Non-Contractual Churn Prediction with Limited User Information

ANDREAS BRYNOLFSSON BORG
Non-Contractual Churn Prediction with Limited User Information

ANDREAS BRYNOLFSSON BORG
Abstract

This report compares the effectiveness of three statistical methods for predicting defecting viewers in SVT’s video on demand (VOD) services: logistic regression, random forests, and long short-term memory recurrent neural networks (LSTMs). In particular, the report investigates whether or not sequential data consisting of users’ weekly watch histories can be used with LSTMs to achieve better predictive performance than the two other methods. The study found that the best LSTM models did outperform the other methods in terms of precision, recall, F-measure and AUC – but not accuracy. Logistic regression and random forests offered comparable performance results. The models are however subject to several notable limitations, so further research is advised.
Förutsägning av avtalslöst tittaravhopp med begränsad användarinformation

Sammanfattning

## 5 Discussion

5.1 Hyperparameter Search

5.2 Model Performance

5.3 Model Limitations

5.4 Concluding Remarks

5.5 Future Work

## Bibliography

## Appendices

A Grid Search Results

B Model Performance Results

C Feature Set Correlation Matrices
Chapter 1

Introduction

Understanding customer behavior is key for operating a successful business. Competition is fierce in an increasing number of sectors in the modern globalized market [1]. Getting an upper hand on one’s competitors is therefore highly sought after. Economic analysis has repeatedly shown that retaining existing customers is significantly cheaper than attempting to capture new ones. For instance, Reichheld and Sasser [2] studied the effects of companies retaining 5% more of their customers and found that this increased their profits by 25%–85%. A defecting customer does not only represent a potential loss of revenue – they can also become a competitor’s customer. In light of this, other studies have shown that instead of fighting for market share growth, margins and profits can increase significantly more by decreasing defection rates [3, 4]. Reducing the number of defecting customers, who are also referred to as churning customers, is therefore very important for business success.

Knowing that customers are churning at a certain rate is not intrinsically useful. Determining which customers are likely to churn is instead more informative to businesses. By identifying these customers before they actually churn, a company can enact business strategies for incentivizing such customers to stay loyal, for example by targeted marketing, personalized discounts or other promotions [5].

The advent of vast processing power in modern-day computers and huge-scale data collection across Internet-connected devices has facilitated machine learning applications in businesses around the world. These types of algorithms can be used for classifying customers as likely churners or non-churners, and is something that has garnered the attention of both industry and academia [6].

The online video on demand (VOD) industry has seen a rapid adoption rate since the beginning of the 21st century. This is an extremely competitive market that not only competes for subscription fees, but also screen time [7]. Sveriges Television (SVT), Sweden’s public service television broadcasting company, operate several VOD services. These services differ from most concurrent ones in that they are not run with commercial success in mind. Losing viewers to other VOD operators is therefore not associated with any direct loss in profitability, since SVT is tax-funded. The consequence of SVT losing viewers is instead associated with a potential loss in public trust and approval, which could jeopardize the company’s continued operation. Minimizing viewer churn is thus undoubtedly important for SVT.
1.1 Problem Description

Since early 2018 SVT has striven to maintain a steady number of unique weekly viewers. This quantitative metric is one of several internal measures used for estimating the public’s approval of SVT. There are two main components SVT uses in attempting to achieve a stable weekly viewer count. The first is to provide user-relevant content to its viewers, both in terms of what is offered, and recommendation engines for highlighting such content. The second component is to strategically publish content, for instance by releasing episodes on a weekly basis or taking seasonal trends into account. SVT is interested in researching how statistical methods can be used for predicting what content is and will be in demand, as well as when genre-specific content should be published. This is a complex task that requires significant effort and resources to solve.

Further complexity arises from the fact that SVT’s VOD services do not utilize user accounts that viewer sign into (as of 2019). The lack of accounts obfuscates a lot of information about the viewers. It also makes tracking an individual’s viewing history across devices impossible.

A possible approach for working toward SVT’s VOD viewership goal is to decompose it into simpler subproblems that are easier to solve. One clear subproblem that underpins SVT’s objective is the task of predicting which of their VOD viewers are likely to churn in the foreseeable future. The purpose of this report is to lay the foundation for how SVT can pursue their goal. This report will investigate how data analysis and statistical methods can be used to classify users as churners or non-churners. The information gained from these results can presumably be used for further analysis, which can help meet SVT’s larger goal. Clustering methods can for instance be used to try to identify common traits that churners share when it comes to watch histories, types of watched programs etc. In turn, such clusters can be used as a tool for deciding what content should be published or procured. Provided that this can be done on a sufficiently long horizon, SVT can hopefully take proactive measures to maximize viewer satisfaction and reduce churn.

Most customer behavior is non-static, i.e. it changes over time. Standard methods for predicting user churn, such as logistic regression and random forests, are usually not compatible with temporal aspects of customer interaction. Instead any sequential data is often aggregated into static data, which incurs a loss of potentially useful information [8]. Investigating how sequential data can be incorporated into user churn prediction is therefore relevant in the scope of this problem. Recurrent neural networks (RNNs) are capable of processing sequential data, with demonstrated good churn prediction results [9]. Whether or not equally impressive results can be achieved in a setting where user information is restricted has yet to be studied.
1.2 Research Question

In the context of the problem statement presented in section 1.1, this study concerns the following research question.

*dHow does the predictive performance of a recurrent neural network fed sequential data compare to that of logistic regression and random forests in a setting where viewer information is limited?*

1.3 Delimitations

Due to time constraints, it is natural that the scope of the project has to be adjusted accordingly. Among the plethora of existing machine learning techniques available, only a selected number of them will be investigated and compared. Logistic regression and random forests are methods which are relatively simple and widely applied in the field of user churn prediction [10]. They will be used in this report as benchmarks, as they have shown to perform well in a wide variety of business sectors, for instance in the telecommunications [11], financial [12], and media markets [13]. A more advanced method will also be investigated, namely RNNs. Such networks can learn from sequential data and will be trained, evaluated and compared to the benchmark methods.

Each of the methods will be tasked with binary classification, i.e. categorizing viewers as churners or non-churners depending on whether or not they are likely to stop using SVT’s VOD services in the next few weeks. Ideally one might prefer to obtain a prediction of *when* in the upcoming weeks a particular viewer is expected to churn, but this is a task in and of itself and is left as a suggestion for future research.

Only data pertaining to SVT’s kids-oriented content, *SVT Barn*, will be used in this project. This is done due to the fact that these shows are equipped with metadata concerning their content, which is of interest in any clustering to be done in the future. The implemented methods should however be general enough so that they may later be used for prediction across all of SVT’s VOD services and programs.

1.4 Report Outline

The remainder of this report is structured as follows. Chapter 2 presents the mathematical and algorithmic background the project is based on. Previous work in the field is also summarized. Chapter 3 contains the methodology used in attempting to answer the posed research question – which includes the pre-processing of data, training of models and validation strategies. This is followed by chapter 4 displaying the results of the project. What follows is a discussion of the obtained results in chapter 5. The conclusions of the project are also presented, along with suggestions for further research in the area.
Chapter 2

Background

Some terminology, theory and concepts must be established in order to better understand the remainder of this report. This chapter summarizes the most necessary information. Mathematical concepts that underpin statistical learning are introduced and the classification algorithms used in this project are explained. User churn, evaluation metrics, and techniques for improving the performance of statistical methods are outlined. The chapter also summarizes related research in the field of user churn prediction.

2.1 Binary Classification

Classification is the task of using data as input to predict a qualitative response output [14]. Binary classification is a special case of classification, where the response variable can only take on one of two possible outcomes. Despite its relative simplicity, binary classification is an important problem in statistical learning [15].

Formally speaking, binary classification is the process of taking members of a set of observations \( \mathcal{X} \) and classifying them as members of one of two possible classes. The members of \( \mathcal{X} \) can range from scalars to vectors and can contain both quantitative and categorical data. Throughout this report the observations \( X \in \mathcal{X} \) will all be vectors of real numbers of length \( d > 1 \), \( d \in \mathbb{N} \) i.e. \( X = (X_1, \ldots, X_d) \), \( X_i \in \mathbb{R} \).

In practice this means that each observation is assigned a label signifying its class affiliation. These class labels are often the members of a set such as \( \mathcal{Y} = \{0, 1\} \), where the elements \( Y \in \mathcal{Y} \) encode some mutually exclusive and collectively exhaustive categorical data (e.g. yes/no, churner/non-churner). Frequently the classes corresponding to each label are referred to as the negative and positive, respectively. In this report the negative class consists of the observations whose label is 0, and the positive class those whose label is 1. Moreover, the positive class is defined to consist of churners, and the negative class is made up of non-churners.

The working assumption is that there exists a relationship between \( Y \) and \( X \) and labeling is accomplished through a classification rule \( f : \mathcal{X} \to \mathcal{Y} \), expressed as

\[
Y = f(X), \quad Y \in \mathcal{Y}, \ X \in \mathcal{X}.
\]  

(2.1)

In cases where \( f \) is not analytically derivable, approximations of the classification rule can be used instead. This is what statistical methods are for.
2.2 Statistical Methods

The fundamental idea behind statistical learning is to be able to use known, observed data in order to infer the response caused by new, unobserved data. In a binary classification setting, this boils down to using data and algorithms to construct a classification rule \( \hat{f} \) that approximates the behavior of \( f \) \cite{14}. To distinguish the predicted class labels produced by \( \hat{f} \) from the true labels \( y \) given by \( f \), the notation \( \hat{y} = \hat{f}(x) \) is often used. Ideally \( \hat{f} \) should be able to assign the correct labels, i.e.

\[
\hat{f}(x) = \hat{y} = y = f(x)
\]  

(2.2)

to the greatest extent possible. Gauging the predictive performance of \( \hat{f} \) is itself a vast area of research \cite{16}, which will only be briefly discussed in this report. As most measurements are subject to both systematic and random errors, and seldom all possible information is available, estimating an \( \hat{f} \) that perfectly follows \( f \) is in general impossible \cite{14}. Technical and algorithmic limitations also restrict the efficacy of the constructed classification rule.

Classification rules are typically constructed using a statistical method as a framework. These methods define how input data is used, learning is done, and class labels are predicted. Some statistical methods are better suited at particular types of classification problems and input structures than others, so choosing an appropriate one can be crucial for obtaining an \( \hat{f} \) that approximates \( f \) well.

Once a statistical method has been settled on, an instance of the method – a model – is trained and evaluated. Typically this is done by partitioning the dataset into a training, validation and test set, as seen in figure 2.1. The model is given the observations in the training set (without the true class labels), and then tasked with predicting corresponding labels. The true-predicted label pairs are then compared, and the model adjusted with the aim of maximizing some objective function (or minimizing a loss function), based on whether or not the labels match. The validation set is later used for estimating how well the model has been trained, and the test set used for evaluating how well the model generalizes to unseen data \cite{14}.

An important aspect of a statistical method is its variance. Since the methods learn based on what data they are provided, a high-variance statistical method will produce classification rules that can classify the same test observations differently, if the data it is trained on differs between runs \cite{14}. High-variance methods are prone to a phenomenon known as overfitting, i.e. they closely conform to the training data but are poor at generalizing to test data \cite{17}. Variance-reduction techniques can be used to mitigate this, and a few of these will be examined in this project.
What follows are descriptions of the three statistical methods used in this project.

### 2.2.1 Logistic Regression

Despite the nomenclature, logistic regression is a method mostly used for binary classification, rather than regression. It was first described by Cox [18] in 1958 and has since seen applications in many different fields [19]. In the most general sense, logistic regression is a mapping \( \sigma : \mathbb{R}^d \rightarrow [0, 1] \), \( d \in \mathbb{N} \). Using the set \( Y \) from section 2.1, the shorthand notation \( \sigma(Y = 1 \mid X = x) = \sigma(x) \) is often used in binary classification literature [14]. That is, the function \( \sigma(\cdot) \) maps a vector of real numbers to the probability of that observation belonging to the positive class, hence \( \sigma(x) = P(Y = 1 \mid X = x) \). It is mathematically defined using the logistic function, and can be expressed as

\[
\sigma(x) = \frac{1}{1 + e^{-(\beta_0 + \beta x^\top)}} \quad \beta = (\beta_1, \ldots, \beta_d), \ x = (x_1, \ldots, x_d), \ \beta_i, x_i \in \mathbb{R}. \tag{2.3}
\]

Since \( |Y| = 2 \), it follows that \( \sigma(Y = 0 \mid X = x) = 1 - \sigma(x) \). The classification rule is typically constructed such that the observation \( X = x \) is predicted to belong to the positive class if \( \sigma(x) \geq h \), for some \( 0 \leq h \leq 1 \). If \( \sigma(x) < h \) the observation is instead classified as belonging to the negative class [20]. Figure 2.2 visualizes a simple example with one-dimensional inputs of how the threshold \( h \) determines the classification rule. By tuning the value of \( h \), widely different predictions can be made. The effects of doing this are further explained in section 2.4.

![Figure 2.2: Sample plot of \( \sigma(x) \) for \( \beta_0 = 0, \beta = 10 \). The classification rule is

\[ \hat{y} = \begin{cases} 1, & x \geq 0 \\ 0, & \text{otherwise} \end{cases} \] for \( h = 0.5 \) and \[ \hat{y} = \begin{cases} 1, & x \geq \frac{-\ln(9)}{10} \\ 0, & \text{otherwise} \end{cases} \] for \( h = 0.1 \). The model is trained by fitting the weights \( \beta_0, \ldots, \beta_d \). It is beneficial to penalize large \( \beta \in \beta \) with \( L_2 \) regularization (i.e. minimizing the distance from the origin to \( \beta \) in \( d \)-dimensional space). Regularizing the weights leads to a reduction of variance which makes the model less prone to overfitting, thus better at classifying unseen data [14], as mentioned in section 2.2. In this project the Scikit-learn library for
Python will be used. Its built-in package for logistic regression estimates parameters by minimizing the following $L_2$ penalized cost function

$$\mathcal{L}(\beta_0, \beta) = \min_{\beta_0, \beta} \frac{1}{2} \beta \beta^T + \frac{1}{\lambda} \sum_{i=1}^{n} \log \left( (1 - y(i)) e^{\beta_0 + \beta x(i)} + y(i) e^{-\beta_0 - \beta x(i)} + 1 \right)$$ (2.4)

where $x(i)$ and $y(i)$ denote the $i$th training observation and class label, respectively. The hyperparameter $\lambda$ in equation (2.4) is the regularization strength and can be tuned. Smaller values of $\lambda$ specify stronger regularization, and vice versa [21].

Many of the strengths of logistic regression – reasons for why it is still frequently used in industry and academia – are largely due to the method’s simplicity. It requires relatively little computational resources for training, and the method offers some interpretability since the modeled weights can be thought of in terms of log-odds [20].

2.2.2 Decision Trees and Random Forests

Another popular method for both classification and regression is decision trees. This flowchart-like approach is based on the partitioning of the $d$-dimensional feature space into disjoint regions. Each region is assigned a class label which is based on the training observations that are located in said region [22]. In binary classification this means that a region is labeled 1 if the fraction of positive observations in the region exceeds some threshold $h$, and vice versa. A new observation is given the same class labels as that of the region it is located in.

Partitions of the feature space are performed with the intended goal of making the resulting $R$ regions as class-homogeneous as possible. Class homogeneity is conventionally measured using either the Gini index or cross-entropy [14]. Because it is fast to compute [23] and good at handling two classes [22], the Gini index will be used in this project. It is defined as follows

$$G_r = \sum_{y \in \mathcal{Y}} \hat{p}_{ry}(1 - \hat{p}_{ry})$$ (2.5)

where $\hat{p}_{ry}$ denotes the fraction of training samples in the $r$th region that belong to class $y$. In a binary classification setting, expression (2.5) is minimized when for one $y \in \mathcal{Y}$, $\hat{p}_{ry} \to 1 \implies G_r \to 0$.

Finding partitions that result in $R$ regions with optimal class homogeneity is however computationally unfeasible, and is even for binary classification problems NP-complete [24]. Since the search space for finding an optimal partition rapidly increases with the number of training observations and dimensions, a greedy algorithm known as recursive binary splitting is often used instead [25]. This algorithm works by searching for a feature dimension $X_i$ and breakpoint $b_j$ along that dimension that together maximize class homogeneity in the two hyperrectangles that result from the partition. The process is recursively performed in the hyperrectangles that emerge until a stopping criterion is reached [14]. After $k$ recursive splits the feature space is partitioned into $k + 1$ hyperrectangles.
As hyperrectangles are defined and given class labels, a decision tree analogous to the partitioning can be grown. Decision trees are a family of directed acyclic graphs that consist of $k > 0$ internal nodes that branch off with exactly two children, and $k + 1$ terminal nodes that lack any children. The feature and breakpoint found using recursive binary splitting together form an internal node in the tree, and the hyperrectangles formed by the partition become that node’s children [1]. In essence a decision tree is a set of topologically ordered if-else statements.

Traversal along a tree is accomplished by feeding it input, and starting from the top of the tree systematically evaluating the first condition that is reached along the given path [22]. Depending on the truth value of the condition, one of the two paths that emanate from the branch is taken, and the process is repeated until a terminal node is reached. In a classification setting, terminal nodes are encoded with same class labels as their corresponding hyperrectangle. The classification rule $\hat{f}$ is formulated so that the input observation is assigned the label present in the terminal node it reaches after traversing the tree [14]. The duality between decision trees and hyperrectangles is exemplified in figure 2.3.

![Figure 2.3](image)

Figure 2.3: A two-dimensional feature space partitioned using recursive binary splitting and its corresponding decision tree. A condition evaluated as true leads to the left branch.

Shortcomings of decision trees are their large variances and tendency to overfit [25]. Even small perturbations in training data can lead to widely different splitting criteria since the recursive binary splitting does not look ahead [14]. The selected stopping criterion can also greatly affect the terminal nodes [22]. Random forests is an ensemble learning method that builds on decision trees, and achieves lower variance in the process.

Random forests are constructed by building $T$ number of decision trees and having them together make a prediction. In order to decorrelate the trees and prevent them from being exact duplicates of one another, they are trained on different data [26]. This is in practice done using the bootstrap, i.e. randomly drawing with replacement $n$ observations from the training set $T$ times to obtain $T$ bootstrapped datasets. As an added means of decorrelating the trees, whenever a new splitting criterion is to be introduced, only a random subset of $\delta < d$ number of features will be considered.
The reasoning behind this procedure is that if there exists a few important features in the feature space, most trees would make their first splits using one of those features [14]. The resulting trees would in effect become similar to each other, hence very correlated [26]. Typically $\delta$ is set to be rather small compared to $d$, and a good rule of thumb for classification according to literature is letting $\delta = \lfloor \sqrt{d} \rfloor$ [14].

A random forest for classification makes predictions on new data by feeding the grown decision trees the same observation $x$ and tallying the output of the corresponding classification rules $\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_T$. The tallied results are then combined to give a final class prediction. This is commonly decided by a plurality vote given by the expression

$$\hat{y} = \arg \max_{y \in \mathcal{Y}} \sum_{t=1}^{T} \mathbb{1}_{\hat{f}_t(x)} = y$$

(2.6)

where $\mathbb{1}_{(\cdot)}$ is the indicator function [26]. In the case of binary classification, this just amounts to a simple majority vote.

Random forests have shown to perform well in a variety of settings, even without any tuning of the hyperparameters $T$ and $\delta$, or the stopping criterion for growing the decision trees [25]. Uncorrelated trees average out each other’s errors, thereby reducing the variance. The effect is amplified as $T$ increases, without the risk of overfitting, at the cost of additional computational resources needed to construct decision trees [14]. In this project $T = 500$ will be used.

Tuning $\delta$ is more case-specific. If many features exhibit strong correlation it may be beneficial to set $\delta$ conservatively to increase the decorrelating effect. At the same time, picking too small a $\delta$ can lead to poor recursive splits, leading to potentially detrimental effects on predictive performance [26]. Standard search and validation algorithms can be used to find values of $\delta$ that give good performance.

### 2.2.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a type of artificial neural networks (ANNs) suited for handling sequential data. Several variations of RNNs exist. However, for the purpose of this project only one variant will be investigated: the so called many-to-one RNN [27]. As the name implies, this type of RNN takes as input a sequence $X = (X^{(1)}, X^{(2)}, \ldots, X^{(\tau)})$, where $X^{(t)} = (X_1^{(t)}, X_2^{(t)}, \ldots, X_d^{(t)})$, $X_i^{(t)} \in \mathbb{R}$, and produces a single output $\hat{Y}^{(\tau)}$. Since the churn status of a viewer should be given as a single label after observing all of the data, there is no point in outputting predicted labels at every week prior to $t = \tau$. Thus the many-to-one RNN setup is appropriate for the task at hand. A strength of RNNs is their ability to handle inputs of different lengths [28]. This property is desirable since it allows SVT to feed the network watch histories of a varying number of weeks without the need for zero-padding input data to match the sequence length of the training data.

In essence an RNN is an ANN with a recurrent connection to itself, as seen in figure 2.4. At each time step the output of the RNN, $h^{(t)} = (h_1^{(t)}, \ldots, h_L^{(t)})$, $h_i^{(t)} \in \mathbb{R}$ (known
as its hidden state, is fed back into the network at the next time step together with the next input $x^{(t+1)}$. This gives an RNN the ability to remember information about previous entries in the sequence [28]. For visualization purposes the recurrent loop can be unfolded through time and the RNN displayed as a sequence of ANNs (where the parameters are identical for each unit in the sequence [29]), also seen in figure 2.4. The prototypical ANN structure commonly used for RNNs is shown in figure 2.5. In that setting the RNN is given by the following equations

$$h_i^{(t)} = \tanh\left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}\right) \quad (2.7a)$$

$$\hat{y}^{(\tau)} = \text{softmax}(c + V h^{(\tau)}) \quad (2.7b)$$

where $U$, $W$ and $V$ are weight matrices corresponding to the input, hidden state and output layer, respectively. The vectors $b = (b_1, \ldots, b_l)$ and $c = (c_1, \ldots, c_J)$ are the biases corresponding to the hidden state and output layer, respectively [28]. The hyperbolic tangent and softmax functions are defined as follows

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (2.8)$$

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^J e^{z_j}}, \quad i = 1, \ldots, J, \quad z = (z_1, \ldots, z_J) \in \mathbb{R}^J. \quad (2.9)$$

For equation (2.7a) to hold at $t = 1$ the initial hidden state $h^{(0)}$ must be defined. The weight matrices and bias vectors must also be initialized. Random initialization of weight matrices can lead to faster convergence to local optima, conditional on that the scale of the initialization is neither too big nor too small. Bias vectors can however be initialized to zero vectors, with generally good results [28]. In this project all weight matrices and $h^{(0)}$ will be initialized using the Glorot uniform initializer, as that method uses a heuristic for drawing initial weights on an appropriate scale [30]. This follows the suggested default settings from the Keras neural network library used in this project, which also defaults the bias vectors to zero vectors [31].

Figure 2.4: A many-to-one type of RNN. The recurrent loop can be unfolded through time to show how the previous hidden state $h^{(t-1)}$ is fed back into the network together with the current input $x^{(t)}$. A prediction vector $\hat{y}^{(\tau)}$ is output at $t = \tau$ using a fully connected layer and the softmax function.
Figure 2.5: Internals of a standard RNN. The rectangle connecting the previous hidden state $h^{(t-1)}$ and current input $x^{(t)}$ to the elementwise hyperbolic tangent activation function denotes a fully connected layer with a bias vector.

A property of the softmax function is that the components of its output sum to 1. Hence the elements of $\hat{y}^{(\tau)}$ can be interpreted as the probabilities of the observation belonging to each of the $J$ classes (provided that cross-entropy is used as the loss function) [14]. The final step of the classification rule $\hat{f}$ is typically constructed so that the observed sequence $X = x$ is assigned the class label $\hat{y}$ corresponding to the class encoded by the maximum element of $\hat{y}^{(\tau)}$. Threshold conditions that work analogously to those described in section 2.2.1 and section 2.2.2 can also be imposed if the classification is binary.

As with standard ANNs, the components of $U, W, V, b$ and $c$ of the RNN are learned using a loss function and an optimizer. As mentioned, the binary cross-entropy loss function is appropriate in this setting seeing as the resulting output $\hat{y}^{(\tau)}$ will correspond to class probabilities. The Adam optimizer has proven to be both fast and achieve formidable results. Adam is also capable of learning its learning rates by itself, in stark contrast to regular stochastic gradient descent which requires careful tuning, and is found to be well-suited to a wide range of problems [32]. In this project the Adam optimizer will be used with the default parameter settings as proposed by its authors.

Even though RNNs are capable of remembering dependencies in their inputs of arbitrary length, in practice this is seldom feasible. During the backpropagation step of the optimization algorithm, gradients can vanish toward zero or explode toward $\pm \infty$ [33]. Vanishing gradients lead to no learning, whereas exploding gradients cause unstable learning [34]. These phenomena are exacerbated as the length of the input sequences increase, since the weights between hidden-to-hidden layers of the RNN are repeatedly self-multiplied [29]. For this reason RNNs are rarely used as-is since the dependencies they can learn without gradient-related issues tend to be of inadequate lengths.
Long Short-Term Memory

The long short-term memory (LSTM) model is an RNN that has been popularized in light of the gradient problems of standard RNNs [28]. These networks use a more complex updating rule for its hidden state than equation (2.7a). Instead, LSTMs use an LSTM cell, depicted in figure 2.6. The cell combines information from the network’s previous hidden state, its current input, and an additional component known as its internal state $s(t)$. The internal state is only modified through linear interactions, which is what allows the network to bypass the vanishing and exploding gradient-issues present in RNNs. This is what enable LSTMs to capture long-term dependencies [29]. The internal state forms an internal recurrence controlled by three gates that control the flow of information in and out of the cell [35].

The forget gate $f(t)$ determines how much of each of the previous internal states should remain, by scaling its elements by values in $(0, 1)$. The greater the values, the more of the old state should persist in the cell, and vice versa. New information is then added to the cell state using the input gate $g(t)$, which works in a similar way as the forget gate. How much of the hidden state should be output is also controlled by a gate – the output gate $q(t)$ [28].

![LSTM cell diagram](image)

Figure 2.6: An LSTM cell. The rectangles connecting $h(t-1)$ and $x(t)$ to the activation functions $\sigma$ denote fully connected layers with corresponding weight matrices and bias vectors. Circular nodes denote elementwise operations.
The LSTM cell is governed by the following set of equations

\[ f_i(t) = \sigma_f \left( b_f + \sum_j U_{ij}^f x_j(t) + \sum_j W_{ij}^f h_j(t-1) \right) \tag{2.10a} \]

\[ g_i(t) = \sigma_g \left( b_g + \sum_j U_{ij}^g x_j(t) + \sum_j W_{ij}^g h_j(t-1) \right) \tag{2.10b} \]

\[ q_i(t) = \sigma_q \left( b_q + \sum_j U_{ij}^q x_j(t) + \sum_j W_{ij}^q h_j(t-1) \right) \tag{2.10c} \]

\[ s_i(t) = f_i(t) s_i(t-1) + g_i(t) \sigma_s \left( b_s + \sum_j U_{ij}^s x_j(t) + \sum_j W_{ij}^s h_j(t-1) \right) \tag{2.10d} \]

\[ h_i(t) = \tanh \left( s_i(t) q_i(t) \right) \tag{2.10e} \]

where the different \( U \) and \( W \) are weight matrices as before, and the \( b \) bias vectors. The choice of activation functions \( \sigma \) can vary, but standard practice is letting \( \sigma_f, \sigma_g, \sigma_q \) be the sigmoid function, and \( \sigma_s \) be the hyperbolic tangent function [29].

The previously mentioned initializers can be used for LSTMs as well, and the vector of class probabilities \( \hat{y}^{(\tau)} \) is obtained in the same way as equation (2.7b). Likewise, class labels are obtained using the same classification rule as for standard RNNs.

### 2.3 User Churn

Because of the many fields churn prediction is used in, the definition of churn in literature is often domain- or even case-specific. Many definitions of churn capture a change in, or loss of, customer engagement and there exists some conventional terminology across the research of user churn. Broadly speaking, user churn can be broken down into two types: **active** and **passive** [36, 37, 38]. Active churn is relatively straightforward to measure and is the result of a user choosing to terminate a contract, delete an account etc. Passive churn on the other hand is more vague and can stem from prolonged user inactivity, failure to pay subscription fees or other implicit indications that a user has lost interest in a service. Clearly passive churn is what SVT is concerned with.

An approach for defining and classifying passive churners, as seen in literature such as Coussement and De Bock [39] and Karnstedt et al. [40] is to introduce three non-overlapping time frames. The partitions are used for observing user behavior, filtering unclassifiable users, and labeling training and test data, as seen in figure 2.7. As the name implies, the observation window (OW) is the period in which user activity is gathered. This window is usually the longest of the three in order to amass sufficient data. The activity window (AW) follows and is used to filter those users that may already have churned in the OW or do not have recent enough activity. This window is usually the shortest of the three. The users with satisfactory amounts of logged activity in both the OW and AW can then be labeled as churners or not depending on if they have enough activity in the churn prediction window (CW). The definition of sufficient activity in this project is given in section 3.1.
Figure 2.7: Churn prediction framework. The CW is only used for labeling users.

With churn labels assigned, statistical methods can be taught to learn what differentiates churners from non-churners. Models are tested and evaluated by predicting the class labels of unseen users and comparing these with their actual labels. At this stage the CW is censored, i.e. only information up to the “current time” ($t_c$) is used, which means that data in the CW is only used for labeling purposes [41]. This is done to mirror the situation when making predictions in real time on unlabeled data.

A drawback of using this method for labeling churners is the potential loss of a significant number of users in the dataset. Since users that have not been sufficiently active in the OW and AW are considered unclassifiable and just dropped, this can make for an inefficient classification framework. The method is also prone to variability in individual users’ labels, depending on how the time frame is partitioned into the different windows. Despite these possible shortcomings, studies that have used this approach have demonstrated promising results [42, 41].

### 2.4 Performance Metrics

Selecting and defining appropriate measures for model performance is an ongoing effort. Some metrics are better suited than others, and some are even domain-specific [43]. In binary classification problems, commonly used evaluation criteria include accuracy, precision, recall and the $F$-measure [44]. They are all expressed using a confusion matrix, which categorizes all predictions into one of four groups based on the true and predicted labels, as seen in figure 2.8.

<table>
<thead>
<tr>
<th>$\hat{y}$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>True negative (TN)</td>
<td>False negative (FN)</td>
</tr>
<tr>
<td>1</td>
<td>False positive (FP)</td>
<td>True positive (TP)</td>
</tr>
</tbody>
</table>

Figure 2.8: A confusion matrix.
The metrics are defined by equations (2.11).

\[
\text{accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.11a}
\]

\[
\text{precision} = \frac{TP}{TP + FP} \tag{2.11b}
\]

\[
\text{recall} = \frac{TP}{TP + FN} \tag{2.11c}
\]

\[
F\text{-measure} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{2.11d}
\]

In other words, accuracy is the fraction of correctly classified observations over the total population. Precision and recall measure the true positives over the sum of predicted positives and total number of positives in the population, respectively. Finally the F-measure is defined as the harmonic mean of precision and recall.

While accuracy might be the most intuitive measure of these four metrics, it has in practice been shown to be potentially misleading [14]. This is especially the case when the distribution between classes is heavily imbalanced. Suppose that there is a 99:1 ratio of negatives to positives in a binary population. A classifier could learn to reach an expected classification accuracy of 99% by predicting all observations as negatives, even though it would never be able to distinguish members of the minority class [28]. Accuracy can therefore be a deceitful metric and should be taken with a grain of salt. For this reason it is seldom used alone in research on predicting user churn, as studied by e.g. Garcia et al. [6] and De Caigny et al. [1].

Precision and recall are tightly coupled. As one of the metrics increase, the other tends to decrease [43]. Depending on the classification problem at hand, either precision or recall may be more important than the other. In such cases an optimization problem where the more important metric is to be maximized can be posed. If neither is deemed more important than the other the F-measure is useful to attempt to maximize instead [28].

The thresholds \( h \) mentioned in section 2.2.1 to section 2.2.3 can be used to create so called precision-recall curves for a learning model. These curves are constructed by letting the threshold value range from 0 to 1, and for every value of \( h \) recording the precision and recall values the model achieves. The obtained set of points are then plotted against each other in \([0,1] \times [0,1]\) [43]. The area under the precision-recall curve (AUC) can also be used as a model evaluation tool. A perfect classifier would produce a precision-recall with an AUC = 1 [45]. For practical reasons the exact AUC is approximated by a finite sum using incrementally changed threshold values \( \vec{h} = (0,h_1,h_2,\ldots,h_k,1) \). The approximation is given by

\[
\text{AUC} = \sum_i (\text{recall}_i - \text{recall}_{i-1}) \times \text{precision}_i \tag{2.12}
\]

where \( \text{precision}_i \) and \( \text{recall}_i \) denote the precision and recall scores obtained using the \( i \)th threshold value in \( \vec{h} \) [21].
2.5 Sampling Strategies

Among many studied markets and companies, the proportion of churners to non-churners is heavily skewed in favor of the non-churners [1, 46]. While user defection being a rare event is beneficial for companies, a consequence of this class imbalance is that churn prediction problems often become significantly harder [16]. When training models to be capable of identifying both churners and non-churners, it is generally unwise to do so using a random set of observations [47]. As mentioned in section 2.4, this can lead to models only picking up members of the majority class and fail to learn the distinguishing traits of churning users.

The class imbalance problem can be mitigated in several different ways. Depending on the context, the cost of false negatives and false positives can differ (e.g. failing to detect a present condition vs. erroneously diagnosing a non-existing condition). To minimize the costlier misclassification type, different weights can be given to members of the two classes [48]. No weights will be given to the classes in this project since doing so requires domain knowledge outside the scope of the thesis. This is instead left as a future area of study for SVT. Two simple, analogous sampling strategies will instead be used to reduce the class disparity: undersampling and oversampling. Each of these approaches comes with its own advantages and disadvantages.

Undersampling accomplishes a more balanced population by keeping all members of the minority class, and discarding random members of the majority class until the desired class proportions are reached [49]. One advantage of this method is that if the dataset is large and the number of minority members is small in comparison, then it simultaneously reduces the size of the dataset [46]. A reduced dataset is beneficial for training purposes, both in terms of time and computational resources. The main disadvantage of undersampling is that potentially useful information is scrapped in the process, which could negatively affect the performance of the classifier. If only a few percent of the population make up the minority class and training sets are to be split equally between classes, massive amounts of information are wasted [44].

Oversampling on the other hand duplicates random members of the minority class, while keeping those in the majority class intact. This approach can lead to models overfitting, since the same minority class members will be used for training several times. In turn this gives them disproportionate importance, which may teach the model to look for those particular observations [46]. The more severe the class imbalance in the original dataset, the greater this effect becomes. Another drawback of oversampling is that the size of the training set increases, which leads to slower learning rates [49]. It is therefore less resource-effective than undersampling, at the cost of no lost data.

Studies in binary classification in settings where the class imbalance has been notable have found that undersampling generally outperforms oversampling [50, 51, 52]. Some studies including Burez and Van Den Poel [46] and Verbeke et al. [4] found no significant positive impact on performance by oversampling the datasets, compared to doing no sampling at all. The effects of the target class proportions following undersampling have also been studied. While a perfectly balanced dataset resulting
from undersampling has repeatedly shown improved classifier performance compared to using the original dataset [46], tuning the target proportion of minority to majority samples can lead to even better results [53, 50]. Studies researching undersampled datasets with a smaller but present class imbalance have demonstrated good results for minority to majority ratios around 40 : 60 – 30 : 70, citing reduced discarded data as a possible explanation for the increase in performance [49, 50]. In order to evaluate the models on test data representative of the actual population, the datasets are to be split into training, validation and test sets prior to any sampling.

2.6 Standardization of Features

Some statistical methods have shown to be more efficient learners if the data they are fed is scaled. Standardization is a scaling method which works particularly well for neural networks [54]. The standardization process results in a transformed dataset \( \tilde{\mathbf{X}} \), where \( \tilde{\mathbf{X}} = (\tilde{X}_1, \ldots, \tilde{X}_d) \in \tilde{\mathbf{X}} \) and \( \forall j \in [\tilde{\mathbf{X}}_j] = 0, \text{Var}[\tilde{\mathbf{X}}_j] = 1 \). Mathematically this is achieved by subtracting the \( j \)th feature’s mean \( \mu_j \) from the corresponding observation \( x_j \), and dividing by that feature’s standard deviation \( \varsigma_j \), i.e.

\[
\tilde{x}_j = \frac{x_j - \mu_j}{\varsigma_j}.
\] (2.13)

Centering the features around a mean of zero tends to result in faster convergence rates, thereby reducing training times [54]. Standardization also ensures that all inputs are treated equally in any regularization processes. It is necessary to standardize the training and test sets separately since this ensures that information about future observations does not bleed into the training of the models [14].

2.7 Feature Selection

While a rich dataset with many features can certainly be useful in statistical learning, it can also have detrimental effects on model performance. Depending on the type of model used, a large set of features may lead to high variance and in turn models that are overfit and poor at generalizing to unseen data [14]. It is natural that certain variables are more influential than others for making predictions. By discarding the less important features (which may account for more noise than actionable information), the problem of overfitting may be mitigated.

Other benefits associated with feature selection techniques have been researched since the 1970s [55]. A model with fewer parameters is simpler to explain. Model interpretability should not be underestimated and can be important in communicating results to laymen, especially as machine learning is adopted by an increasing number of businesses. Finally, resourcefulness is a factor to consider. By reducing the dimensions of a dataset, significant temporal and computational resources may be saved, at possibly negligible expense of model performance [56].
Two well-established dimensionality-reduction techniques will be used in this project. What follows are short descriptions of both.

2.7.1 Principal Component Analysis

Principal component analysis (PCA) is a method capable of reducing the number of features in a dataset. The idea behind PCA is to transform the dataset into a new set of features in a different coordinate system. These new features are called the principal components, and are linear combinations of the original features. The transformation used in PCA makes the principal components linearly uncorrelated, and they are also ranked in terms of importance [57]. By discarding the components that fall below a set importance threshold, feature selection can be performed [58].

Component importance is defined by the magnitude of the variance of the original dataset when it is projected onto said component. The larger the projected variance, the greater the component’s importance and higher its rank [59]. Since feature variances are central in PCA, it is recommended to standardize the data prior to applying PCA, in particular if features are of different units [57]. Moreover, if standardization is not performed, features with smaller variances can be systematically ignored, whereas those with larger variances may be attributed disproportionate importance [60].

Principal components are frequently computed using the sample covariance matrix $S$ of the matrix of standardized observations $\tilde{X} = (\tilde{x}_1^T, \ldots, \tilde{x}_n^T)$. Since $\tilde{X} \tilde{X}^T$ is a positive semidefinite matrix, singular-value decomposition can be used to find its eigenvalues and eigenvectors in an efficient manner [57]. That is, $S$ can be expressed as

$$S \propto \tilde{X} \tilde{X}^T = \Sigma \Lambda \Sigma^T \quad (2.14)$$

where $\Sigma = (v_1^T, \ldots, v_d^T)$ is a matrix of the eigenvectors of $\tilde{X} \tilde{X}^T$, and $\Lambda$ a diagonal matrix of the corresponding eigenvalues, i.e. $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_d)$. The eigenvector $v_{(1)}$ corresponding to the largest eigenvalue (in terms of magnitude) accounts for the largest variance in the data and is used for constructing the first principal component, and so on for subsequently smaller eigenvalues [57]. The $i$th principal component for the $j$th standardized observation $\tilde{x}_{(j)}$ is then computed by

$$t_{i,(j)} = \tilde{x}_{(j)} v_{(i)}^T. \quad (2.15)$$

The cumulative variance explained by the first $k$ principal components is given by the sum of variances of the first $k$ principal components, divided by $d$ [57]. Feature selection is done by truncating the new feature vectors $t_{(i)} = (t_{1,(i)}, \ldots, t_{d,(i)})$, keeping only the first $m$ features that together account for a desired level of variance.

PCA is also a popular tool for visualizing higher-dimensional data in two- or three-dimensional space. Since it is common for the first two to three principal components to account for a majority of the variance of the data, scatter plots in two or three dimensions can be helpful aids in displaying characteristics and distributions of the different classes [57].
2.7.2 Recursive Feature Elimination

Recursive feature elimination (RFE) is another method for reducing the number of features in a dataset, first proposed by Guyon et al. [61]. Performing an exhaustive search for an optimal feature subset in $d$-dimensional space requires fitting and comparing $2^d - 1$ models. This quickly becomes computationally unfeasible even for moderate values of $d$. RFE significantly reduces the search space by iteratively fitting models, ranking the features (which is specific to the statistical method used, e.g. for logistic regression $|\beta_i|$ are used [21]), and dropping a set number of least important features. This results in at most $d - 1$ models being fit. Those models can then be ranked in terms of chosen performance metrics, and the best-performing subset of features can be used in a final model. Due to the greedy nature of RFE, optimality cannot be guaranteed, however by dropping only a single feature at each iteration one can maximize their chances of finding a well-performing feature subset [61].

Since features are scrambled in ANNs, the notion of feature importance loses meaning for LSTMs. Because of this, RFE will not be used on the sequential dataset. A strength of deep learning methods such as LSTMs is their ability to combine low-level features into abstract high-level features [56]. The abstraction process is itself a type of implicit feature selection technique, as the models learn what signals are important for making good predictions.

2.8 $k$-Fold Cross-Validation

A drawback of partitioning the dataset into training, validation and test sets is that less data will be available for model fitting. If data is not plentiful, one way of circumventing the need for a dedicated validation set is to use $k$-fold cross-validation [14]. This validation approach works by partitioning the set of training observations $\mathcal{X}_t$ into $k$ (approximately equally large) disjoint blocks. Denote by $\mathcal{X}_{(i)}$ the $i$th block of $\mathcal{X}_t$, then

\[ \mathcal{X}_t = \bigcup_{i=1}^{k} \mathcal{X}_{(i)}, \quad \forall i \neq j, \mathcal{X}_{(i)} \cap \mathcal{X}_{(j)} = \emptyset \]

and let $\mathcal{X}_t$ with the $i$th block removed be denoted by

\[ \mathcal{X}_{(-i)} = \mathcal{X}_t \setminus \mathcal{X}_{(i)}. \]

The algorithm then iterates across all $k$ number of folds. In the $i$th iteration, the model is fit using $\mathcal{X}_{(-i)}$, yielding a classification rule $\hat{f}_{\mathcal{X}_{(-i)}}$. The statistical method is then validated using some performance metric $\mathcal{M}$, evaluated with the true-predicted label pairs $(y_{(j)}, \hat{f}_{\mathcal{X}_{(-i)}}(x_{(j)}))$ for all observations $x_{(j)} \in \mathcal{X}_{(i)}$ and corresponding true class labels $y_{(j)}$ [14]. An estimate $\hat{\theta}_\mathcal{M}$ for the statistical method’s performance, with
respect to $\mathcal{M}$, is then given by

$$\hat{\theta}_\mathcal{M} = \frac{1}{k} \sum_{i=1}^{k} \left( \forall x_{(j)} \in X_{(i)} \mathcal{M}(y_{(j)}, \hat{f}^{X_{(i)}}(x_{(j)})) \right).$$

A natural extension of $k$-fold cross-validation is to repeat the algorithm $n$ times. This is often referred to as $n \times k$-fold cross-validation.

### 2.9 Related Work

Churn prediction by means of statistical methods has been studied since the late 1990s. Its roots stem from the telecommunications industry but has since been seen applications in financial services [8], retail industries [62] and on-demand media [63].

Martins [41] and Stojanovski [42] studied the effectiveness of LSTMs compared to logistic regression and random forests in churn prediction for a music streaming service. They found that logistic regression was the worst of the three, however no conclusive ranking of random forests and LSTMs could be made. The models developed by Martins [41] resulted in slightly stronger $F$-measures for LSTMs, with random forests having the advantage in terms of AUC. The improved performance of the LSTMs did come at a significant increase in training time, and the author could not rule out the possibility that the two methods were comparable after all. Stojanovski [42] on the other hand obtained random forest models that outperformed LSTMs for both $F$-measures and AUC.

Seungwook et al. [64] also compared the same three statistical methods, albeit in a non-contractual mobile game setting and focusing solely on the AUC. The study found minimal differences in performance between the methods, although random forests proved to be the weakest of the three.

Similar inconclusive results can be found in a plethora of churn-related articles and studies [65], and no one method has been found to be a jack of all trades that generalizes well to other areas. Despite the fact that advanced models tend to perform slightly better in terms of $F$-measures, they are sometimes rejected in favor of simpler methods due to the added resources needed for model fitting, and lack of interpretability in the results [41].

Several studies have however identified three influential variables in predicting customer churn that seem to generalize to many industries: recency (time elapsed since last the interaction with the company), frequency (how often the user interacts with the company) and monetary value (amount spent by a customer in a given time frame) [53, 8]. The first two variables are possibly of interest in this study and will be examined, however the latter one has no meaningful interpretation in this study and will not be considered.

The time span a customer has been with a company has also shown to be an influential variable for predicting churn, especially in contractual settings [6, 66]. This too is a feature that is possible to make use of in this project.
Chapter 3

Methodology

This chapter describes the methods used for manipulating the data, and training and evaluating the algorithms for predicting user churn. While the dataset used is not publicly accessible, this chapter aims to make the project as reproducible as possible for similar studies.

3.1 Definitions

Due to the case-specific nature of churn prediction problems, a few terms have to be defined for this project in order to be able to precisely label users and discuss the results obtained.

Users

A user is defined by a unique alphanumeric ID assigned to the device a viewer uses to access the VOD services. The notation

\[ u_i = \text{user } i \]  \hspace{1cm} (3.1)

will be used to refer to an arbitrary user. Since viewers have no personal accounts they log in to, it is impossible to track that individual’s activity across several devices. It is therefore likely that a single individual may be be represented by more than one user. A person accessing the VOD services from both their phone and computer would for instance count as two separate users.

The converse is also true. If two or more people use the same device, there is no way of precisely identifying the different individuals and they would instead count as a single user. There is no way of overcoming either of these measurement errors using SVT’s existing framework.

Sessions

A session is defined as a video started by a user at a timestamp \( t_j \) that reflects both user intent and some interest to watch said video. In practice this corresponds to video starts with playing times at least 300 seconds long, or those that elapsed 85%
or more of the total video length. This filtering is intended to discard accidental
video starts and cases where users quickly realized that the video was uninteresting
to them. At the same time this filter keeps all the short sessions that many kids
shows represent due to their short episode lengths. Sessions can be encoded as binary
variables to help in the labeling of training and test data. The following notation
will used to refer to timestamps and sessions in the remainder of the report

\[ t_j = \text{time } j \]  \hspace{1cm} (3.2)
\[ s_{i,j} = \begin{cases} 
1, & \text{if } u_i \text{ watched } \geq 300s \text{ or elapsed } \geq 85\% \text{ of video length at } t_j \\
0, & \text{otherwise.} 
\end{cases} \]  \hspace{1cm} (3.3)

Churn

Since SVT’s VOD viewers neither sign up for the services nor are able to opt-out
of paying the public service tax, they are subject to passive churning, as explained
in section 2.3. In this report, a churner is defined as a user who has logged at least
one session in the OW, at least one session in the AW, and no sessions in the CW. A non-churner is a user who satisfies the conditions in the OW and AW, and also
has logged at least one session in the CW. Using set notation and the previously
established definitions, the entire labeling process is given by algorithm 1.

**Algorithm 1:** Labeling procedure.

**Data:** Set of users \( U \), set of sessions \( S \), OW start time \( t_i \),

AW start time \( t_a \), CW start time \( t_c \), CW end time \( t_f \).

**Result:** User-churn label pairs \((u_i, y_i)\).

1 begin
2 \( U_{OW} \leftarrow \{ u_i \in U \mid \exists j \text{ s.t. } s_{i,j} \in S, s_{i,j} = 1, t_j \in [t_i, t_a) \} \)
3 \( U_{AW} \leftarrow \{ u_i \in U \mid \exists j \text{ s.t. } s_{i,j} \in S, s_{i,j} = 1, t_j \in [t_a, t_c) \} \)
4 \( U_{lab} \leftarrow U_{OW} \cap U_{AW} \)
5 \( U_{CW} \leftarrow \{ u_i \in U_{lab} \mid \exists j \text{ s.t. } s_{i,j} \in S, s_{i,j} = 1, t_j \in [t_c, t_f) \} \)
6 foreach \( u_i \in U_{lab} \) do
7 \hspace{1cm} if \( u_i \in U_{CW} \) then
8 \hspace{1.5cm} \( y_i \leftarrow 0 \) // Non-churner
9 \hspace{1cm} else
10 \hspace{1.5cm} \( y_i \leftarrow 1 \) // Churner
11 \hspace{1cm} end
12 end
3.2 Dataset Description

SVT has records of each started video from all of their VOD services from December 2014 onward. Throughout the years there have been some changes in the collection of data in terms of formats and features. The latest overhaul was made in early December 2017, and each logged start from then on contains a set of 115 features. These features include information about the

- user: such as device used, rough geographic location and user ID
- program: for instance content type, program length and production company
- stream: e.g. when it was started, its length, whether or not it was automatically started, etc.

The limitations imposed by the accountless nature of SVT’s VOD services mean that key information frequently used in other churn prediction studies is missing or impractical to obtain. As mentioned in section 2.9, the time span a user has been with a service has been shown to be significant in churn prediction. There is no data point among the 115 features that explicitly states the first time a user watched a video. The same applies to the recency of any given user. These features can however be engineered fairly easily in the data pre-processing step.

In order to have data in one and the same format, only data from December 2017 onward is deemed relevant. This still constitutes several terabytes of data, which would require processing power not available for this project. Instead only 26 weeks worth of data will be used, of which the OW, AW and CW constitute 20, 2 and 4 weeks respectively. This project will use data starting from $t_i = 2018-01-01$ together with the specified window lengths, resulting in the window breakpoints $t_a = 2018-05-21$, $t_c = 2018-06-04$ and $t_f = 2018-07-01$. Since predictions are given after observing data in the OW and AW, this gives $\tau = 22$.

3.3 Data Pre-Processing

Before any models are trained, it is standard practice to first clean and process the data. Some data points may be missing or malformed and can affect the learning in inadvertent ways. Raw data may also need to be aggregated for compatibility with certain statistical methods, and for efficiency reasons.

3.3.1 Sequential Dataset Creation

Seeing as SVT is interested in viewer behavior on a weekly basis, aggregating watch histories at that level is appropriate. By doing this, potentially hundreds of video starts can be condensed into a single vector, at the cost of losing some – mainly temporal – information. Aside from reducing the storage space required, this procedure will significantly reduce the time needed to train models. The aggregation is accomplished by for each user combining data from all video starts in a given week,
for every week between $t_i$ and $t_f$. Data is combined by e.g. summation (weekly total watch time), counting (number of distinct programs watched in a given week) and computation of various statistics (weekly mean session time), etc. Features are also to be standardized for reasons explained in section 2.6.

In constructing the sequential dataset it is crucial not to omit information about user inactivity. Since a user’s absence is not logged in the database, that information must be added in the data pre-processing step. In this project, a week without user activity is encoded by inserting a zero-filled feature vector in lieu of any actual aggregated activity. Applying algorithm 1 to the sequential dataset provides churn labels for every user in the sequential dataset. The sequential dataset for all users can be after this pre-processing be represented as a three-dimensional tensor of size $(|U_{lab}| \times \tau \times d)$, as seen in figure 3.1.

![Figure 3.1: Format of the sequential dataset after aggregation.](image)

### 3.3.2 Static Dataset Creation

The sequential dataset cannot be used for logistic regression and random forests. In order to feed a user’s watch history into one of these simpler methods, it first needs to be aggregated so that it is no longer sequential. In practice this means that each user’s data is condensed into a one-dimensional vector. This step allows for feature engineering, since the sequence of observed values for each feature, $(X^{(1)}_i, X^{(2)}_i, \ldots, X^{(\tau)}_i)$ for each user can be combined (e.g. the mean weekly watch time across the $\tau$ weeks can be computed) and added as a new feature. Features typically considered important in churn prediction such as user lifespan (in the time span $[t_i, t_f]$) and recency can be constructed as well. Since logistic regression will use $L_2$ regularization the features should be standardized in this dataset [14]. Random forests do not require standardized inputs. However, the random forests will receive standardized data as inputs, both for the sake of comparison to logistic regression and since PCA will be applied to the data.
3.4 Experiment Setup

Several experiments will be conducted in this project, each with a unique configuration. The experiments consist of three to four main components: the statistical method, hyperparameters for the method in question, the sampling technique – and for the static dataset, the feature subset used as input. What follows are the specific component options, along with how the experiments will be evaluated.

3.4.1 Hyperparameter Search

Hyperparameters are parameters of a statistical method that are set before the training of a model and that remain constant throughout. Methods typically have several hyperparameters associated with them, some more influential than others with respect to performance [14]. Some of the more important hyperparameters for the methods used in question will be tuned in this project, with the aim of improving model performance.

In logistic regression, promising candidates for the regularization strength $\lambda$ will be searched for. The number of random features $\delta$ to consider at each split in the random forests is also to be tuned. Finally, the number of hidden units $\ell$ used in the LSTM networks will be tweaked.

A coarse grid search will be used to train and evaluate the models using test data for the various hyperparameter settings. While hyperparameter tuning can be made at arbitrary granularity, model performance is not guaranteed to be affected significantly. Devoting time and energy toward training and evaluating models is considered more important than performing extensive searches for well-performing hyperparameters. Any ranking of hyperparameter values based on performance should therefore be seen as indicative and not absolute, however the best-performing values with respect to the $F$-measure will be used for final model training. Table 3.1 gives the specific hyperparameter values to be tested in the grid search.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>1</th>
<th>10</th>
<th>100</th>
<th>1,000</th>
<th>10,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$</td>
<td>1</td>
<td>10</td>
<td>100</td>
<td>1,000</td>
<td>10,000</td>
</tr>
<tr>
<td>$\delta$</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>$\ell$</td>
<td>100</td>
<td>250</td>
<td>500</td>
<td>1,000</td>
<td>2,000</td>
</tr>
</tbody>
</table>

3.4.2 Sampling

Based on results obtained from previous research, training any of the statistical methods used in this project on unsampled data is not deemed worthwhile, as the resulting models tend to be rather weak. Instead the undersampling and oversampling methods covered in section 2.5 will be used on both the sequential and static datasets.
As Burez and Van Den Poel [46], Jagelid and Movin [50], among others have found that perfectly balanced training sets do not necessarily lead to optimal performance, other class distributions will be experimented with as well. The exact class proportions of the training sets that will result from undersampling and oversampling are given in table 3.2.

Table 3.2: The class proportions used in training sets.

<table>
<thead>
<tr>
<th></th>
<th>0.75</th>
<th>0.7</th>
<th>0.65</th>
<th>0.6</th>
<th>0.55</th>
<th>0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-churners</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Churners</td>
<td>0.25</td>
<td>0.3</td>
<td>0.35</td>
<td>0.4</td>
<td>0.45</td>
<td>0.5</td>
</tr>
</tbody>
</table>

### 3.4.3 Feature Selection

Feature engineering is no exact science, which is another reason why feature selection methods should be applied to datasets to remove possibly useless features. Both PCA and RFE will be used on the static dataset to reduce its dimensionality. For RFE one feature will be dropped at each iteration, as model performance is deemed important and the number of features will not be particularly large. As a means of benchmarking those models, the full static dataset will also be used for training logistic regression and random forest models.

The dimensionality of the feature set produced by PCA will be decided using the full, unsampled static dataset. The smallest number of principal components that account for at least 99% of the variance of that dataset will be used as the truncation value for subsequent PCA-transformed training, validation and test sets.

Construction of the reduced feature sets using RFE will be conducted on a statistical method-basis. Filters will be developed that extract only the $m$ most important features from the full feature set, for each method. The value of $m$ will be decided by running the RFE algorithm for all sampling strategies and class proportions. The number of most important features that result in the greatest average $F$-measure of all tried configurations will be assigned to $m$.

Since PCA does not generalize to higher-dimensional data structures than matrices, and RFE is inapplicable in neural networks, no feature-reducing methods will be used on the sequential dataset.

### 3.4.4 Evaluation

Models will be assessed on their accuracy, precision, recall, $F$-measure and AUC, following the definitions from section 2.4. In order to gauge the consistency of the models, the grid search will be carried out using $5 \times 2$-fold cross-validation, where the training set in the current loop iteration will be sampled to reduce the class disparity. This will be repeated for all feature sets, sampling strategies and class proportions. The performance scores obtained will then be averaged to feature set-level, to reduce the number of final models to be fit. The best-performing hyperparameter for each statistical method-feature set pair will be used for further model development.
Final models will be developed for each statistical method, sampling strategy, feature set and studied class proportion. These models will be trained on 90% of the available data (which will be sampled, hence may not correspond to nine times the size of the test set), and evaluated using the remaining test observations. This process will be repeated five times, and the mean performance scores will be tabulated.

Scikit-learn does not support model ranking for more than one metric for RFE [21]. Since no weights are given to classes and their misclassifications, the $F$-measure will be used for ranking feature subsets in the feature selection step. Just as with the hyperparameter search, $5 \times 2$-fold cross-validation with unsampled validation sets will be used when searching for the best-performing number of most important features according to RFE.
Chapter 4

Results

This chapter presents the results obtained from the experiment setups outlined in chapter 3. Extensive tables containing performance metrics for specific model configurations and hyperparameters are found in appendix A and appendix B, respectively.

4.1 Processed Datasets

In order to make the dataset feasible to work on without the need for massive computing power, only a subset of users \( U \) from the entire dataset were used. Users who had started a video in \([t_i, t_c]\) in the dataset were randomly selected with success probability \( 10^{-4} \), resulting in \( |U| = 93,124 \) unique users. Using BigQuery, all sessions as defined in section 3.1 made by users in \( U \) corresponding to kids-related programming on the same period were extracted, resulting in 38 million rows. Applying algorithm 1 to this data reduced the dataset to 23 million rows spanning \( |U_{lab}| = 34,231 \) unique labeled users. Among these users, 4,709 were labeled as churners and the remaining 29,522 as non-churners.

Data exploration showed that a handful of sessions were found to span several hours worth of playing time. These were clear outliers that probably did not correspond to actual viewing sessions. On the basis of this discovery, all sessions spanning longer than 10,800 seconds (three hours) were removed. While the vast majority of viewers who logged these outlying sessions also had sessions shorter than three hours, 114 users did not and were consequently removed, resulting in a final user count of 4,644 churners and 29,473 non-churners.

Approximately two thirds \( \left( \frac{2}{3} \right) \) of the columns of SVT’s dataset contained strictly categorical data with many factors (such as episode names, program descriptions, production companies etc.). These columns, aside from the ones containing user IDs and episode IDs, were dropped for two reasons. Firstly, in order for any of the statistical methods to be able to make use of categorical data, it would have to be encoded using one-hot vectors. This approach would be highly impractical as for just kids programs alone, the number of columns of the dataset would increase to an order of magnitude of \( 10^4 \). As mentioned, this would lead to additional resources being required for aggregation and model training. Secondly, programs are continuously added and removed from SVT’s publicly accessible VOD library, and programs listed in the watch histories might differ between runs. This gives rise to the need for a
program to keep track of the one-hot mapping of categorical data in different training datasets which is outside the scope of this project.

Of the remaining columns, those with frequently missing values were discarded. A few numerical columns also contained duplicated information, in which case only one of those columns were kept. After these filtering steps, eleven features remained.

For every labeled users, all sessions logged in each of the $\tau = 22$ weeks were condensed into a single vector spanning $d = 22$ features. The 22 week vectors for each user were then aggregated into a single static feature vector. At this stage statistics such as the features’ mean values and standard deviation across all weeks for the 22 features could be computed. The earliest and last recorded visits each user made were at this step added as features in the static dataset, resulting in a static dataset with 69 features.

\section{4.2 Selected Features}

\subsection{4.2.1 PCA}

As seen in figure 4.1, more than 99\% of the variance in the full static dataset was explained with 22 principal components. Based on this information, 22 principal components were used in all subsequent experiments where PCA was used as a feature selection technique.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure4.1.png}
\caption{Cumulative sum of explained variance of the standardized static dataset, per number of principal components following PCA.}
\end{figure}

The first two principal components alone accounted for approximately 70.9\% of the variance. How 1,000 randomly selected members from each of the two classes were distributed in the two-dimensional feature space of the first two principal components is shown in figure 4.2.
4.2.2 RFE

Figure 4.3 displays the cross-validated $F$-measures per number of the most important features, averaged across all sampling strategies and class proportions. For logistic regression, the obtained $F$-measure plateaued after seven features, however a maximum was reached at 36 features. For random forests the maximum cross-validated $F$-measure was achieved using the 25 most important features. Neither of the RFE feature sets that maximized $F$-measures included the days since the last recorded visit, or the earliest recorded visit as important features. The frequency of visits (number of distinct days visited in the OW and AW) was however included in both RFE feature sets.

The shorthand notation “All”, “PCA$_{22}$” and “RFE$_m$” in subsequent figures and tables refer to the full (static or sequential) feature set, the static dataset’s 22 first principal components, and the static dataset’s $m$ most important features obtained from RFE (where $m \in \{36, 25\}$, depending on the method used).
4.3 Hyperparameter Search

The full results from the conducted grid searches are presented in table A.1 to table A.3 in appendix A. The best-performing hyperparameters with respect to average $F$-measures for each method and feature set are displayed in table 4.1. As mentioned in section 3.4.1, these hyperparameter values were later used in final model training.

<table>
<thead>
<tr>
<th>Features</th>
<th>$\lambda$</th>
<th>$\delta$</th>
<th>$\ell$</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>10</td>
<td>20</td>
<td>500</td>
</tr>
<tr>
<td>PCA$_{22}$</td>
<td>1,000</td>
<td>20</td>
<td>—</td>
</tr>
<tr>
<td>RFE$_{36}$</td>
<td>10,000</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>RFE$_{25}$</td>
<td>—</td>
<td>3</td>
<td>—</td>
</tr>
</tbody>
</table>

4.4 Model Performance

The performance metrics of all combinations of methods, samplers, population proportions and applicable feature selection techniques are found in table B.1 to table B.3 in appendix B. The configurations that lead to the largest cross-validated $F$-measures for each method are presented in table 4.2.

The mean performance scores of the model configurations from table 4.2 are presented in table 4.3. Sample precision-recall curves generated using the configurations from table 4.2 are shown in figure 4.4.
Table 4.2: Best-performing method configurations.

<table>
<thead>
<tr>
<th>Method</th>
<th>Sampler</th>
<th>Features</th>
<th>Churners</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic regression</td>
<td>Under</td>
<td>All</td>
<td>0.35</td>
</tr>
<tr>
<td>Random forests</td>
<td>Under</td>
<td>All</td>
<td>0.3</td>
</tr>
<tr>
<td>LSTM</td>
<td>Over</td>
<td>All</td>
<td>0.45</td>
</tr>
</tbody>
</table>

Table 4.3: Performance of best method configurations.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic regression</td>
<td>0.79426</td>
<td>0.35273</td>
<td>0.63399</td>
<td>0.45327</td>
<td>0.34891</td>
</tr>
<tr>
<td>Random forests</td>
<td>0.80129</td>
<td>0.35844</td>
<td>0.54105</td>
<td>0.43121</td>
<td>0.34497</td>
</tr>
<tr>
<td>LSTM</td>
<td>0.77213</td>
<td>0.35934</td>
<td>0.73523</td>
<td>0.48185</td>
<td>0.43292</td>
</tr>
</tbody>
</table>

Figure 4.4: Precision-recall curves resulting from the models in table 4.2.
Chapter 5

Discussion

This chapter highlights and discusses the most important findings from chapter 4. Identified shortcomings of the models and modeling procedures are also reviewed. The project at large is summarized and potential future areas of research are outlined.

5.1 Hyperparameter Search

The regularization strength $\lambda$ did not drastically affect any metric of the different logistic regression models. Any overarching trend on performance across feature sets dependent on the value of $\lambda$ did not seem to exist either. Similar results were seen for the tested values of $\delta$. The number of hidden units $\ell$ in the LSTMs proved to be the least influential examined hyperparameter.

In summary, the conducted hyperparameter search turned out to be relatively unfruitful. Of course this does not rule out the possible existence of hyperparameter values that greatly can affect model performance.

5.2 Model Performance

The results found by e.g. Verbeke et al. [4], suggesting that oversampled training sets lead to models that are comparable to those trained on unsampled data seem to generalize to random forests in this setting. Those models produced particularly poor $F$-measures and recall scores. Among the 21 models with the worst average $F$-measures, 18 of these were the random forests with oversampled training data. Of the remaining three – which were all LSTMs – two were also using oversampled training data, one of which predicted no churners at all, resulting in undefined $F$-measures. Based on these results, future model development at SVT using random forests can outright disregard oversampled training data. The problem with the oversampled data appears to stem from the fact that the resulting models predicted almost all test observations as non-churners, seeing as the corresponding accuracy scores were approximately equal to the fraction of non-churners in the population, at around 86%. Likewise, the corresponding recall scores seldom reached scores above 0.1, meaning that the models very often failed to pick up even a small minority of churners in the test sets.
Undersampling was used in the development of the best-performing logistic regression and random forest models, reaffirming the claims made by both Drummond and Holte [51] and Japkowicz and Shah [16]. The effects of using under- or oversampled training data for LSTMs on the performance metrics were not as pronounced, which could arise from the overall information-richer observations those models were trained on.

The results from e.g. Jagelid and Movin [50] also seem to generalize to this study. None of the 14 method-sampler-feature set combinations achieved their maximum $F$-measures for perfectly balanced training sets. Logistic regression and random forests in particular produced greater $F$-measures for training sets with greater class disparities, whereas LSTMs appear to have performed better with slightly less imbalanced sets.

For virtually all model configurations, recall was notably greater than precision. That is, the models were better at correctly identifying users as churners than distinguishing churners from non-churners. Similar trends have been recorded in e.g. Stojanovski [42], and likely stems from the inherent class imbalance problem. As the class proportions got more balanced, there was an overall increase in recall for all models and feature sets. A natural explanation for this is simply that as the number of churners in the training sets increased, the models got better at picking up unseen churners in the test sets.

Aside from the models that predicted no or very few churners, accuracy steadily decreased as the class imbalance in the training sets were reduced. This trend is not unexpected and stems from the sometimes misleading nature of the accuracy metric, mentioned in section 2.4.

The difference in performance between logistic regression and random forests was rather small for their respective best configurations, as seen in table 4.3. The LSTM model with the greatest $F$-measure did however produce higher precision, recall and AUC scores as well, compared to the best-scoring simpler methods. Both the best logistic regression and random forest models scored accuracies that were higher than that of the LSTM. As indicated by previous churn prediction studies, obtaining a single model or statistical method that with certainty outperforms the others across all performance metrics is hard to come by [42, 41, 64]. This turned out to be the case in this project as well.

5.3 Model Limitations

An apparent model limitation is the fact that sessions are completely void of qualitative information describing whether or not a user was satisfied with the content they watched. Surely such information would be beneficial to incorporate in model development. It is for instance reasonable to think that a streak of sessions where the user left the VOD service satisfied would result in an increased probability that the user would return to it and not churn. A repeated series of dissatisfying sessions could on the flip side increase the probability of a user churning.
Another obvious limitation is the rather lax criterion used for labeling users. As mentioned by Kapoor et al. [67], defining churn for non-contractual setting is a difficult task, and the definition settled on will certainly affect resulting models’ performance. If SVT’s ideal predictive model should be able to devise strategies for keeping steady unique weekly visitor counts, requiring at least one session per week and user in the OW and AW could be a more appropriate criterion. Such a criterion would however come with a set of undesirable properties. Firstly, it would likely exacerbate the problem of loss of data in the aggregation process. Seeing as algorithm 1 resulted in a 63% loss of unique users in the dataset, the current criterion is unresourceful as is. Secondly, the already present sample selection bias would intensify. By only extracting users – churners or not – that have been sufficiently active, other types of churning users are completely disregarded in the dataset aggregation process. The drawback of this is that models will never be exposed to the more fleeting viewers in any model training, and will likely not be able to identify them as such on test data.

Other choices for lengths of the OW, AW and CW have not been experimented with either. Longer OWs entail more data, in theory making churn prediction easier [64]. At the same time, user behavior closer in time to the start of the CW has shown to be of greater importance than older watch patterns in previous studies [39]. A longer OW also means that when models are deployed to classify churners in real time, longer waiting times are needed prior to making predictions. Even though recency was not deemed an important feature for the RFE datasets, the black box nature of ANNs does not rule out the possibility that the LSTM models made use of that information. The added amounts of data arising from longer time frames will also require more time and computational resources, both for aggregating it to compact formats and training models.

It is also worth re-emphasizing that the $F$-measure assigns equal weights to precision and recall. In turn this gives the different misclassification types equal weights, which is not guaranteed to be the case for SVT. Incorrectly classifying non-churners as churners can lead to unnecessary countermeasures being taken, incurring additional costs for SVT. Conversely, by falsely identifying churners as non-churners, viewership and trust in the company could be lost. On these grounds, other performance metrics could be more suitable.

None of the tested methods generate predictions for when an identified churner is likely to defect in the CW. Obtaining those types of detailed predictions could allow for more nuanced analyses and immediate feedback on e.g. how current publishing strategies are working. How well any of the models adapt to the seasonality trends SVT has identified has also not been investigated, as only data from the first six months of 2018 was used.

A final limitation worth mentioning is the possibility of only training models on fragmented user activity. Tracking a user across different devices would present the models with fuller pictures of their future churn statuses. The current models have for example no way of piecing together watch histories from the same individual using more than one device.
5.4 Concluding Remarks

With the research question from section 1.2 in mind, the obtained empirical results do indicate that the best LSTM models configurations trained on sequential data produce precision, recall, F-measures and AUC scores that exceed those coming from logistic regression and random forests. Those LSTM models did however not outperform the simpler methods in terms of accuracy. The predictive performance of the best logistic regression and random forest model configurations were comparable.

5.5 Future Work

Several extensions of this project and similar ones are possible. Exploring how detailed predictions could be made is very interesting. Knowing when a user is about to churn – for instance by assigning churn probabilities to discrete time steps in the CW – could provide highly valuable information. Research into how the lengths of the time frames impact the predictive performance may also be worth investigating. Finding ways of making accurate churn predictions on a CW that is as long as, or longer than the OW is also a reasonable pursuit. This is especially the case when ample time is needed between identifying likely churners to implementing retentive actions for keeping them engaged.

If user accounts that are unified across devices is implemented for SVT’s VOD services in the future, assessing the models using a more complete picture of users’ watch histories is a natural progression. Experimenting with unequal class weights and different misclassification costs could also be worthwhile, but requires additional domain knowledge.
Bibliography


[31] F. Chollet et al. Keras. https://keras.io, 2015.


# Appendix A

## Grid Search Results

### Table A.1: Grid search results for logistic regression.

<table>
<thead>
<tr>
<th>Features</th>
<th>λ</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>1</td>
<td>0.73576</td>
<td>0.29120</td>
<td>0.62807</td>
<td>0.38061</td>
<td>0.34906</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.75675</td>
<td>0.33197</td>
<td>0.63256</td>
<td>0.42089</td>
<td>0.35596</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0.76617</td>
<td>0.32030</td>
<td>0.60935</td>
<td>0.40951</td>
<td>0.36065</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.74111</td>
<td>0.32848</td>
<td>0.60539</td>
<td>0.40655</td>
<td>0.34147</td>
</tr>
<tr>
<td></td>
<td>10,000</td>
<td>0.70275</td>
<td>0.30659</td>
<td>0.59857</td>
<td>0.39499</td>
<td>0.33803</td>
</tr>
<tr>
<td>PCA_{22}</td>
<td>1</td>
<td>0.72461</td>
<td>0.18010</td>
<td>0.50970</td>
<td>0.29347</td>
<td>0.24087</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.72445</td>
<td>0.22192</td>
<td>0.57059</td>
<td>0.32963</td>
<td>0.26100</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0.72362</td>
<td>0.24891</td>
<td>0.54357</td>
<td>0.32990</td>
<td>0.26239</td>
</tr>
<tr>
<td></td>
<td>1,000</td>
<td>0.69165</td>
<td>0.25254</td>
<td>0.56165</td>
<td>0.33815</td>
<td>0.28776</td>
</tr>
<tr>
<td></td>
<td>10,000</td>
<td>0.67735</td>
<td>0.23783</td>
<td>0.55818</td>
<td>0.32326</td>
<td>0.26389</td>
</tr>
<tr>
<td>RFE_{36}</td>
<td>1</td>
<td>0.58784</td>
<td>0.18514</td>
<td>0.68372</td>
<td>0.28211</td>
<td>0.35028</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.65426</td>
<td>0.20031</td>
<td>0.69950</td>
<td>0.31766</td>
<td>0.35988</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>0.72362</td>
<td>0.24891</td>
<td>0.54357</td>
<td>0.32990</td>
<td>0.26239</td>
</tr>
<tr>
<td></td>
<td>1,000</td>
<td>0.69165</td>
<td>0.25254</td>
<td>0.56165</td>
<td>0.33815</td>
<td>0.28776</td>
</tr>
<tr>
<td></td>
<td>10,000</td>
<td>0.66830</td>
<td>0.26865</td>
<td>0.72508</td>
<td>0.38032</td>
<td>0.37655</td>
</tr>
</tbody>
</table>

### Table A.2: Grid search results for random forests.

<table>
<thead>
<tr>
<th>Features</th>
<th>δ</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>3</td>
<td>0.78570</td>
<td>0.36011</td>
<td>0.29922</td>
<td>0.26290</td>
<td>0.32495</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.80681</td>
<td>0.37409</td>
<td>0.29835</td>
<td>0.28653</td>
<td>0.33624</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0.80831</td>
<td>0.34811</td>
<td>0.29947</td>
<td>0.31625</td>
<td>0.33591</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>0.80936</td>
<td>0.34112</td>
<td>0.31931</td>
<td>0.32435</td>
<td>0.34057</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.80929</td>
<td>0.35356</td>
<td>0.33965</td>
<td>0.34449</td>
<td>0.35503</td>
</tr>
<tr>
<td>PCA_{22}</td>
<td>3</td>
<td>0.78755</td>
<td>0.20564</td>
<td>0.19482</td>
<td>0.19618</td>
<td>0.23535</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.76759</td>
<td>0.24054</td>
<td>0.19619</td>
<td>0.21934</td>
<td>0.26549</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0.78334</td>
<td>0.26428</td>
<td>0.22100</td>
<td>0.22008</td>
<td>0.26147</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>0.78509</td>
<td>0.23160</td>
<td>0.24984</td>
<td>0.23761</td>
<td>0.26941</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.79029</td>
<td>0.26039</td>
<td>0.25366</td>
<td>0.23983</td>
<td>0.28758</td>
</tr>
<tr>
<td>RFE_{25}</td>
<td>3</td>
<td>0.80268</td>
<td>0.36095</td>
<td>0.34451</td>
<td>0.33612</td>
<td>0.34120</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.80025</td>
<td>0.31863</td>
<td>0.34065</td>
<td>0.29551</td>
<td>0.36226</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0.79187</td>
<td>0.33520</td>
<td>0.34363</td>
<td>0.29037</td>
<td>0.34715</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>0.78718</td>
<td>0.34257</td>
<td>0.33704</td>
<td>0.28696</td>
<td>0.32644</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.78951</td>
<td>0.34854</td>
<td>0.34694</td>
<td>0.28772</td>
<td>0.34067</td>
</tr>
</tbody>
</table>
Table A.3: Grid search results for LSTM.

<table>
<thead>
<tr>
<th>$\ell$</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.82252</td>
<td>0.30014</td>
<td>0.31253</td>
<td>0.29293</td>
<td>0.35814</td>
</tr>
<tr>
<td>250</td>
<td><strong>0.82905</strong></td>
<td>0.29200</td>
<td>0.32986</td>
<td>0.31718</td>
<td>0.36268</td>
</tr>
<tr>
<td>500</td>
<td>0.80081</td>
<td><strong>0.32161</strong></td>
<td>0.38500</td>
<td><strong>0.32667</strong></td>
<td>0.39769</td>
</tr>
<tr>
<td>1,000</td>
<td>0.75761</td>
<td>0.30819</td>
<td><strong>0.39216</strong></td>
<td>0.31410</td>
<td><strong>0.39908</strong></td>
</tr>
<tr>
<td>2,000</td>
<td>0.74966</td>
<td>0.29743</td>
<td>0.34484</td>
<td>0.32201</td>
<td>0.38848</td>
</tr>
</tbody>
</table>
## Appendix B

### Model Performance Results

Table B.1: Mean performance scores, logistic regression.

<table>
<thead>
<tr>
<th>Sampler</th>
<th>Features</th>
<th>Churners</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>All</td>
<td>0.25</td>
<td>0.84408</td>
<td>0.42605</td>
<td>0.41505</td>
<td>0.42048</td>
</tr>
<tr>
<td></td>
<td>PCA22</td>
<td>0.25</td>
<td>0.82562</td>
<td>0.39243</td>
<td>0.40452</td>
<td><strong>0.39838</strong></td>
<td><strong>0.37438</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3</td>
<td>0.81800</td>
<td>0.38959</td>
<td>0.51351</td>
<td>0.44305</td>
<td>0.3999</td>
</tr>
<tr>
<td></td>
<td>RFE36</td>
<td>0.25</td>
<td>0.78488</td>
<td>0.32767</td>
<td>0.49569</td>
<td>0.33626</td>
<td>0.30326</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3</td>
<td>0.73798</td>
<td>0.28641</td>
<td>0.71875</td>
<td>0.39429</td>
<td>0.35865</td>
</tr>
<tr>
<td></td>
<td>PCA22</td>
<td>0.35</td>
<td>0.76299</td>
<td>0.24824</td>
<td>0.74894</td>
<td>0.37288</td>
<td>0.37862</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3</td>
<td>0.75703</td>
<td>0.28129</td>
<td>0.47689</td>
<td>0.35386</td>
<td>0.30586</td>
</tr>
<tr>
<td></td>
<td>RFE36</td>
<td>0.35</td>
<td>0.69598</td>
<td>0.22558</td>
<td>0.79054</td>
<td>0.35100</td>
<td>0.37375</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.4</td>
<td>0.68933</td>
<td>0.27165</td>
<td>0.71875</td>
<td>0.39429</td>
<td>0.35865</td>
</tr>
<tr>
<td></td>
<td>PCA22</td>
<td>0.4</td>
<td>0.75615</td>
<td>0.32043</td>
<td>0.71800</td>
<td>0.44311</td>
<td>0.35406</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.4</td>
<td>0.72450</td>
<td>0.21082</td>
<td>0.54149</td>
<td>0.30349</td>
<td>0.22715</td>
</tr>
<tr>
<td></td>
<td>RFE36</td>
<td>0.4</td>
<td>0.62427</td>
<td>0.21252</td>
<td>0.69283</td>
<td>0.32526</td>
<td>0.24864</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5</td>
<td>0.62778</td>
<td>0.25472</td>
<td>0.7254</td>
<td>0.39415</td>
<td>0.39219</td>
</tr>
<tr>
<td></td>
<td>PCA22</td>
<td>0.5</td>
<td>0.65299</td>
<td>0.25477</td>
<td>0.80603</td>
<td>0.38716</td>
<td>0.37593</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5</td>
<td>0.54367</td>
<td>0.18992</td>
<td>0.78490</td>
<td>0.30584</td>
<td>0.28674</td>
</tr>
<tr>
<td></td>
<td>RFE36</td>
<td>0.5</td>
<td>0.58939</td>
<td>0.23684</td>
<td>0.80838</td>
<td>0.34635</td>
<td>0.41598</td>
</tr>
</tbody>
</table>

Note: The table shows the mean performance scores for different samplers and feature selection methods. The columns represent accuracy, precision, recall, F-measure, and AUC. The values under each feature selection method indicate the performance under different feature reduction thresholds.
Table B.2: Mean performance scores, random forests.

<table>
<thead>
<tr>
<th>Sampler Feature</th>
<th>Churners</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>All</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.25</td>
<td>0.83851</td>
<td>0.39046</td>
<td>0.40000</td>
<td>0.39517</td>
<td>0.37381</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>0.80129</td>
<td>0.35844</td>
<td>0.54105</td>
<td>0.43121</td>
<td>0.34497</td>
<td></td>
</tr>
<tr>
<td>0.35</td>
<td>0.77022</td>
<td>0.32139</td>
<td>0.61720</td>
<td>0.42268</td>
<td>0.35914</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>0.73036</td>
<td>0.28558</td>
<td>0.66017</td>
<td>0.39869</td>
<td>0.32993</td>
<td></td>
</tr>
<tr>
<td>0.45</td>
<td>0.70604</td>
<td>0.28254</td>
<td>0.72211</td>
<td>0.40616</td>
<td>0.31348</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.66092</td>
<td>0.24138</td>
<td>0.76364</td>
<td>0.36681</td>
<td>0.30287</td>
<td></td>
</tr>
<tr>
<td><strong>PCA_{22}</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.25</td>
<td>0.83118</td>
<td>0.36441</td>
<td>0.38393</td>
<td>0.37391</td>
<td>0.30936</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>0.79103</td>
<td>0.29899</td>
<td>0.37623</td>
<td>0.33177</td>
<td>0.25058</td>
<td></td>
</tr>
<tr>
<td>0.35</td>
<td>0.76964</td>
<td>0.27577</td>
<td>0.45676</td>
<td>0.34391</td>
<td>0.23200</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>0.68054</td>
<td>0.20609</td>
<td>0.52995</td>
<td>0.29677</td>
<td>0.19076</td>
<td></td>
</tr>
<tr>
<td>0.45</td>
<td>0.69168</td>
<td>0.23846</td>
<td>0.63409</td>
<td>0.34658</td>
<td>0.23477</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.57298</td>
<td>0.20580</td>
<td>0.75325</td>
<td>0.32327</td>
<td>0.26678</td>
<td></td>
</tr>
<tr>
<td><strong>RFE_{25}</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.25</td>
<td>0.82767</td>
<td>0.38078</td>
<td>0.44880</td>
<td>0.41200</td>
<td>0.34174</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>0.80774</td>
<td>0.33965</td>
<td>0.53440</td>
<td>0.41533</td>
<td>0.35632</td>
<td></td>
</tr>
<tr>
<td>0.35</td>
<td>0.78107</td>
<td>0.31548</td>
<td>0.60641</td>
<td>0.41504</td>
<td>0.34030</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>0.73242</td>
<td>0.30057</td>
<td>0.64766</td>
<td>0.41059</td>
<td>0.32622</td>
<td></td>
</tr>
<tr>
<td>0.45</td>
<td>0.68757</td>
<td>0.27273</td>
<td>0.72973</td>
<td>0.39706</td>
<td>0.29048</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.67585</td>
<td>0.25687</td>
<td>0.76718</td>
<td>0.38487</td>
<td>0.30057</td>
<td></td>
</tr>
<tr>
<td><strong>All</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.25</td>
<td>0.85932</td>
<td>0.34146</td>
<td>0.06167</td>
<td>0.10448</td>
<td>0.32863</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>0.86665</td>
<td>0.50427</td>
<td>0.12939</td>
<td>0.20593</td>
<td>0.36455</td>
<td></td>
</tr>
<tr>
<td>0.35</td>
<td>0.86108</td>
<td>0.56250</td>
<td>0.11111</td>
<td>0.18557</td>
<td>0.40985</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>0.85404</td>
<td>0.46575</td>
<td>0.06897</td>
<td>0.12014</td>
<td>0.37540</td>
<td></td>
</tr>
<tr>
<td>0.45</td>
<td>0.86401</td>
<td>0.47368</td>
<td>0.07826</td>
<td>0.13433</td>
<td>0.33052</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.86313</td>
<td>0.19444</td>
<td>0.01573</td>
<td>0.02911</td>
<td>0.25945</td>
<td></td>
</tr>
<tr>
<td><strong>PCA_{22}</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.25</td>
<td>0.86166</td>
<td>0.44118</td>
<td>0.03205</td>
<td>0.05976</td>
<td>0.31697</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>0.86489</td>
<td>0.33333</td>
<td>0.00436</td>
<td>0.00860</td>
<td>0.22554</td>
<td></td>
</tr>
<tr>
<td>0.35</td>
<td>0.86108</td>
<td>0.21739</td>
<td>0.01085</td>
<td>0.02066</td>
<td>0.21406</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>0.86313</td>
<td>0.46552</td>
<td>0.05832</td>
<td>0.10365</td>
<td>0.31071</td>
<td></td>
</tr>
<tr>
<td>0.45</td>
<td>0.85170</td>
<td>0.08333</td>
<td>0.00412</td>
<td>0.00784</td>
<td>0.18938</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.85551</td>
<td>0.08571</td>
<td>0.00647</td>
<td>0.01202</td>
<td>0.18728</td>
<td></td>
</tr>
<tr>
<td><strong>RFE_{25}</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.25</td>
<td>0.85961</td>
<td>0.44444</td>
<td>0.11159</td>
<td>0.17839</td>
<td>0.33107</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>0.85991</td>
<td>0.44186</td>
<td>0.12311</td>
<td>0.19257</td>
<td>0.33734</td>
<td></td>
</tr>
<tr>
<td>0.35</td>
<td>0.86108</td>
<td>0.36957</td>
<td>0.07556</td>
<td>0.12546</td>
<td>0.30360</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>0.86284</td>
<td>0.41333</td>
<td>0.06813</td>
<td>0.11698</td>
<td>0.30604</td>
<td></td>
</tr>
<tr>
<td>0.45</td>
<td>0.85844</td>
<td>0.38182</td>
<td>0.04468</td>
<td>0.08000</td>
<td>0.32495</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.86184</td>
<td>0.44262</td>
<td>0.05857</td>
<td>0.10345</td>
<td>0.31060</td>
<td></td>
</tr>
</tbody>
</table>
Table B.3: Mean performance scores, LSTM.

<table>
<thead>
<tr>
<th>Sampler</th>
<th>Churners</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Under</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.25</td>
<td>0.86020</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.40961</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td><strong>0.87251</strong></td>
<td><strong>0.52083</strong></td>
<td>0.22573</td>
<td>0.31496</td>
<td>0.40429</td>
<td></td>
</tr>
<tr>
<td>0.35</td>
<td>0.82649</td>
<td>0.40991</td>
<td>0.41317</td>
<td>0.41153</td>
<td>0.39244</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>0.80744</td>
<td>0.36398</td>
<td>0.66895</td>
<td>0.47144</td>
<td><strong>0.43528</strong></td>
<td></td>
</tr>
<tr>
<td>0.45</td>
<td>0.75381</td>
<td>0.35123</td>
<td>0.75244</td>
<td>0.47891</td>
<td>0.42647</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.73359</td>
<td>0.30814</td>
<td><strong>0.81195</strong></td>
<td>0.44674</td>
<td>0.37935</td>
<td></td>
</tr>
<tr>
<td>Over</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.25</td>
<td><strong>0.86753</strong></td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.36454</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>0.86313</td>
<td><strong>0.49933</strong></td>
<td>0.08280</td>
<td>0.13483</td>
<td>0.37372</td>
<td></td>
</tr>
<tr>
<td>0.35</td>
<td>0.85580</td>
<td>0.45505</td>
<td>0.38731</td>
<td>0.41841</td>
<td>0.42512</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>0.83162</td>
<td>0.38251</td>
<td>0.45973</td>
<td>0.41482</td>
<td>0.37924</td>
<td></td>
</tr>
<tr>
<td>0.45</td>
<td>0.77213</td>
<td>0.35934</td>
<td>0.73523</td>
<td><strong>0.48185</strong></td>
<td><strong>0.43292</strong></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.67966</td>
<td>0.25797</td>
<td><strong>0.84279</strong></td>
<td>0.39491</td>
<td>0.37940</td>
<td></td>
</tr>
</tbody>
</table>

1 Due to no predicted churners, the denominator of equation (2.11d) was 0, hence the expression was undefined and set to 0 by Scikit-learn.
Appendix C

Feature Set Correlation Matrices

Figure C.1: Feature correlation matrix of All (static).
Figure C.2: Feature correlation matrix of PCA$_{22}$. 
Figure C.3: Feature correlation matrix of RFE_{36}. 
Figure C.4: Feature correlation matrix of RFE_{25}.