zero relationship between the quantity of features being utilized by the user and its probability of churning, so we engineer and add it to our dataset.

All continuous feature values are aggregated by taking the mean over each time step period of 8 hours. For the `sec_played` group of features, the standard deviation is also calculated, and it corresponds to the features with the `_std` suffix.

By extracting and combining the features using the methods above, a grand total of 52 features were generated. The correlation matrix between these features can be seen in Figure 4.4.

4.2.4 Data Pre-processing

The goal of the final stage of the pipeline is to transform the data into a format which is suitable to be used for training and that also improves convergence time of the learning algorithms. This is achieved by two main steps: normalization of the feature values and exporting the data into compressed files.

Normalization

While it is possible to use raw features as input to train predictor models, often there is a benefit on normalizing the values using some standard technique. [52] argues that for the backpropagation algorithm commonly used in neural networks, well-behaved features centered around and with unit variance usually provides a faster convergence time. This is mainly due to the way weights are updated in the network: if the input vector has all features with the same sign, the weights can only be increased or decreased together for a single input, zigzagging the values back and forth which turns to be extremely inefficient. In deep architectures, the output a network layer is used as input to the next layers, and for the same reasons it is also desirable that this input is centered around zero. Inputs with unit variance will keep the layer's output well behaved and thus speed up convergence.

Therefore, the following transformations were applied to the input before the training process:

$$\mu_i = \frac{1}{N} \sum_{n=1}^{N} x_n^i \tag{4.1}$$

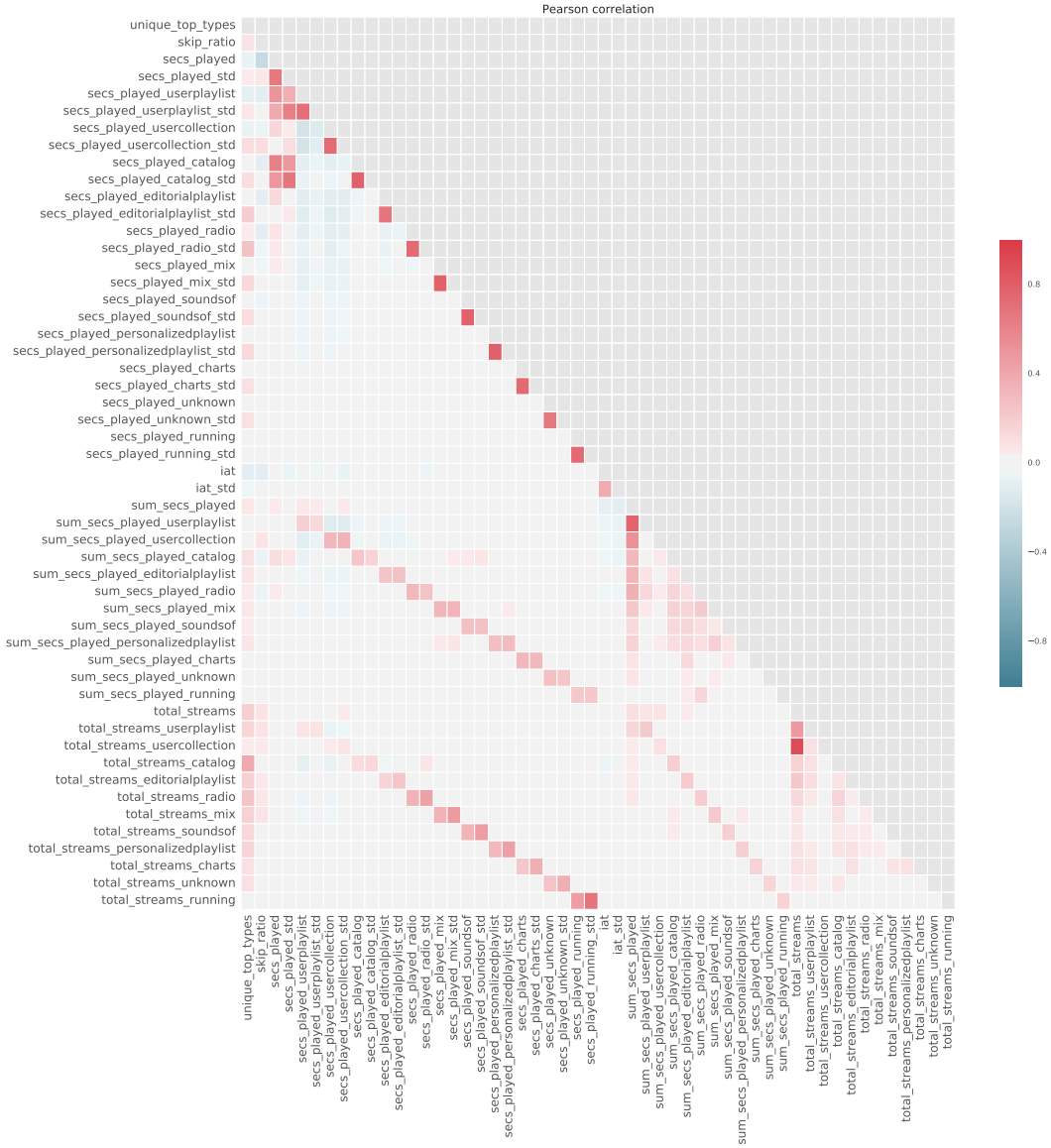

Figure 4.4: Pearson correlation matrix between features

$$\sigma_i^2 = \frac{1}{N}\sum_{n=1}^{N}(x_n^i - \mu_i)^2 \tag{4.2}$$

$$x_n^i = \frac{x_n^i - \mu_i}{\sigma_i^2} \tag{4.3}$$

where N is the total number of samples, x_n^i is a scalar value representing the feature i of the training sample n, feature which has mean μ_i and variance σ_i^2.

Aggregating and Exporting

After normalization, the feature values are in the desired format for training our predictor models, however they are still stored in an unsuitable location to be used as a source since all data is still stored as a table in BigQuery. Streaming the data directly from a cloud

provider into the training algorithms, while is possible, depends heavily on the choice of programming language and the available distributed frameworks, and it also susceptible to engineering problems like the network connection between the cloud provider and the host where the models are going be trained on.

For this project, we decided to take a simpler approach and use compressed JSON line-delimited files as a data source for the training procedures. This format can be exported to through Google's web interface and also programatically through BigQuery's API, and be easily loaded using standard libraries in the most popular programming languages. The rows are processed and exported to files in parallel, greatly accelerating the procedure.

We need to take some precautions for the export process to work flawlessly, however. The source table containing all the training data is stored in a long format, that is one row per user per time step. Since the export procedure is performed in parallel, the entries for a single user might end up spread into different files. To avoid having to fix this in the training functions, we apply a simple aggregation by user ID as to group all time steps belonging to a user into a single row. This can easily be accomplished using a query clause and functions in BigQuery.

4.3 Data Exploration

"Get to know your data" is a common jargon in the data science field, and for a good reason: even the smartest learning algorithms may fail to perform well if the data used as source is of questionable quality. Understanding how the data is distributed is of utmost importance, not only to gather insightful information that may be used in our advantage for training our predictor models, but also to validate that our dataset creation algorithm is free of mistakes and is representative of the target audience. This section will be dedicated to visualizing the data created through the pipeline described in the previous sections.

The large amount of rows in the dataset makes it difficult to plot it using standard well-known visualization libraries. Therefore, a sample of 1% of users was selected for the following plots by applying the same hash plus modulo operator described in subsection 4.2.1, totaling 5.125 unique users and 861.000 rows of data corresponding to the 56 days of observation time.

Due to privacy concerns and the signed non-disclosure agreement between the interested parties in this project, some of the tick labels in the graphs may have been intentionally hidden as to not reveal important business information of Spotify.

4.3.1 User Distribution

A recurring problem in churn prediction is how the non-churning class dominates over the churners for almost all applications, a problem properly examined by [43]. Figure 4.5 plots the distribution of classes in our generated dataset. In it we can observe that indeed the churning class is the overwhelming minority of our samples, composing approximately 2.4% of the data. This heavily imbalanced dataset may pose several challenges when training our models, since learning algorithms may fail to capture the nuances of user churning behaviour with a lack of representative samples.

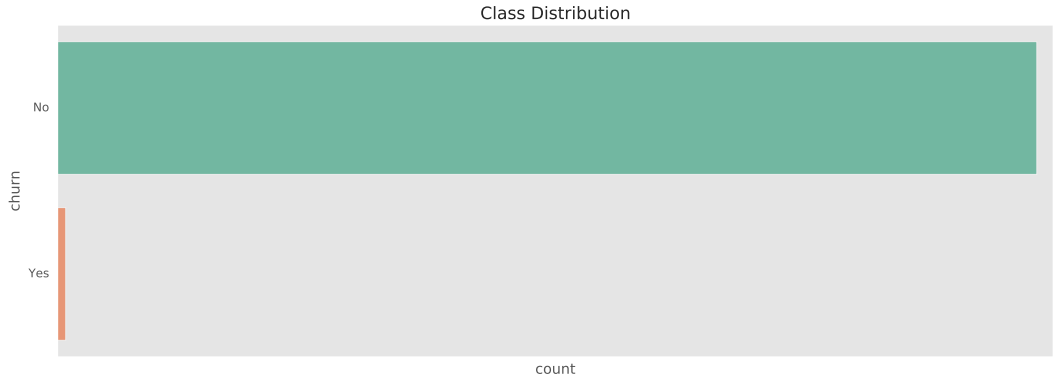

Figure 4.5: Count of users distributed in the churn and non-churn classes

Spotify has several different product types in its catalog. They can however be aggregated into two main categories with minimum loss of information, and that would be the ad-supported *free* version and all the different subscription-based paid versions called as *direct* by the provider. Visualizing the data by plotting against the different product types can be seen in Figure 4.6 and it reveals something interesting, albeit predictable: the majority of the churning users belong to the free tier, which is understandable due in parts to how easy it is to abandon the service without a formal contract attached, and also by the negative ad-filled experience that users are exposed to. **To do** add a ref here from pilot study (14)

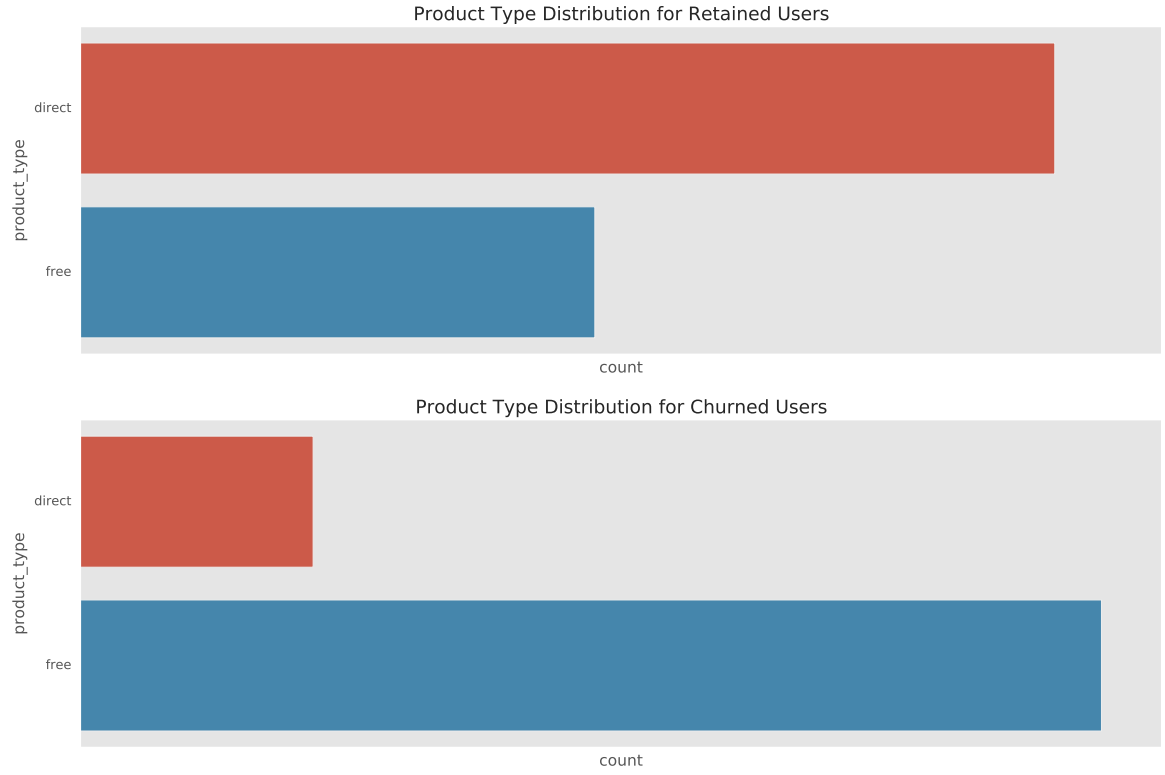

Figure 4.6: Users distribution per product type and churn label

In this project, three different markets were the focus of the study, so it would sensible to visualize how the distribution of the data differs for each single country. In Figure 4.7 we

can distinguish how the majority of sampled account belongs to users from US. Brazil and Mexico have a similar distribution for the retained class, however there is significantly more churning users in Brazil. **To do** Ask Sahar why is that (15).

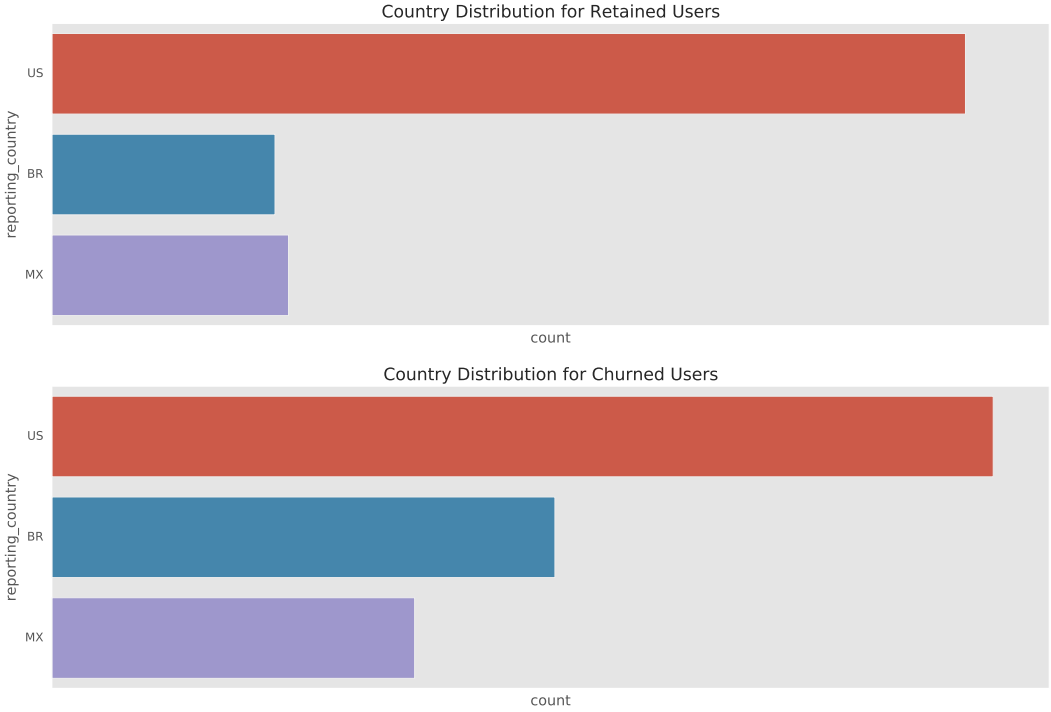

Figure 4.7: Users distribution per country and churn label

Users in Spotify possess different behaviors whether they registered for a long time or if they just created their accounts. Internal studies showed that long-term users reveal a more stable behavior on application usage when comparing to recent arrivals. Also, activating new users into the platform is a challenge in itself, since the service is unknown they are more susceptible to abandoning if they dislike the features they were exposed to. Figure 4.8 plots the number of days since registration between the churning and retained classes, it can be visualized how churning users are concentrated on the range of new users, while the retaining class is more uniformly distributed along the x-axis.

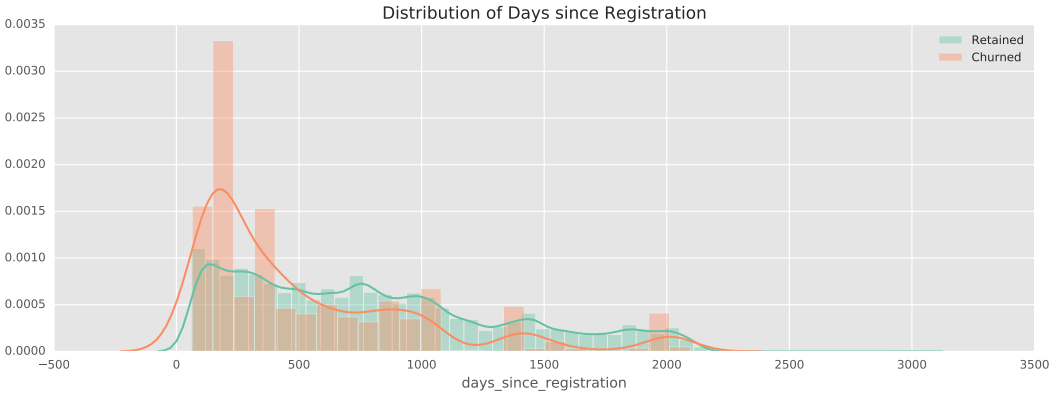

Figure 4.8: Users distribution over days since registration and churn label

4.3.2 User Behavior through Time

Humans are creatures of habits. Our daily routines follow distinguishable patterns mainly influenced by day-night cycles that are far from being random. To better visualize these patterns, Figure 4.9 plots the feature values for churning and retaining users over the course of the observation window sized at 56 days.

The first pattern that can be noticed is that users that are labeled as churning have lower values for features when to those who are retained in the service. This difference can greatly be perceived in the `sec_played` (the average number of seconds in all streams in the time step), `total_streams` (the count of streams in each time step) and `sum_sec_played` (the accumulating sum of seconds played in streams. These are all features that indicate usage: users that interact with the application more often and for more time are more likely to be retained.

The second trait that can be observed is that the gap between the classes for the `skip_ratio` feature is smaller than others. A previous study from the service provider suggests that users who churn have higher skip ratio values, however this is counter-balanced by the fact that free users have an artificial limit for skips in the application, currently limited at 5 per hour **To do** Check with Sahar (16). This may be the root cause for this behavior, since free accounts are the majority of the churning class, as could be seen in Figure 4.6.

4.3.3 Feature Correlation to Churn

One question that can be answered is how does any of the generated features correlates to the churn label. Do they exhibit a positive or negative association? Even though it is known that correlation does not necessarily implies a causal relationship, it may still be interesting to observe what is the overall tendency of the features when they are associated with churn. To accomplish this, a *point biserial correlation coefficient* is the correct tool to use when one of the variable is continuous and the other dichotomous, and its plot can be seen in Figure 4.10.

Some interesting relationships can be observed. First, the plot shows that `iat` features (inter-arrival time between streams) are the ones with the strongest association to the positive churning label, which is understandable: users that do not come back to the application as often are not making use of the service at its fullest and are more likely to abandon it. Second, the `sum_sec_played` group of features are in its majority negatively correlated with churn, which strengthens our hypothesis that customers who use the service more often are less likely to abandon it.

Another interesting fact is that not all set types have the same association with our label: while `user_playlist` and `personalized_playlist` have the strongest negative association with churn of all available set types, while users that have longer streams of `editorial_playlist` are more correlated with a positive value churn. Again, this does not imply that there is a causal relationship between the two, but may be a good tip towards investigating how this feature is delivered to the user.

Lastly, a rather surprising result is that the number of unique set types have a positive association with churn. Our hypothesis is that users who are new in the platform posses a more explorative behavior by trying several different features when compared to users who are in the platform for a long time. As could be seen in Figure 4.8, churning users are commonly younger in the service when compared to retained ones.

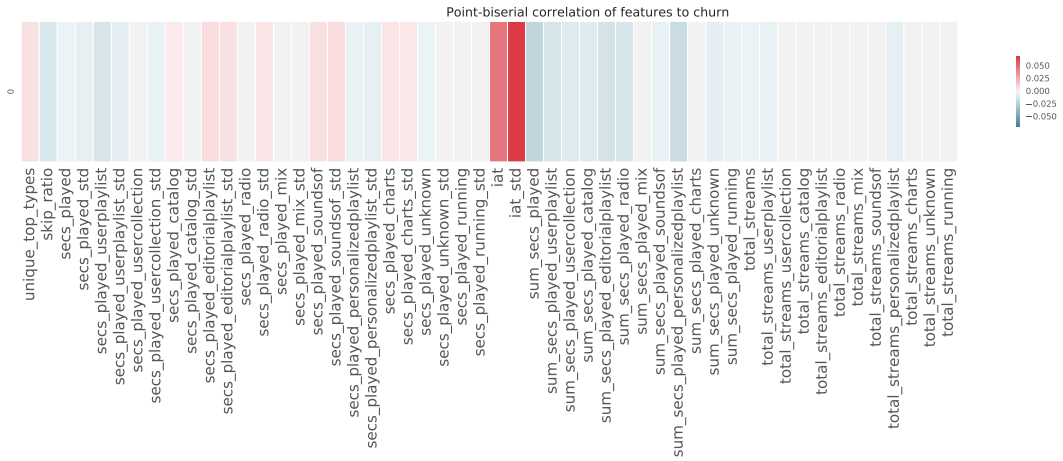

Figure 4.10: Point-biserial correlation between features and the churn label

4.3.4 Dimensionality Reduction

Although the features generated through our data pipeline encompasses different aspects of the users behavior in the platform, it may pose a challenge regarding the high dimensionality of our dataset: machine learning models may have trouble extracting the latent information from feature values if data is not sufficient. Since our dataset is heavily imbalanced, the number of available samples for the churning class may be insufficient, and using a more compact representation of the data may prove fruitful when the goal is to maximize the score of our predictor models.

Principal Component Analysis (PCA) is popular technique that strives to find a lower dimension representation by keeping most of the variability of the data, or as described by Hotelling [35], it is, for a given set of data vectors, the d orthonormal axes onto which the variance retained under projection is maximal.

In Figure 4.11 we can visualize the data distribution when its dimensionality is reduced to its two principal components through PCA for both retained and churned classes. It can be seen that even though the overall shape of the distributions are clearly different, their center points are not that close to each other, which may result in difficulties when trying to find a hyperplane that bests separates both retained and churning classes.

4.4 Software and libraries

Manipulating large quantities of data is a task that cannot be performed easily using standard libraries of a programming language. The pipeline creation was only made possible in the scope of this project by making use of frameworks for massive data manipulation available at Spotify.

The sampling, extraction and pre-processing steps of the data pipeline were performed using Big Query[51], a fully-managed and cloud-based interactive query service for massive datasets created by Google. The parsing step that extracted the type of the stream from its play context was performed in Scio[53], a Scala API for Apache Beam and Google Cloud Dataflow.

Figure 4.9: Mean values of features throughout the observation window

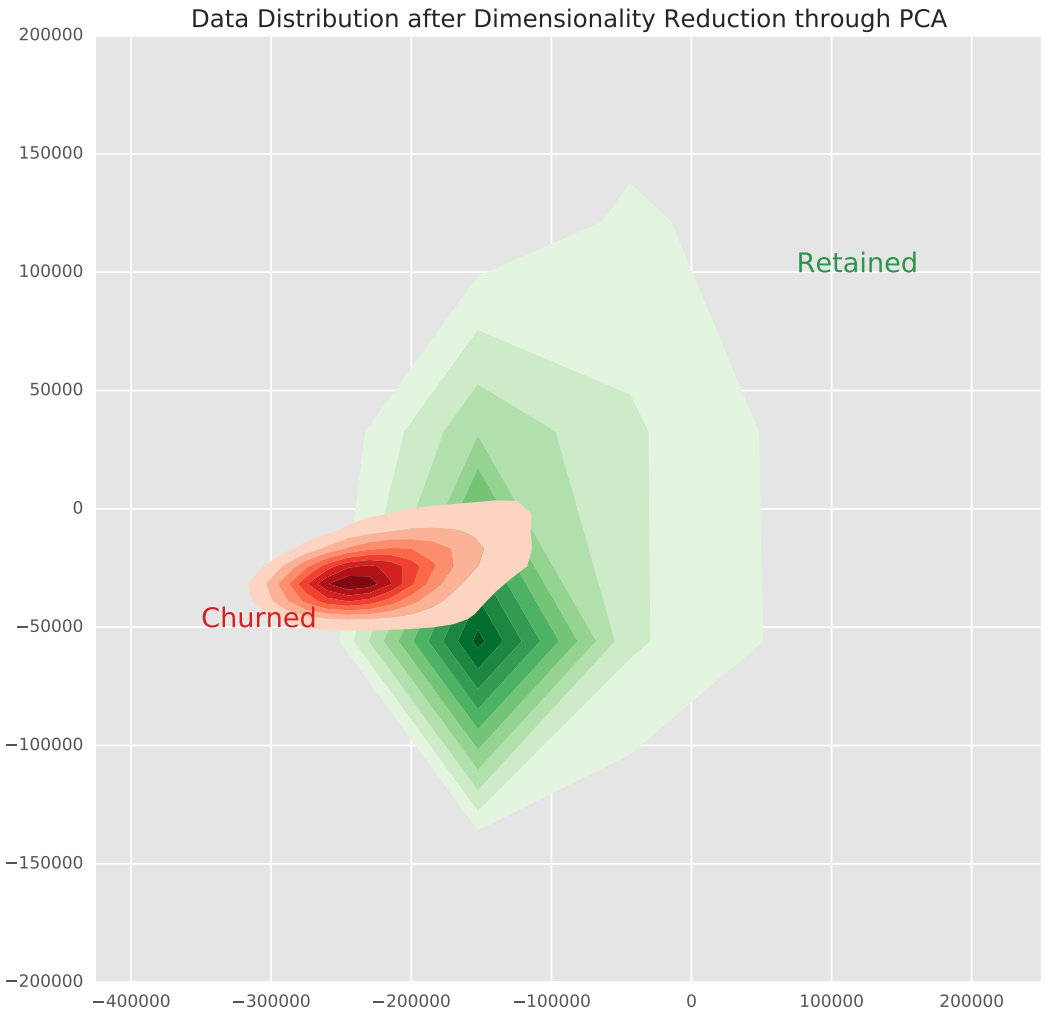

Figure 4.11: Data distribution after reducing to 2 principal components through PCA

Chapter 5

Methodology

Creating experiments that are reliable, reproducible and as unbiased as realistically possible is a challenge for any researcher interested in obtaining results that can safely be reasoned on. It is made harder by the complexity of our own source dataset: human behavior is far from being a process easily predictable due to the sheer amount of latent variables that may influence the outcome of the experimentation process.

This chapter is dedicated to present to the reader a detailed picture how the experiments were performed, the choices of classifiers, the way they were optimized to the best set of parameters, and how the final score for each model was obtained.

5.1 Handling the Class Imbalance Problem

For any relatively successful company the non-churning class heavily outweigh the amount of churners in the dataset. Depending on the estimator, this property can have a negative impact in performance where the models only can only learn how to distinguish the majority class and failing to detect the class of interest. However empirical observations have shown that a better performance can be achieved if the class ratio difference is reduced [43] [54].

There are three commonly used sampling techniques for dealing with heavily skewed datasets:

- Undersampling by removing samples from the majority class using a pre-defined selection technique
- Oversampling by creating artificial samples that closely mimics the true data present in the minority class
- A combination of over and undersampling, commonly oversampling the minority class first and reducing the data in the majority class second.

Unless otherwise stated, the experiments in this project will be *randomly undersampled* to a class ratio of 1-to-1. This technique, while easy to implement, throws away several samples from the majority class that could be used for learning. Testing different ratios for the class distribution is however one of the experiments that will be performed in this project. Different methods for under and oversampling will be further discussed in Future Works **To do** add over undersampling talk in future works (17).

5.2 Cross-Validation

It is a common practice in supervised machine learning experiments to split the data sets into *train* and *test sets* as to obtain an unbiased estimate of the performance of the trained models on unseen data: evaluating a model on the data it was trained on can return an overly optimistic score that does not generalize well for data that is not included in the training, a phenomenon called *overfitting*. Therefore, every score in this project is reported using data that was not present in the training stage.

Almost all machine learning predictor models have arguments that needs to be chosen even before the training process starts, values that are commonly called *hyperparameters*. To fine tune these parameters, we need a method to evaluate how good a set of values are against all other (or a reasonably sized subset of) available options, and for that purpose we need a set of unseen data for the same reasons mentioned before. If the test set were used for such an evaluation, it could not be considered as unseen data anymore since it was used for the purpose of choosing the best model, thus our bias may have "leaked" to the prediction score. Therefore another set is needed to tune the hyperparameters, which is commonly called the *validation set* in the machine learning domain.

Dividing the dataset into three splits however drastically reduces the number of samples that ares available for model learning. Another concern is that if the sets are unique, the overall performance metric will depend in a seemingly random split of validation and test sets. To mitigate this problem, a technique called *cross-validation* can be applied[55]. In its most basic approach, instead of having a single validation set, the training set is instead divided into k folds, and at each hyperparameter tuning round the model is trained on $k - 1$ folds and validated on the fold that was left out. For each set of parameters, k models are generated, and the final metric can then be an aggregation function over all estimators performance, like the mean. After selecting the best set of parameters, another training session can be made with the union of the data available in all folds, thus acquiring a tuned version of the model.

The above technique helps to strengthen our performance metric against biases when selecting the best set of hyperparameters to train our model with. However we still have an unique test set where we evaluate our model to obtain a final score. When comparing different types of estimators, it may happen that one estimator may better learn from the samples that are on the training set while the other can better learn from the patterns contained on the samples of the test set, a split that was purely made by chance. This may lead to a form of selection bias resulting in overly-optmistic scores [56]. The same procedure used for hyperparameter tuning can be applied for splitting the train and test sets, a technique called *nested cross-validation*. In it, the full dataset is split into k outer folds of training and test sets, and each training set is further split into l inner folds for hyperameter tuning. A diagram of this procedure can be seen in Figure 5.1.

Even though this technique increases the confidence of our metric scores, there is a price to pay in terms of performance: training all these models may take a considerable amount of time. Spotify however has no lack of computational resources available, so this project makes use of this method. Inspired by [43], all our models were trained using 2×2 *cross-validation*, with 2 outer folds for the train/test split and 2 inner folds for train/validation splits.

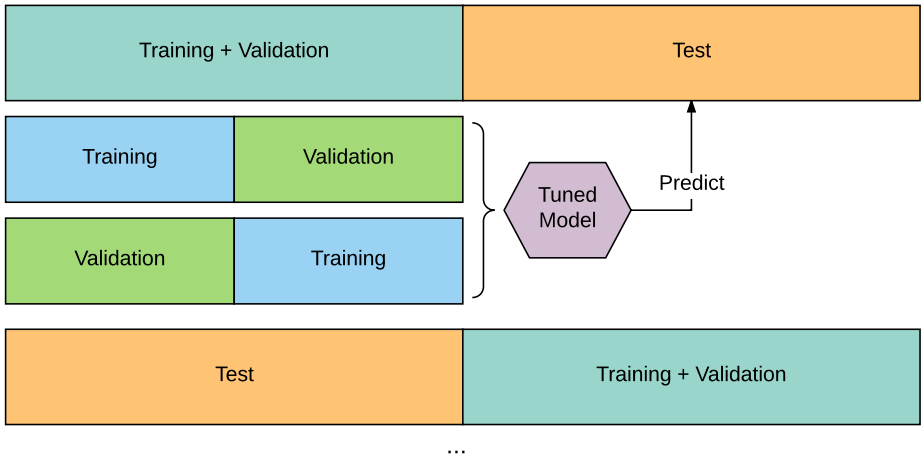

Figure 5.1: 2x2 nested cross validation

5.3 Hyperparameter search

Lastly, a proper method needs to chosen to perform search over the domain of parameter values. A common practice in machine learning model training is to perform a parameter sweep by exhaustively searching a subset of values manually chosen by the experimenter. This method is called *grid search*, and due to its simplicity and reproducibility is one of the most used techniques, be it for classical machine learning algorithms like Support Vector Machines as to deep feed-forward neural networks [57].

However, performing a grid search over every combination of values can quickly turn it into a method which is too computationally expensive to perform for any real-world applications. Deep architectures like LSTMs are known to be costly to train, and combining it a full parameter sweep is prohibitive if the dataset in question is of a respectable size. Bengio and Bergstra[58] have demonstrated that, for neural networks, performing a *random search* over the same parameters domain can result in a model which is as good or better than the one validated through grid search by using a fraction of the computation time.

In a random search, parameter values are randomly sampled from a specified distribution instead of being chosen directly from a grid. For a set number of rounds, samples are gathered for every parameter and used for model validation. This allows us to set a processing budget which is independent of the number of parameters available and the number of possible values for each. Therefore, adding parameters that have no impact in performance will not decrease the efficiency of the cross-validation process for no gain.

The experiments performed in this project make use of both methods: for the classical Logistic Regression and Random Forests, a grid search over a pre-selected set of values was the chosen method, since their training time is commonly faster when compared to regularly sized deep neural network. Random search is the technique used for validating parameters of our LSTM models, since it is too expensive to compute with a parameter sweep due to its long training time. The set of parameters used for the experiments follows:

- Logistic Regression
  - C (inverse of regularization strength): 0.01, 0.1, 1, 10, 100
  - Regularization method: L1-norm, L2-norm

- Random Forest
  - Number of estimators: 10, 100, 500, 1000
- LSTM
  - Number of layers: 1 to 3
  - Units per layer: 64 to 256
  - Optimizer: RMSprop, Adagrad

5.4 Model training

Even though all models were trained with the same source dataset, temporal and time-invariant models requires a different representation of the data, a topic which was explored in section 2.2. Logistic regression and random forest models were both trained with the SPTD version of the dataset using the mean as the aggregation function $f()$, while the LSTM models were trained with the full unrolled MPTD data.

When analyzing the data contained in the MPTD dataset, it can easily be noticed that is severely sparse. This is due to the representation chosen for the absent user in a specific time step: users that did not stream any song during an 8 hours interval will have its features all set to zero. We can leverage this property through a data structure that better utilizes memory resources by storing only the non-zero entries instead, which is accomplished by representing the data with a *compressed sparse row* structure [59]. This allows us to store the data in memory in a single machine without resorting to distributed algorithms which would make the implementation more difficult.

To increase the reproducibility of our experimentation process, the same seed value was used for all random factors contained in the experiments. That includes the dataset fold splitting, random search, weights initialization in LSTMs and the undersampling. The seed value chosen was 42, which is the answer to the ultimate question of life, universe and everything [60].

5.4.1 The LSTM recurrent network

This section describes some details about the LSTM models used in the experiments of this project.

Since the number of layers is a parameter that is being tuned during the cross-validation phase, the exact architecture of the LSTM varies depending on the training and validation data. It is beneficial however to visualize the overall structure of one choice of model parameters as to better understand how the layers are organized, which can be seen in Figure 5.2.

In this example, three LSTM layers were stacked on after the other, each one with a different number of units per layer chosen through random search. A softmax layer transforms the output of the last LSTM layer and transform it into a probability distribution over the two retained and churned classes.

Every model was trained for 70 epochs using a batch size of 512, a number chosen for performance reasons. The optimizers were initialized with a learning rate of 0.001. For Adagrad, the value for the fuzz factor $\epsilon = 1 \times 10^{-8}$ applies. RMSProp also has the same ϵ, adding $\gamma = 0.9$ which regulates how much of the previous average of gradients shall be used to calculate the current gradients .

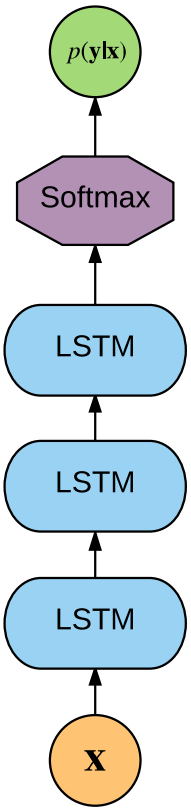

Figure 5.2: Model architecture with 3 stacked LSTM layers

The weight matrices for the linear transformations of the input were initialized with Xavier initialization method, while the ones corresponding to the linear transformation of the recurrent state were initialized with a random orthogonal matrix. Bias vectors were initialized to zeros since the asymmetry breaking is already performed by the chosen weight initialization algorithm.

The loss function is binary cross entropy. All parameters were regularized with L_2 weight decay method, using a regularizing factor of 0.01.

5.4.2 Training Procedure

We summarize the training process in Algorithm 5.1. This procedure is applied for every different classifier that we are currently experimenting on, and results in a probability score assigned for every sample in the dataset.

The first step of the procedure finds all the parameter values that the models are going be validated on, which can be either a full parameter sweep for Random Forests and Logistic Regression or through randomly sampling from parameters distributions for an LSTM. This will define the size of the innermost loop of the algorithm.

The outermost loop iterates two times by splitting the input data into two halves and selecting one fold to be the training set and the other the test set at each turn. After this split, the training data (and *not* the test set) is randomly undersampled to the specified ratio, commonly 1-to-1 unless this is a parameter that is actually being experimented on. Since our dataset is heavily imbalanced, this reduces the number of samples significantly.

The second for-loop iterates also two times and splits the undersampled training data

further into two sets: training and validation. Finally, a model is trained for each choice of hyperparameter for that specific classifier, resulting in a score for that parameter for that fold. The best hyperparameter is then selected to be the one with the best average score over the 2 folds, which is then used to train a new model with the training the whole training data obtained from the first topmost split.

After the training of the tuned model is finished, it is used to calculate the probability scores of each sample in the test set. These scores are stored in memory, and the process is repeated by switching the roles of the test and training sets. When the whole procedure is finished, every sample from the dataset will have a probability score assigned to it calculated when that sample was part of the test set.

Algorithm 5.1 Training and evaluating procedure using nested cross-validation

Require: $indata$: input data
Ensure: $preds$: predictions for every input sample when they were part of the test set

```
params ⇐ gridOrRandSearch()
for trval, te in 2fold(indata) do
    trval ⇐ undersample(trval)
    for tr, val in 2fold(trval) do
        for p in params do
            model ⇐ train(tr, p)
            scores[p] ⇐ evaluate(model, val)
        end for
    end for
    p ⇐ argmax(avg(scores))
    model ⇐ train(trval, p)
    preds ⇐ predict(model, te)
end for
return preds
```

5.5 Evaluation Metrics

5.5.1 Confusion Matrix

A *confusion matrix* (also called contingency table) is a table layout representing the performance of a classifier's output, judging by its predictions against the actual true values. In a binary classification problem, the confusion matrix is a 2 x 2 table where commonly the rows represent the true labels while the columns the predicted classes. Each cell contains a count of how many samples were classified on that category, and the values in the diagonal represent the correctly classified samples (if the features are ordered). Figure 5.3 depicts a sample confusion matrix for a binary classifier.

The confusion matrix is an excellent visualization tool to estimate the performance of a model. The values that should be maximized, the *true positives* (TP) and *true negatives* (TN), represent the samples that were correctly labeled as possessing the feature of interest or not, respectively. On the other hand, the *false positives* (FP) and *false negatives* (FN) are the misclassified samples where the algorithm predicted that the feature was present while in truth it was not and vice-versa. In this work, the positive class will always represent a

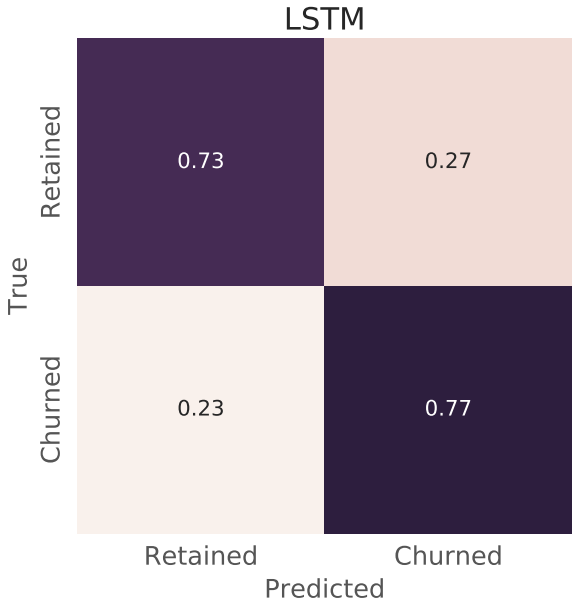

Figure 5.3: An example of a normalized confusion matrix

churning user, and it follows that the negative class represents the non-churners.

5.5.2 Classification Accuracy

Several different metrics can be derived from the confusion matrix table. The *classification accuracy* (CA) of a model is a common metric that corresponds to the fraction of the correctly classified samples on the test set, and can be calculated as follows:

$$CA = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

While trivial to understand, this metric may lead to erroneous conclusions when class imbalance is present in the test set, which is a common occurrence on the churn prediction domain. For example, if 9 out of 10 users of a dataset are non-churners, any classifier that simply outputs a negative class for all samples will result in an accuracy of 90%, however its ability of detecting churners is non-existent. For a service provider, detecting churn cases is always more important than detecting the loyal users, and this metric by itself cannot represent this goal [43] [4].

5.5.3 Precision, recall and other metrics

To address the class imbalance problem of the classification accuracy, others metrics are also commonly used. The *positive predictive value* (PPV, also known as precision) is the proportion of the samples labeled as positive which are true positives, and describes the performance of the algorithm. The *true positive rate* (TPR, also called sensitivity and recall) of a model corresponds to the number of correctly predicted positive samples divided by all positive samples. *True negative rate* (TNR, also called specificity and fall-out) is the number of correctly predicted negatives divided by all true negatives. The *false positive rate* (FPR, also called false alarm ratio) is the probability of receiving a false positive as output of an experiment, and is calculated by dividing the number of false positives by the total number of positive

samples. The *F1 score* (F1, also called F-score or F-measure) builds upon precision and recall by returning an harmonic mean between these two metrics.

$$PPV = \frac{TP}{TP + FP} \tag{5.2}$$

$$TPR = \frac{TP}{TP + FN} \tag{5.3}$$

$$TNR = \frac{TN}{TN + FP} \tag{5.4}$$

$$FPR = \frac{FP}{TN + FP} = 1 - TNR \tag{5.5}$$

$$F1 = 2 \times \frac{PPV \times TPR}{PPV + TPR} \tag{5.6}$$

Depending on the distribution of classes of the dataset, it is often difficult (although desirable) to maximize both precision and recall metrics at the same time. A compromise must be reached that achieves the best trade-off between the two, which is commonly a business decision. For a music streaming service like Spotify, maximizing sensitivity is preferred due to the costs associated with a churning user, however a reasonable specificity is also a metric that should be strived for. **To do** Check with Sahar (18)

5.5.4 Receiver Operating Characteristic

The *receiver operating characteristic* (ROC) is a visualization tool that plots the relationship between the true positive rate (commonly the y-axis) and the false positive rate (the x-axis) of a binary classifier system. The curve is drawn by selecting different parameters of a model or levels of threshold for the decision boundary between the positive and negative classes. For example, when the output of a classifier is a probability value (like in logistic regression), different thresholds can be chosen to decide whether a user is a churner or not, depending if the goal is to minimize FPR or maximize TPR. An example of a ROC curve can be seen in Figure 5.4.

A good classifier would score values close to the upper-left corner of the plot, where the point (0,1) represents a perfect classifier with 100% TPR and 0% FPR. On the other hand, an algorithm that outputs a curve alongside the diagonal where TPR and FPR are almost the same at different threshold levels can be considered close to a random guess, like the flip of a coin. A classifier would underperform if its scores are closer to the bottom-right corner of the plot, however this result can always be mirrored by updating the model to simply invert the positive and negative labels of the classified samples.

ROC curves can be used to compare the performance of different models by measuring the *area under the curve* (AUC) of its plotted scores, which ranges from 0.0 to 1.0. The greater this area, the better the algorithm is to find a specific feature. Moreover, models with an area close to 0.5 can be assumed to perform not much better than random guess, since this is the total area under the diagonal line. The AUC can also be interpreted as the probability of a randomly picked positive sample is ranked higher by the model when compared to a randomly picked negative case.

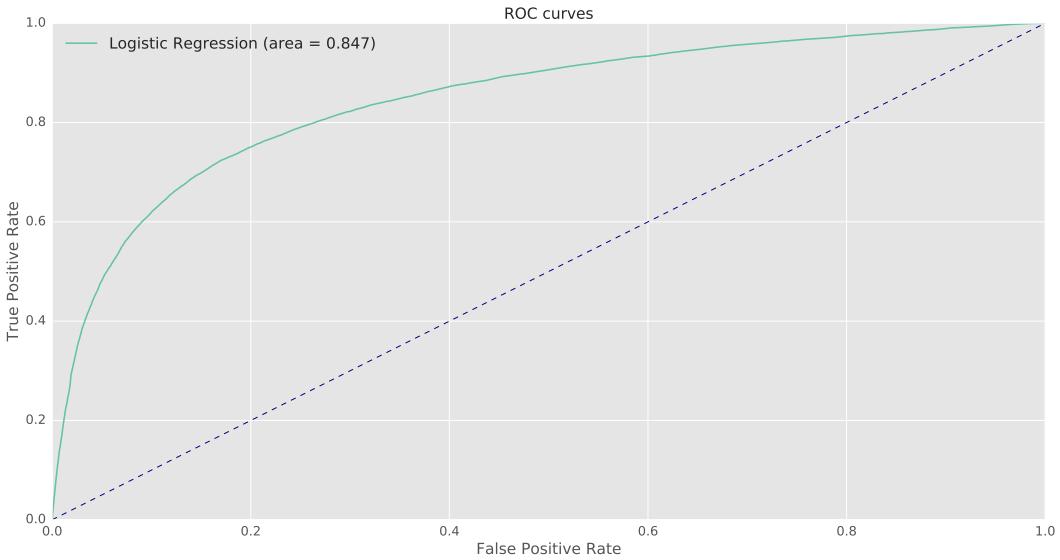

Figure 5.4: An example of a ROC curve

5.5.5 Precision-Recall curves

While ROC curve is the most commonly used method for evaluating the performance of a classifier, it still fails to capture the true quality of the model when data is severely sparse. For any heavily skewed dataset, predicting the negative majority correctly can be accomplished easily, however the same cannot be said about the rare positive class. Since ROC curves utilizes the false positive rate calculated using the large number of true negatives, the plot may give the idea that the model is doing quite well, while failing to show its ability to predict the positive class.

Visualizing the precision plotted against its recall value may be more informative when dealing with sparse data. The *precision-recall curve* is the visualization tool that accomplishes exactly that, for every threshold value, the recall (commonly the x-axis) is plotted against its precision (the y-axis).

Figure 5.5 shows an example a precision-recall curve plotted for the same data from Figure 5.4, however focusing only in the churning class which is the focus of this study. Here we can clearly see how the precision is affected by an skewed dataset: even if our model manages to have only a small fraction of false positives, since they are many the precision will quickly be reduced.

Similarly to ROC curves, the area under the line correlates to the overall quality of the classifier: values closer to 1 indicates that a model has high precision and recall for every different tested threshold. The balance between precision and recall can be manipulated by changing the threshold value: if we want to increase the recall, simply using a lower threshold will yield an increase in the number of true positives, at the cost of lower precision.

5.6 Statistical significance

To do Will I do statistical significance? If so, how? (19)

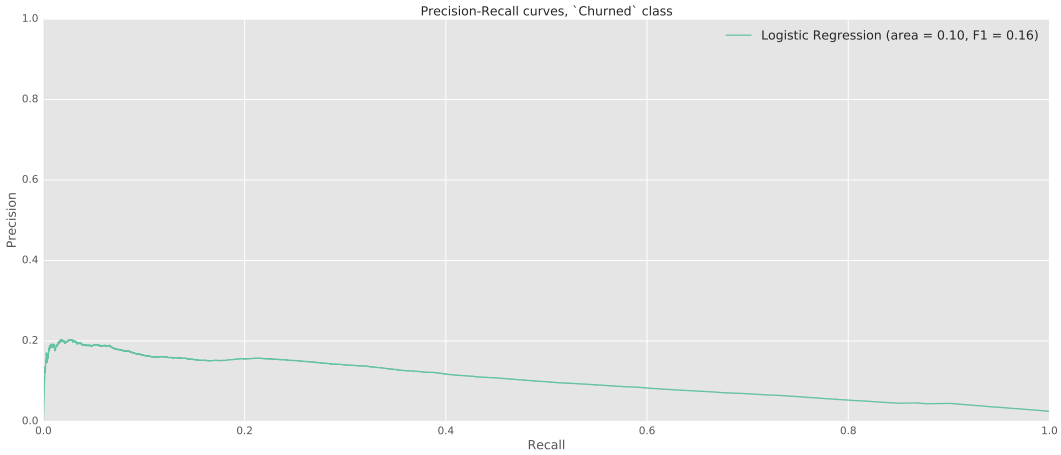

Figure 5.5: An example of a Precision-Recall curve

5.7 Software and Libraries

The experiments on this thesis project were only made possible by utilizing well known open-source software libraries for data processing and machine learning. The recurrent LSTM models were implemented in Keras[61], a high-level deep learning wrapper library written in Python. Tensorflow[62], a library for numerical computation using data flow graphs created by Google, was used as a backend for Keras. Random forests and logistic regressions models, as well as the procedures for handling cross-validation and hyperparameter search, were implemented on Scikit-Learn[63], a Python module for machine learning. The random undersampling was performed using a modified version of a library called Imbalanced-Learn[64], a Python package that offers several re-sampling techniques for imbalanced datasets.

Chapter 6

Results and Discussion

> Talk is cheap. Show me the code.
>
> *Linus Torvalds*

This chapter will be dedicated to detail the experiments performed in this project, while also visualizing and reasoning over the obtained results.

The experiments on this project aim to verify how the performance of current models used for predicting churn compares to a newer approach like LSTMs, and also explore how can we influence the scores of our predictor by changing different aspects of the input data. The experiments in this project are:

- Comparing LSTMs to baseline models commonly used for churn prediction
- Changing the size of the prediction and observation windows
- Exploring how the class balance between churning and retaining users influences performance
- Verifying if reducing the dimensionality of the data can increase the scores of the predictor models

As to avoid overwhelming the reader with data, the main results will be presented only, while the more extensive data can be thoroughly explored in Appendix A. Following are some details regarding all experiments.

- The precision, recall and F1-score metrics were all obtained by thresholding the output scores to 50%, which makes no assumption of whether the provider would rather decrease precision in favor of recall or vice-versa. Precision-recall and ROC curves plot however its corresponding metrics against all possible thresholds.
- Unless explicitly stated, the plots will show the metrics for the churned class, which is the focus of this project.

6.1 LSTM vs. baseline models

In this experiment, our goal is to test the performance of the LSTM model compared against the current state of the art in churn prediction. The pilot study has revealed that logistic

		F1-Score	PR AUC	Precision	Recall
LSTM	Retained	**0.901823**	**0.994568**	**0.992342**	**0.826438**
	Churned	**0.173997**	0.204015	**0.098445**	**0.748219**
Logistic Regression	Retained	0.831038	0.992812	0.991091	0.715492
	Churned	0.114972	0.105123	0.062285	0.746072
Random Forest	Retained	0.897689	0.994356	0.991901	0.819821
	Churned	0.166283	**0.204470**	0.093734	0.735728

Table 6.1: Metrics for the LSTM vs. baseline experiment

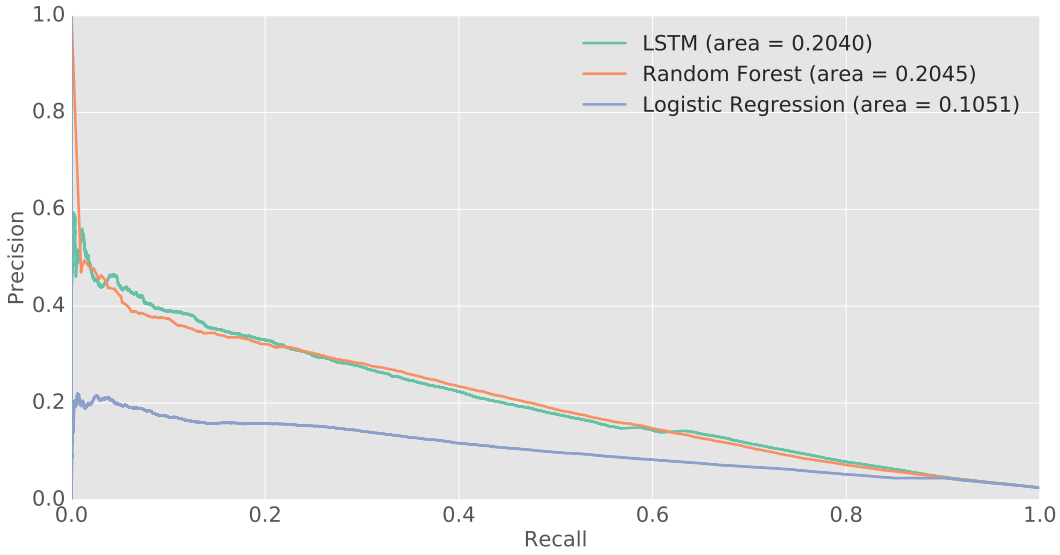

Figure 6.1: Precision-Recall curves for the LSTM and baseline models

regression and random forests are algorithms that are still fairly used in the industry and academia.

All three models were trained on the on the dataset composed of 56 days of observation time and 30 days of prediction, where undersampling was performed on the training data as to have an equal number of churners and retainers.

The results for this experiment shows that random forest has the best overall performance of all evaluated models for almost all metrics in Table 6.1. This follows the results observed during the pilot study of this project. The intuition behind this is that random forests can more easily learn the correlations between the data and the binary label given that proper features are engineered.

6.2 Experimenting on different window sizes

The size chosen for the time windows may have an influence on the performance metrics of our trained estimators that is non-trivial to predict. Even though an intuitive guess can be made for the effect of the change, the set of values that maximizes the accuracy of the classifiers while being compliant to the company's goals and also the available computing resources is *a priori* unknown. This experiment attempts to answer this question by training several LSTM models while changing the observation and the prediction window sizes, and

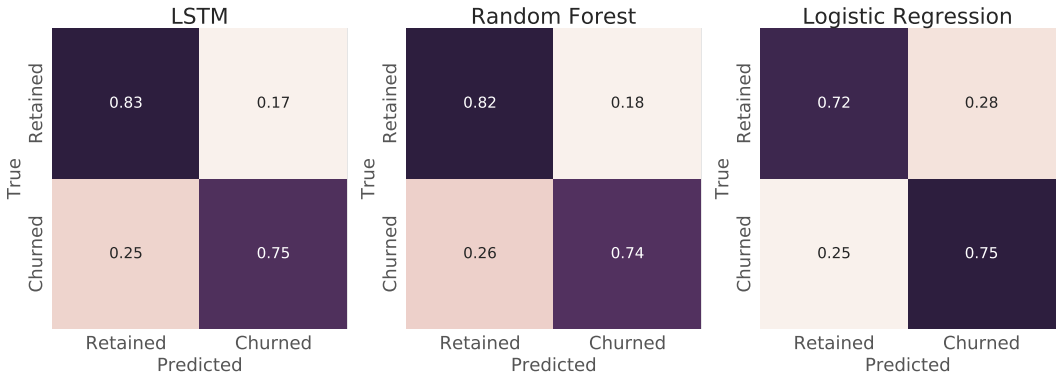

Figure 6.2: Normalized confusion matrices for the LSTM and baseline models

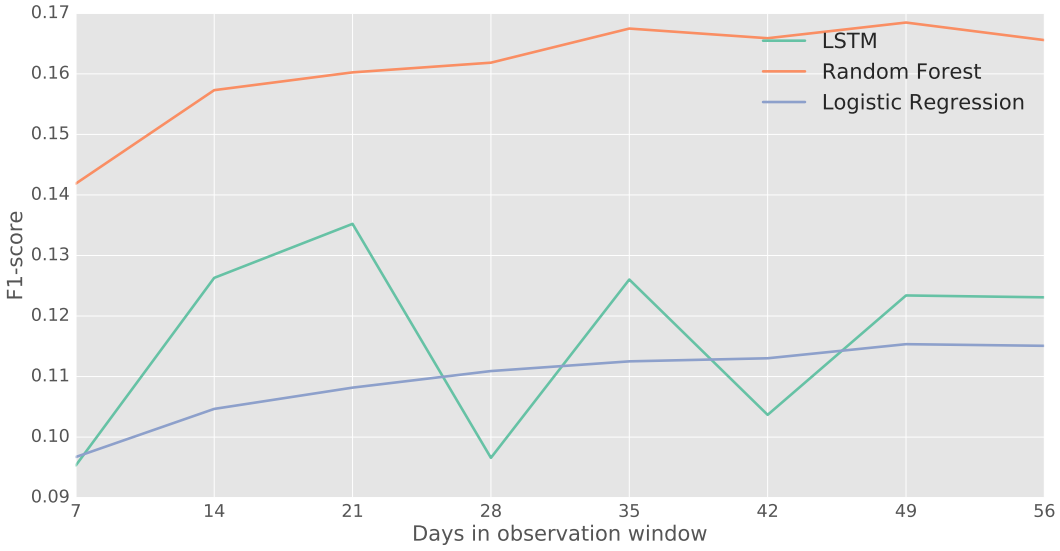

Figure 6.3: F1-scores for the churned class using different observation window sizes and models

by plotting the best score obtained for each estimator.

6.2.1 Observation Window

Increasing the size of the observation window basically means that more data is going to be used for training. It is expected that with more data, a better accuracy can be achieved, however that comes with the cost of an ever increasing training time. So the question that this experiment is exploring is how far in the user history must we train our models on as to obtain a good trade-off between accuracy and training time, a topic thoroughly explored in the work of [11].

Our experiment involved increasing the size of the observation window by 7 days while keeping the size of the prediction window constant in 30 days, the resulting scores can be seen in Figure 6.3 and Figure 6.4.

Both the random forest and logistic regression models exhibit an increase in F1-scores when the size of the observation window is made larger, however the gain can hardly be considered significant. The LSTM model on the other hand show a rather erratic behavior,

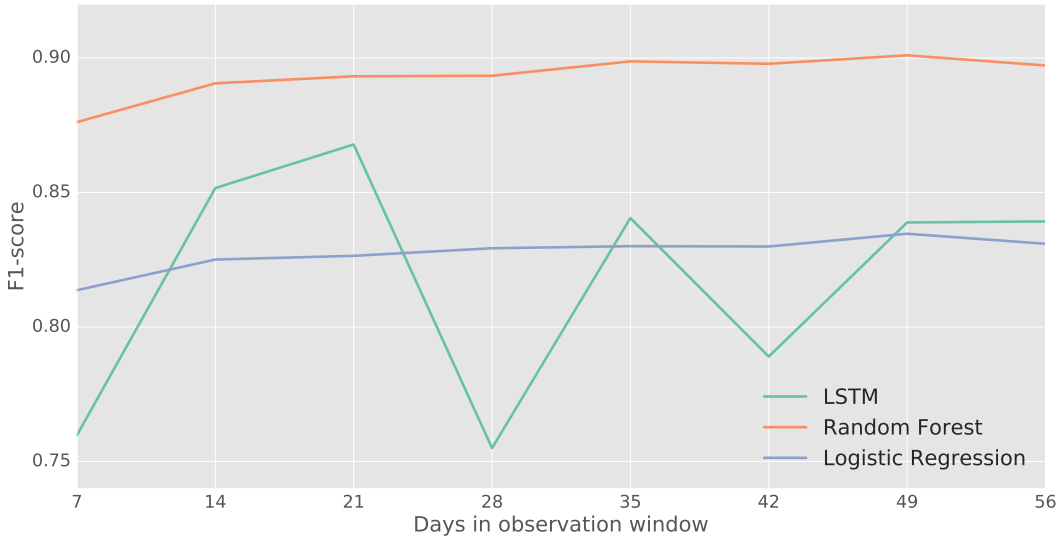

Figure 6.4: F1-scores for the retained class using different observation window sizes and models

reaching its top when the window is sized at 21 days.

6.2.2 Prediction Window

Modifying the size of the prediction window indicates how tolerant we are for considering a user a churner or not. Small windows are commonly used for services lacking a formal contract between the provider and the customer, which is similar approach used in the works of [38], [37] and [10]. Larger prediction window sizes are often seem in "classical" service providers like the telecom industry where a contract is commonly established, an approach that is widely seem in the churn prediction literature, [6] and [4] to name a few.

This experiment involves changing the size of the prediction window in intervals of 7 days while keeping the observation window size constant at 56 days. It is important to note that smaller window sizes means that a larger share of users will be considered churners, and due to our undersampling technique this also means that the number of samples used for training will be larger.

The result of this experiment can be seen inFigure 6.6.

To do Reason about the pred window experiment (20)

6.3 Effect of Class Distribution

In this experiment...

6.4 Experimenting with Dimensionality Reduction

This experiment...

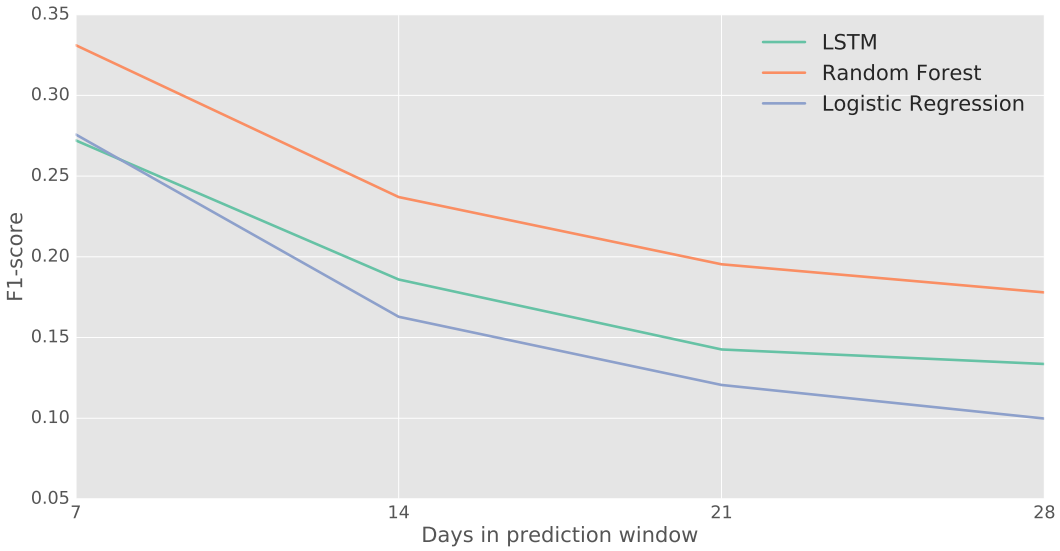

Figure 6.5: F1-scores for the churned class for different prediction window sizes and models

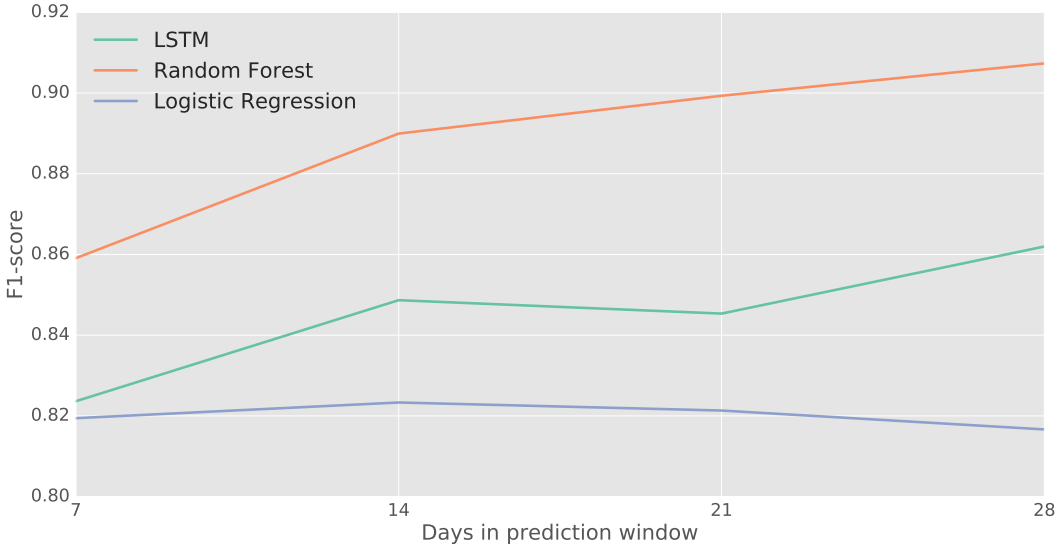

Figure 6.6: F1-scores for the retained class for different prediction window sizes and models

		F1-Score	PR AUC	Precision	Recall
LSTM	Retained	0.872797	0.992778	0.991944	0.779202
	Churned	0.143338	0.152913	0.079239	0.750171
Random Forest	Retained	0.883207	0.992759	0.991383	0.796315
	Churned	0.148800	0.168207	0.082885	0.726749

Table 6.2: Metrics for the dimensionality reduction experiment

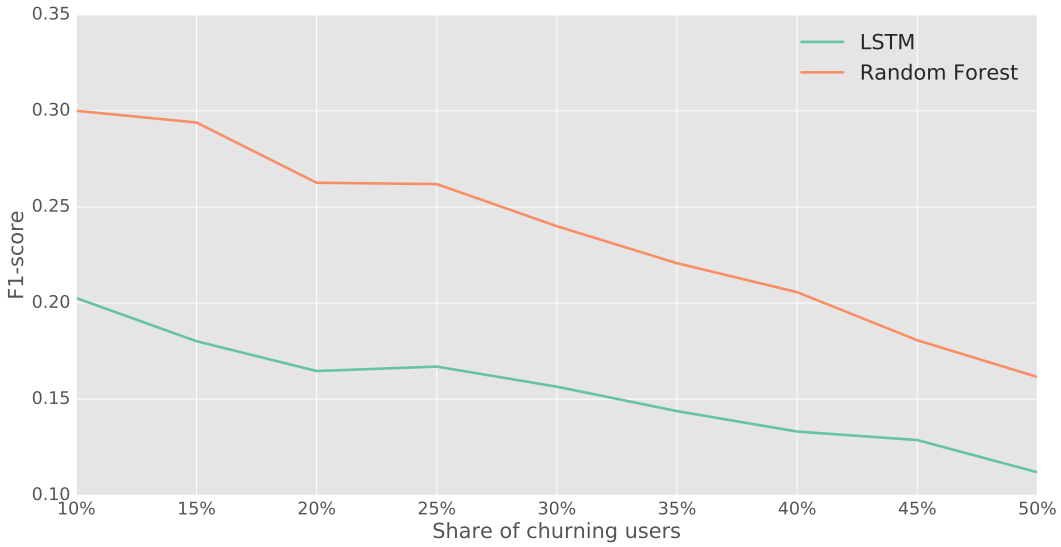

Figure 6.7: F1-scores of the positive churning class for different shares of churning users

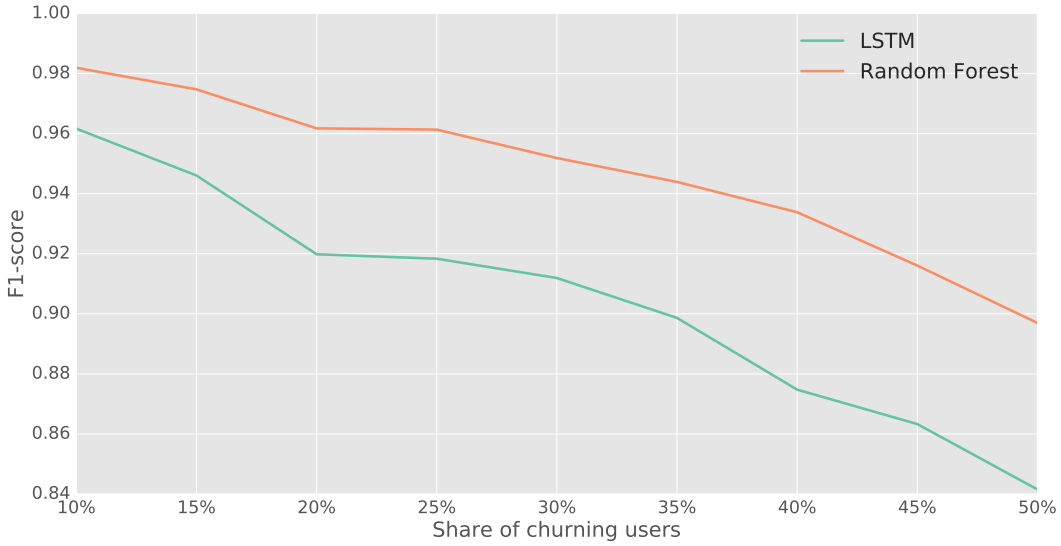

Figure 6.8: F1-scores of the negative retained class for different shares of churning users

Chapter 7

Conclusions and Future Work

* Predicting when the user is churning (Regression LSTM vs. Survival Analysis)
    * Ensemble methods using classifiers that can strongly predict the positive and negative classes

Bibliography

[1] Netflix Quarterly Earnings. https://ir.netflix.com/results.cfm. Acessed: 2017-03-01.

[2] Spotify Press. https://press.spotify.com/us/about/. Accessed: 2017-03-01.

[3] IFPI Global Music Report 2016. http://www.ifpi.org/downloads/GMR2016.pdf. Accessed: 2017-03-01.

[4] Mohammed Hassouna, Ali Tarhini, Tariq Elyas, Mohammad Saeed Aboutrab, United Kingdom, United Kingdom, and Saudi Arabia. Customer Churn in Mobile Markets: A Comparison of Techniques. 8(6):224–237, 2015. doi: 10.5539/ibr.v8n6p224.

[5] Ning Lu, Hua Lin, Jie Lu, and Guangquan Zhang. A customer churn prediction model in telecom industry using boosting. *IEEE Transactions on Industrial Informatics*, 10(2): 1659–1665, 2014. ISSN 15513203. doi: 10.1109/TII.2012.2224355.

[6] Muhammad Raza Khan, Joshua Manoj, Anikate Singh, and Joshua Blumenstock. Behavioral Modeling for Churn Prediction: Early Indicators and Accurate Predictors of Custom Defection and Loyalty. *Proceedings - 2015 IEEE International Congress on Big Data, BigData Congress 2015*, pages 677–680, 2015. doi: 10.1109/BigDataCongress.2015.107.

[7] Vishal Mahajan, Richa Misra, and Renuka Mahajan. Review of data mining techniques for churn prediction in telecom. *Journal of Information and Organizational Sciences*, 39(2): 183–197, 2015.

[8] Özden Gür Ali and Umut Aritürk. Dynamic churn prediction framework with more effective use of rare event data: The case of private banking. *Expert Systems with Applications*, 41(17):7889–7903, 2014. ISSN 09574174. doi: 10.1016/j.eswa.2014.06.018.

[9] David R Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 215–242, 1958.

[10] Jagat Pudipeddi, Leman Akoglu, and Hanghang Tong. User Churn in Focused Question Answering Sites: Characterizations and Prediction. pages 469–474, 2014.

[11] Michel Ballings and Dirk Van Den Poel. Customer event history for churn prediction: How long is long enough? *Expert Systems with Applications*, 39(18):13517–13522, 2012. ISSN 09574174. doi: 10.1016/j.eswa.2012.07.006. URL `http://dx.doi.org/10.1016/j.eswa.2012.07.006`.

[12] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[13] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[14] Kristof Coussement and Koen W De Bock. Customer churn prediction in the online gambling industry: The beneficial effect of ensemble learning. *Journal of Business Research*, 66(9):1629–1636, 2013.

[15] Jonathan Burez and Dirk Van den Poel. Separating financial from commercial customer churn: A modeling step towards resolving the conflict between the sales and credit department. *Expert Systems with Applications*, 35(1):497–514, 2008.

[16] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[17] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[18] Ekachai Phaisangittisagul. An analysis of the regularization between l2 and dropout in single hidden layer neural network. In *Intelligent Systems, Modelling and Simulation (ISMS), 2016 7th International Conference on*, pages 174–179. IEEE, 2016.

[19] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[20] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[21] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[22] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[23] Geoffrey Hinton. Neural networks for machine learning. `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`, 2012. [Online; accessed 01-July-2017].

[24] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[25] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[26] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[27] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[28] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[29] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[30] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[31] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4694–4702, 2015.

[32] Wojciech Zaremba and Ilya Sutskever. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014.

[33] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[34] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[35] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[36] Zoheb Borbora, Jaideep Srivastava, Kuo Wei Hsu, and Dmitri Wil Iams. Churn prediction in MMORPGs using player motivation theories and an ensemble approach. *Proceedings - 2011 IEEE International Conference on Privacy, Security, Risk and Trust and IEEE International Conference on Social Computing, PASSAT/SocialCom 2011*, pages 157–164, 2011. doi: 10.1109/PASSAT/SocialCom.2011.122.

[37] Julian Runge, Peng Gao, Florent Garcin, and Boi Faltings. Churn prediction for high-value players in casual social games. *IEEE Conference on Computatonal Intelligence and Games, CIG*, 2014. ISSN 23254289. doi: 10.1109/CIG.2014.6932875.

[38] Gideon Dror, Dan Pelleg, Oleg Rokhlenko, and Idan Szpektor. Churn prediction in new users of Yahoo! answers. *Proceedings of the 21st International Conference on World Wide Web*, pages 829–834, 2012.

[39] Vladislav Lazarov and Marius Capota. Churn prediction. *Bus. Anal. Course. TUM Comput. Sci*, 2007.

[40] Anders Drachen, Eric Thurston Lundquist, Yungjen Kung, Pranav Simha Rao, Diego Klabjan, Rafet Sifa, and Julian Runge. Rapid prediction of player retention in free-to-play mobile games. *CoRR*, abs/1607.03202, 2016.

[41] Artit Wangperawong, Cyrille Brun, Dr. Olav Laudy, and Rujikorn Pavasuthipaisit. Churn analysis using deep convolutional neural networks and autoencoders. pages 1–6, 2016.

[42] Kristof Coussement and Dirk Van den Poel. Churn prediction in subscription services: An application of support vector machines while comparing two parameter-selection techniques. *Expert systems with applications*, 34(1):313–327, 2008.

[43] J. Burez and D. Van den Poel. Handling class imbalance in customer churn prediction. *Expert Systems with Applications*, 36(3 PART 1):4626–4636, 2009. ISSN 09574174. doi: 10.1016/j.eswa.2008.05.027. URL http://dx.doi.org/10.1016/j.eswa.2008.05.027.

[44] Gaurangi Anand Auon, Haidar Kazmi, Pankaj Malhotra, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Deep Temporal Features to Predict Repeat Buyers. (February 2016), 2015.

[45] Martin Längkvist, Lars Karlsson, and Amy Loutfi. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42(1): 11–24, 2014. ISSN 01678655. doi: 10.1016/j.patrec.2014.01.008.

[46] Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. Predictive Business Process Monitoring with LSTM Neural Networks. 2016. URL http://arxiv.org/abs/1612.02130.

[47] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[48] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

[49] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[50] Boxun Zhang, Gunnar Kreitz, Marcus Isaksson, Javier Ubillos, Guido Urdaneta, Johan a. Pouwelse, and Dick Epema. Understanding user behavior in Spotify. *2013 Proceedings IEEE INFOCOM*, pages 220–224, 2013. ISSN 0743-166X. doi: 10.1109/INFCOM.2013.6566767. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6566767.

[51] Kazunori Sato. An inside look at google bigquery. *White paper, URL: https://cloud. google. com/files/BigQueryTechnicalWP. pdf*, 2012.

[52] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[53] Spotify. Scio: A Scala API for Apache Beam and Google Cloud Dataflow. http://spotify.github.io/scio, 2015. [Online; accessed 01-July-2017].

[54] Charles X Ling and Chenghui Li. Data mining for direct marketing: Problems and solutions. In *KDD*, volume 98, pages 73–79, 1998.

[55] Mervyn Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the royal statistical society. Series B (Methodological)*, pages 111–147, 1974.

[56] Gavin C Cawley and Nicola LC Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11 (Jul):2079–2107, 2010.

[57] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th international conference on Machine learning*, pages 473–480. ACM, 2007.

[58] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[59] Aydin Buluç, Jeremy T Fineman, Matteo Frigo, John R Gilbert, and Charles E Leiserson. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, pages 233–244. ACM, 2009.

[60] Douglas Adams. The hitchhiker's guide to the galaxy. 1995.

[61] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015. [Online; accessed 01-July-2017].

[62] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[63] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[64] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017. URL `http://jmlr.org/papers/v18/16-365.html`.

Appendix A

Additional Results

A.1 Precision-recall curves

A.2 ROC curves

A.3 Metrics tables

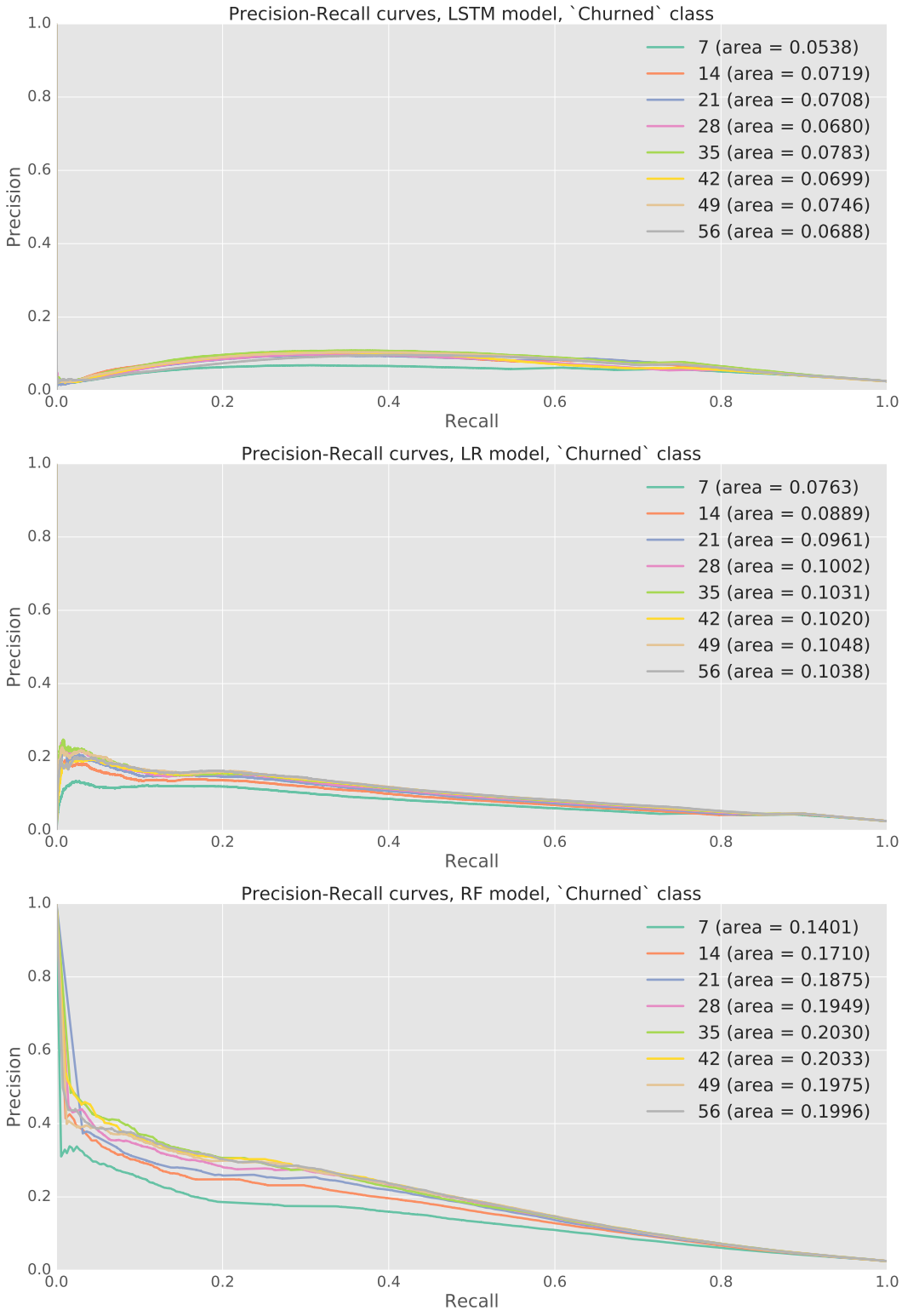

Figure A.1: Precision-recall curves of the churned class for the observation window experiment. The models are LSTM, logistic regression and random forest, respectively

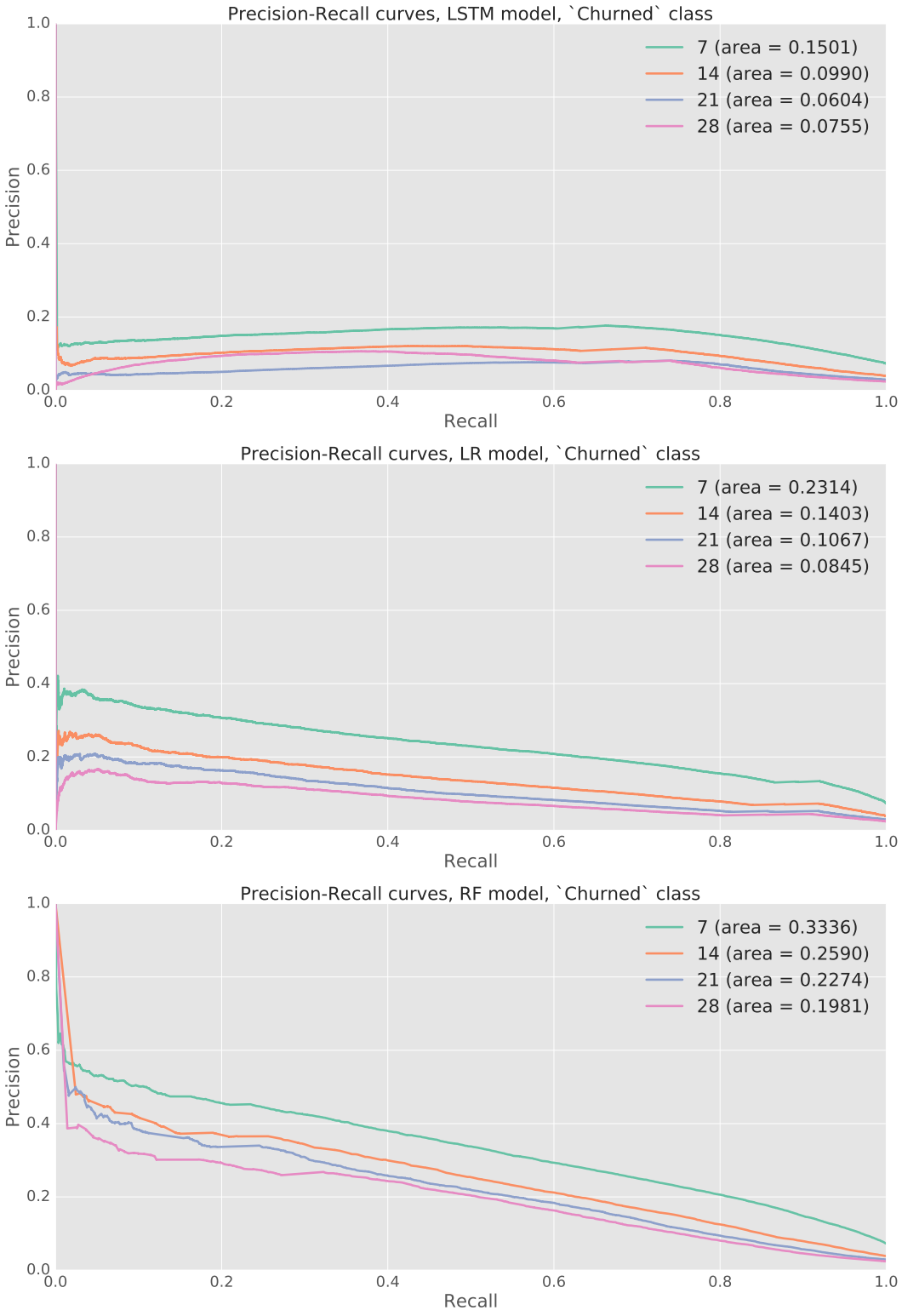

Figure A.2: Precision-recall curves of the churned class for the prediction window experiment. The models are LSTM, logistic regression and random forest, respectively

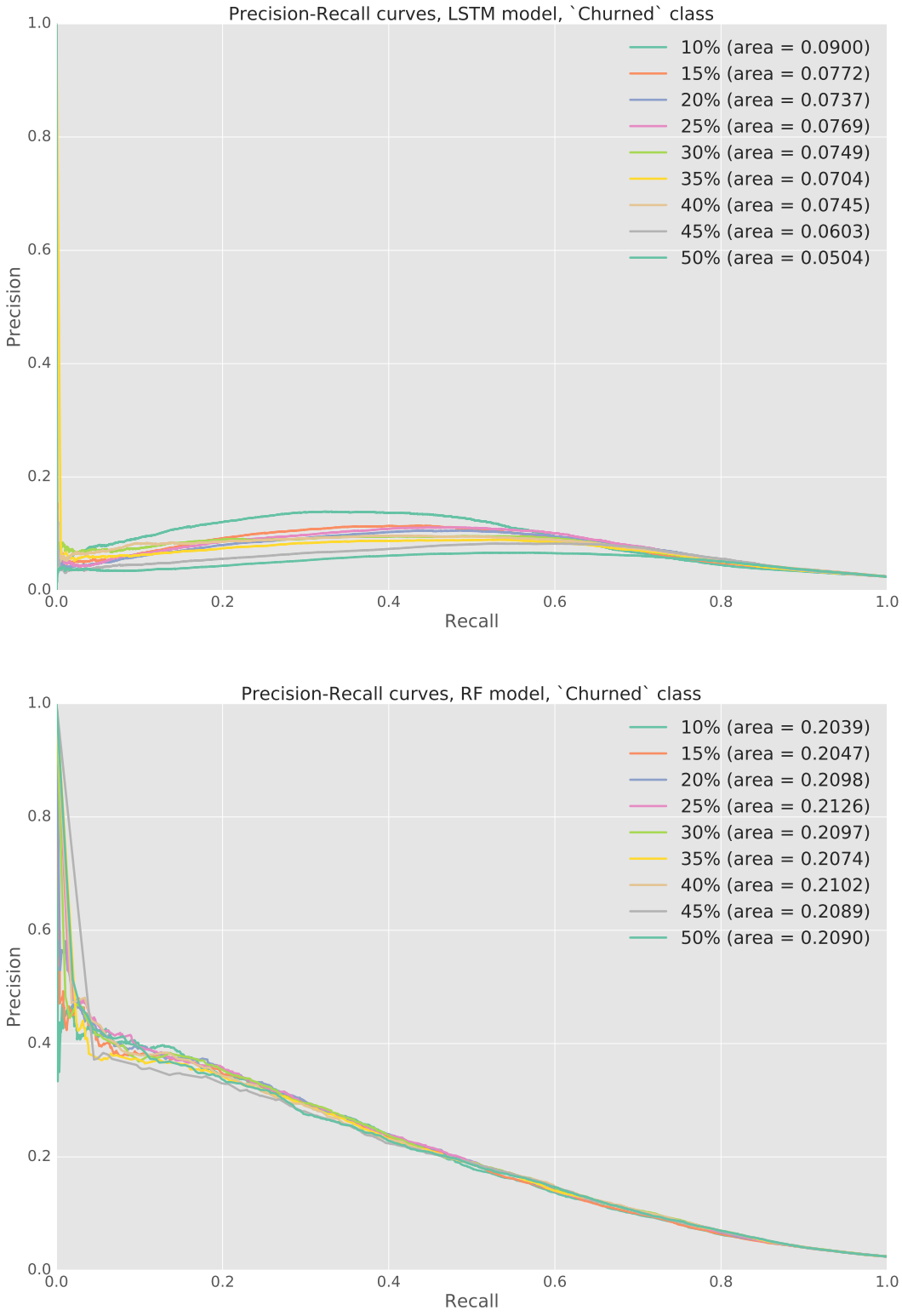

Figure A.3: Precision-recall curves of the churned class for the class balance experiment. The models are LSTM and random forest, respectively

			F1-Score	PR AUC	Precision	Recall
LSTM	10%	Retained	*0.961646*	0.991275	0.984150	*0.940148*
		Churned	*0.202574*	*0.090036*	*0.137299*	0.386163
	15%	Retained	0.946061	0.991598	0.985619	0.909556
		Churned	0.180157	0.077199	0.111897	0.461973
	20%	Retained	0.919837	0.991991	0.988625	0.859998
		Churned	0.164649	0.073727	0.095445	0.598871
	25%	Retained	0.918358	0.992326	0.989153	0.857021
		Churned	0.166948	0.076928	0.096486	0.618989
	30%	Retained	0.911930	0.992033	0.988888	0.846085
		Churned	0.156494	0.074946	0.089663	0.614573
	35%	Retained	0.898646	0.991438	0.989102	0.823349
		Churned	0.143794	0.070388	0.081122	0.632237
	40%	Retained	0.874766	0.992236	0.990585	0.783195
		Churned	0.133156	0.074518	0.073595	0.698234
	45%	Retained	0.863309	0.991402	0.991214	0.764641
		Churned	0.128737	0.060313	0.070638	0.725221
	50%	Retained	0.839230	*0.992415*	**0.992118**	0.727171
		Churned	0.123085	0.068765	0.066874	**0.771933**
Random Forest	10%	Retained	**0.981921**	0.993655	0.983035	**0.980810**
		Churned	**0.300023**	0.203894	**0.287416**	0.313788
	15%	Retained	0.974738	0.993544	0.985423	0.964282
		Churned	0.293922	0.204665	0.225561	0.421737
	20%	Retained	0.961760	0.993784	0.987977	0.936898
		Churned	0.262594	0.209803	0.173706	0.537782
	25%	Retained	0.961340	0.993670	0.988060	0.936027
		Churned	0.261896	**0.212639**	0.172719	0.541462
	30%	Retained	0.951873	0.993767	0.989192	0.917266
		Churned	0.239996	0.209713	0.150395	0.593719
	35%	Retained	0.943922	0.993684	0.989599	0.902276
		Churned	0.220736	0.207393	0.134480	0.615554
	40%	Retained	0.933831	0.993775	0.990513	0.883286
		Churned	0.205684	0.210247	0.121927	0.657017
	45%	Retained	0.916083	0.993626	0.991306	0.851471
		Churned	0.180668	0.208895	0.103779	0.697252
	50%	Retained	0.897189	**0.994240**	*0.991878*	0.819002
		Churned	0.165577	0.199575	0.093293	*0.735240*

Table A.1: Metrics for the class balance experiment

			F1-Score	PR AUC	Precision	Recall
LSTM	7	Retained	0.823630	0.974100	0.971157	0.715013
		Churned	*0.272094*	*0.150116*	*0.167242*	0.729376
	14	Retained	0.848669	0.987558	0.986669	0.744534
		Churned	0.185941	0.099012	0.106108	0.750910
	21	Retained	0.845348	0.989559	0.990381	0.737367
		Churned	0.142604	0.060440	0.078705	0.758035
	28	Retained	*0.861970*	*0.992292*	*0.992303*	*0.761899*
		Churned	0.133607	0.075458	0.073233	*0.760976*
Logistic Regression	7	Retained	0.819430	0.980246	0.973687	*0.707365*
		Churned	*0.275769*	*0.231384*	*0.168623*	*0.756390*
	14	Retained	*0.823319*	0.988834	0.984885	0.707291
		Churned	0.162862	0.140317	0.091636	0.731201
	21	Retained	0.821328	0.991033	0.987873	0.702837
		Churned	0.120611	0.106732	0.065916	0.708495
	28	Retained	0.816645	*0.992240*	*0.989444*	0.695228
		Churned	0.099809	0.084523	0.053735	0.700000
Random Forest	7	Retained	0.859076	0.984100	0.979099	0.765265
		Churned	**0.331079**	**0.333607**	**0.209295**	**0.791815**
	14	Retained	0.889945	0.991412	0.988637	0.809169
		Churned	0.237010	0.259038	0.140071	0.769709
	21	Retained	0.899322	0.993527	0.991293	0.822968
		Churned	0.195370	0.227368	0.112185	0.755783
	28	Retained	**0.907344**	**0.994559**	**0.992572**	**0.835596**
		Churned	0.177947	0.198052	0.101002	0.747073

Table A.2: Metrics for the prediction window experiment

			F1-Score	PR AUC	Precision	Recall
LSTM	7	Retained	0.759781	0.991025	0.992226	0.615572
		Churned	0.095322	0.053770	0.050642	0.809603
	14	Retained	0.851632	0.992178	0.991327	0.746445
		Churned	0.126301	0.071872	0.069023	0.742168
	21	Retained	*0.867829*	*0.992460*	0.991078	*0.771844*
		Churned	*0.135221*	0.070760	*0.074557*	0.725676
	28	Retained	0.754988	0.991924	**0.993143**	0.608960
		Churned	0.096572	0.067951	0.051253	**0.834000**
	35	Retained	0.840498	0.992837	0.992663	0.728783
		Churned	0.126028	*0.078335*	0.068496	0.787352
	42	Retained	0.789003	0.991724	0.992310	0.654839
		Churned	0.103672	0.069907	0.055429	0.799649
	49	Retained	0.838852	0.992317	0.992239	0.726539
		Churned	0.123394	0.074572	0.067029	0.775642
	56	Retained	0.839230	0.992415	0.992118	0.727171
		Churned	0.123085	0.068765	0.066874	0.771933
Logistic Regression	7	Retained	0.813669	0.991255	0.988039	0.691613
		Churned	0.096712	0.076252	0.052121	0.669464
	14	Retained	0.825072	0.991869	0.989110	0.707703
		Churned	0.104654	0.088905	0.056605	0.692398
	21	Retained	0.826448	0.992184	0.989865	0.709342
		Churned	0.108169	0.096096	0.058522	0.713282
	28	Retained	0.829265	0.992497	0.990268	0.713294
		Churned	0.110907	0.100216	0.060058	0.723236
	35	Retained	0.830033	0.992643	0.990586	0.714266
		Churned	0.112509	0.103061	0.060937	0.732019
	42	Retained	0.829915	0.992647	0.990726	0.714018
		Churned	0.113018	0.101979	0.061208	0.736118
	49	Retained	*0.834664*	0.992744	0.990785	*0.721046*
		Churned	*0.115348*	*0.104808*	*0.062583*	0.735240
	56	Retained	0.830921	*0.992801*	*0.991129*	0.715299
		Churned	0.115075	0.103801	0.062337	*0.747243*
Random Forest	7	Retained	0.876152	0.993256	0.991183	0.785044
		Churned	0.141873	0.140149	0.078638	0.724310
	14	Retained	0.890559	0.993822	0.991668	0.808161
		Churned	0.157307	0.170996	0.088123	0.731922
	21	Retained	0.893164	0.993925	0.991717	0.812427
		Churned	0.160246	0.187477	0.089969	0.732117
	28	Retained	0.893381	0.994163	*0.991941*	0.812637
		Churned	0.161846	0.194900	0.090869	*0.739338*
	35	Retained	0.898717	0.994195	0.991906	0.821534
		Churned	0.167482	0.202993	0.094503	0.735337
	42	Retained	0.897824	0.994100	0.991815	0.820105
		Churned	0.165886	**0.203295**	0.093529	0.732800
	49	Retained	**0.900970**	0.994093	0.991656	**0.825481**
		Churned	**0.168477**	0.197516	**0.095300**	0.725773
	56	Retained	0.897189	**0.994240**	0.991878	0.819002
		Churned	0.165577	0.199575	0.093293	0.735240

Table A.3: Metrics for the observation window experiment

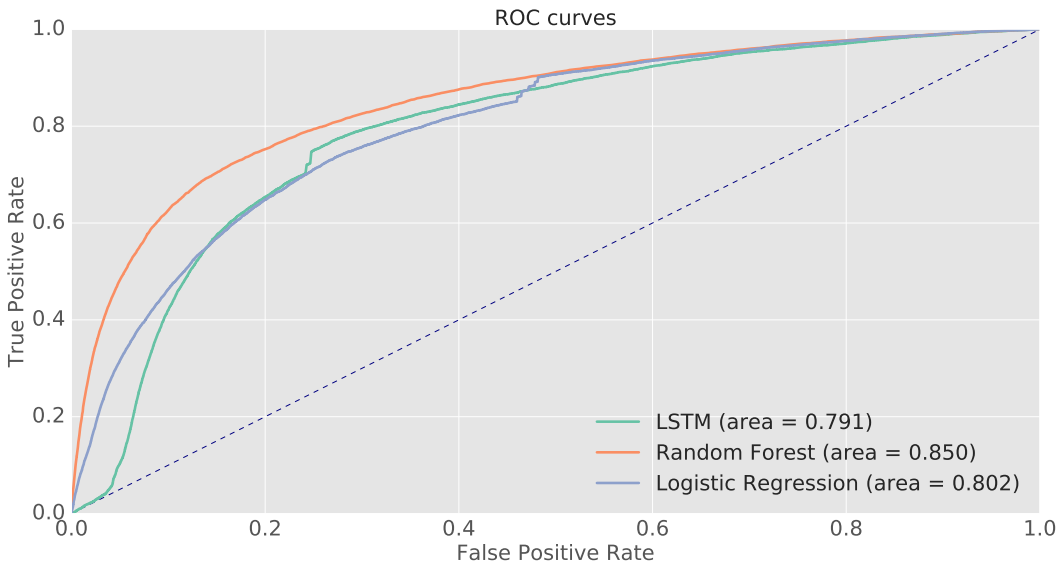

Figure A.4: ROC curves for the LSTM vs. baseline models experiment

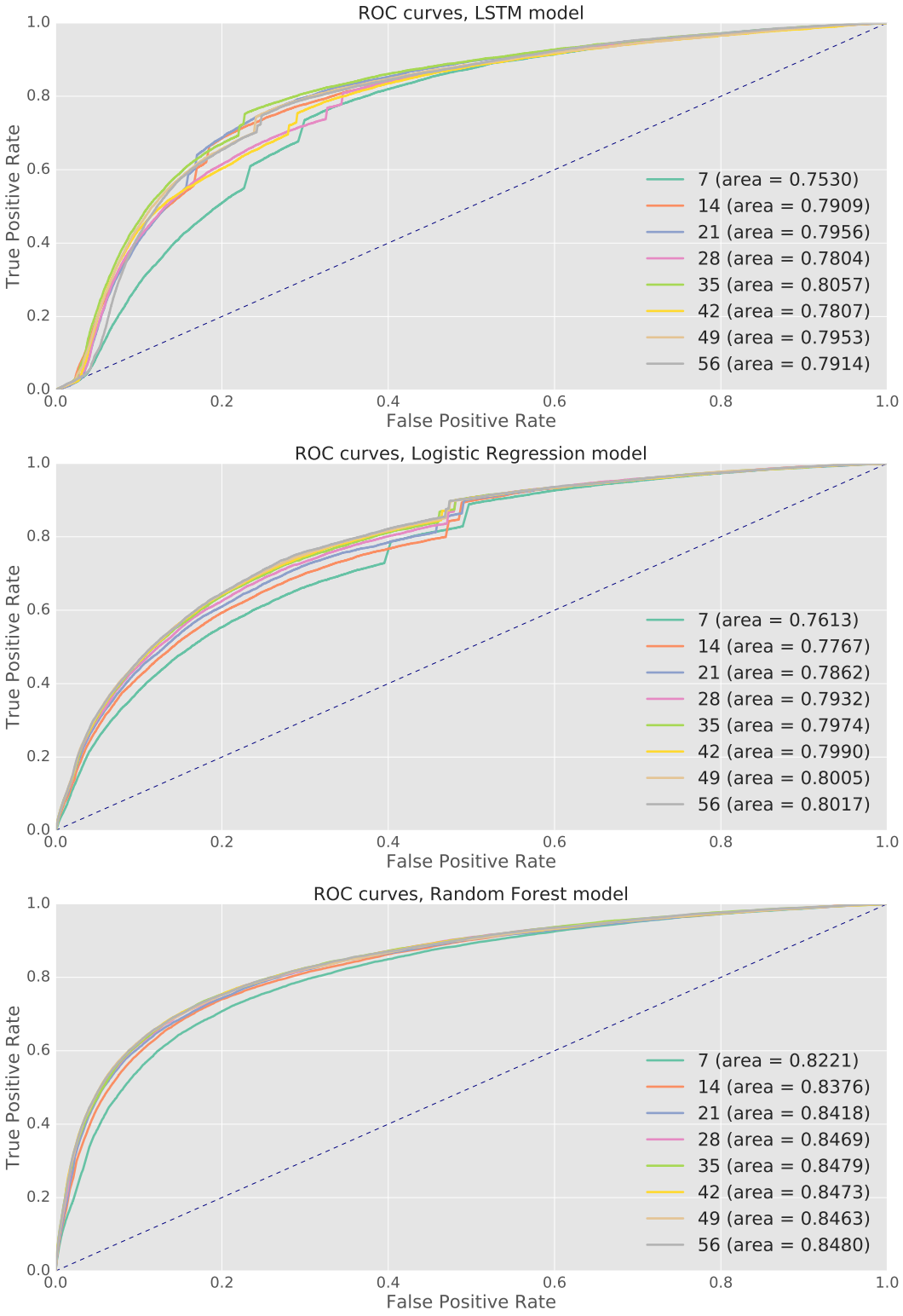

Figure A.5: ROC curves for the observation window experiment

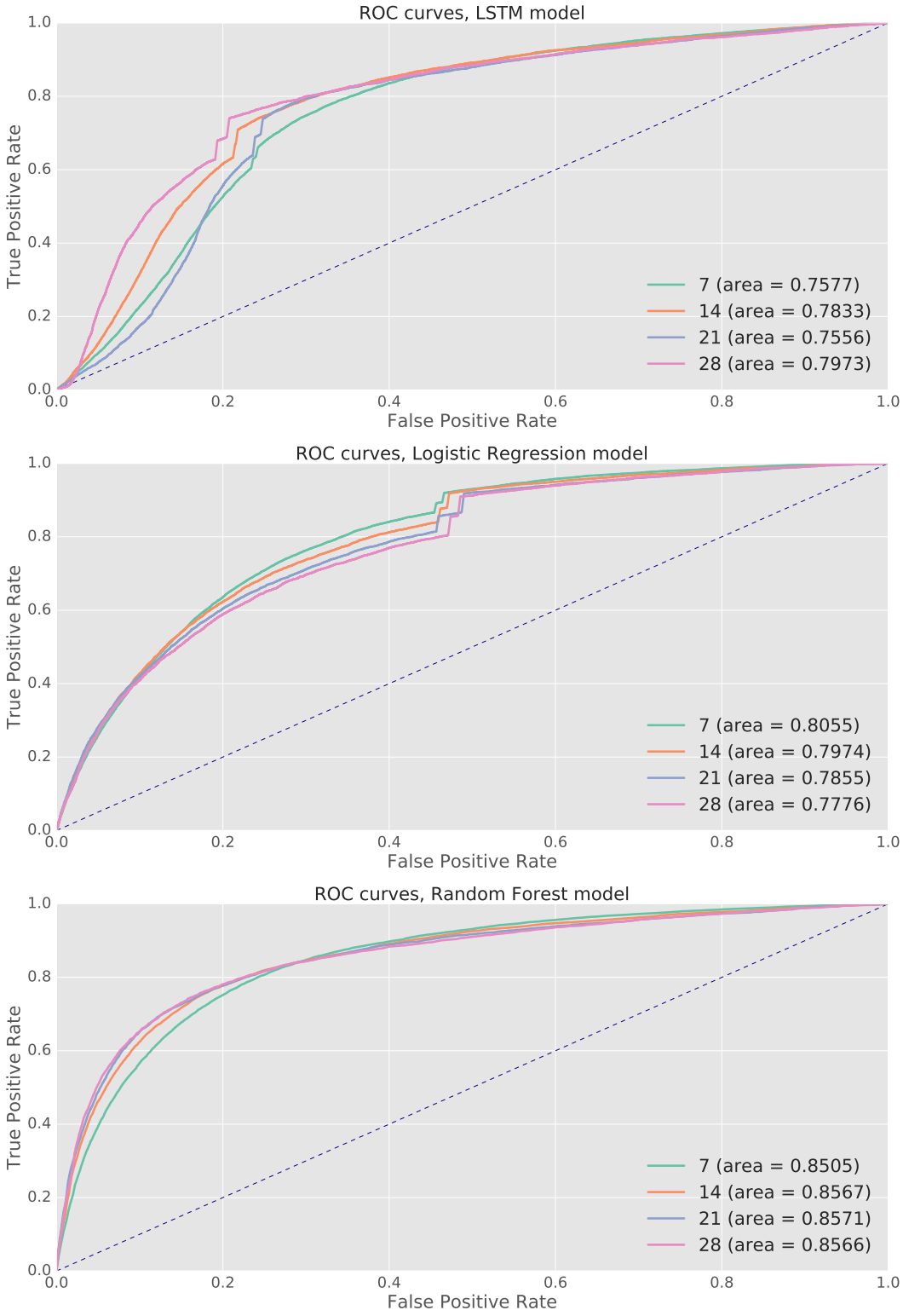

Figure A.6: ROC curves for the prediction window experiment

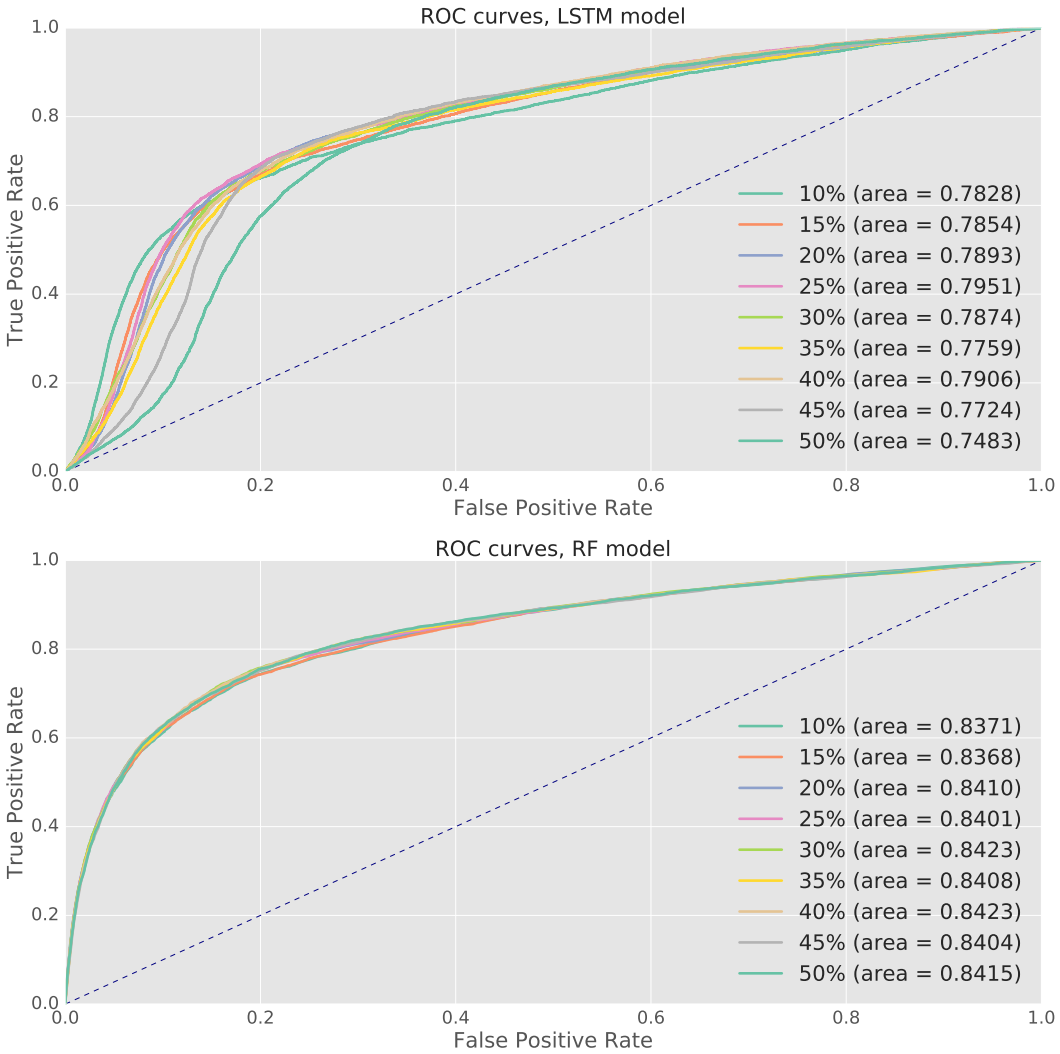

Figure A.7: ROC curves for the class balance experiment

To do...

- ☐ 1 (p. 1): add ref
- ☐ 2 (p. 4): Add this paragraph somewhere else maybe? It seems strange here
- ☐ 3 (p. 5): Improve the research question section, add some bullet points
- ☐ 4 (p. 5): Write ethics section
- ☐ 5 (p. 5): Write the Outline
- ☐ 6 (p. 9): add refs
- ☐ 7 (p. 9): Add logistic function
- ☐ 8 (p. 27): Review the paragraph about countries
- ☐ 9 (p. 28): Check Sahar: This is only for sampling, so will this be true?
- ☐ 10 (p. 28): add ref
- ☐ 11 (p. 28): add ref
- ☐ 12 (p. 30): Add this Future Work
- ☐ 13 (p. 30): improve the zero-filling figure
- ☐ 14 (p. 34): add a ref here from pilot study
- ☐ 15 (p. 35): Ask Sahar why is that
- ☐ 16 (p. 36): Check with Sahar
- ☐ 17 (p. 40): add over undersampling talk in future works
- ☐ 18 (p. 47): Check with Sahar
- ☐ 19 (p. 47): Will I do statistical significance? If so, how?
- ☐ 20 (p. 48): Reason about the pred window experiment