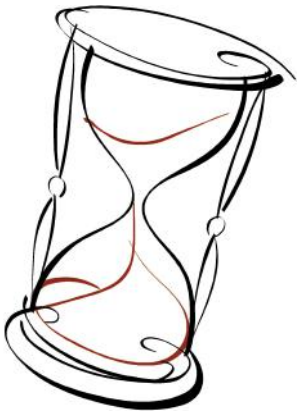
WHAT IS WORTH SPENDING TIME ON

ŞTEFAN P. NICULAE



A Machine Learning Approach to Finding Impactful Features

BACHELOR OF COMPUTER SCIENCE THESIS

University of Bucharest
Faculty of Mathematics and Computer Science
Department of Computer Science

Adobe Romania

February - July, 2016

"An approximate answer to the right problem is worth a good deal more than an exact answer to an approximate problem."

— John Tukey, mathematician

ABSTRACT

Making a decision on what to invest development time in is difficult. Today's market is so competitive that you cannot afford to focus on negligible product features. Based on reported customer behavior, I propose a ranking of the most important features with the help of statistics and machine learning. Following this advice leads to making informed decisions leading to good use of developers' time.

ACKNOWLEDGEMENTS

Traian Şerbănuță - university advisor
Paul Alexandru Chirița - scientific advisor
Alexandru-Daniel Mirea - ideas & feedback
Daniel Dogaru - ideas & feedback
Ştefan Teodor Craciun - manager

CONTENTS

I PREFACE 1

1 INTRODUCTION 2
- 1.1 Problem Statement . . . 2
- 1.2 Motivation . . . 2
- 1.3 Paper Outline . . . 2

2 CONTEXT 4
- 2.1 The Application . . . 4
- 2.2 Technologies Used . . . 6

II BACKGROUND 8

3 ALGORITHMS 9
- 3.1 Machine Learning Fundaments . . . 9
- 3.2 Learning Models . . . 18
- 3.3 Statistical Based Methods . . . 25

4 RELATED WORKS 28
- 4.1 Feature Selection . . . 28
- 4.2 Customer Retention . . . 29

III DEVELOPMENT 30

5 DATA ANALYSIS 31
- 5.1 Retention Definition . . . 31
- 5.2 Event Based Features . . . 36
- 5.3 Time Based Features . . . 43
- 5.4 Action Sequences . . . 46

6 ALGORITHMS 50
- 6.1 Model Optimization Pipeline . . . 50
- 6.2 Combining Classifier . . . 52
- 6.3 Learning Based Feature Selection . . . 53

7 APPLICATION AND RESULTS 55
- 7.1 Statistical Based Methods . . . 55
- 7.2 Machine Learning Based . . . 60

IV CLOSING WORDS 70

8 CONCLUSIONS 71
- 8.1 Results Summary . . . 71
- 8.2 Contribution to the Field . . . 71

9 FURTHER WORK 73

V APPENDIX 74

BIBLIOGRAPHY 77

Part I

PREFACE

The first part of the thesis provides the *what*, *why* and *where*. The first chapter explains the problem being tackled and the motivation for choosing it as my graduate diploma project. The second chapter contains an outline of the context I worked in: the application being analyzed and the stack of technologies used.

1 INTRODUCTION

Knowing which parts of a product are most impactful is a very valuable information to have. We do not settle for data analysis results for answering this question, instead statistical and machine learning based approaches will be used. Before we dive into the gist of the paper, it is very important to understand the objective clearly. This chapter also provides the reason behind choosing to work on it and a brief overview of matters to be discussed.

1.1 PROBLEM STATEMENT

When developing a new product, time is a critical resource. It is very easy to focus on low-impact instead of more impactful ones. The Pareto principle states: *A high percentage of effects in any large system are caused by a low percentage of variables.* [17] That means that if the critical 20 percent of a product's features are used 80 percent of the time, design and testing resources should focus primarily on those features. The remaining 80 percent of the features should be reevaluated to verify their value in the design [17].

This leads us to the core of the problem: *what are the features worth focusing on during development?* Before we can answer this question, we need to define what it means for a feature to be worthy. That is easy - it has to make the product successful. Success in subscription-based models is directly correlated to the retention rate, i.e. the percentage of people that keep return to the application. In conclusion, we should find the features that have the most impact on retention.

1.2 MOTIVATION

I chose to work on finding the most impactful features because it is quite an unique problem, different from the ones found in textbooks. It allowed me to try techniques from different disciplines (statistics, information theory, machine learning) and use previously encountered algorithms in different ways.

1.3 PAPER OUTLINE

Part I of the thesis sets the environment of the problem – chapter 1 has stated the problem and the reason for choosing it, chapter 2 describes the context in which the algorithms and analysis are applied.

Part II lays the theoretical foundation for the rest of the thesis. Chapter 3 includes definitions and intuitive explanations for fundamental machine learning concepts, learning algorithms and common statistical feature selection techniques. Chapter 4 lists works related to this thesis, on either the machine learning side, or customer retention.

Part III is the core of the thesis. In chapter 5 we are introduced to the data, its particularities and churner/retainer relationships. Chapter 6 presents algorithms developed to optimize a supervized learning algorithm on a particular dataset. Finally,

chapter 7 discusses how theoretical methods and the developed pipeline were applied in the current context and provides an interpretation of the results.

Part IV wraps up matters discussed and succinctly states the overall results of the thesis. Chapter 8 provides a comparative overview of findings from the three sources (data analysis, statistical methods and machine learning) and a mention of my slight contribution to the field. Chapter 9 gives insight into how the project can be continued – what are the possible branches and improvements it can undertake.

2

CONTEXT

This paper will present not only theoretical algorithms but also applied results. The first section describes the product chosen as the recipient of the analysis and its particularities. The second section lists the technology stack used and the reason for choosing them.

2.1 THE APPLICATION

Adobe Experience Design (XD) is an all-in-one tool for UX designers which can be used to design and prototype website and mobiles apps quickly.[1]

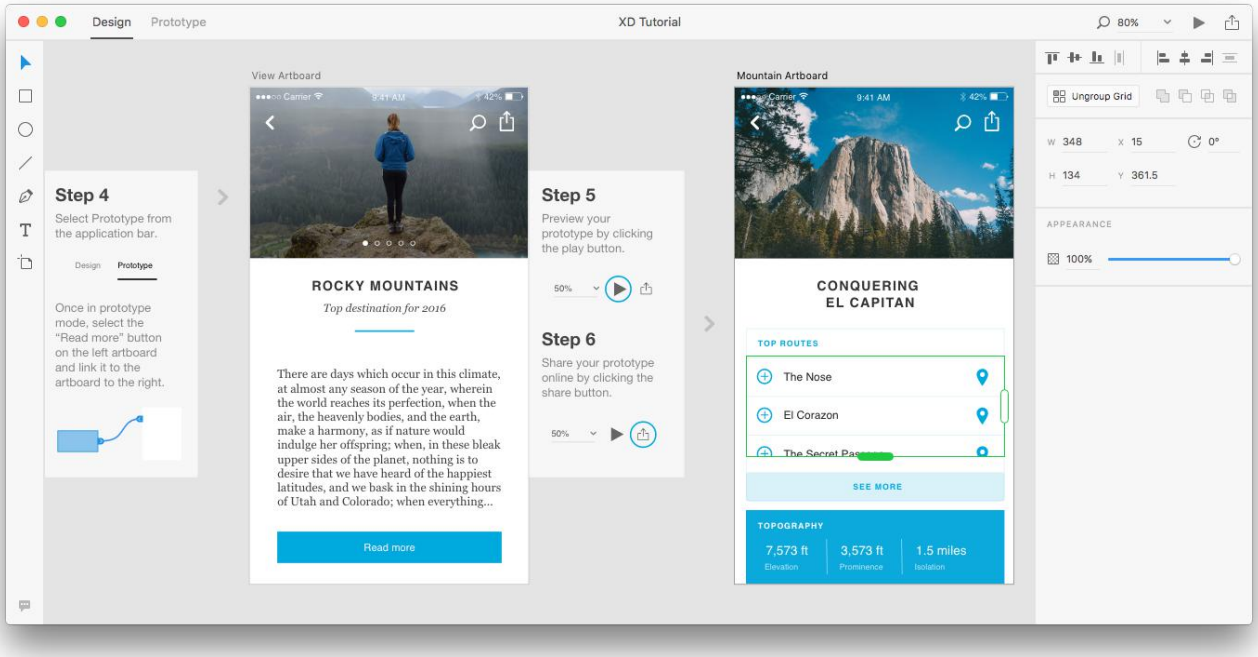


Figure 1: Adobe XD tutorial

Figure 1 showcases XD's more interesting features: design (current), prototype (explained in step 4) and preview (explained in step 5) modes. Two artboards can be observed in the center of the image. A repeat-grid is demonstrated on the right, highlighted in green. The left side of the image lists available drawing tools (rectangle, ellipse, line, path, text and artboard). On the right side we can see horizontal and vertical alignment buttons, masking (add, subtract, intersect, exclude overlap), transform controls (size, position and rotation) and appearance controls (on this object opacity only, other types of objects also feature modifiable fill, border, shadow, blur or text modifications).

1 Adobe Experience Design – adobe.com/products/experience-design.html

At the time of writing this paper, Adobe XD is not fully released. It also provides a number of distinct features and has an analytics system in place. Thus, it is a great candidate for this paper's scope.

Uncommon Features

XD organizes screens of an app in **Artboards** (AB) of selected device's screen size. One item can be repeated multiple times by using a **Repeat Grid** (RG) which can make each item duplication have different content.

The application provides three modes of operation: Design, Prototype and Preview. **Design** is where most of the action happens - creating, editing an positioning objects. **Prototype** handles navigation between screens through **Wires** from one artboard to another. Preview is where the mock-up can be interacted with in real time.

2.1.1 *Instrumentation*

Information about the product is received by analysing event-logs. An event-log, containing time and additional information, is collected every time the user executes a certain action inside the application. Event-log triggers have to be explicitly scripted and provide an additional effort on the development team. Placing these triggers, also known as instrumenting, on each feature can become a significant slow-down. This is true especially in today's preferred fast-development scenarios where experimentation and flexibility is encouraged over rigorous planning.[2] Therefore, lack of instrumentation is an intrinsic problem in this context (early to intermediate development) and should not be seen as a particularity of the chosen application.

Ideal Features

This serves more as a guideline for future application developers – what should be instrumented such that analyses can be performed efficiently.

We will start by mentioning object operations: creating, duplicating/pasting, deleting, transforming (size, position, rotation, corner radius), editing (color, opacity, border, fill, shadow) and advanced object operations: masking, aligning with external items (by snapping), aligning inside other items (by using the dedicated button), selecting one or multiple objects and group locking/unlocking. To these we add application specific actions such as expanding a RG, making an AB the home screen or changing a wire's connection points. Data for these operations should be available for both primitive objects: rectangles, ellipses, lines, paths, text objects, images and for advanced objects too: RGs, ABs and prototyping wires.

Document operations are important as well: the number of documents created, the number of opens and the number of different documents opened, the number of saves, imports, exports and shares. For the mentioned events, one could compute their occurrence count, average use, preference in their group (e.g. rectangles among primitive object draws).

XD's modal structure invites measuring time spent in design, prototype or preview mode, on the welcome screen, in the tutorial document and idle time (with the application unfocused). These can be paired with application-related data such as

2 Agile Software Development – agilemanifesto.org

the number of launches, time spent in the first sessions, preferred time of day (morning, evening etc), number of days active in a month, average time between sessions etc.

Interaction with the application: mouse clicks/taps, keyboard shortcuts, pan and zoom actions can be counted as well. Through these we can also compute the percentage of failed keyboard combinations to indicate a customer need. The targeted medium (iOS, Android, Web, macOS, Windows), based on the UI kit open or AB size, can provide insight about retainers behavior.

Performance is outside of the scope of this feature so load times, application crashes are not too relevant. Localization information, though, can prove useful.

Available Features

Out of the metrics listed above only a fraction were available for analysis thus the results are unfortunately impacted in a negative way. A notable absence is information about editing or transforming an object. We believe it is very important as the numbers of rectangles alone does not tell much about a customer's behavior. What they are doing with them, does. With the addition of this information, action sequences could tell a more complete story, data analysis could reap deeper information and learning models could achieve better performance.

Time Periods

Since the product is not yet released, in the very early months of development, testers and invited members are the ones providing usage data. Adobe XD saw a public preview on 14 march 2016. Opening the application to the public caused a huge increase in the number of users, providing an abundance of usage data.

Unless explicitly stated, analysis and results in this paper are concluded on the latest available data (at the time of writing). A sample size of one month is used.

2.1.2 *Big Data*

Working with the data hundreds of thousands of users providing millions of logs (users and events for one month only) was a challenge itself. When working with such a large volume of data, downloading, processing and using it for machine learning training has to be carefully planned before hitting the start button.

2.1.3 *Processing Performance*

Performance becomes critical as is is accentuated by the high number of entries to process. During the development of feature extraction, caches had to be implemented and used throughout the code so that a slight modification in one function would not cause the whole process to be rerun, wasting days.

2.2 TECHNOLOGIES USED

Technologies used will only be mentioned and given a brief motivation for choosing. Detailed examinations of each of them do not make the object of this paper but the interested reader is invited to check the cited papers.

Before Processing

I will refrain from commenting on data gathering as I have not worked in this area. Hadoop [27] is the choice of distributed storage framework, Hue web interface and Hive editor.

Feature Extraction

Python [22] was chosen because of the fast prototyping it allows and availability of libraries, both for processing and for machine learning. Matlab, R and Java were also considered but ultimately checked fewer boxes than Python.

Pandas [18] is the data processing library used to extract features and manipulate the data. Working with the underlying structure provided by NumPy [26] was sometime required.

Machine Learning and Feature Selection

For machine learning, we chose the **Scikit-Learn** [21] framework for its straightforward API and ease of use. More specialized feature selection methods were provided by **Scikit-Feature** [16].

2.2.1 *Visualization*

Playing a big part in the data analysis and results interpretation, visualization was given appropriate attention through development. Matplotlib [13] providing the engine for **Seaborn** has helped create some of the more advanced charts while the iWork suite provided basic charting functionality.

Part II

BACKGROUND

The second part of the thesis contains sources consulted for development. The first chapter provides an intuitive explanation of the theoretical concepts required to understand the rest of the paper. The second chapter provides a brief overview of results obtained by others on similar tasks or on parts of the problem.

3

ALGORITHMS

This chapter provides definitions and motivations for studying of fundamental concepts on which learning algorithms are built. Two main areas are expanded afterwards: statistical methods for feature selection and specific machine learning algorithms.

3.1 MACHINE LEARNING FUNDAMENTS

Definitions in this section follow a hierarchical structure: the necessity of the concept, intuitive explanation of it and then more and more concrete descriptions of its building blocks. The stopping point is reaching the underlying mathematical formulas.

In regards to the breadth **breadth** of this section, we need to consider that machine learning is a very broad field and we do not intend to provide a summary of its entirety. Each concept will have a number of branching sub-concepts. We will provide a brief overview of each branch then only focus on the one, or ones, applied in this paper. Regarding the **depth** of discussed concepts - the reader is expected to have an intermediate understanding of statistics and thus we will only focus on machine learning-specific concepts and stop when reaching the statistical fundaments. Very formal, mathematical definitions do not make the object of this chapter, as a rigorous definition of each concept touched would make the concept of an entire book by itself (for this purpose we recommend [3] and [12]). Instead, we aim to provide the reader with an **intuitive understanding** of machine learning concepts and the reasons for choosing to apply certain techniques.

Concepts in this section constitute fundamental definitions not belonging to myself, instead **paraphrased** mainly from Bishop [3]. To avoid cluttering the text with references after each concept, they will be included only when a different source is cited.

3.1.1 *What is Machine Learning*

Definition

Machine learning is a subfield of computer science that evolved from the study of pattern recognition and computational learning theory in artificial intelligence. In 1959, Arthur Samuel defined machine learning as a "Field of study that gives computers the ability to learn without being explicitly programmed". Machine learning explores the study and construction of algorithms that can learn from and make predictions on data. Such algorithms operate by building a model from example inputs in order to make data-driven predictions or decisions expressed as outputs, rather than following strictly static program instructions.

Tom M. Mitchell provided a widely quoted, more **formal definition**: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E". This definition is notable for its defining machine learning in

fundamentally operational rather than cognitive terms, thus following Alan Turing's proposal in his paper "Computing Machinery and Intelligence" that the question *"Can machines think?"* be replaced with the question *"Can machines do what we (as thinking entities) can do?"* [28]

Relation to Statistics

Machine learning is closely related to and often overlaps with computational statistics; a discipline which also focuses in prediction-making through the use of computers. It has strong ties to mathematical optimization, which delivers methods, theory and application domains to the field.

Relation to Data Analytics

Within the field of data analytics, machine learning is a method used to devise complex models and algorithms that lend themselves to prediction. These analytical models allow researchers, data scientists, engineers, and analysts to produce reliable, repeatable decisions and results and uncover hidden insights through learning from historical relationships and trends in the data.

Types of learning

Machine learning tasks are typically classified into two broad categories, depending on the nature of the learning feedback available to a learning system. These are:

- **Supervised learning**: the computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs.
- **Unsupervised learning**: no labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).

Other types of learning include semi-supervized learning, where the teacher gives an incomplete training signal: a training set with some (often many) of the target outputs missing or reinforcement learning, where a computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle), without a teacher explicitly telling it whether it has come close to its goal. Another interesting category of machine learning problems is **meta learning** which learns its own inductive bias based on previous experience. [28]

For the purpose of this paper we are interested in supervized learning. An approach similar to meta learning will be described later chapters.

Types output

Another categorization of machine learning tasks arises when one considers the desired output of a machine-learned system:

- In **classification**, inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more (multi-label classification) of these classes. This is typically tackled in a supervised way.

- In **regression**, also a supervised problem, the outputs are continuous rather than discrete.
- In **clustering**, a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.
- **dimensionality reduction** simplifies inputs by mapping them into a lower-dimensional space.

Other types include density estimation which finds the distribution of inputs in some space.

In this paper, we will be dealing with classification tasks and dimensionality reduction.

3.1.2 *Supervized Learning*

Supervised learning is the task of inferring a function from labeled training data. In the training set, each example is a pair consisting of an input object (vector of properties) and a desired output value (called label). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels (in the case of classification) for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way. [28]

In order to solve a given problem of supervised learning, one has to perform the following steps:

1. Determine the **type of training** examples and their feature representation.
2. Gather a **training set**. The training set needs to be representative of the real-world use of the function.
3. Determine the structure of the learned function and corresponding **learning algorithm**.
4. Run the learning algorithm on the gathered training set. Some supervised learning algorithms require **tuning parameters**. These parameters should be adjusted by optimizing performance on a validation subset.
5. **Evaluate** the accuracy of the estimator on a test set that is separate from the training set.

Between the last two steps lies feature selection which in a way is the scope of this whole paper. The **number of features** should not be too large, because of the curse of dimensionality; but should contain enough information to accurately predict the output.

Other things need to be considered as well – heterogeneity of feature data, whether they lead to similar ranges of values; redundancy among data, whether there are highly correlated features; expected shape of data relationships, linear or more complex; etc.

Curse of dimensionality

In learning problems that involve learning an infinite distribution from a finite number of data samples in a high-dimensional feature space with each feature having a number of possible values, an enormous amount of training data are required to ensure that there are several samples with each combination of values. With a fixed number of training samples, the predictive power reduces as the dimensionality increases. [28]

Meta learning

Meta learning is a subfield of Machine learning where automatic learning algorithms are applied on meta-data about machine learning experiments. The main goal is to use such meta-data to understand how automatic learning can become flexible in solving different kinds of learning problems, hence to improve the performance of existing learning algorithms. [28]

Flexibility is very important because each learning algorithm is based on a set of assumptions about the data, its inductive bias. This means that it will only learn well if the bias matches the data in the learning problem. A learning algorithm may perform very well on one learning problem, but very badly on the next. From a non-expert point of view, this poses strong restrictions on the use of machine learning techniques, since the relationship between the learning problem and the effectiveness of different learning algorithms is not yet understood.

3.1.3 *Model Selection*

A wide range of supervised learning algorithms are available, each with its strengths and weaknesses. There is no single learning algorithm that works best on all supervised learning problems. In computational complexity and optimization the **no free lunch** theorem that states that for certain types of mathematical problems, the computational cost of finding a solution, averaged over all problems in the class, is the same for any solution method. No solution therefore offers a shortcut. In machine learning to gain an improved performance, we need to leverage prior information to mach procedures to problems. Such analysis for the task at hand is provided in later, in a separate chapter.

Bias-variance tradeoff

The bias–variance tradeoff (or dilemma) is the problem of simultaneously minimizing two sources of error that prevent supervised learning algorithms from generalizing beyond their training set:

- The **bias** is error from erroneous assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting)
- The **variance** is error from sensitivity to small fluctuations in the training set. High variance can cause overfitting: modeling the random noise in the training data, rather than the intended outputs.

This tradeoff applies to all forms of supervised learning. It has also been invoked to explain the effectiveness of heuristics in human learning. [28]

The bias–variance tradeoff is a central problem in supervised learning. Ideally, one wants to choose a model that both accurately captures the regularities in its training data, but also generalizes well to unseen data. Unfortunately, it is typically impossible to do both simultaneously. High-variance learning methods may be able to represent their training set well, but are at risk of overfitting to noisy or unrepresentative training data. In contrast, algorithms with high bias typically produce simpler models that don't tend to overfit, but may underfit their training data, failing to capture important regularities.

Models with low bias are usually more complex (e.g. higher-order polynomials), enabling them to represent the training set more accurately. In the process, however, they may also represent a large noise component in the training set, making their predictions less accurate - despite their added complexity. In contrast, models with higher bias tend to be relatively simple (low-order or even linear polynomials), but may produce lower variance predictions when applied beyond the training set.

3.1.4 *Generalization Power*

A core objective of a learner is to generalize from its experience. Generalization in this context is the ability of a learning machine to perform accurately on new, unseen examples/tasks after having experienced a learning data set. The training examples come from some generally unknown probability distribution (considered representative of the space of occurrences) and the learner has to build a general model about this space that enables it to produce sufficiently accurate predictions in new cases. Because training sets are finite and the future is uncertain, learning theory usually does not yield guarantees of the performance of algorithms. Instead, probabilistic bounds on the performance are quite common.

For best generalization, complexity of the hypothesis should match the complexity of the function underlying the data. If the hypothesis is less complex than the function, we've underfitted. Then, we increase the complexity, the training error decreases. But if our hypothesis is too complex, we've overfitted. After then, we should find the hypothesis that has the minimum training error.

Learning algorithms train a model based on a finite set of training data. During this training, the model is evaluated based on how well it predicts the observations contained in the training set. In general, however, the goal of a machine learning scheme is to produce a model that generalizes, that is, that predicts previously unseen observations. Overfitting occurs when a model fits the data in the training set well, while incurring larger generalization error. [28]

Overfitting

When overfit, a model describes random error or noise instead of the underlying relationship. Overfitting occurs when a model is excessively complex, such as having too many parameters relative to the number of observations. A model that has been overfit has poor predictive performance, as it overreacts to minor fluctuations in the training data.

The possibility of overfitting exists because the criterion used for training the model is not the same as the criterion used to judge the efficacy of a model. In

particular, a model is typically trained by maximizing its performance on some set of training data. However, its efficacy is determined not by its performance on the training data but by its ability to perform well on unseen data. Overfitting occurs when a model begins to "memorize" training data rather than "learning" to generalize from trend. [28]

The most obvious consequence of overfitting is poor performance on the validation dataset. Another negative consequence is that an overfitted function is likely to request more information about each item in the validation dataset than does the optimal function; gathering this additional unneeded data can be inconvinient.

In order to avoid overfitting, it is necessary to use additional techniques (e.g. cross-validation, regularization, early stopping and pruning parameters or model comparison), that can indicate when further training is not resulting in better generalization. The basis of these techniques is either:

- to explicitly penalize overly complex models, or
- to test the model's ability to generalize by evaluating its performance on a set of data not used for training, which is assumed to approximate the typical unseen data that a model will encounter.

We will make use of cross-validation, regularization, early stopping and pruning.

3.1.5 *Regularization*

Regularization can be used to learn simpler models by inducing sparsity. Without bounds on the complexity of the function space available, a model will be learned that incurs zero error on the training set. If measurements were made with noise, this model may suffer from overfitting and display poor performance on unseen data. Regularization introduces a penalty for exploring certain regions of the function space used to build the model, which can improve generalization. [28]

Early stopping

Early stopping can be viewed as regularization in time. Intuitively, a training procedure like gradient descent will tend to learn more and more complex functions as the number of iterations increases. By regularizing on time, the complexity of the model can be controlled, improving generalization.

In practice, early stopping is implemented by training on a training set and measuring accuracy on a statistically independent validation set. The model is trained until performance on the validation set no longer improves. The model is then tested on a testing set. [28]

Pruning

Pruning is a technique in machine learning that reduces the size of decision trees by removing sections of the tree that provide little power to classify instances. Pruning reduces the complexity of the final classifier, and hence improves predictive accuracy by the reduction of overfitting.

One of the questions that arises in a decision tree algorithm is the optimal size of the final tree. A tree that is too large risks overfitting the training data and poorly

generalizing to new samples. A small tree might not capture important structural information about the sample space. However, it is hard to tell when a tree algorithm should stop because it is impossible to tell if the addition of a single extra node will dramatically decrease error. This problem is known as the horizon effect. A common strategy is to grow the tree until each node contains a small number of instances then use pruning to remove nodes that do not provide additional information.

Pruning should reduce the size of a learning tree without reducing predictive accuracy as measured by a cross-validation set. Various techniques for tree pruning differ in the measurement that is used to optimize performance. [28]

3.1.6 *Cross-validation*

Cross-validation (CV) is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. In a prediction problem, a model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (testing dataset). The goal of cross validation is to define a dataset to "test" the model in the training phase (i.e., the validation dataset), in order to limit problems like overfitting, give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem), etc.

One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set). To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds.

Furthermore, one of the main reasons for using cross-validation instead of using the conventional validation (e.g. partitioning the data set into two sets of 70% for training and 30% for test) is that the error on the training set in the conventional validation is not a useful estimator of model performance and thus the error on the test data set does not properly represent the assessment of model performance. This may be because there is not enough data available or there is not a good distribution and spread of data to partition it into separate training and test sets in the conventional validation method. In these cases, a fair way to properly estimate model prediction performance is to use cross-validation as a powerful general technique.

In **k-fold** cross-validation, the original sample is randomly partitioned into k equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k - 1 subsamples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged (or otherwise combined) to produce a single estimation. The advantage of this method over repeated random sub-sampling is that all observations are used for both training and validation, and each observation is used for validation exactly once.

Monte Carlo cross-validation randomly splits the dataset into training and validation data. For each such split, the model is fit to the training data, and predictive accuracy is assessed using the validation data. The results are then averaged over the splits. The advantage of this method (over k-fold cross validation) is that the pro-

portion of the training/validation split is not dependent on the number of iterations (folds). The disadvantage of this method is that some observations may never be selected in the validation subsample, whereas others may be selected more than once. In other words, validation subsets may overlap. This method also exhibits Monte Carlo variation, meaning that the results will vary if the analysis is repeated with different random splits.

Stratified variants of k-fold and Monte Carlo cross-validation each fold contains roughly the same proportions of the two types of class labels. This is particularly useful in binary classification with an unbalanced representation of the two response values in the data. [28]

We have chosen a 3-fold stratified CV in our approach.

3.1.7 *Hyper-parameter Optimization*

Hyperparameter optimization or model selection is the problem of choosing a set of hyperparameters for a learning algorithm, usually with the goal of optimizing a measure of the algorithm's performance on an independent data set. Often cross-validation is used to estimate this generalization performance. Hyperparameter optimization contrasts with actual learning problems, which are also often cast as optimization problems, but optimize a loss function on the training set alone. In effect, learning algorithms learn parameters that model/reconstruct their inputs well, while hyperparameter optimization is to ensure the model does not overfit its data by tuning, for example, regularization. [28]

Horizon effect

The horizon effect, is a problem where, in many games, the number of possible states or positions is immense and computers can only feasibly search a small portion of it, typically a few plies down the game tree. Thus, for a computer searching only five plies, there is a possibility that it will make a detrimental move, but the effect is not visible because the computer does not search to the depth of the error (i.e. beyond its "horizon"). The same concept applies when in hyper-parameter search.

Grid Search

The traditional way of performing hyperparameter optimization has been grid search, or a parameter sweep, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set.

Random Search

Since grid searching is an exhaustive and therefore potentially expensive method, a randomized search simply samples parameter settings a fixed number of times. It has been found to be more effective in high-dimensional spaces than exhaustive search. [28]

3.1.8 *Feature Selection*

Feature Selection is the process of selecting a subset of relevant features for use in model construction in order two achieve for three main reasons:

- simplification of models to make them easier to interpret,
- shorter training times,
- enhanced generalization.

The central premise when using a feature selection technique is that the data contains many features that are either redundant or irrelevant, and can thus be removed without incurring much loss of information. Redundant or irrelevant features are two distinct notions, since one relevant feature may be redundant in the presence of another relevant feature with which it is strongly correlated.

Recursive Feature Elimination

Given an estimator that assigns weights to features (e.g., the coefficients of a linear model), recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features and weights are assigned to each one of them. Then, features whose absolute weights are the smallest are pruned from the current set features. That procedure is recursively repeated on the pruned set until the number of features reaches one. [21]

3.1.9 *Performance Metrics*

In order to evaluate a model's performance and also compare different learning algorithms, we need to know what it means to perform better. In this section we define a number of performance metrics and give an intuitive explanation of what they mean.

Confusion Matrix

NAME	PREDICTED	ACTUAL	ERROR
True Positive (tp)	retention	retained	
False Positive (fp)	retention	churner	type I
False Negative (fn)	churn	retained	type II
True Negative (tn)	churn	churner	

Table 1: Confusion matrix explanation

Intuitively, true positives can be thought of as being too optimistic, predicted retention on a churner while false negatives would mean being too cautious, predict churn on a retainer.

Accuracy, defined as

$$\frac{tp + tn}{tp + fp + tn + fn}$$

measures correct hits, of any kind.

Precision, with the formula

$$\frac{tp}{tp + fp}$$

expresses how trustworthy a prediction of retention is. Having a high precision means that when outputting retention, the estimator is probably correct.

Recall, having the formula

$$\frac{tp}{tp + fn}$$

is the ability to find all retainers, i.e. correctly classify when facing a retainer. The higher the recall, the better the estimator is at finding most retainers.

In cases of skewed classes, e.g. very low retention rate, an estimator always predicting churn would have F1 is a compound metric defined as

$$2 \times \frac{precision \times recall}{precision + recall}$$

and becomes very useful when a single numeric value is needed to compare classifier performance.

Many other scores exist. Basic ones such as sensitivity (or true positive rate), specificity (or true negative rate), negative predictive value, fall-out (or false positive rate), false discovery rate, miss rate (or false negative rate) [28] etc. Complex ones as well, such as F-beta, Hamming loss, hinge loss, Jaccard similarity score, Matthews correlation coefficient [21] and others. In this paper we will evaluate performance through accuracy, precision, recall, F1 and a custom score defined in later chapters.

3.2 LEARNING MODELS

After providing high level definitions of machine learning algorithms and the necessary concepts, we detail actual algorithms used in the industry.

We divide learning models into families of algorithms. Each family has its training and prediction process explained. We also mention each algorithm's strengths and weaknesses and how parameters impact the bias-variance tradeoff.

3.2.1 *Neural Networks*

Artificial **neural networks** (NNs) are a family of models inspired by biological neural networks which are used to estimate or approximate functions that can depend on a large number of inputs and are generally unknown. Artificial neural networks are generally presented as systems of interconnected "neurons" which exchange messages between each other. The connections have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs and capable of learning.

A **feedforward neural network** is an artificial neural network wherein connections between the units do not form a cycle. The feedforward neural network was the first and simplest type of artificial neural network devised. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

Perceptron

The simplest kind of neural network is a single-layer perceptron network, which consists of a single layer of output nodes; the inputs are fed directly to the outputs via a series of weights. In this way it can be considered the simplest kind of feedforward network. It is an algorithm that provides binary classification by calculating the linear combination of its real-valued or boolean inputs and passes it through a threshold activation function. The perceptron discriminates the data based on the dot product of the inputs and the learned weights. The Perceptron is limited in the sense that it can only distinguish between two linearly-separable classes. [28]

Multilayer Perceptron

A multilayer perceptron (MLP) is a feedforward artificial neural network model. An MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. Except for the input nodes, each node is a neuron with a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training the network. MLP is a modification of the standard linear perceptron and can distinguish data that are not linearly separable. A multilayer perceptron consists of three or more layers (an input and an output layer with one or more hidden layers) of nonlinearly-activating nodes. [28]

The variance increases and the bias decreases with the number of hidden units. Regularization can be applied to decrease their variance at the cost of increased bias.

The **activation function** of a node defines the output of that node given an input or set of inputs. A standard computer chip circuit can be seen as a digital network of activation functions that can be "ON" (1) or "OFF" (0), depending on input.

Backpropagation, an abbreviation for "backward propagation of errors", is a method of training artificial neural networks used in conjunction with an optimization method such as gradient descent. The method calculates the gradient of a loss function with respect to all the weights in the network. The gradient is fed to the optimization method which in turn uses it to update the weights, in an attempt to minimize the loss function. Backpropagation requires a known, desired output for each input value in order to calculate the loss function gradient. [28]

Gradient descent is an optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. In mathematics, the gradient is a generalization of the usual concept of derivative of a function in one dimension to a function in several dimensions. [28]

3.2.2 *Stochastic Gradient Descent*

Stochastic gradient descent (SGD) is a stochastic approximation of the gradient descent optimization method for minimizing an objective function that is written as

a sum of differentiable functions. Stochastic approximation attempts to find zeroes or extrema of functions which cannot be computed directly, but only estimated via noisy observations.

3.2.3 *Decision Trees*

Decision tree learning uses a decision tree graph as a predictive model which maps observations about an item to conclusions about the item's target value. Tree models where the target variable can take a finite set of values are called classification trees. In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Each interior node corresponds to one of the input variables; there are edges to children for each of the possible values of that input variable. Each leaf represents a value of the target variable given the values of the input variables represented by the path from the root to the leaf.

A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node has all the same value of the target variable, or when splitting no longer adds value to the predictions. This process of top-down induction of decision trees is an example of a greedy algorithm, and it is a very common strategy for learning decision trees from data. [28]

The depth of the tree determines the variance. Decision trees are commonly pruned to control it.

Random forests operate by constructing a multitude of decision trees at training time and outputting the mode of the classes predicted by each.

3.2.4 *Support Vector Machines*

Support vector machines (SVMs) are a type of non-probabilistic binary linear classifier. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on. [28]

A **probabilistic classifier** is a classifier that is able to predict, given a sample input, a probability distribution over a set of classes, rather than only outputting the most likely class that the sample should belong to. Probabilistic classifiers provide classification with a degree of certainty, which can be useful when combining classifiers into ensembles.

More formally, a support vector machine constructs a hyperplane or set of hyperplanes in a high-dimensional space which can be used for classification. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (called functional margin), since in general the larger the margin the lower the generalization error of the classifier.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

Kernel methods map the data into higher dimensional spaces in the hope that in this higher-dimensional space the data could become more easily separated or better structured. This mapping function, however, hardly needs to be computed because of a tool called the kernel trick. The **kernel trick** provides a bridge from linearity to non-linearity to any algorithm that can expressed solely on terms of dot products between two vectors. It comes from the fact that, if we first map our input data into a higher-dimensional space, a linear algorithm operating in this space will behave non-linearly in the original input space. The "trick" resides in the fact that the mapping does not need to be ever computed. In an algorithm expressed only in terms of inner products between vectors, all we need to do is replace the product with a kernel function whenever it is used. This way, the algorithm can be carried into a higher-dimension space without explicitly mapping the input points into this space. [23]

3.2.5 *Naive Bayes*

A Bayesian network is a probabilistic graphical model that represents a set of random variables and their conditional dependencies via a directed acyclic graph (DAG). Formally, Bayesian networks are DAGs whose nodes represent random variables in the Bayesian sense: they may be observable quantities, latent variables, unknown parameters or hypotheses. Edges represent conditional dependencies; nodes that are not connected (there is no path from one of the variables to the other in the bayesian network) represent variables that are conditionally independent of each other. Each node is associated with a probability function that takes, as input, a particular set of values for the node's parent variables, and gives (as output) the probability (or probability distribution, if applicable) of the variable represented by the node. [28]

Naive Bayes NB is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. It is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. Naive Bayes variants perform very well in text classification tasks.

Gaussian NB does the following: when dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a Gaussian distribution.

In the case of **Multinomial** NB, samples (feature vectors) represent the frequencies with which certain events have been generated by a multinomial $(p_1 ... p_n)$ where p_i is the probability that event i occurs.

In the multivariate **Bernoulli** event model, features are independent booleans (binary variables) describing inputs.

3.2.6 *Discriminant Analysis*

Linear Discriminant Analysis (LDA) is a method to find a linear combination of features that characterizes or separates two or more classes. LDA is closely related to analysis of variance (ANOVA) and regression analysis, which also attempt to express one dependent variable as a linear combination of other features or measurements. However, ANOVA uses categorical independent variables and a continuous dependent variable, whereas discriminant analysis has continuous independent variables and a categorical dependent variable (i.e. the class label). Logistic regression is more similar to LDA than ANOVA is, as they also explain a categorical variable by the values of continuous independent variables. These other methods are preferable in applications where it is not reasonable to assume that the independent variables are normally distributed, which is a fundamental assumption of the LDA method. [28]

A **quadratic** classifier is used to separate measurements of two or more classes a quadric surface. It is a more general version of the linear classifier where it is assumed that the measurements from each class are normally distributed. Unlike LDA however, in QDA there is no assumption that the covariance of each of the classes is identical.

3.2.7 *Logistic Regression*

Logistic regression (LR) is a regression model that measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function. The output of binary regression is a probability of class membership; combined with some decision rule that puts observations into classes turns this regressor into a classifier.

3.2.8 *Neighborhood*

K-Nearest Neighbors (KNN) is a non-parametric method used for classification. The input consists of the k closest training examples in the feature space, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k is one, then the object is simply assigned to the class of that single nearest neighbor. A high value of k leads to high bias and low variance. [28]

A drawback of the basic "majority voting" classification occurs when the class distribution is skewed. That is, examples of a more frequent class tend to dominate the prediction of the new example, because they tend to be common among the k nearest neighbors due to their large number. One way to overcome this problem is to weigh the classification, taking into account the distance from the test point to each of its k nearest neighbors. The class of each of the k nearest points is multiplied by a weight proportional to the inverse of the distance from that point to the test point.

A **nearest centroid** classifier or assigns to observations the label of the class of training samples whose mean (centroid) is closest to the observation.

3.2.9 *Anomaly Detection*

Anomaly detection (also outlier detection) is the identification of items, events or observations which do not conform to an expected pattern or other items in a dataset. Typically the anomalous items will translate to some kind of problem such as bank fraud, a structural defect, medical problems or errors in a text. [28]

Anomaly detection is not applicable in our case because retainers are not as rare as to be labeled as exceptional.

3.2.10 *Recommender systems*

Recommender systems seek to predict the rating or preference that a user would give to an item. The most popular areas of application are: movies, music, news, books, research articles, search queries, social tags, and products in general.

They are not usable in our case because unlike movies or music, product features can not be treated as homogenous entities. They differ greatly in the way they are used and thus do not adhere to recommender systems assumptions.

3.2.11 *Other*

There are many other methods of classification but we will not detail them, as they are not used in this paper. They include Self Organizing Maps (SOM), Genetic algorithms (GA), Fuzzy Networks etc.

3.2.12 *Ensemble Learning*

Ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms.

Supervised learning algorithms are commonly described as performing the task of searching through a hypothesis space to find a suitable hypothesis that will make good predictions with a particular problem. Even if the hypothesis space contains hypotheses that are very well-suited for a particular problem, it may be very difficult to find a good one. Ensembles combine multiple hypotheses to form a (hopefully) better hypothesis. Evaluating the prediction of an ensemble typically requires more computation than evaluating the prediction of a single model, so ensembles may be thought of as a way to compensate for poor learning algorithms by performing a lot of extra computation. [28]

An ensemble is itself a supervised learning algorithm, because it can be trained and then used to make predictions. The trained ensemble, therefore, represents a single hypothesis. This hypothesis, however, is not necessarily contained within the hypothesis space of the models from which it is built. Thus, ensembles can be shown to have more flexibility in the functions they can represent. This flexibility can, in theory, enable them to over-fit the training data more than a single model would, but in practice, some ensemble techniques (especially bagging) tend to reduce problems related to over-fitting of the training data.

Empirically, ensembles tend to yield better results when there is a significant diversity among the models. Many ensemble methods, therefore, seek to promote diversity among the models they combine. Although perhaps non-intuitive, more random

algorithms (like random decision trees) can be used to produce a stronger ensemble than very deliberate algorithms. Using a variety of strong learning algorithms, however, has been shown to be more effective than using techniques that attempt to dumb-down the models in order to promote diversity.

Bagging

Bootstrap aggregating, often abbreviated as bagging, involves having each model in the ensemble vote with equal weight. In order to promote model variance, bagging trains each model in the ensemble using a randomly drawn subset of the training set. As an example, the random forest algorithm combines random decision trees with bagging to achieve very high classification accuracy.

Boosting

Boosting is a machine learning ensemble meta-algorithm for primarily reducing bias, and also variance in supervised learning, and a family of machine learning algorithms which convert weak learners to strong ones. Boosting is based on the question posed by Kearns and Valiant *Can a set of weak learners create a single strong learner?* A weak learner is defined to be a classifier which is only slightly correlated with the true classification (it can label examples better than random guessing). In contrast, a strong learner is a classifier that is well-correlated with the true classification.

Boosting involves incrementally building an ensemble by training each new model instance to emphasize the training instances that previous models mis-classified. In some cases, boosting has been shown to yield better accuracy than bagging, but it also tends to be more likely to over-fit the training data. [28]

Gradient Bossting

The idea of gradient boosting originated in the observation by Leo Breiman that boosting can be interpreted as an optimization algorithm on a suitable cost function. Algorithms that optimize a cost function over function space by iteratively choosing a function (weak hypothesis) that points in the negative gradient direction. This functional gradient view of boosting has led to the development of boosting algorithms in many areas of machine learning and statistics beyond regression and classification. [28]

ADA-Boost

AdaBoost, short for "Adaptive Boosting", is a meta-algorithm formulated by Yoav Freund and Robert Schapire who won the Gödel Prize in 2003 for their work. It can be used in conjunction with many other types of learning algorithms to improve their performance. The output of the other learning algorithms (weak learners) is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers. In some problems it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing (e.g.,

their error rate is smaller than 0.5 for binary classification), the final model can be proven to converge to a strong learner.

While every learning algorithm will tend to suit some problem types better than others, and will typically have many different parameters and configurations to be adjusted before achieving optimal performance on a dataset, AdaBoost (with decision trees as the weak learners) is often referred to as the best out-of-the-box classifier. When used with decision tree learning, information gathered at each stage of the AdaBoost algorithm about the relative hardness of each training sample is fed into the tree growing algorithm such that later trees tend to focus on harder-to-classify examples.

3.3 STATISTICAL BASED METHODS

Feature selection is a very interesting part of machine learning. In this chapter we explore statistical methods for accomplishing this task which have proved effective in the industry.

3.3.1 *Variance and Covariance*

Variance Threshold

Variance is a measure of how far a set of numbers are spread out from their mean. Variance Threshold is a simple baseline approach to feature selection. It removes all features whose variance doesn't meet the threshold threshold, i.e. features that don't change too much from sample to sample.

Correlation Matrices

Covariance is a measure of how much two random variables change together. The sign of the covariance shows the tendency in the linear relationship between the variables, i.e. variables that change together will have corresponding greater positive values.

When analyzing a number of variables, it is useful to build a covariance matrix, where the element in the n, m position corresponds to the covariance of the nth and mth variable. Closely related to the covariance matrix is the correlation matrix whose elements range between -1 and 1. The principal diagonal is the correlation of the variable with itself, which is always 1.

3.3.2 *Statistical Based*

Chi-Squared

Chi-squared tests how well the observed distribution of data fits with the distribution that is expected. It measures how likely it is that the observed distribution is due to chance i.e. independent from the expected distribution. [9] Chi-squared can be used in feature selection by removing features that are the most likely to be independent of class and therefore irrelevant for classification. [21] The higher the chi-squared score, the more important the feature.

T-Score

T-tests are used to compare the means of two groups when the population variance is unknown. When running a T-test, the bigger the T-score, the more likely it is that the results are repeatable. In feature selection. This mean that it can be used in feature selection by removing features with a low T-score. [7]

Analysis of Variance

Analysis of variance (ANOVA) is a collection of statistical models used to analyze the differences among group means. In the ANOVA setting, the observed variance in a particular variable is partitioned into components attributable to different sources of variation. In its simplest form, ANOVA provides a statistical test of whether or not the means of several groups are equal. ANOVAs are useful for comparing (testing) three or more means (groups or variables) for statistical significance. [28] A higher score means higher importance.

Gini Index

The Gini index measures the inequality among values of a frequency distribution. A Gini index of zero expresses perfect equality, where all values are the same. A Gini index of 1 expresses maximal inequality among values (e.g. the values lie only in the higher and lower quantiles instead of uniformly spread between the minimum and the maximum values). Used in feature selection, the lower the Gini index, the more important the feature is. [28]

Correlation Feature Selection

The Correlation Feature Selection (CFS) evaluates the worth of a subset of features. It takes into account the usefulness of individual features for predicting the class label along with the level of inter-correlation among them. It is based on the following hypothesis: Good feature subsets contain features highly correlated with the class, yet uncorrelated with each other. [28]

3.3.3 *Similarity Based*

Fisher

Maximum-likelihood estimation (MLE) is a method of estimating the parameters of a statistical model given only a sample of the overall data. Fisher's score is a numerical way of solving the MLE problem. Fisher's criterion seeks to projects high-dimensional data onto a line and discriminate data in this one-dimensional space. The projection maximizes the distance between the means of the two classes while minimizing the variance within each class. The higher the Fisher's score, the more important the feature is. [11]

ReliefF

Relief is an older feature selection algorithm initially developed for binary classification. It has noise-tolerant and robust to feature interactions, however, it does not dis-

criminate between redundant features. The algorithm iteratively updates the weight of each feature by looking at the distance between the nearest hit and nearest miss for each class. ReliefF is an updated version that uses a different norm and updating rule. A higher reliefF score, means a greater importance. [14]

3.3.4 *Sparse Learning Based*

Sparse learning refers to a collection of methods of learning that seek a trade-off between predicting performance and sparsity of the result. In a sparse learning task, the measure of performance is not the sole concern: it is desired to obtain a good result with as few features as possible. [8]

Notable algorithms include:

- Robust Feature Selection (RFS) [20]
- LL L21 and LS L21 [1]

3.3.5 *Information Theoretical Based*

Information theory spurs a number of techniques useful in feature selection. Most of them are based on manipulating the mutual information of the given features.

Mutual Information

Mutual information (MI) of two variables is a measure of the mutual dependence between them. More specifically, it quantifies the "amount of information" obtained about one variable, through the other. [28]

Conditional Mutual Information (CMI) is, intuitively, the expected value of the mutual information of two variables given the value of a third. [28]

Intuitively, the expected value of a variable is the long-run average value of repetitions of the experiment it represents. Less roughly, the law of large numbers states that the arithmetic mean of the values almost surely converges to the expected value as the number of repetitions approaches infinity. [28]

Notable algorithms for feature selection based on CMI:

- Conditional Mutual Information Maximization (CMIM) [10]
- Joint Mutual Information (JMI) [29]
- Mutual Information Feature Selection (MIFS) [2]
- Mutual Information Matrix (MIM) [15]
- Max-dependency, Max-Relevance and Min-Redundancy (MRMR) [16]
- Conditional Infomax Feature Extraction (CIFE) [16]

Double Input Symmetrical Relevance (DISR)

This measure evaluates the additional information that a set of variables provides about the output with respect to the sum of each single variable contribution. [19]

4

RELATED WORKS

To avoid duplicating research and also to benefit from the findings of others, one should always consult related works before embarking on the development of a new project. I have not used the works others directly but their findings and experiment results provided great **guidance** in approaching this subject.

In this chapter we will provide a brief **overview** of the consulted articles. Sources of machine learning fundaments are not included here, as they have been covered in the previous chapter. This part focuses on papers regarding specific techniques, overviews of popular methods or experiments/case studies.

4.1 FEATURE SELECTION

Feature selection is a branch of machine learning that stirs quite the interest among researchers due to both its applicability and relative difficulty. This section presents two comparisons: one in the field of feature selection results and the other evaluating learning algorithms.

"A Survey on Feature Selection Methods"

Chandrashekar and Sahin [5] provide an overview on feature selection techniques in the article. They divide methods into Filter, Wrapper and Embedded. Filter methods include correlation and MI approaches, Mentioned Wrapper methods are Sequential Forward Floating Selection, and genetic techniques. Embedded methods employ MRMR, RFE, Network Pruning and Lazy Feature Selection. Classifiers used are SVMs and RBF networks.

They demonstrate the applicability of the methods on medical datasets. On a diabetes prediction task best performance is achieved by a SVM (80%) by using 7 features out of the available 8. On a liver disorder related dataset, the RBF network performs the best (73%) by training on 4 of the available 6 features.

"An Empirical Comparison of Supervised Learning Algorithms"

Although not a paper that strictly targets feature selection techniques, Caruana and Niculescu-Mizil [4] conduct a factual experiment of most popular learning algorithms. They compare ten different models, evaluating their performance using a variety of performance metrics.

The highest overall scores were achieved by boosted trees with random forests as a close second, SVMs in third and shallow neural nets on the fourth position. These findings are not to be taken as absolute truth because, as the authors themselves warn, there is significant variability across problems and metrics – even the best models some- times perform poorly, and models with poor average performance occasionally perform exceptionally well.

4.2 CUSTOMER RETENTION

Most works in the field of churn detection are geared towards subscription-model businesses with constant customer feedback available. Two such environments are provided by Telecommunications and Web platform services. The requirements of the problem addressed in this paper are dissimilar from both thus requiring picking only generalizable concepts from related works.

"A New Neural Network Based Customer Profiling Methodology for Churn Prediction"

Authors Tiwari, Hadden, and Turner [24] propose three NN architectures: (1 neuron, 1 layer), (3 neurons, 2 layers), and (6 neurons, 4 layers) for customer churn prediction.

Their case study is based on real industry data as well, with a dataset of 8400 customers, 35 features for each and a churn rate of 29%. The models were trained on one month of usage and the architecture that generalized the best (the second one) achieved 65% accuracy.

"Customer Churn Prediction by Hybrid Neural Networks"

An interesting approach is handled by Tsai and Lu. They consider two hybrid models for churn prediction: NN combined with another NN and SOM combined with NN. The first model has the task of data reduction by filtering unrepresentative training data while the second one does the actual classification.

The dataset they experiment on contains 37 000 subscribers with a retention rate of 70%. The hybrid model that performs the best is the second one, with an accuracy of 92%. The architecture chosen contains a single hidden layer with 32 nodes.

"If You Build It They Might Stay: Retention Mechanisms in World of Warcraft"

An analysis run by Debeauvais et al. [6] focuses more on the psychology of customers and ways the application developers used to make it thrive. To define retention, their study uses weekly play time and length of active time period.

The paper emphasises the impact of social aspects for long term retention. Such findings are outside the scope of our problem but they do hint that more attention should be shown to social-related features (e.g. the sharing functionality).

Part III

DEVELOPMENT

The third part is the core of the thesis – it contains the *how* of the matter. The first chapter details the data analysis process and its outcomes. The second chapter is comprised of the methods I have developed that are generalizable to similar problems. In the third chapter results from both statistical and machine learning methods are reviewed and interpreted.

5 DATA ANALYSIS

In order to pick the right learning algorithm, we have to understand the data first. In this chapter we explore event-based features, time-based features and action sequences by comparing their distributions and the retainer/churner differences.

Throughout the chapter, users will be divided between retainers and churners. A **retainer** is a customer that will keep using the application and a **churner** is one that will not. What it means for a user to be retained is a discussion in itself and will be discussed in 5.1. Before we dive in, let's define retention rate.

$$\text{retention rate} = \frac{\text{retainers}}{\text{total users}}$$

To promote consistency, charts comparing retainers to churners will feature yellow and blue colors, respectively.

Outliers

The data that enters the analysis process is filtered at both ends by ignoring:

- users have spent less than 15 seconds in the app
- top 2% most active users

We cut uninterested users that do not give the product even a fair chance. 15 seconds from the starting the application (including loading) to closing it barely gives you enough time to read the title. Therefore we consider these accidental opens and do not include them in the analysis. The other possibility is that they have realized in less than 15 seconds that this product is completely unfit for them that it is not worth giving it a good inspection. The inclusion of such users would needlessly skew the data and not help the main goal: making users stay.

By disregarding anything past the 98% quantile, we alleviate another measuring error: users involved in the testing process and users with habits very far from normal use. Inclusion of these two kinds of users would not bring any additional information about retainers usage, instead pollute the data.

5.1 RETENTION DEFINITION

As the data comes unlabelled and we are not given a formal definition of customer retention, we propose several metrics that can be used for labelling:

- **time spent** — total in-app time over the measured period
- **launches** — the number of unique sessions
- **days span** — the number of days between the first and the last launch

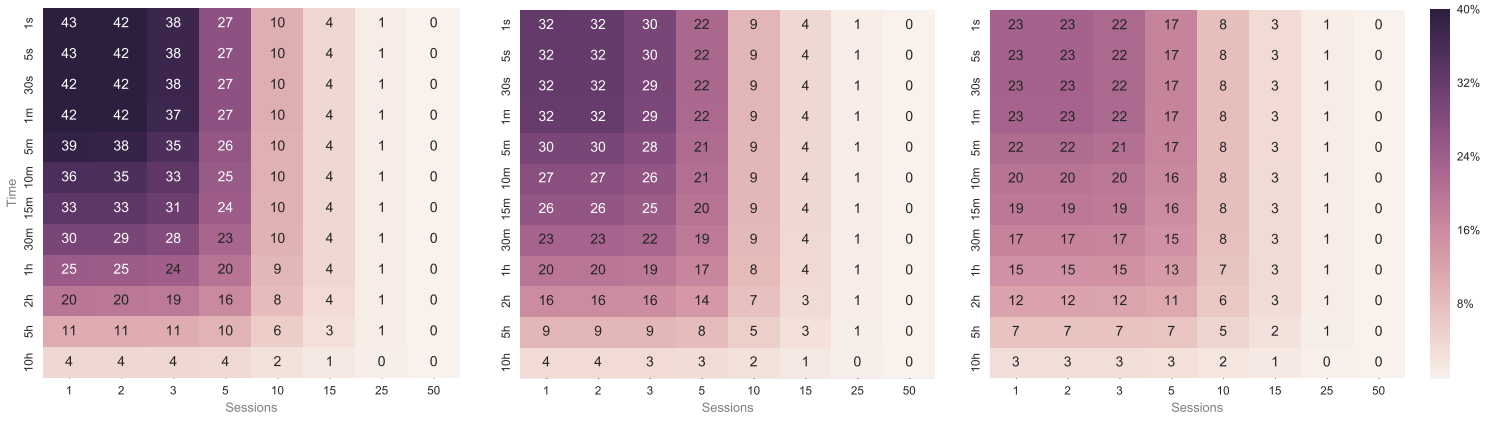
Best practices dictate that labels should not be picked after the data, instead they should come from an external, unbiased source. With this in mind, we will look at how various parameter values divide the customers into retainers and churners.

5.1.1 *Discrete*

We'll start by looking at binary labelling (0 or 1), as this classification seems very suitable for the task at hand.

Distribution

Visualising all three parameter is pretty difficult without 3D axes, which have the downside of being hard to read. One dimension would be enough to plot one parameter. Two parameters can be accommodated using heat-maps. The third parameter (days span) varies across charts.



(a) 10 Days (b) 15 Days (c) 20 Days

Figure 2: Discrete retention rate

Figure 2 gives us a first insight on how the data can be split. Very high retention is observed when all three parameters have low values — at least one second spent, at least one launch and at least ten days span. We can see that going over fifteen sessions, very few customers can still be classified as retained. Over five hours of total time thins percentages considerably as well. Both time and number of sessions seem to slice the data in the same way, modulus a scaling factor.

We refrain from choosing the parameters such that the heat-maps are centered. That would mean to set the definition based on the data, not the other way around. The chosen retention discrete parameter values are: **15** days span, **3** launches and **10** minutes spent. This combination generates yields a retention rate of **26.1%**.

5.1.2 *Continuous*

A continuous retention score (between 0 and 1) can be used in predicting the probability that a given customer will retain to the application. Using the same parameters as discrete retention, we have to come up with a way to map the three of them to a retention probability.

First we decide on what is the **importance** of each parameter. Picking the days span as the most important one is not a good idea. A user that starts the application today, opens it again over three weeks and only spends five minutes overall should not be categorized as retained. The number of sessions is also not solely representative of a retainer. Usage frequency may vary between two retainers — one might use

the application in bursts of 20 minutes five times a day and the other might have one long session of 2 hours. By placing high emphasis on the number of launches we incorrectly penalize the second user's retention probability. We consider time spent the most distinctive feature of a retainer. You can definitely say someone will come back once they've invested a significant amount of time in the product.

The second thing we tackle is the **reference** values of each parameter. Having decided that time spent is the most important factor, how much *time* is enough to confidently call a user retained? We picked the following values:

1. 30 days span
2. 4×20 hours spent
3. 100 launches

This means that if a user has spent 4×20 hours (4 hours on each workday of the month) in the application or launched it 100 times, we can safely say they are retained.

The third aspect is the **scaling**. Sure, spending 80 hours gains you the rank of definitely retained but how about 40 hours, or 8 hours? 40 hours should bring you more than 50% retention, same for the other two parameters. The scaling of the parameters is illustrated in figure 3.

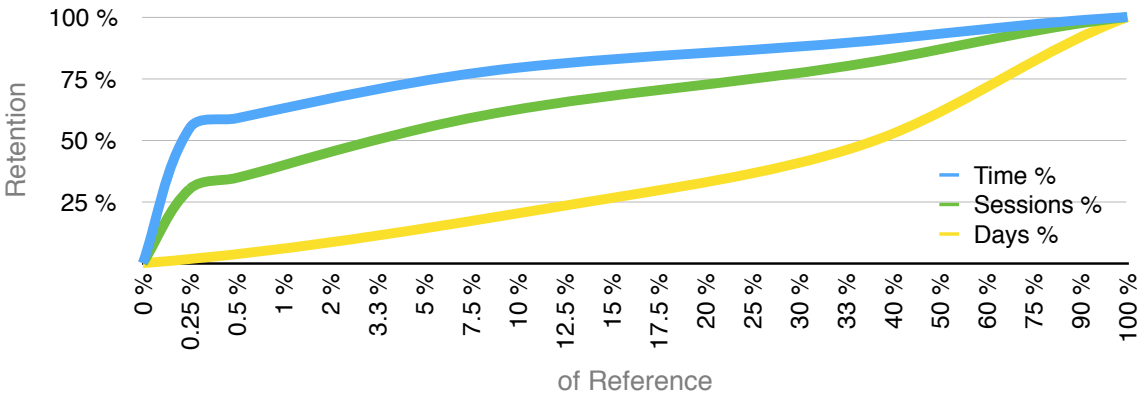


Figure 3: Parameters Scale

Notable value pairs include: 24 minutes (0.5% of reference) for 59% retention, 10 launches (10% of reference) for 65% retention and 15 days (40% of reference) for 62% retention.

The final formula for a parameter with is

$$\min\left(\frac{value}{reference}, 1\right)^{scale} \times importance$$

The retention probability for a user is the sum of the formula applied on each of the three parameter values

Distribution

The distribution of retention probability can be visualized in figure 4

We can observe the peak at the 35% - 40% retention probability, where 16.1% of the users fall. This is good news as churners edging 40% are more likely to be converted

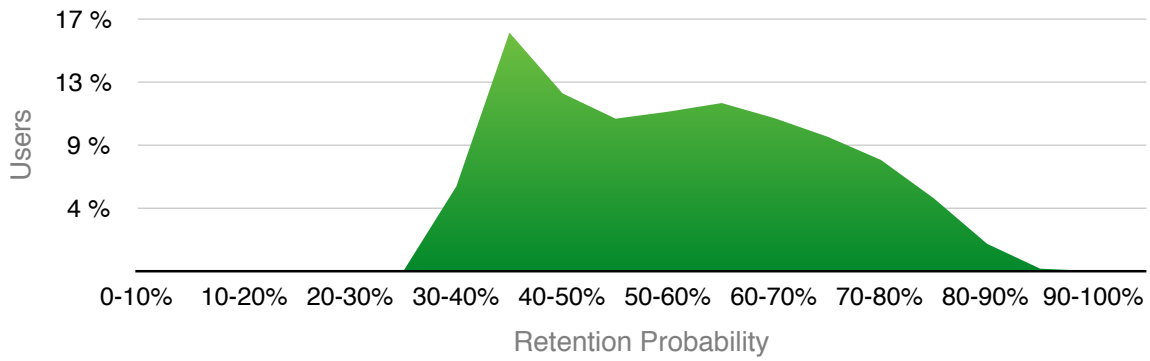


Figure 4: Continuous retention distribution

than users with very low retention probability. Other than the discussed peak, the distribution looks pretty **normally distributed**, telling us that the choice in defining it is close to reality.

Agreement with the Discrete Labelling

To further validate both the continuous and discrete labels, we take a look at how they agree. For this we need to set a **threshold** for retention. Any users with a continuous score higher than it is considered retained and anything under — churned. We try several values for the threshold: 50%, 60%, 66% and 70%. An agreement is achieved when the discrete score is 1 and the continuous one is over the threshold *or* the discrete score is 0 and the continuous one below the threshold.

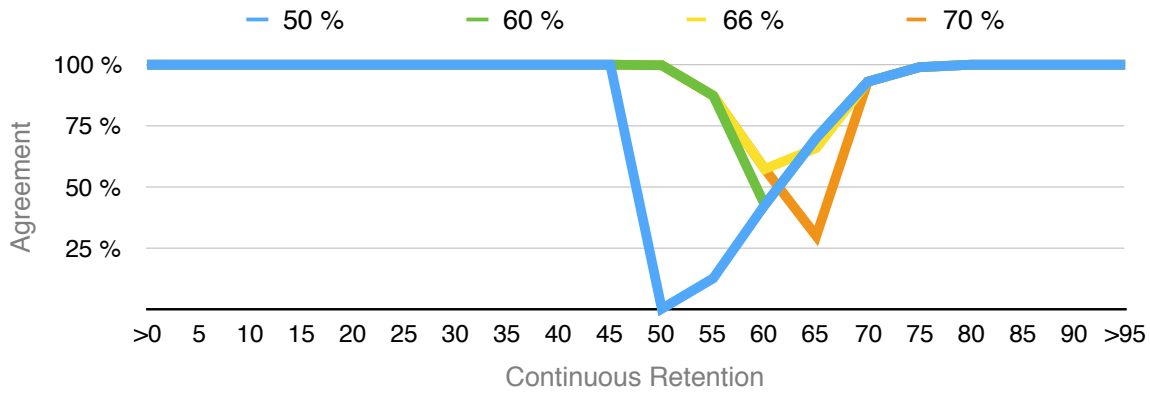


Figure 5: Continuous - Discrete Agreement

Figure 5 reveals that the best choice of threshold is 66%. For this value, scores agree most of the time, with a drop of under 50% in the area where continuous score reports 55% - 70% retention probability

5.1.3 *Evolution Throughout Periods*

It should be noted that measuring retention right after a big launch can easily turn into misleading conclusions. For example, the retention rate before launching the public preview was 34%, in the first month of the after the pre-release it dropped to 4% and finally stabilize around the 26% the months after. Both 34% and 4% do not tell the full story.

Before the preview, measurements were polluted by testers and did not reflect real use, artificially inflating the retention rate. Right after the public preview, a wave of new users has downloaded the application but many of them only curious about the new product and not planning to use it at all. This phenomenon could be attributed to an over-achieving marketing campaign which reached more users than strictly interested ones.

5.1.4 *Historical Features*

Historical features do not convey information about the usage pattern. They are very similar to the parameters used for labelling the data:

- time spent in each mode
- number of sessions

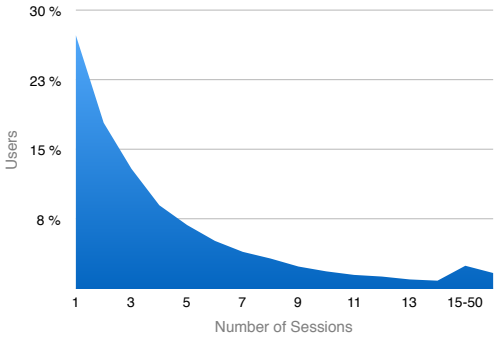
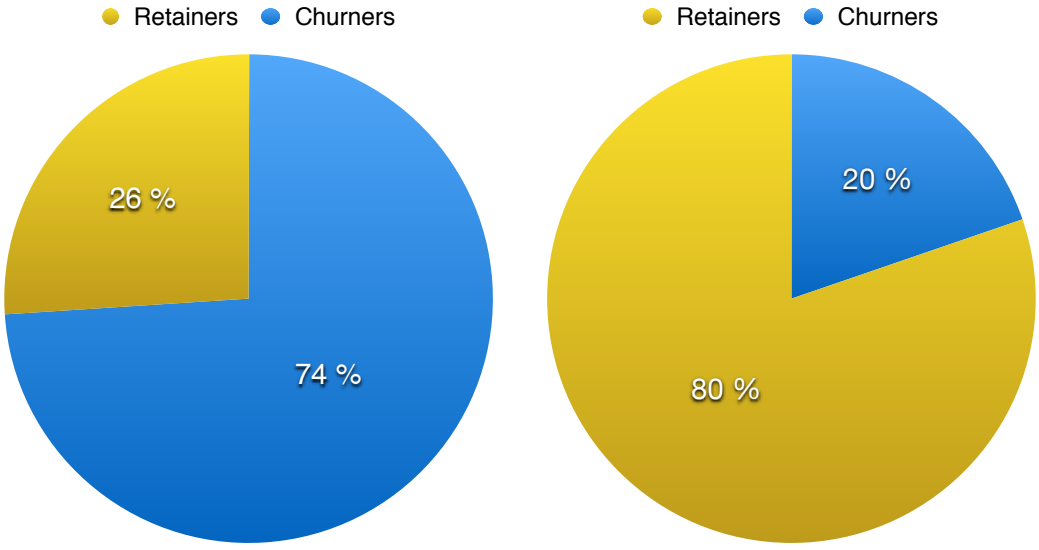


Figure 6: Sessions distribution

Pareto in Practice

We can see the Pareto Principle in glaring evidence: 26% of the users amount to 80% of the total time spent in the application.



(a) Users (b) Time Spent

Figure 7: Sizes

5.2 EVENT BASED FEATURES

After looking at historical data, we analyze even-
based features. These are actions that trigger a logging event every time they are performed. Split into categories, they are:

- **document** related actions:
  - open
  - save
  - new document
  - export
  - import
  - share
- **primitive object** draws:
  - rectangles
  - ellipses
  - lines
  - paths
  - text objects
- **advanced object** interactions:
  - repeat grids
  - prototyping wires
  - arboards created
  - artboards duplicated

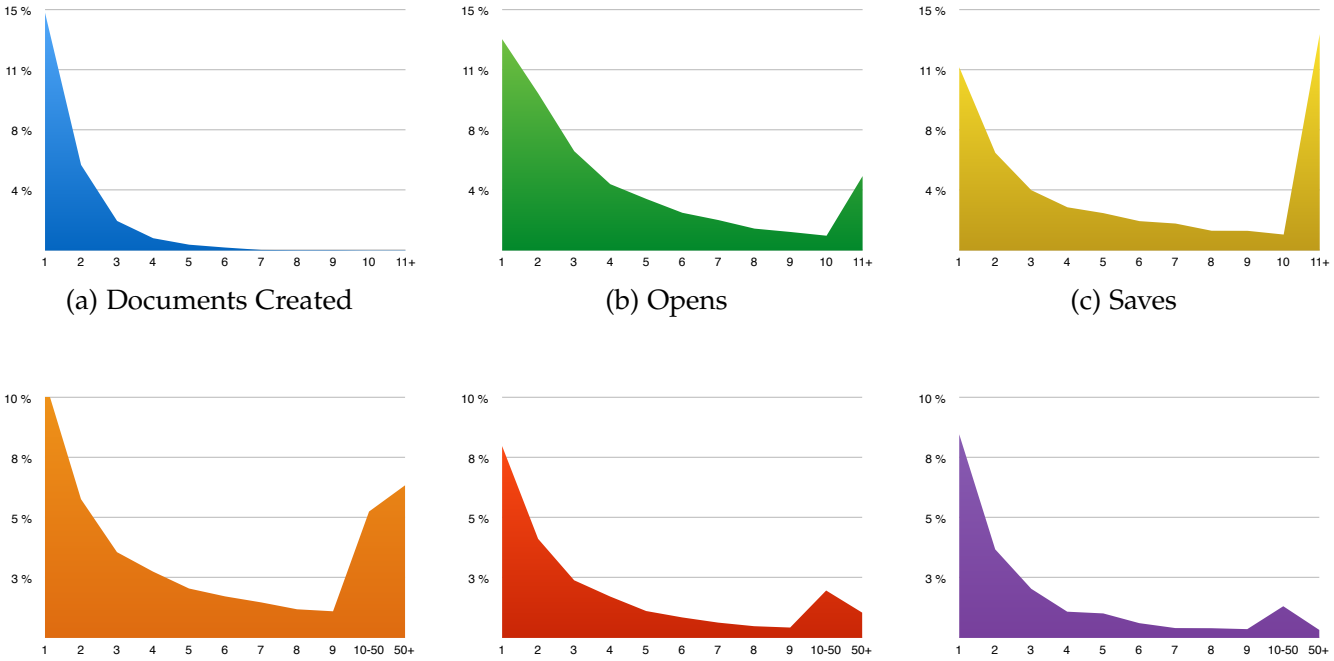
5.2.1 *Document*

To begin understanding customer behavior we start by looking at the histograms of document related actions.

As it can be seen in figure 8 these actions are not normally distributed, instead the histograms are crowded at low event counts and very scarce at higher numbers. About 15% of the users have created only one document, 6% two documents and under 1% have created more than 5. The number of documents opened has a flatter descent, but it for over 10 documents opened the percentage of users is still very low.

The number of times a user has saved is distributed in a different manner: there is still the fact that many users have saved few times but another spike occurs at 11+ save counts. This tells us that there are two types of behaviour: a user is either not active and not save or their activity involves a high number of saves. There are not many users that save a lot but their save count much higher compared to the others.

The number of imports, exports and shares can be seen as document operations as well, but in contrast to the primary ones (create, open, save) their user count peaks at lower values. A noteworthy fact is that the number of imports behaves similarly to the number of saves but there are more users doing a high number of imports



(a) Documents Created (b) Opens (c) Saves

(d) Imports (e) Exports (f) Shares

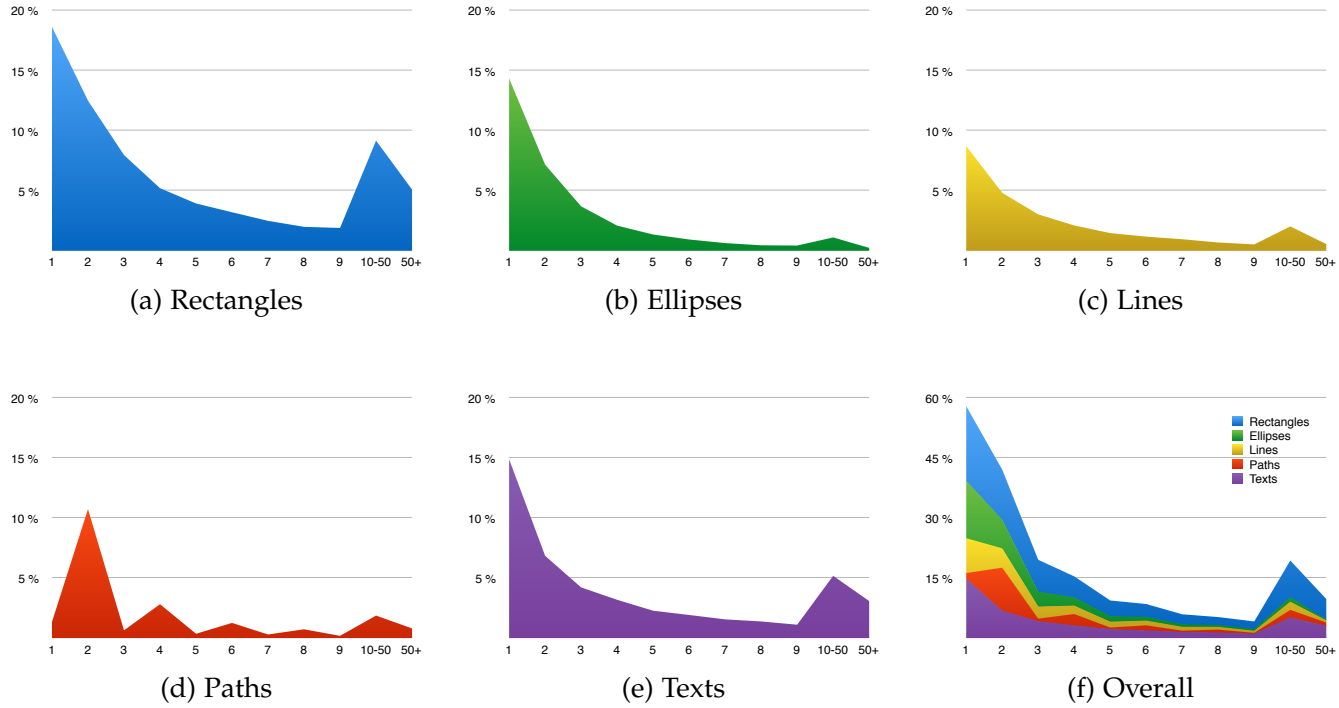
Figure 8: Document related distributions

(10+). The histograms for exports and shares looks similar to the one for documents created apart from lower value peaks.

The difference to 100% of users have completed zero actions and are not included in figure 8.

5.2.2 *Primitive Objects*

The next thing we look at is the distributions of primitive object draws. These actions are the ones we will focus on as primitive and advanced objects are product features that can be improved, as opposed to support-features such as exporting, importing or opening a document.



(a) Rectangles (b) Ellipses (c) Lines

(d) Paths (e) Texts (f) Overall

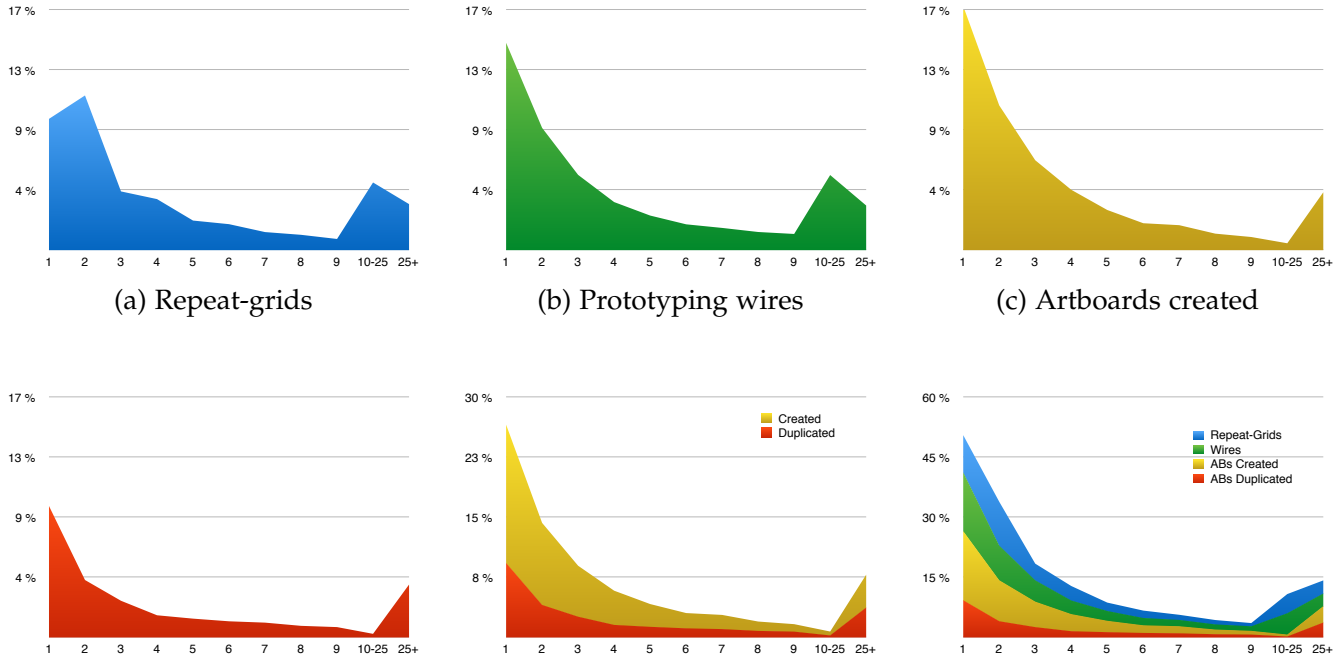
Figure 9: Primitive object draws

The behaviour seems to be similar to document related actions. The difference lies in the number of users drawing at least one object. It rises close to 20% of all users for rectangles drawn. Subfigure 9f tells us that that about half the users have not drawn any object at all.

The number of paths drawn is the lowest of all the primitive objects in spite the fact that it is a very prominent tool in for experienced designer.

5.2.3 *Advanced Objects*

Even more interesting to look is the advanced objects draws. As these are features unique to the product, they deserve more attention in order to differentiate the application from the competition.



(a) Repeat-grids (b) Prototyping wires (c) Artboards created

(d) Artboards duplicated (e) Artboards (f) Overall

Figure 10: Advanced object draws

In figure 10 we can see that more customers have used artboard functionality, even more than the number of documents created or rectangles drawn. Repeat-grid functionality is not as popular but there is a good percentage of customers making extensive use of them (10+).

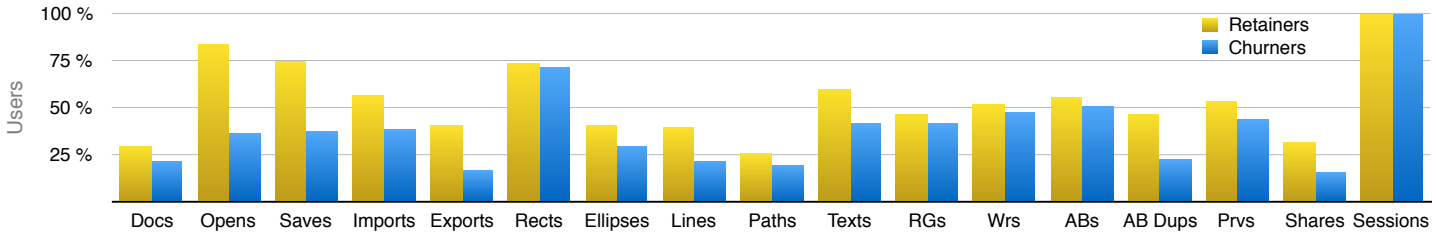
The overall histogram (figure 10f) tells a similar story to the one for primitive objects. About half of users have created only one advanced object, of any kind.

5.2.4 *Touchers, Significants, Extensivers*

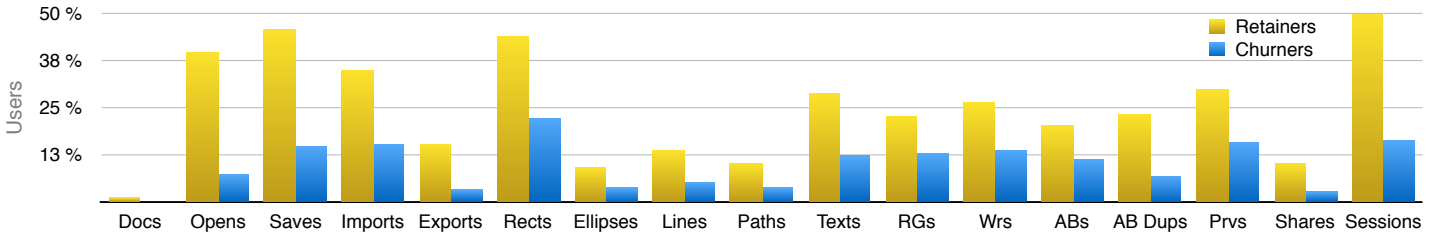
In the previous subsections we looked at the distributions of all users, retained or not. More useful for this paper's purpose is looking at usage patterns of users not as a whole but split into retainers and churners.

Since many users have not even touched a feature once, we start by counting the users that did. In figure , a *toucher* has used a feature at least once, a *significant* - at least five times and an *extensiver* used it for twenty-five or more times.

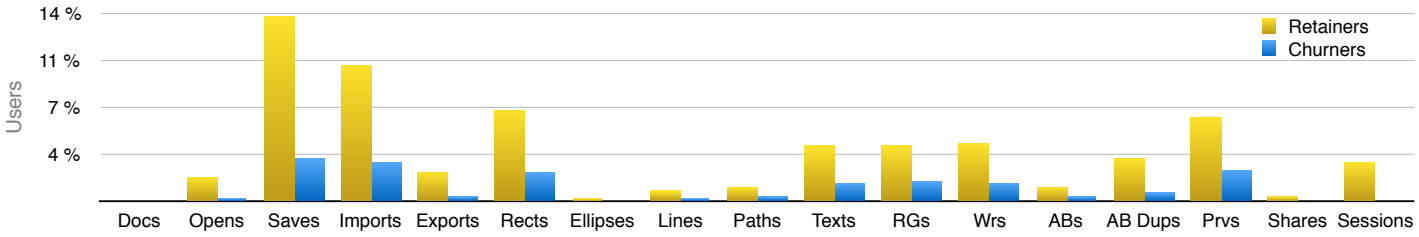
Figure 11a tells us right away that retainers are more active: a bigger percentage of them use a feature at least once, compared to the non-retainers. It also tells us retainers and non-retainers have launched the app at least once, but serves more as a confirmation that the computed statistics are correct rather than something we can draw conclusions upon.



(a) Touchers (1 uses)



(b) Significants (5 uses)



(c) Extensivers (25 uses)

Figure 11: Number of users for usage count thresholds

The fact that retainers use the product more is a given — the interesting part though is *how* much they use them. Increasing the threshold to five uses weeds out a higher percentage of retainers and with twenty-five, only a few remain.

To better understand the trend, we plotted the difference between the number retainers and touchers, for each threshold value. E.g., the violet series in figure 12 is the number of retained touchers from which we subtracted the number of churned touchers. This already gives us a basis to select **discriminant features** - it seems like the number of documents created, the number of documents opened, exports and shares are the most distinctive metrics (ignoring the obvious one - sessions).

As an extension to the previous, the chart in figure 13 allows us to more easily see the increase of retainers, but this time between threshold values. E.g. the dark blue series is the number of retained touchers from which we subtract the number of re-

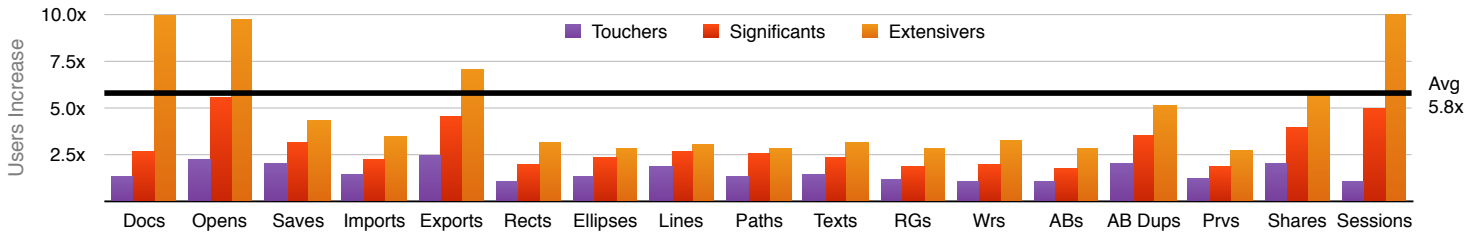


Figure 12: Retainers increase

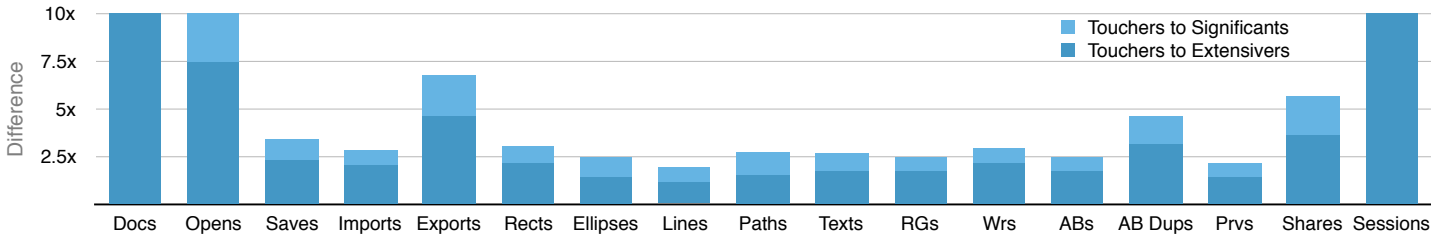


Figure 13: Retainers increase difference

tained extensivers. This chart reinforces the previous conclusion that by highlighting that mostly the same features are more distinctive for "deeper" retention.

5.2.5 *Average Uses*

Average uses present the data in a different light, rather than imposing some thresholds, we see how much customers use a feature.

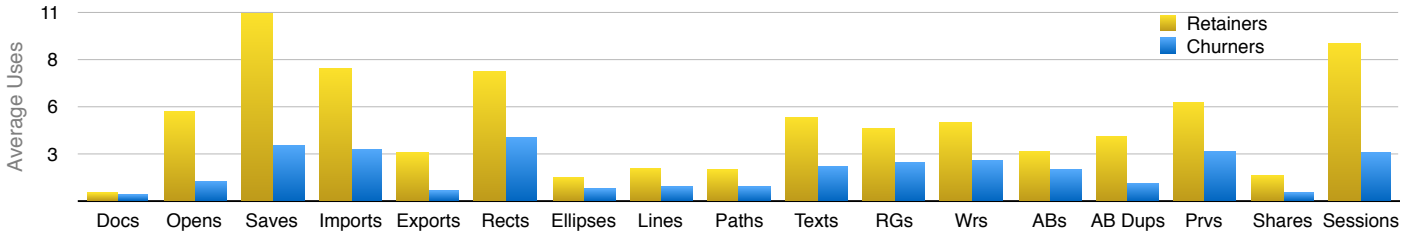


Figure 14: Average uses

Figure 14 provides an explanation for the phenomenon observed while analyzing histograms in the previous section. Retainers, although making up a smaller fraction of the total users, push feature usage higher with their more intensive habits. As opposed to the previous charts, here we see that the number of saves is the most used feature by retainers. For churners, drawing rectangles is a popular activity. Curiously, the number of documents created is the lowest frequency action done by both retainers and churners.

Previous assumptions are confirmed about the number of imports and documents opened. New ones emerge as well: the number of saves, rectangles drawn and preview actions can be a differentiating factor.

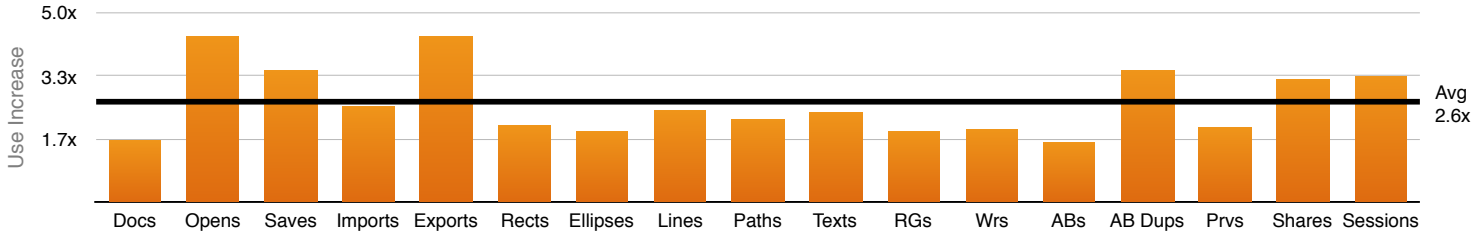


Figure 15: Average use increase

Just as before we are interested in seeing the relative difference of retainers — churners. Figure 15 shows the number of exports, artboard duplications and share actions, even though low in magnitude are some of the most discriminating.

5.2.6 *Violinplots*

The previous two sub sections have compared the number of users for each feature and the number of times they have been used, on average. Before that we have looked at the distributions visualized as histograms.

Violinplots attempt to combine all these aspects into one chart, for easier cross comparison. They are similar to boxplots in the sense that they show both the maximum usage count for each feature, the average (white dot in the middle) and the 25 and 75 quantiles (inner thick black lines). In addition, the width of each measurement represents how many users have performed the action 2, 4 or 8 times. They can be seen as histograms of retainers and churners placed side by side.

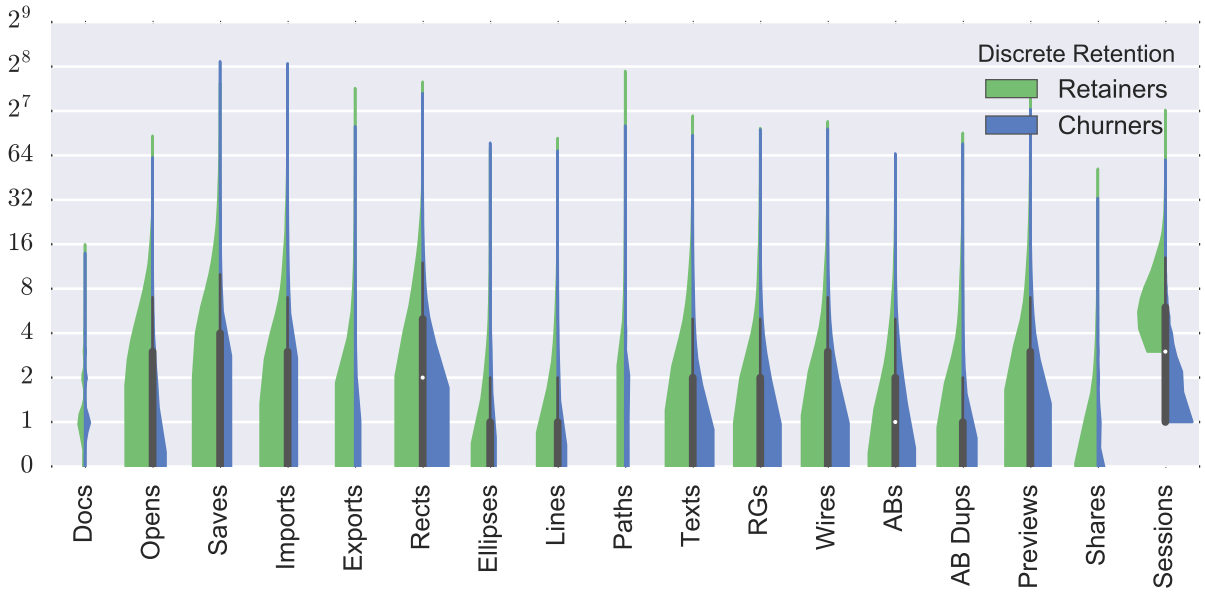
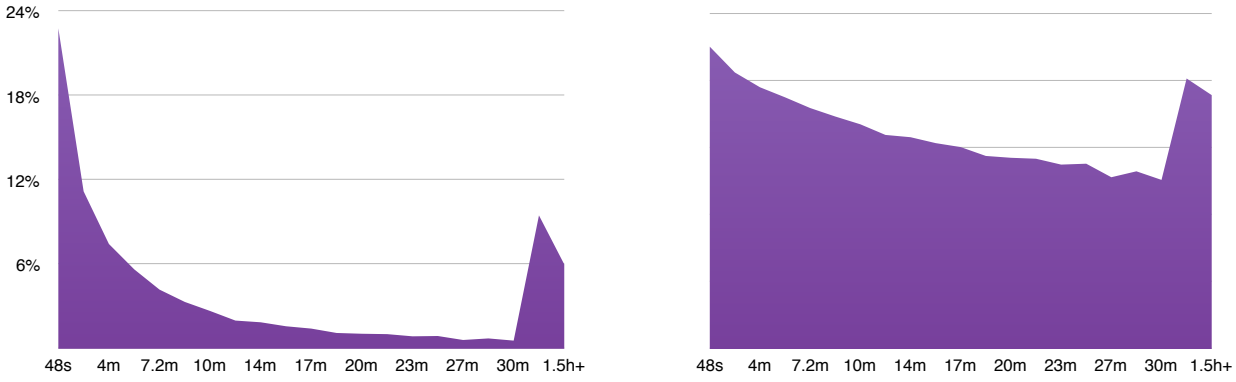


Figure 16: Events violinplots

Unfortunately the shape of the distributions is not ideal for such a chart but it is still useful for drawing conclusions. The number of documents opened, saves and imports is high for retainers and low for churners. On the right part of the chart, churner representation is also present, alongside with retainers. On this basis we can launch the hypothesis that the number of documents opened, imports, and shares are the most discriminatory features.

5.3 TIME BASED FEATURES

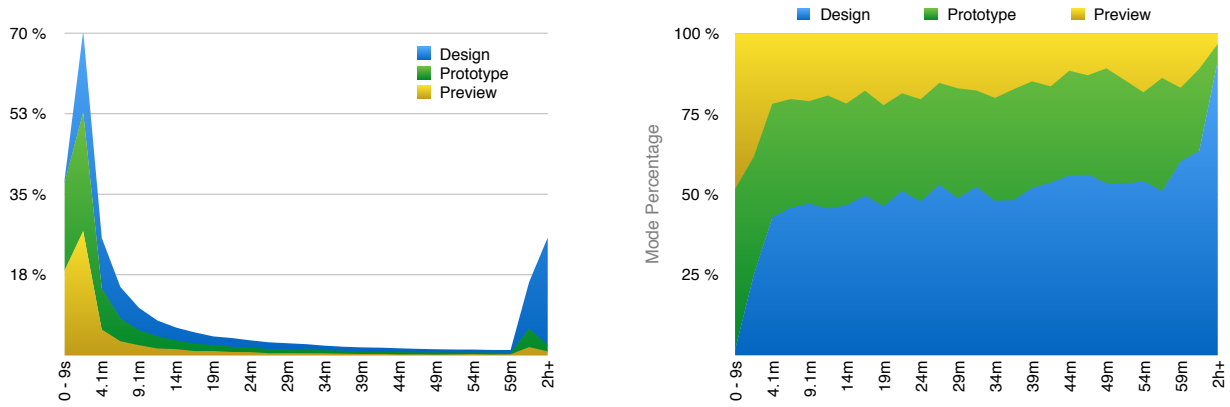
Having looked at event-based features, we now turn our attention to the time spent. This may be another indicator of how retainers.



(a) Linear (b) Logarithmic

Figure 17: Time spent in the first session

We know first impressions count, so let's see just how much do users spent the first time they open the app. Figure 17 shows that most users spend less than one minute in their first session. More useful for vizualizing the slope is a logarithmic scale, plotted in figure 17b. There is a steady decrease until the 30 minute mark, then extensive users cause a spike for the 1.5 hours+ time spent.



(a) Overall (b) Overall (percentage)

Figure 18: Time spent in each mode

The logarithmic slopes for time spent in each mode (design, prototype and preview) are similar. The linear scale (figure 18a) looks similar to the previously seen histograms: many users on low values. Subfigure 18b gives us the first insight into mode preference: at low amount of times, there are more users using the preview mode and at the highest amounts of time - design and prototype dominate.

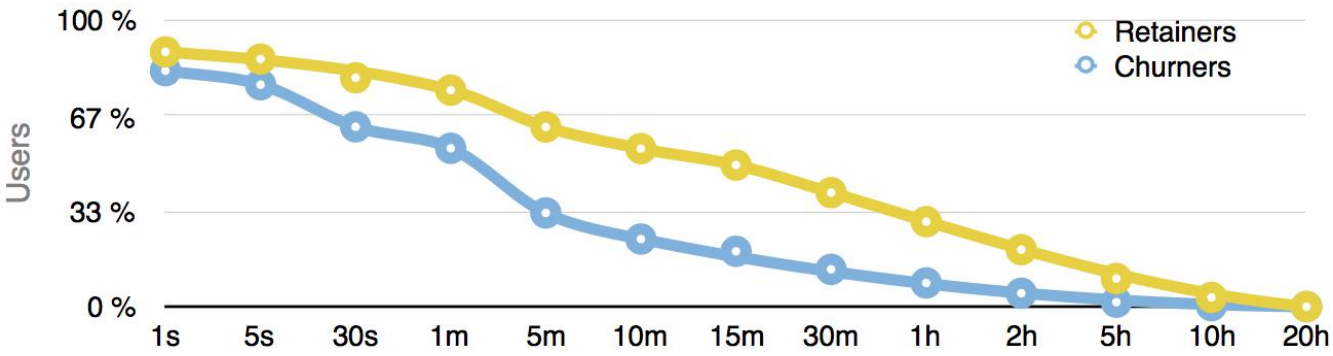
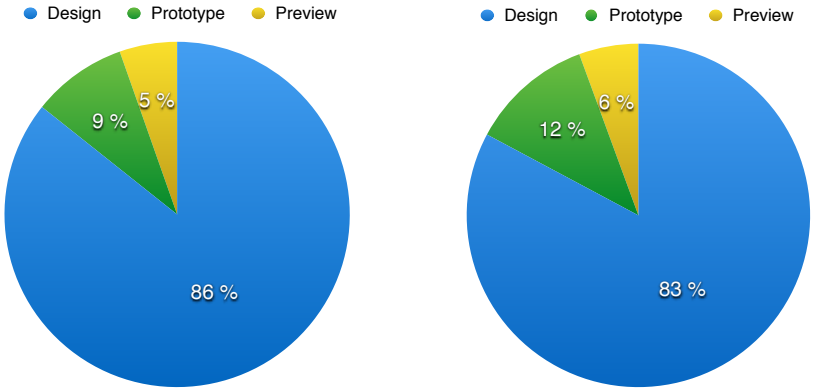


Figure 19: Users that spent at least

Applying the same reasoning as for event-based features, we plot the number of users that have spent at least 1 second, 5 seconds, 30 seconds etc on average in every mode. As expected, churners have a lower representation while retainers flatten out at a slower pace.



(a) Retainers (b) Churners

Figure 20: Mode preference

A useful insight is provided by much time, across all sessions, retainers and churners have spent in each mode. As seen before, design mode is the most prominent with prototype at a distant second. Interesting though is the difference between retainers and churners.

Even though churners spend less time in any mode, the relative time spent varies between classes. This is highlighted in figure 21 where we can see that retainers prefer design mode about 3% more while churners have a prototyping preference of about the same amount (3%).

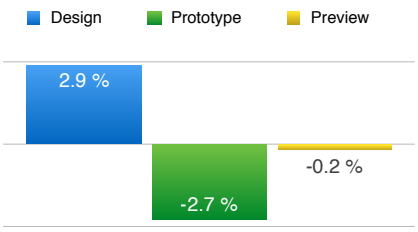


Figure 21: Retainers mode difference

Violinplots

In this case, violinplots are a better fit to interpret the data. The action frequency shows the average distance between any two actions has a distribution shape close to a gaussian. The retainers peak out at about 25 seconds and the churners at 12 seconds.

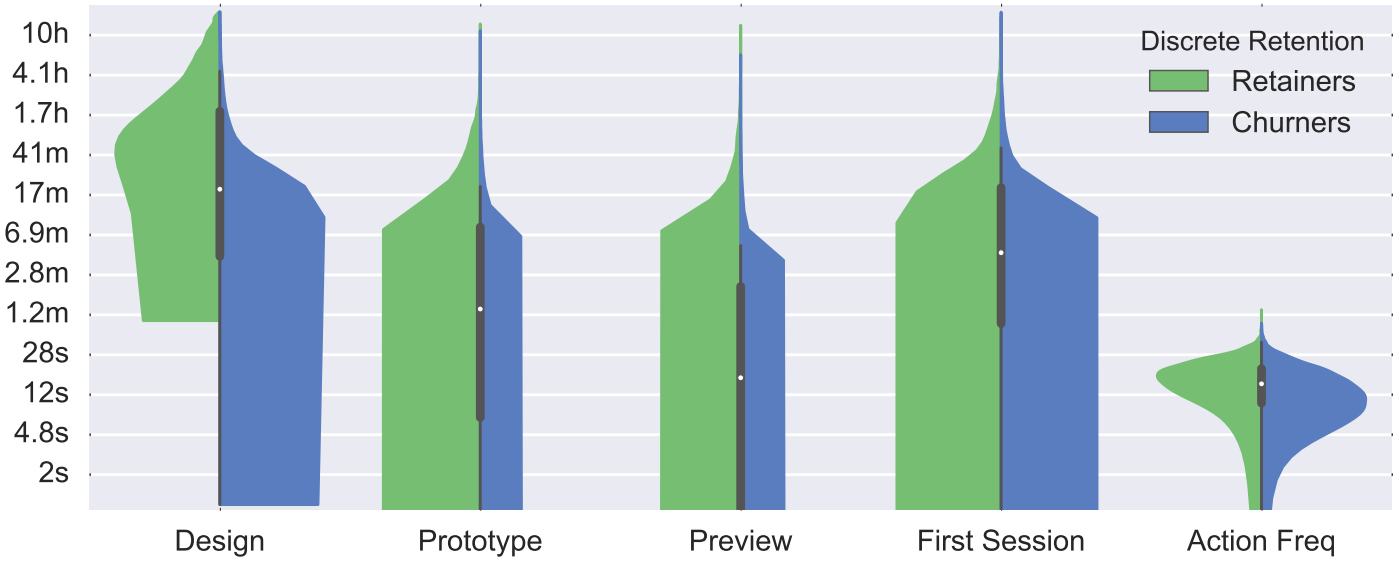


Figure 22: Time violinplots

From the three distribution pairs on the left (figure 22), we draw similar conclusions as before: design mode is used more by retainers and the other two modes are unpopular among churners

5.3.1 *Performance*

While performance of an application is a very important factor in customer onboarding, it is beyond the scope of this paper as it lies in the performance optimization category.

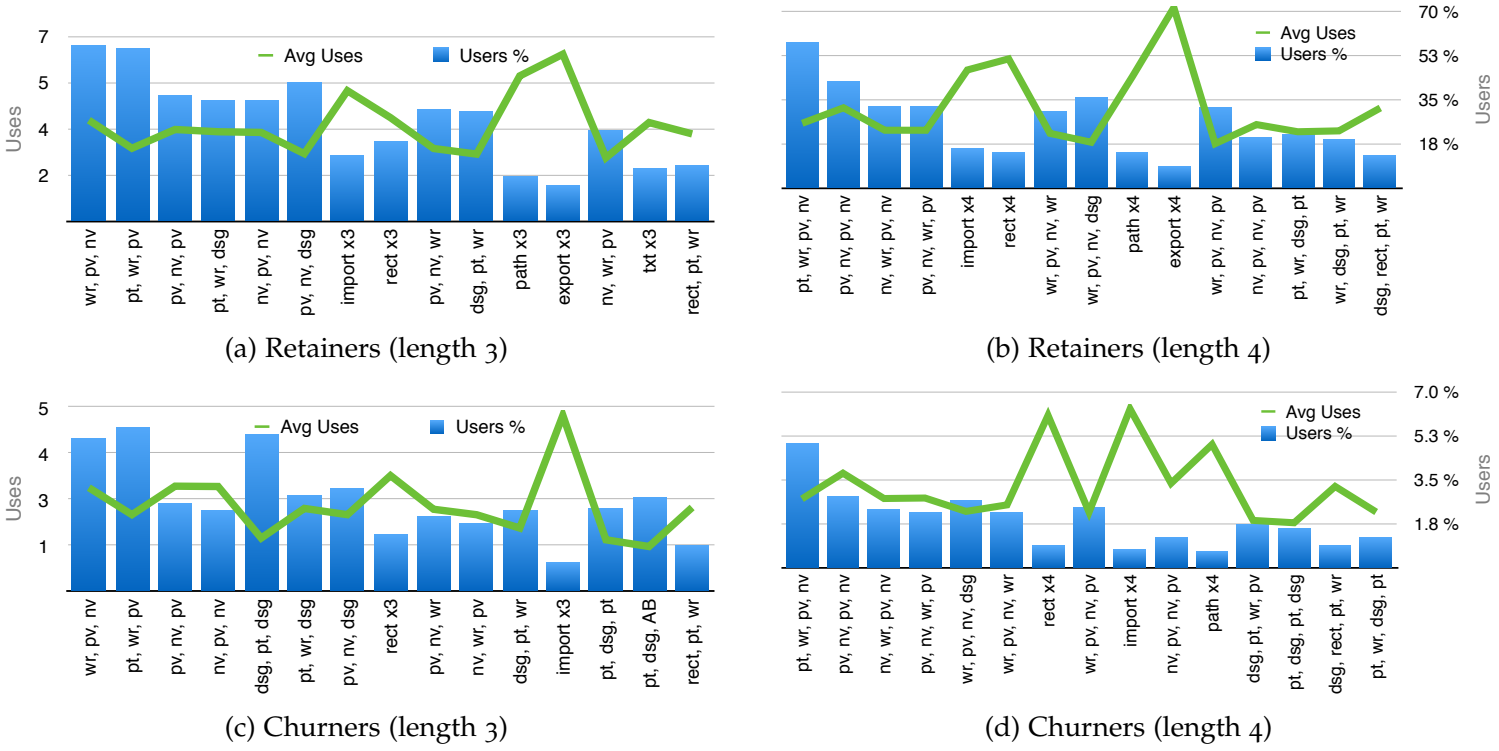
5.4 ACTION SEQUENCES

Until now we have looked at event-based and time-based features but have missed a very important aspect: the sequentiality of the actions. Drawing a rectangle then making a prototyping wire and finally turning on preview mode reveals a lot more about the user's behavior rather than just looking at isolated object counts.

We have picked sequences of length three and four as more than that would likely provide no additional benefit and less than that can hardly be described as a sequence. Actions to not be more than two minutes apart to be considered in the same sequence.

5.4.1 *Usage and Users*

We start by looking at the sequences which have been performed most times. Figure 23 depicts both the number of users and how many times a user has done the action sequence, on average. On the top row there are sequences common among retainers and the bottom row represents churners.



(a) Retainers (length 3)

(b) Retainers (length 4)

(c) Churners (length 3)

(d) Churners (length 4)

Figure 23: Most performed sequences

Immediately obvious is the fact that adjusting a prototyping wire, switching to preview mode and navigating is the most common sequence for both retainers and churners. Creating multiple rectangles, paths or text objects is also among top sequences for retainers. Churners seem to switch between modes more often. Another interesting fact is that among both retainers and churners, there are few users that make extensive use of importing and exporting.

5.4.2 *Common Sequences*

After looking at retainers and churners behavior separately it's time to compare the two classes.

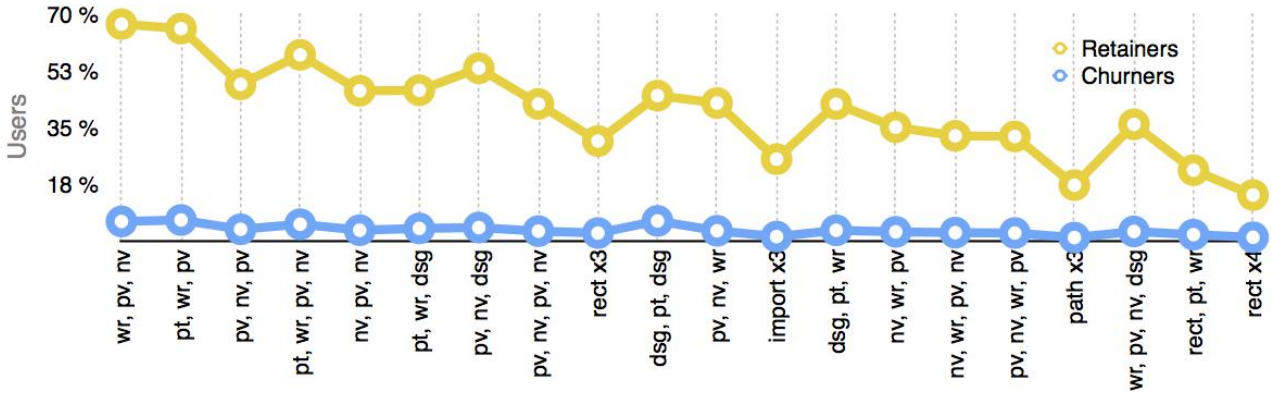


Figure 24: Number of users for common sequences

Figure 24 shows that among top action sequences, there are always more retainers performing them.

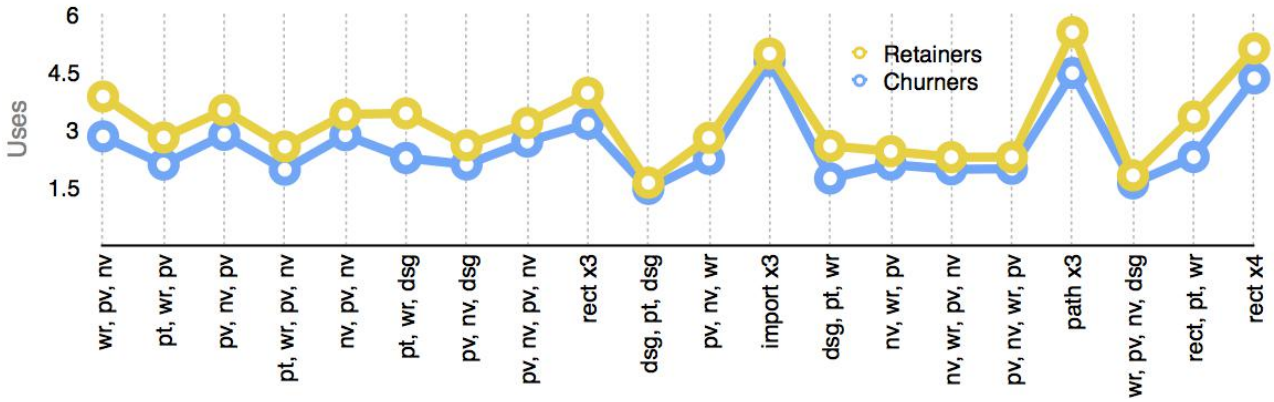


Figure 25: Common sequences usage count

Figure 25 indicates that the usage count, even though higher for retainers, is close between the classes.

A real conclusion can be drawn by consulting figure 26 where the discriminatory patterns emerge by comparing the difference in the number of users: creating a prototyping wire and firing up preview mode is performed about two thirds more by retainers.

Even more relevant is the comparison of usage count. Figure 27 points out that even though most sequences are performed more by retainers, for of multiple art-board creations and one rectangle followed by two text objects, it is not the case. On the other side, retainers more often create two rectangles and then a text object.

CONCLUSION

We propose the following subsets of features as the most impactful ones, according to matters discussed in this chapter.

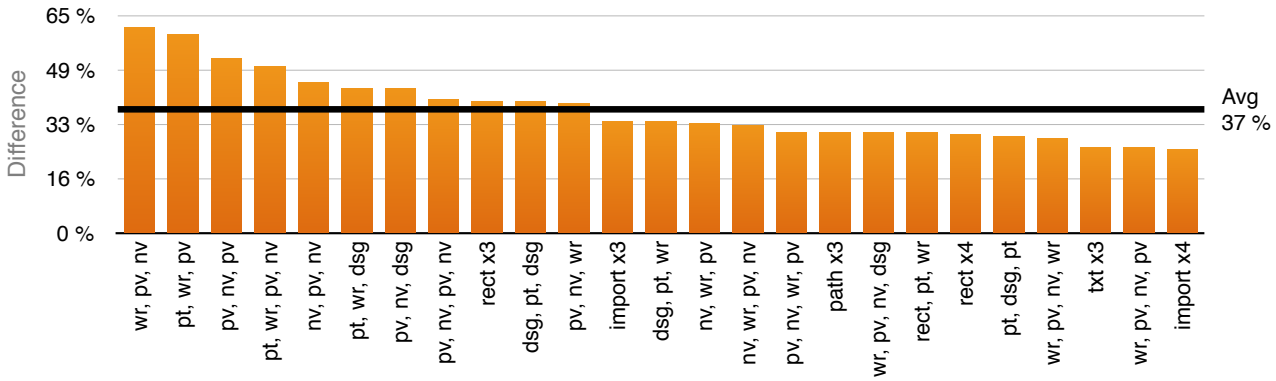


Figure 26: Number of retainers difference

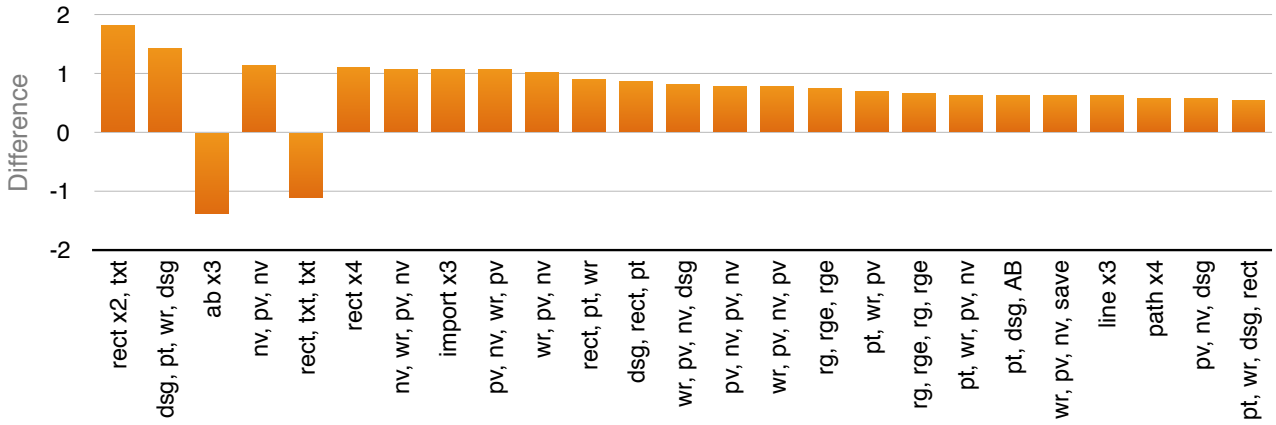


Figure 27: Usage count difference

Events & preferences

Based on results from touchers, significants and extensivers counts difference:

- documents created
- ellipses drawn
- artboards created

Based on usage counts:

- documents opened
- documents saved
- exports
- artboards duplicated

Action Sequences

Based on users difference:

- wire › preview mode › navigate preview
- prototype mode › wire › preview mode

- preview mode › navigate preview › preview mode
- prototype › wire › preview › navigate preview

Based on average usage:

- rectangle › rectangle › text
- design mode › prototype mode › wire › design mode
- create artboard › create artboard › create artboard
- navigate preview › preview mode › navigate preview
- rectangle › text › text

These feature sets will be used in the machine learning chapter.

6

ALGORITHMS

This chapter details the algorithms developed in order to bring the task at hand to completion. We will start with the model optimization pipeline which culminates with the usage of the combining classifier. After its execution, valuable feature selection cues are outputted – discussed in the third section.

6.1 MODEL OPTIMIZATION PIPELINE

This pipeline helps automate the process of choosing a learning model and tuning its parameters for a specific dataset. It aims to reduce the necessity of human input as much as possible, without sacrificing performance gains. The approach is meant for users new to the field which don't have much experience in picking a model or understand each parameter's impact.

This is not to say that expert input is to be disregarded. We merely trade experience for processing time. The extreme opposite is not true either – no matter how experienced, one can never pick the best combination of model, parameters and features for any given learning task. Some level of trial and error approach will always be required.

We start this section with a brief, intuitive explanation of a few key concepts needed to understand the reason behind pipeline's choices. Model performance is mainly influenced by three aspects:

- quality and quantity of samples given
- model parameters
- features for each sample

The **first aspect** is not easily tuned, but fortunately, for our task a wealth of data was available and for training. The quality of a dataset translates into how relevant and diverse the examples are for the entire population. In our case that means it should include users that are very likely to return, users that are not likely at all to come back and everything in-between. We need to see what is the retention rate of a user of a user that has drawn many rectangles but no circles, a user that has spent a lot of time in prototype and created many artboards, a user that exported many assets but spent very little time in their first session and so on. Basically we need many examples varying in all dimensions that can be indicative for an unseen case which the classifier will be asked to predict.

Having the task of extraction and processing handled, we deal with the **second aspect**: the model's parameters. Pinpointing the exact combination of parameters that will produce the best performance for a given problem is a very difficult task. One would need have a great understanding of the data as well as having deep knowledge of each learning algorithm. This proves to be a very complex task, especially when using a model highly sensitive to two, three or more parameters. In a perfect, theoretical setting, we would try every possible parameter combination and pick the one performing the best. Clearly, this is impossible to achieve in practice.

An easy mistake to make would be to try the following technique. Start at some value. Increase or decrease it and see which direction improves performance. Keep going in that direction until performance no longer improves. Declare the value stopped on as the best choice for this parameter. Using this value for the first parameter, apply the same procedure for the next parameter. Continue until having determined the best value for each parameter. This technique is based on two major fallacies. The first one is that searching should stop when reaching a peak, also known as a local optimum, instead of overcoming it and keep searching for the global optimum. Even after solving this issue, the second fallacy stills stands: not taking into account the correlation between parameters. The model does not use only one parameter at a time - it needs a value for each of them. Some values, found as the best for their respective parameters produce some performance. Different, even though they are not the 'best' ones, their combination provides a better performance.

This leads us to a more realistic approach: for each parameter, pick a range of values instead of a single one and try every combination then chose the one performing the best. This still can prove unfeasible as running an exhaustive search on a cartesian product quickly grows out of proportions. Say a model has a training time of s seconds, we wish to vary p parameters, each of them in a range of v values. The time required for checking every combination is $s \times p \times v$. This scales very poorly when many parameters are varied, parameter ranges are particularly wide or the training time of the model is especially high. The described technique is called *grid search* and is useful when plenty of time is available or the model's training speed is very good.

A less computationally-expensive technique randomized searching. Pick a number of combinations to try (obviously, less than every possible combination) and pick that many combinations, sampling from the given ranges (or distributions if continuous) for each parameter. This approach comes at the cost of performance: the lower the number of tries, the lower the chance to find that best combination.

After finding the more-or-less best combination of parameters, we tackle the **third aspect**: the features given to the algorithm to learn from, for each sample. More is not always better and after a certain threshold, the bigger the number of features, the least "profitable" (in terms of performance) it is to add another one. This is called the *curse of dimensionality*

After a certain point not only does interpreting become difficult for a human, but the excessive amount of features can lead to poor training performance or generalization power. With too many parameters, a new example, with distant values, will not have enough precursors and the model will not know how to interpret it.

6.1.1 *Pre-requisites*

Before launcing the pipeline we need to fulfill a couple of pre-requisites. After having extracted the features and scaled them, human input is required for two more matters: model selection and feature subset proposal.

Model Selection

Decide what to try:

1. pick applicable learning **methods**
2. for each method, select a number of **algorithms**

3. for each algorithm, consider most impactful **hyper-parameters**
4. for each hyper-parameter, pick a sensible **range of values**[1]

Feature Selection

Determine feature subsets likely to perform well:

- features that seem to discriminate between classes the best, based on **data analysis**
- the most important features, according to **statistical-based methods**: variance, MI, other scores [2]

The last thing is deciding how long the whole fitting process should take.

6.1.2 *Steps*

For each learning algorithm:

1. run exhaustive search on the hyper-parameter grid using a small sample of the dataset. **Restrict** hyper-parameter search iterations and RFE step size based on how long fitting took on the sample
2. fit **hyper-parameters** by running a random-search on value ranges (stop after the restricted number of iterations)
3. add the fitted models to **meta-classifiers**[3] hyper-parameter ranges fit them as well
4. fit **feature subsets** by training on the whole feature set, and each proposed subset; run RFE[4] (with restricted step size)

The resulting learning model is the version that performs the best in the given conditions.

After running the algorithm on stand-alone models, use their best versions as parameters for the **combining classifier**. The resulting model will hopefully perform even better than any previously trained ones. The details of the combining classifier are described in the next section.

6.2 COMBINING CLASSIFIER

The idea does not belong to me, similar techniques have been used in machine learning competitions such ones organized by Kaggle. The implementation however, is my own.

Since we have all these sophisticated learning algorithms which can be fitted on wildly different tasks, why don't we treat ourselves to a bit of meta-learning? Have

1 or distribution in continuous cases
2 Relief, Chi-squared, Fisher, T-score, ANOVA, Gini
3 bagging, boosting etc
4 if model supports it

the machine learn *how* to learn as well. Having all these learning methods and algorithms (see 3.2), it would be a shame to only use the output of one of them and discard the rest.

What we want is a way to aggregate their results such that the end result is better than any of the individual algorithms. This sounds like a classic machine learning task. Only instead of taking as input user profiles, we pass the result of other classifiers and let the model figure out to best combine them.

Now all there is selecting which algorithms will take the role of "deciders" and which algorithm will be the "combiner". The first part is already covered, this is what we have been doing so far: tuning learning algorithms on the classification task. The difficulty lies in choosing *which ones* to pass to the combiner. As discussed earlier, an exhaustive search is not feasible: our new classifier will have two hyper-parameters to tune: the combiner and the list of deciders. Just as before, trying every possible hyper-parameter combination quickly turns unachievable. In order to reduce the number of combinations, we pick the best performing n classifiers (e.g., best 3).

Selecting deciders in this manner could lead to lack of diversity among models. E.g., for $n = 4$ and best performing family being the decision trees, top models could look like: Decision Tree, Random Forest, Extra Trees and AdaBoost with a tree-based base-estimator. Because they are all related, they will all perform well on one type of example and bad on some other. The problem is that they will *all* perform bad on the same type of example and be unable to compensate for each other's faults.

A better way to pick deciders would be to find the best algorithm in each family and pick the best n such families. This way we allow one model's strengths to compensate for another's weaknesses and avoid the problem of family over-representation in the top performers.

The task of picking the combiner model is conceptually different from picking a decider. While deciders have to perform well on the learning problem at hand, the combiner operates on meta-information – the labels predicted by the other deciders. As the number of deciders should not be too large (for matters thoroughly discussed until now), the combiner aggregating them should not be overly complex. A fairly shallow NN, a linear SVM or a relatively small random forest should suffice.

6.3 LEARNING BASED FEATURE SELECTION

After finishing the model optimization process, information valuable in selecting features and understanding their relationships can be easily extracted. We describe some of them, in no particular order.

The number of **restricted training iterations** and model fitting time is a good indicator of required training time for each estimator. It is useful when seeking a classifier with low training time either because the environment for it requires that or if scalability is a concern.

Evolution of performance based on different values for hyper-parameters is useful in identifying trends and perhaps improve subsequent pipeline runs. To an expert eye, sensitivity to low or high values for a certain parameter can reveal useful information about the shape of the data.

The **learning curves** both on the dataset and on training set are a widely used method of determining early stopping time. They can also be employed in varying hyper-parameter values, depending on the axis they are plotted on – either size of the training set or a particular parameter's value.

For models supporting feature importance, aggregated **scores** and **rankings** can be immediately computed for each feature. The advantage of having multiple learning algorithms is the option of trusting the best performing one.

In a sense, this is the essence of the whole thesis – find a way to see which feature is the most important. If an entity such as a learning algorithm can accurately predict customer retention, it must understand the underlying relationships among the data, even if the human analyst does not.

7

APPLICATION AND RESULTS

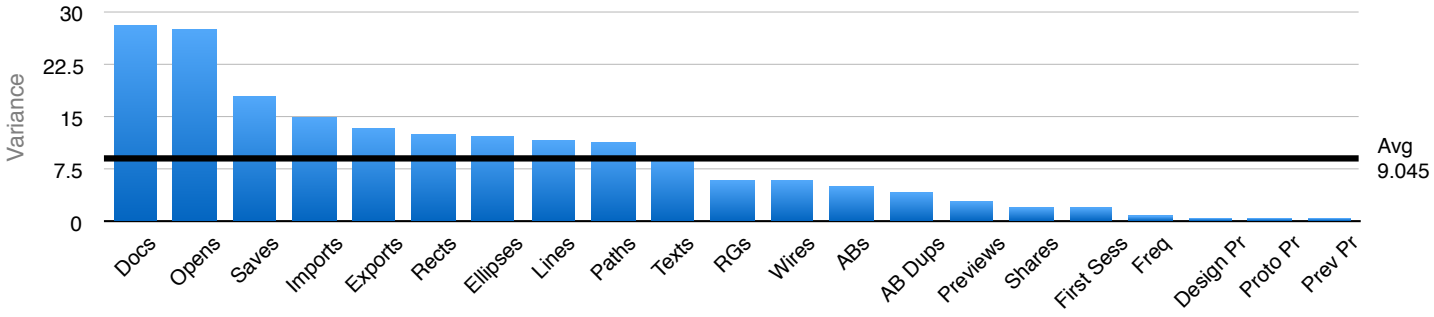
This chapter explores how we applied both techniques researched from literature as well as the custom model optimizing pipeline described in the previous chapter. We start with statistical based feature selection methods and then move to machine learning.

7.1 STATISTICAL BASED METHODS

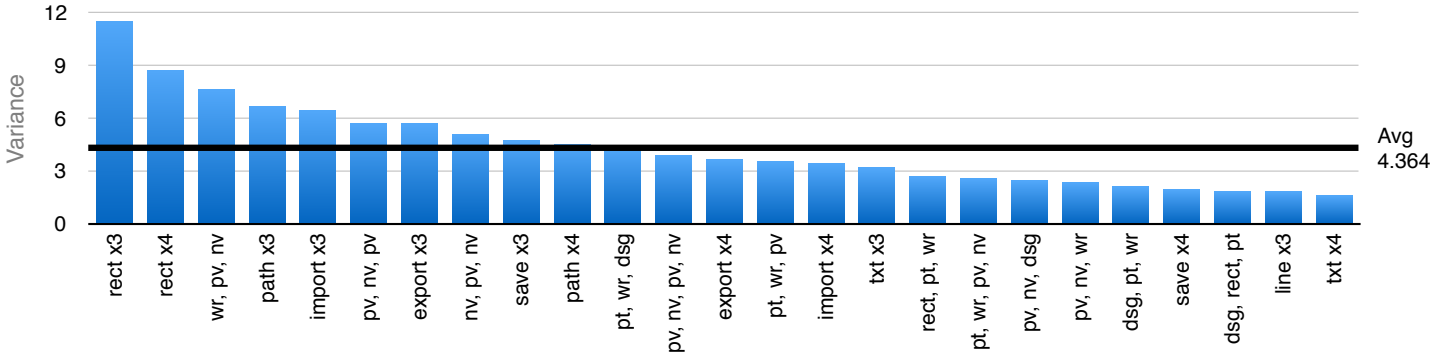
After anaylzing the data, seeking to manually find the most discriminatory features, it is time to use tried-and-true statistical methods for automating this task.

7.1.1 *Variance*

The first technique we employ is a very simple one: discarding low variance features. This is based on the idea that if a feature has almost the same value for every sample (e.g., nearly all users have used the share three times), it is not useful in the classification task. If the values are very close across all examples, it can be ignored without losing much information.



(a) Events & preferences



(b) Sequence

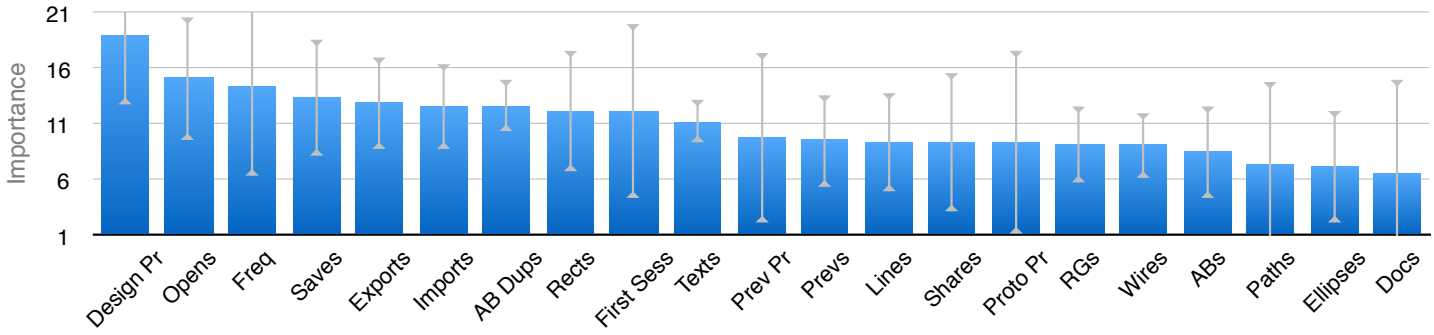
Figure 28: Feature variance

Figure 28a reveals the fact document operations: creation, open and save have a high variance. It also confirms the information concluded through data analysis: the mode preference (percentage of total time) is almost the same across all users and the same is true for the action frequency and the first session duration.

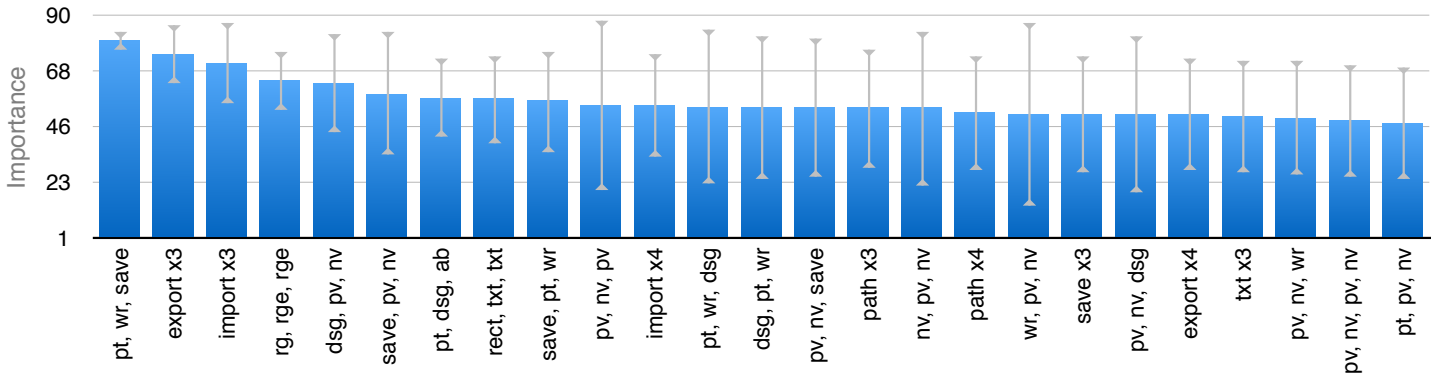
Among sequences, the ones varying the most are rectangle and path draws, imports and exports and using the preview mode. In figure 29 the mean of all algorithms is represented, with error bars marking the standard deviation. The smaller the error bar, the higher the agreement among algorithms.

7.1.2 *Mutual Information*

Moving forward, we use techniques from the field of information theory. Instead of looking at each variant of mutual information ranking algorithm, we aggregate their results and take a single conclusion.



(a) Events & preferences



(b) Sequences

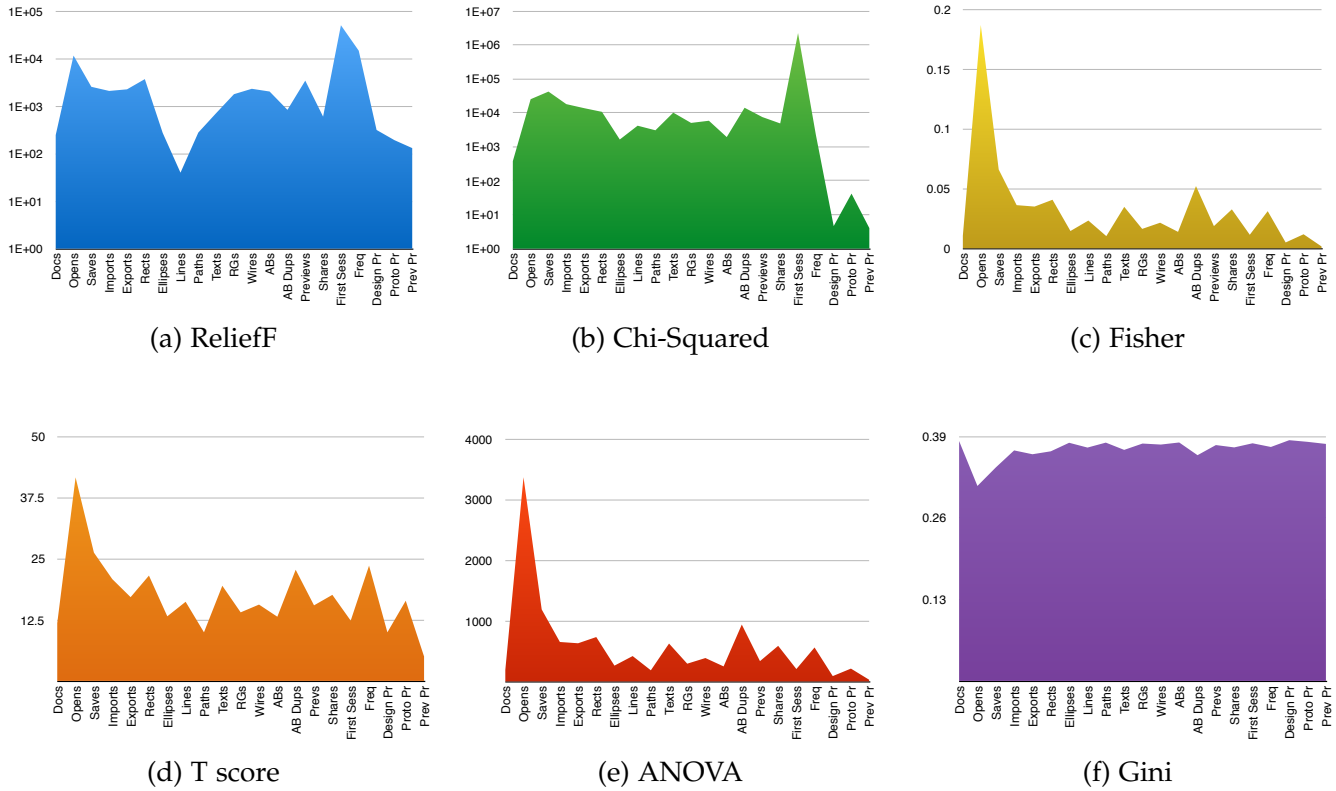
Figure 29: MI-based feature importance

As opposed to variance-based results, MI affirms that design preference and frequency of actions are relevant for classification (figure 29a). The number of documents opened, the number of saves, exports and imports are still scoring high.

As for the action sequences, apart from repeated exports and imports, the sequence of switching to prototype, making a wire and saving is considered by all algorithms as the most discriminatory. Contrasting with previous techniques of selection, by employing MI, sequences containing "save" have emerged to top position.

7.1.3 *Statistical Scores*

Another popular way of selecting features is through the help of statistical scores. These are not dedicated feature selection techniques but they fit this objective very well. In figure 30, the output six scoring algorithms is plotted. For each of them, except Gini (30f), the higher the score, the more important it is deemed by the algorithm. More detail about each is provided in the background chapter.



(a) ReliefF (b) Chi-Squared (c) Fisher

(d) T score (e) ANOVA (f) Gini

Figure 30: Events & preferences

The one thing on which all methods agree is that the number of documents opened is a strong indicator of retention. The ReliefF and Chi-Squared tests place the duration of the first session as the best criteria for prediction, contradicting the variance-based findings. Another aspect all algorithms agree on is that the mode preference is likely irrelevant. This is consistent with previous findings.

When comparing events & preferences, it was easy to plot charts for every statistical score where each feature is evaluated. Action sequences, on the other hand, greatly outnumber them and would make looking at every score value for each action sequence highly impractical.

Figure 31 shows an aggregation of the statistical scores for top sequences. The length of each bar is the sum of the scores and the error bars corresponds to the standard deviation among them. The most important sequence is undisputedly saving, switching to prototype mode and creating a wire. The second and third are variations of it, although not all algorithms agree. Apart from mixes of prototyping wires and navigating preview, other notable appearances among most important sequences are creating a rectangle followed by two text objects exporting or saving multiple times.

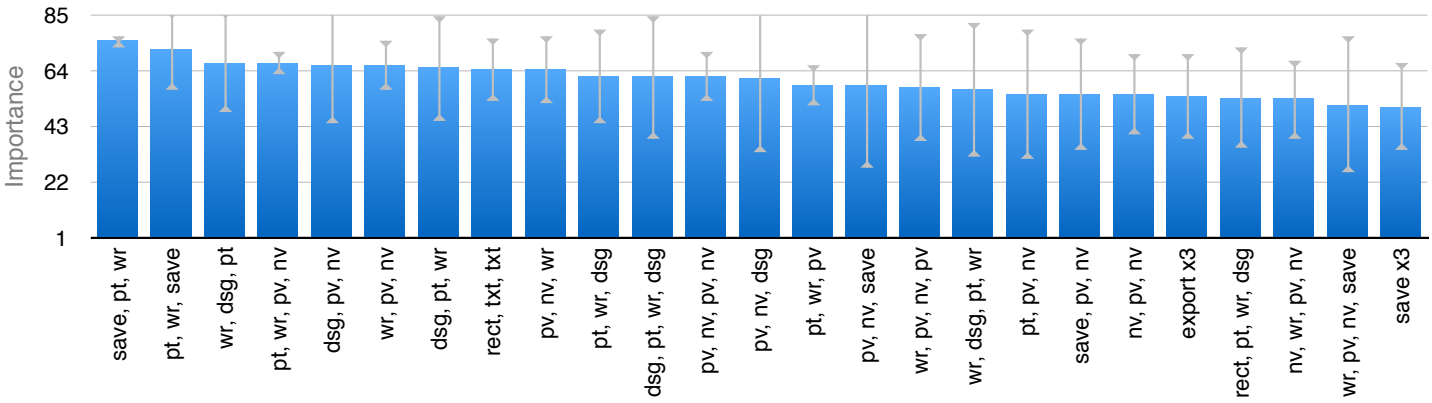


Figure 31: Sequences statistical scores

CONCLUSION

Just like we did for data analysis results, based on statistical methods results, we propose feature subsets for learning algorithms.

Events & Preference

Based on high variance:

- artboards duplicated
- exports
- paths drawn

Based on high average MI importance, Fisher and Chi-Squared:

- design preference
- documents opened
- average action frequency
- documents saved
- first session duration

Action Sequences

Based on high variance:

- rectangle drawn x3
- wire › preview mode › navigate preview

Based on high MI importance:

- prototype mode › wire › save
- export ×3
- import ×3

Based on high statistical score:

- design mode › preview mode › navigate preview
- preview mode › navigate preview › save
- prototype mode › preview mode › navigate preview
- preview mode › navigate preview › design mode
- prototype mode › wire › design mode
- prototype mode › design mode › create artboard

7.2 MACHINE LEARNING BASED

Before diving into results, a few words about applying the methods on the current problem. Feature extraction was detailed in the data analysis chapter - event features and time features along with action sequences. Apart from changing time *amount* to mode *preference*, they remain largely unchanged when feeding them into the learning algorithms.

Feature scaling is another matter. We apply a robust scaling method which mitigates the impact of outliers in the data. It brings each feature into a $(-1, 1)$ range. Models not supporting negative inputs are trained using a $(0, 1)$ scaling method.

Forbidden Features

We need to keep in mind the main objective of this paper. Our goal is not to design the best learning algorithm that makes use of any sources of information available to predict the retention of a new user. Instead we want to find out what are the features that impact performance the most. In this endeavor, machine learning algorithms prove to be a very good tool because a model that accurately predicts retention will inherently have a good "understanding" of the relationships in the data.

This is why we do not present the algorithms with historical data such as the time spent in each mode or the number of total launches. Allowing access to this information would lead to models predicting the function by which users are labeled (discussed in 5.1.1 and 5.1.2) and not focus on the product features. It is obvious that in doing so we drastically reduce classifier performance but as stated earlier, model accuracy should be seen as a means and not the end goal of this paper.

Strong Samples

An approach we attempted was using only strong examples to train an estimator but still test on the whole dataset. Strong examples are made up by users whose continuous retention score is either very high or very low. For these users, we are certain they are either retained or churned. The idea behind this is to provide the model with non-ambiguous examples so it is able to more easily see the data patterns. This was made possible by having both a continuous and a discrete label, something very uncommon in classification tasks.

We tried multiple values for both the low and the high boundary. In figure 32 the y-axis is the high threshold: 65 meaning only users that had 65% or more in the retention probability column were used. Similarly, the x-axis refers to the lower bound. 35 means profiles with a retention score of 35% or less were trained on. Highest performance is observed on the rightmost column and bottom row, corresponding precisely to where one of the thresholds was effectively unused, 50%. The best performance in the lower right corner, when both thresholds are 50%, meaning using the whole dataset.

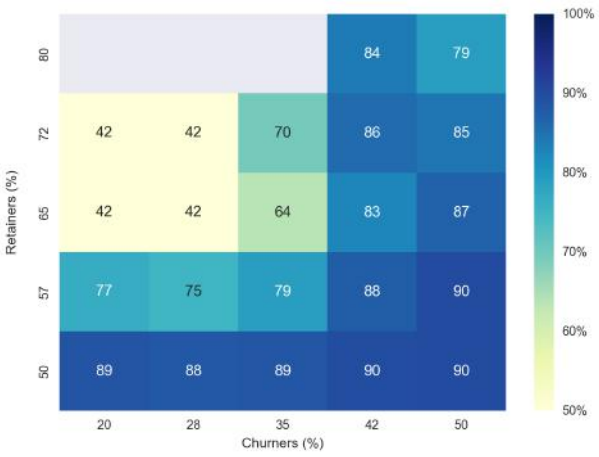


Figure 32: Strong samples performance

As it turns out, deliberately withholding training examples from the model does not lead to better performance. The training test will contain ambiguous examples as well and the estimator needs to know how to interpret them.

7.2.1 *Hyper-parameter Fitting*

We will not list every parameter combination for every algorithm. Nevertheless, one interesting case is the Perceptron's parameter grid.

In figure 33 we can clearly see limitations imposed by the horizon effect and the necessity of applying grid search. Let's see what would happen if a naive searching method would be used.

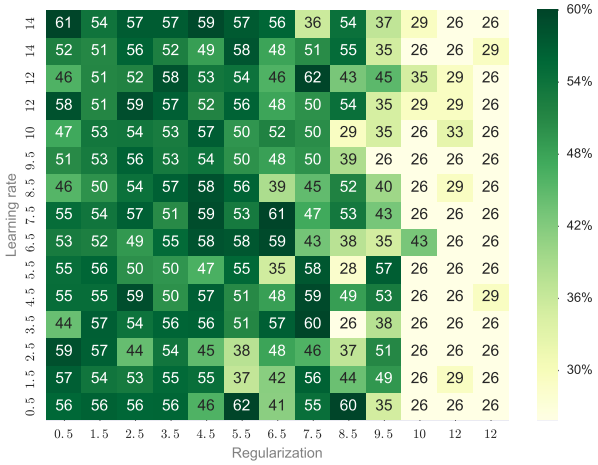
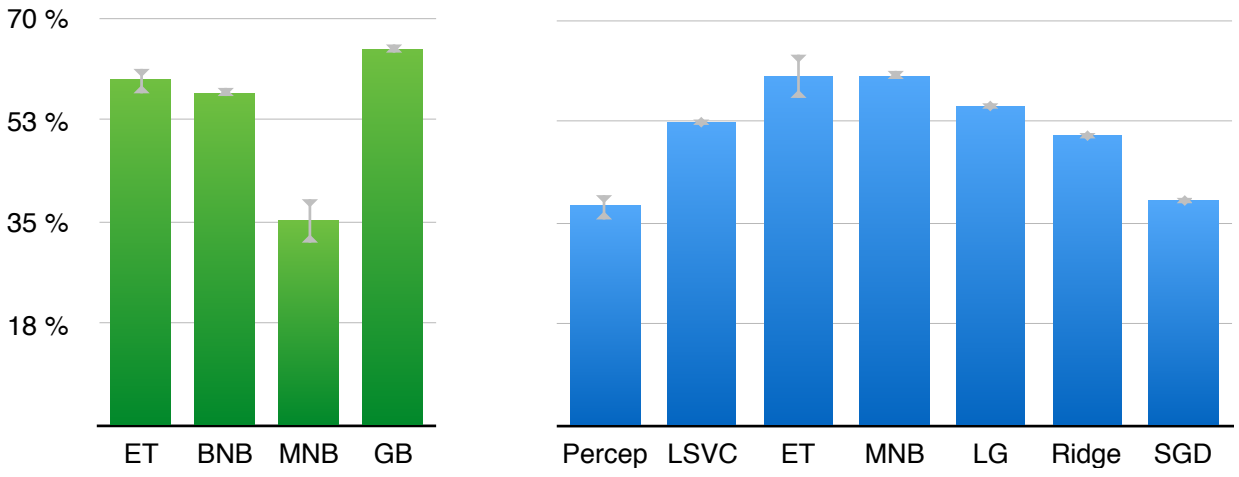


Figure 33: Perceptron parameter grid

Say you start at the midpoint values (0.05 regularization and 0.1 learning rate), with a performance of 52%. Varying the regularization parameter (horizontally) we see that both increasing or decreasing it would achieve lower performance. What we do not see is that after that decrease in performance (to 47%), we reach a score much higher (61%). Since we decided that this is the "best" value for regularization, we will start modifying the learning rate. Decreasing it yields worse performance (39%), while increasing brings a better performance score (53%) followed by a lower one (38%), so we stop after just one increase. What we do not see is that if we had continued to go down the same column, we we would have reached near optimal performance (60%).

After applying this technique we end up with a performance of 53%, much lower than the one found by exhaustively searching - 62%.

Another thing to note here is how well meta-classifiers behave depending on the base-estimator used.



(a) AdaBoost (b) Bagging

Figure 34: Meta-classifiers parameters

Figure 34 shows which models work the best as base-estimators. A multinomial NB classifier works very well with a Bagging approach but shows poor performance in the case of AdaBoosting. Extra Tree classifiers do well in both cases.

7.2.2 *Feature subsets*

The first thing we will look on this step of the pipeline is how many times the algorithms have "chosen" to use each type of proposed subset. By chosen subset, we mean the one that generates the best performance. Figure 35 allows us to compare choices on the two kinds of datasets used: one containing measurements about events and performance and one having additional information in the form of action sequences for each user.

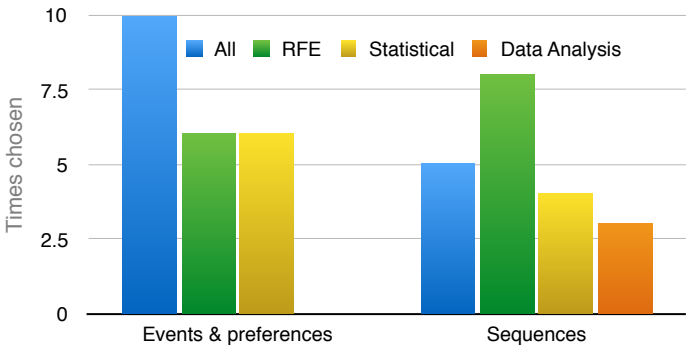


Figure 35: Feature subset choices

The first noticeable fact is that none of the algorithms have chosen the data analysis derived feature subset in the case of dealing with events and preferences. Looking at the same cluster, a very high number of models perform the best when using every feature, in contrast with results from the other dataset. As expected, RFE is the preferred source of feature subsets when dealing with a very high number of characteristics (nearly 100 with sequences, about 20 without). The subset proposed through data analysis is represented here, even though it shows low numbers.

This chart however, is not an indication of absolute effectiveness of the three feature selection techniques. Subset scores were very close to each-other to the point where the preference of one technique over the others can easily be the product of chance and not the merit of the selection method.

This brings us to the next point: low variance models. Variance, this time, measuring how close together the scores a model has produced when trained on multiple feature subsets.

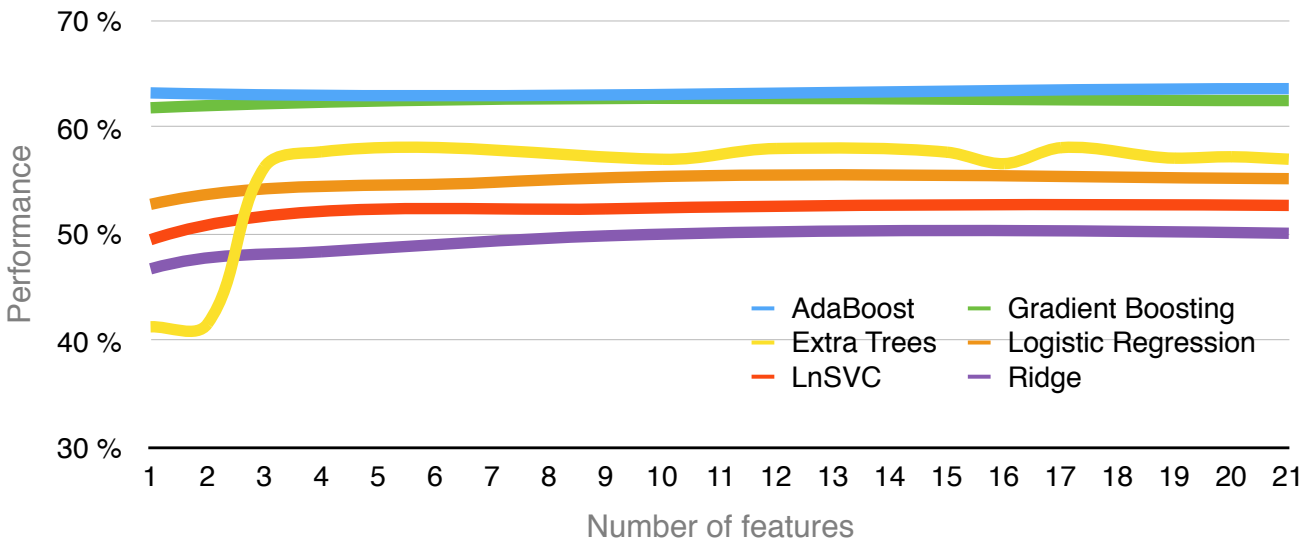


Figure 36: Low RFE-variance models

Figure 36 traces the performance of the most "stable" algorithms, while applying RFE on the events & preferences dataset. The striking conclusion we draw is that some models become very accurate after training on only *less than three* features. Any more additional information brings marginal gains. This nears sparse learning territory, where a low number of features is preferred versus many additional information which would gain minuscule improvements.

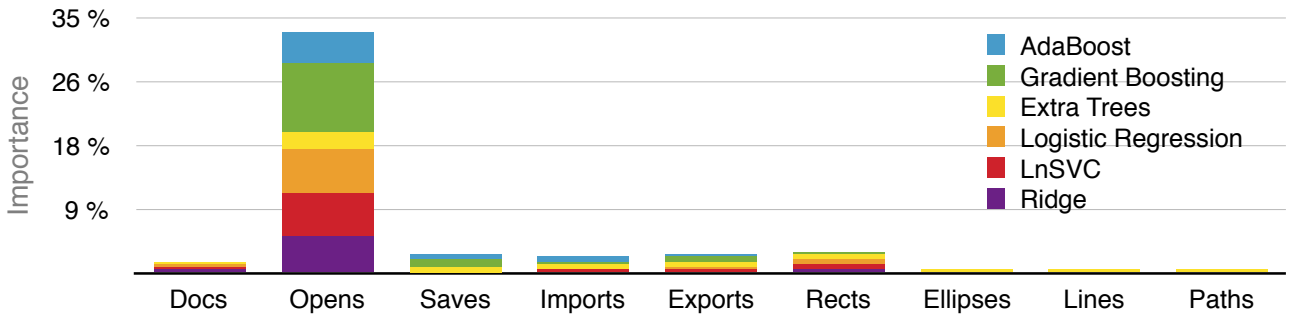
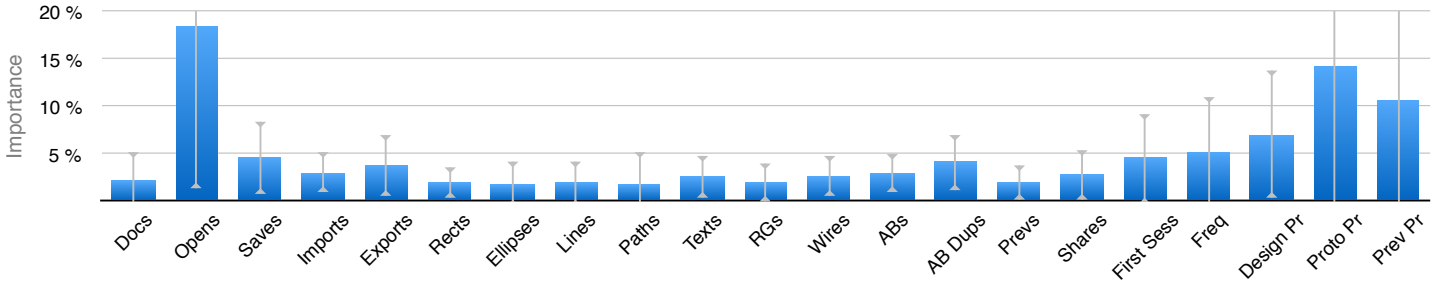


Figure 37: Importance according to sparse models

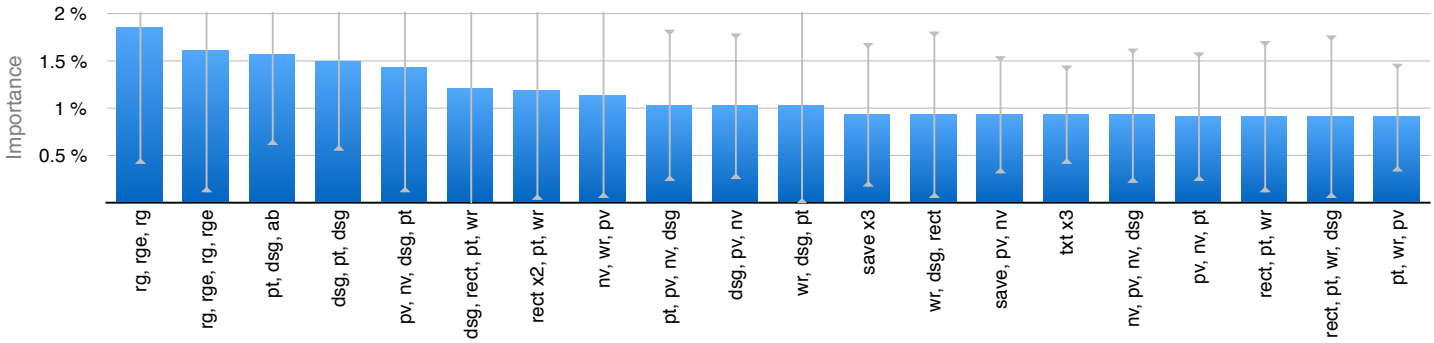
The natural next step is to ask what is this feature that has single-handedly helped estimators predict more than any other. For this, we plot the importance given by each model (figure 37). Outstandingly high is the column representing the number of documents opened, with about 35% aggregated importance. The next three highest rated features are the number of saves, imports and exports. We have to get to the fourth place to get a non-supporting feature - the number of rectangles drawn.

The fact that the number of opens has scored so high should not come as a surprise. A churner, by definition, does not come back to the application. That means a retainer does. When returning, the user has to do something, and as seen in the histograms (figure 8) and violinplots (figure 16) from the data analysis chapter, they are more likely to open an existing document rather than create a new one. This way, opening a document becomes somewhat analogous to launching the application. Since the number of launches is one of the three metrics we have based the labels on, it is quite

natural to see this reaction. It proves the fact that the learning algorithms were able to capture the essence of what was given to them: data labeled partially according to one measurement provided for learning.



(a) Events & preferences



(b) Sequences

Figure 38: RFE-based ranking

Continuing to look at the results, we plot the importance given to each feature by all models, not just low-variance ones. Confirming the previous conclusion, a very high importance is attributed to the numbers of documents opened, although not all models agree on this, denoted by the long error bar. Another interesting result is the importance given to prototyping preference. Not all models rank it highly but it manages to climb to second most important feature.

When adding action sequences into the mix, the disagreement among model rankings grows considerably. Even in this circumstances, some consistent sequences rise to the top. Creation and extension of repeat-grids take the first two places while switching modes, making artboards and prototyping wires take up the next few ones. Not very far from that we see multiple saves and multiple text creations.

As a side note, the presence of action sequences containing nothing but save events might be caused by a learn behaviour some users exhibit. Reminiscent of older times when software products would not provide action feedback quickly enough, some users have gotten used to pressing the keyboard combination for saving (ctrl-s or cmd-s) multiple times in order to avoid unpleasant surprises. This fact doesn't invalidate the action sequence's importance in determining retention, but it should be seen perhaps as a single action.

7.2.3 *Learning Algorithms Comparison*

The time has finally come to compare the results for what we've been building up to until now: hyper-parameter fitted learning algorithms trained on optimal feature subsets. As covered in the background chapter, there is more than one way an algorithm's performance can be measured. Figure 39, presents three performance metrics for a number of representative models. The three metrics chosen are fairly basic ones: accuracy, precision and recall. We have chosen the best estimator from each learning family but omitted ones having results too similar to the rest. A comparison of every model's three basic metrics is available in the appendix, figure 43.

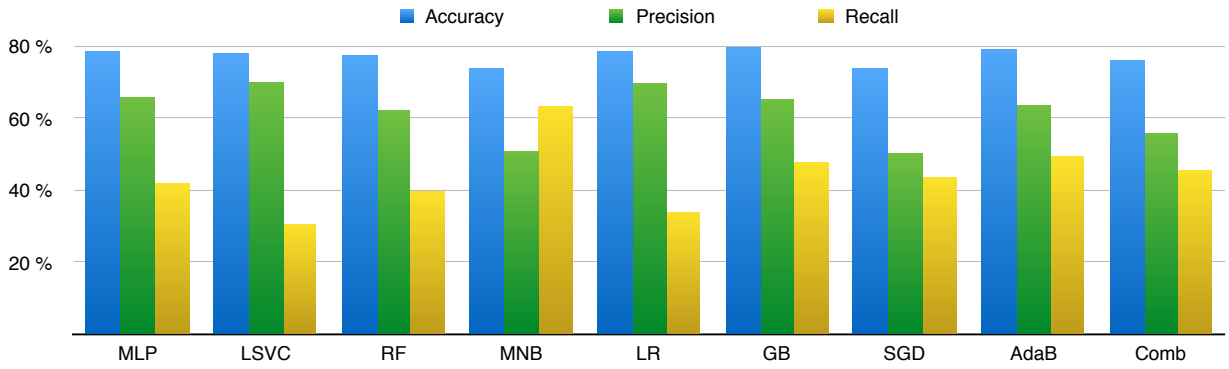


Figure 39: Representative models comparison

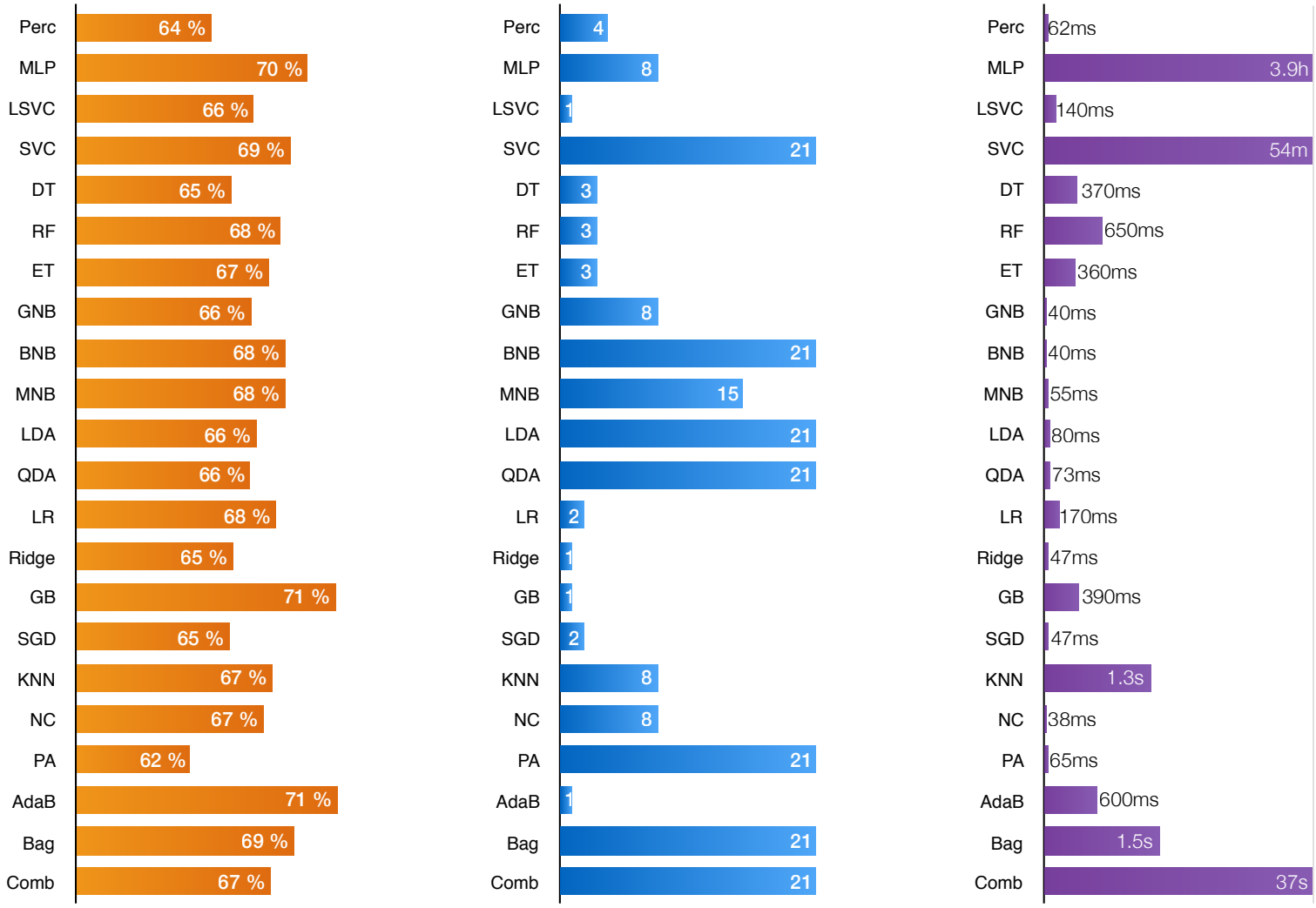
One thing all models have in common is high accuracy. Apart from Multinomial NB, all of them suffer from low recall. The combining classifier does better than most algorithms but pales in comparison to AdaBoosting. One characteristic shared by nearly all algorithms is not being very good at predicting the outcome when faced with a retainer. This could be a result of the skewness of the classes.

A model that outputs "churner" every time would still get an accuracy of $\tilde{7}5\%$ (because that's how many retainers there are in the dataset), but score zero on the recall and precision scale. Relatively high precision scores assure us that is not the case for our models.

While computing or when comparing multiple algorithms, the need arises for a single numerical value by which models can be evaluated and compared. As explained previously, accuracy alone does not tell the whole story. A popular composite metric is the F1 score (defined earlier), but we felt that not even it can provide a fair representation of performance. We went with a custom score, composed of three thirds accuracy and one third F1.

Figure 40 shows a summary of the matters discussed in this paper in the form of three metrics for each algorithm. They are performance (using the metric just described above), number of features (where sparsity is preferred) and training time (on 18 000 samples of 21 real features each). As with all time measurements, they should be compared relatively instead of absolute values as they are highly dependant on the hardware ran on, the algorithm implementation (Scikit-learn was used in this case) and other factors (such as parallelisation which was not used due to a known bug regarding Python multithreading in Unix).

The exceedingly large time required by the MLP net (3.9 hours) is still insignificant compared to training times for deep networks. Not to mention hyper-parameter



(a) Performance (b) Number of features (c) Training time

Figure 40: Algorithm comparison

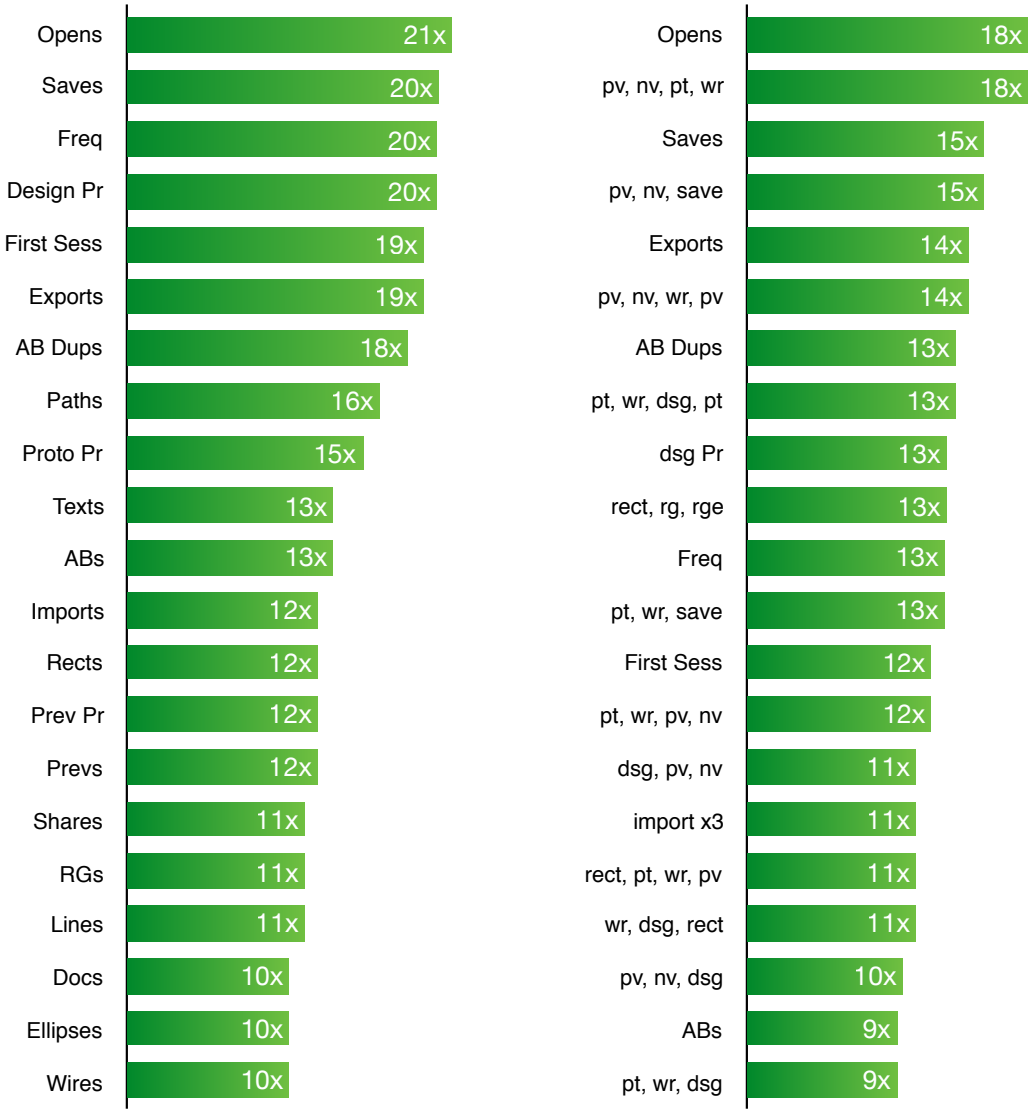
searches which *very quickly* become computationally impractical and instead should be carefully selected manually.

One unfortunate result is that the combining classifier did not perform as expected, being outranked by Gradient Boosting and AdaBoosting techniques. An outstanding performer is GB, being able to reach the highest performance score on a single feature in a relatively small amount of time. MLP, Ridge and AdaBoost are the only other models with prefer (sparsely) only one feature. Logistic Regression and Stochastic Gradient Descent come in close - using two features.

These observations call for a reiteration of this paper's core objective. It is not to fit and train the best performing models of the data using boundless information. Instead we aim to extract the most retention-impactful features of the product. Model scores should not be seen as absolute values, instead they should be used to compare among estimators. This is especially true as we voluntarily withheld user information. Thus a model having the highest score on only one feature (the number of documents opened, discussed in the data analysis chapter) is not the end result we are seeking.

7.2.4 *Most Used Features*

After having a look at how algorithms fare against each-other we direct our attention onto the concrete features selections made by the models.



(a) Events & preferences (b) Sequences

Figure 41: Feature use count

This is where model performance comes into play. Rather than taking the crude usage count by adding one for each estimator that places a feature into its best performing subset, we do a weighting. Each algorithm's contribution has a weight of $(max - score)^2$, where max is the custom-score of the best classifier. This makes better performing algorithms have more impact on the usage count, while poorly comparing ones bring little contribution (and squaring helps make the separation even clearer).

We receive another clear confirmation that the numbers of documents opened is an indication of retention. As it seems to be an effect rather than a cause, we continue down the list. When working with events & preferences only, the average frequency of actions looks like a good predictor of retention. This leads us to believe that

maybe some effort should be allocated into mapping common actions to keyboard shortcuts. Artboard duplication and creation, drawing paths and creating text objects also indicate strong predictive characteristics.

Adding action sequences to the available learning features, we see that they were worth inspecting. The sequence involving previewing and coming back to change a prototyping wire is the second most used one. This can also be partially because it is contained in a proposed feature subset that was selected by many algorithms. As the majority of models has not chosen either data analysis or statistical techniques as their source of feature selection, we can safely rule this off our suspicions list.

Similar to results of previous chapters, action sequences involving modifying wires, saving or switching modes prove to be good indicators of retention. Curiously though, the number of wires alone is the least used feature when evaluating profiles with only events & preferences information available.

CONCLUSION

Running this machine learning experiment has proven effective. It revealed some obvious facts: the number of documents opened is mildly correlated to the number of launches; and some not so obvious ones as well. We learned that the time spent in the first session *does* count a lot and that paths are the preferred tool of the experienced designer. They also organize their working space and are better at reusing content through artboard duplications.

Through the study of action sequences, we also learned that the prototype mode plays an integral role in a retainer's workflow. Previewing the mock-up also scores high on the list of most impactful aspects of the product.

Part IV

CLOSING WORDS

The last part envelops final thoughts on the problem. The first chapter contains a review of the results obtained and a summary of my contribution to the field. The last chapter is a discussion of next steps to be made on the subject.

8 CONCLUSIONS

Determining which feature of a product is the most important one was a complex task. This paper has presented the results of applying various methods for feature selection on a concrete application. The second outcome of the thesis is a reusable method for supervized learning algorithm optimization which hopes to reduce human input to a minimum.

8.1 RESULTS SUMMARY

We consulted three sources in order to robustly asses feature importance. We analyzed the data by looking at event-based and time-based features and then at action sequences. Statistical feature selection methods were employed afterwards, leading into learning algorithms and their feature ranking.

Data analysis revealed that the highest difference between retainers and churners happen when counting documents created and opened, and when interacting with arboards – either creating or duplicating. On the action sequences side, working in prototype mode then previewing along with creating a combination of rectangles and text objects turned out to be the best class discriminators.

Statistical methods, based on variance, mutual information and other scores, indicate high importance for artboard duplications, exports and first session duration. Multiple imports or exports in a row are also prominent.

Making the **machine learn** for us and then asking what it deems important resulted in interesting take-aways. The number of documents opened are tied in with the number of sessions; artboard creations and duplications play a firm role in retainer classification; and prototyping then previewing is one of the most class-differentiating action sequences.

Overall, the features that stand out the most are the ones unique to the product. Prototype and preview are always present in retainers' action sequences and artboard creations and duplications have a high usage count. Basic functionality is still needed, such as the ability to draw paths, rectangles and text objects as well as importing and exporting parts of the document in progress.

8.2 CONTRIBUTION TO THE FIELD

Besides results on a specific application, the tools developed are useful in any other similar setting.

8.2.1 *Combining Classifier*

A meta-estimator which shines when many other learning algorithms are available. The basic idea is that a classifier making predictions while looking at other algorithms' outputs (instead of consulting the data itself) can outperform any individual one. The implementation, in Python, adheres to Scikit-learn's standards and can be easily integrated into other projects.

8.2.2 *Model Optimization Pipeline*

The pipeline seeks to compensate for the user's lack of experience and reduce the necessity of human input. For a given learning task, it finds the best model, hyper-parameter combinations and feature subsets, all in an approximate amount of time inputted at the beginning. After finishing its execution, information for more in depth feature selection and ranking is computed.

Apart from the concrete results and the generalized algorithm, this thesis has helped me understand machine learning concepts a little better. It has provided me the setting to put theoretical knowledge into practice on a real product. While the results may not be the most decisive and the pipeline developed might not stand out as the biggest breakthrough, I consider they have helped me in my personal development.

9

FURTHER WORK

Although a lot of work has been put into solving this task, the conclusions provided are, by their nature, not absolute. The problem can approached from more angles in order to discover interesting facts. More complex learning algorithms can be employed and additional visualization techniques can prove helpful for human interpretation.

Learning

Deep neural networks could reveal more intricate relationships in customer behaviour, especially when using the expanded dataset provided by action sequences. A lateral branch the approach can diverge into is using **sequence learning** which nears the natural language processing (NLP) field. Less rough approximations of retention predictions can be achieved by treating the problem as a **regression** (between 0% and 100% retention probability) one instead of the explored classification task (either retainer or churner).

Implementation

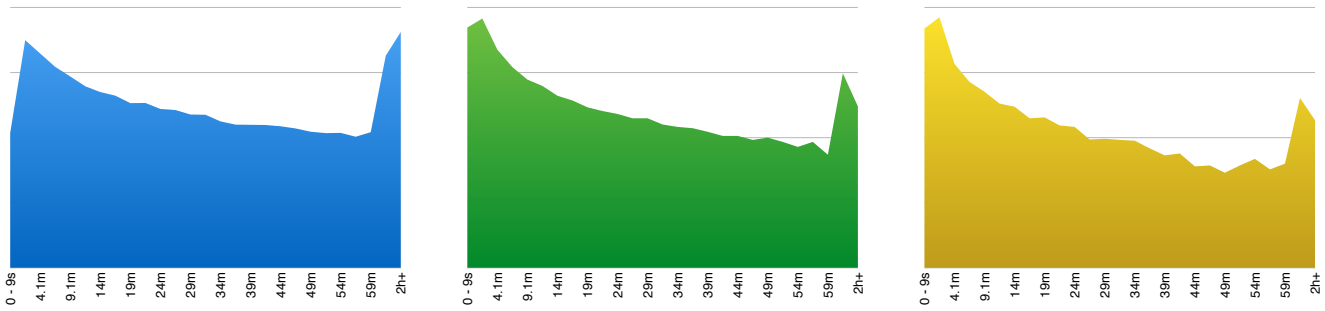
Further **automation** obtained by piping the three separate portions (data analysis, statistical feature selection and machine learning selection which uses previous two) into a single unified entity, requiring even less human input. Processing time on the whole can be reduced by making use of **parallelization** or by exploiting **GPU** power.

Visualization

Principal component analysis (**PCA**) can map feature relationships to lower dimensions making them easier to interpret. On a related note, a technique similar to **word embedding** used in NLP could be employed to provide a single chart of action sequences. Composite **pair grids** can bring a new perspective on pairwise relationships through the use of correlation matrices, kernel density estimation plots and linear regression plots.

Part V

APPENDIX



(a) Design (b) Prototype (c) Preview

Figure 42: Time spent in each mode (logarithmic)

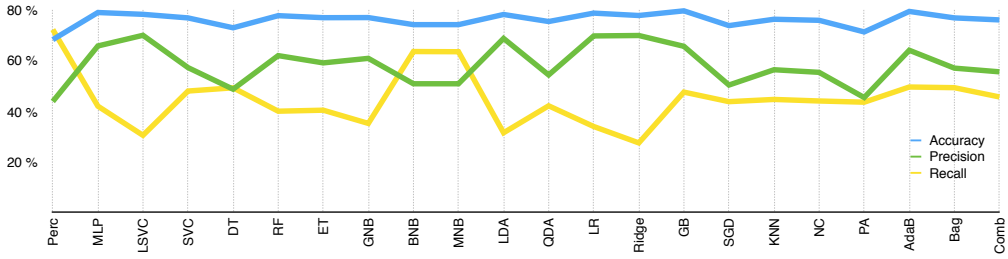


Figure 43: Learning algorithms performance comparison

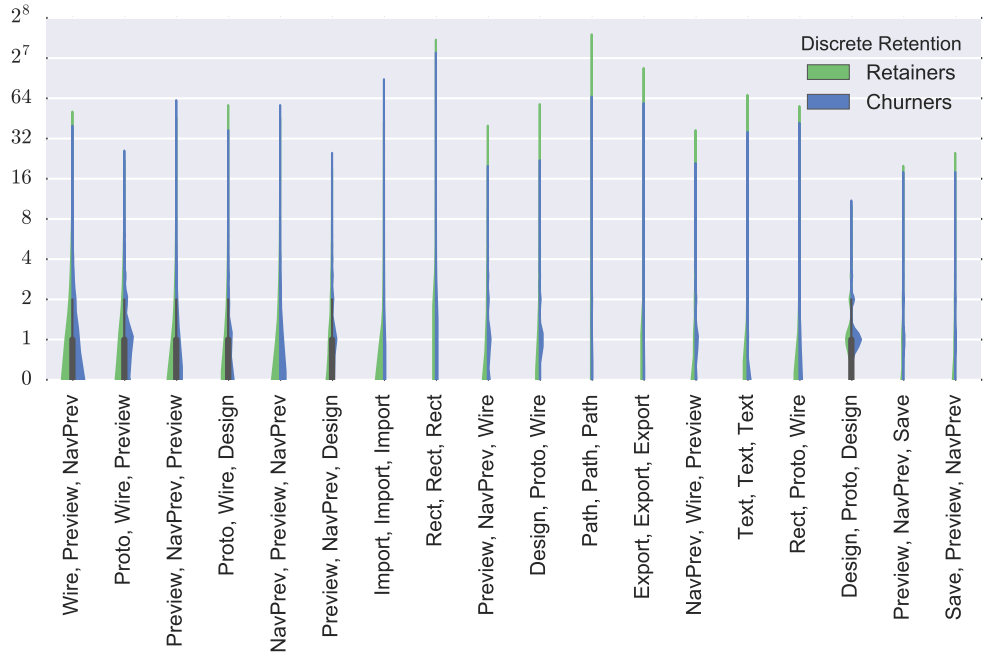


Figure 44: Common Sequences Violinplots

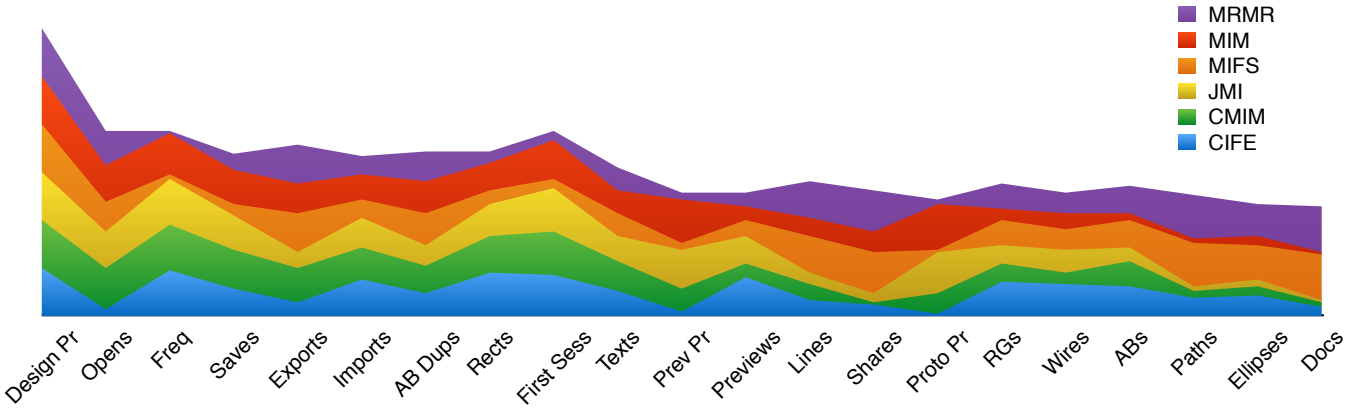


Figure 45: MI importance algorithm contribution

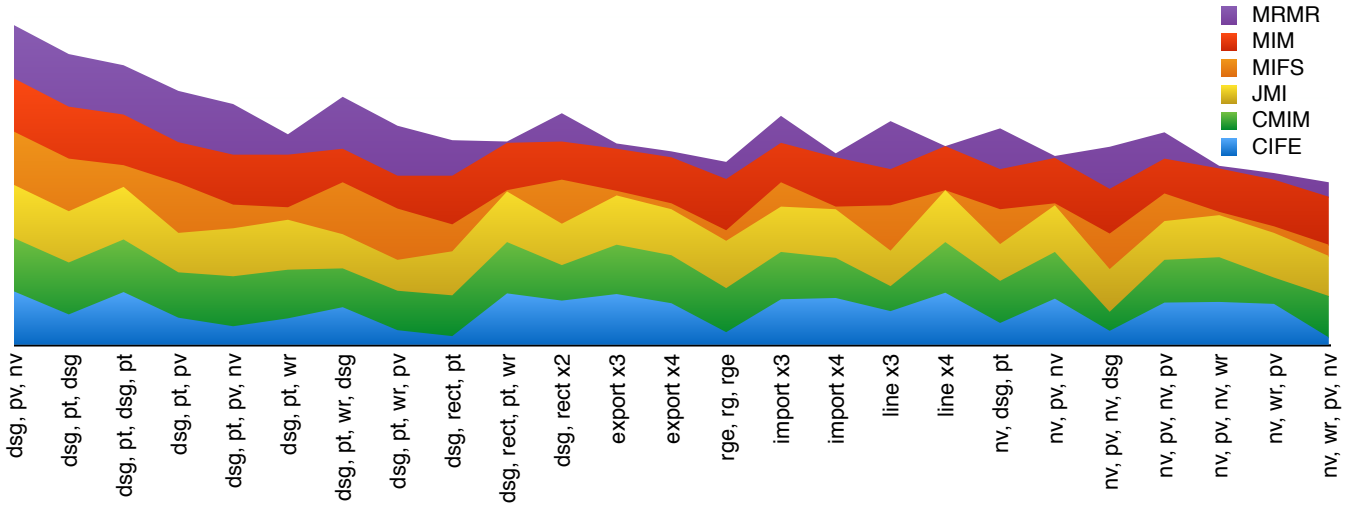


Figure 46: MI ranking algorithm contribution

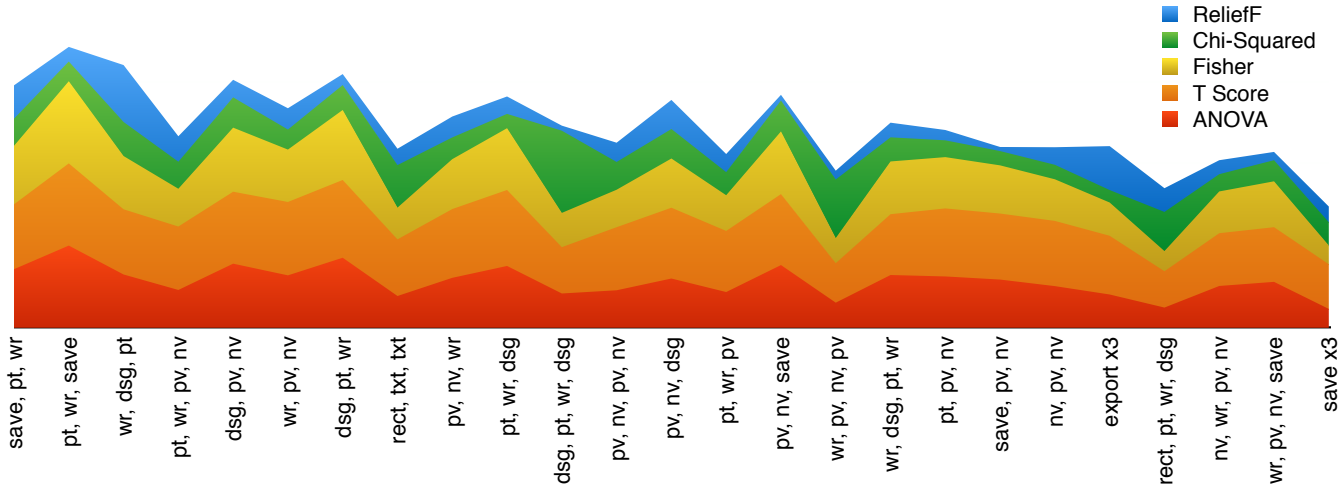


Figure 47: Statistical scores algorithm contribution

BIBLIOGRAPHY

[1] Salem Alelyani, Jiliang Tang, and Huan Liu. "Feature Selection for Clustering: A Review." In: *Data Clustering: Algorithms and Applications*. 2013, pp. 29–60.

[2] Roberto Battiti. "Using mutual information for selecting features in supervised neural net learning." In: *IEEE Transactions on Neural Networks* 5 (1994), pp. 537–550.

[3] Christopher Bishop. *Pattern Recognition and Machine Learning*. New York: Springer, 2006.

[4] Rich Caruana and Alexandru Niculescu-Mizil. "An Empirical Comparison of Supervised Learning Algorithms." In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. New York, NY, USA: ACM, 2006, pp. 161–168.

[5] Girish Chandrashekar and Ferat Sahin. "A Survey on Feature Selection Methods." In: *Computers and Electrical Engineering* 40.1 (Jan. 2014), pp. 16–28.

[6] Thomas Debeauvais, Bonnie Nardi3, Diane J. Schiano, Nicolas Ducheneaut, and Nicholas Yee. "If You Build It They Might Stay: Retention Mechanisms in World of Warcraft." In: *Proceedings of the 6th International Conference on Foundations of Digital Games*. FDG '11. Bordeaux, France: ACM, 2011, pp. 180–187.

[7] S Deviant. *The Practically Cheating Statistics Handbook, The Sequel*. 2nd. CreateSpace Independent Publishing Platform, 2010. ISBN: 1453767142.

[8] L. El Ghaoui, G.-C. Li, V.-A. Duong, V. Pham, A. Srivastava, and K. Bhaduri. "Sparse Machine Learning Methods for Understanding Large Text Corpora." In: *Proc. Conference on Intelligent Data Understanding*. 2011.

[9] Sir 1890-1962 Fisher Ronald Aylmer. *Statistical tables for biological, agricultural and medical research*. Ed. by Frank Yates. 6th. Edinburgh: Oliver and Boyd, 1963.

[10] Francois Fleuret. "Fast Binary Feature Selection with Conditional Mutual Information." In: *J. Mach. Learn. Res.* 5 (Dec. 2004), pp. 1531–1555.

[11] Quanquan Gu, Zhenhui Li, and Jiawei Han. "Generalized Fisher Score for Feature Selection." In: *CoRR* abs/1202.3725 (2012).

[12] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd. New York: Springer, 2009.

[13] J. D. Hunter. "Matplotlib: A 2D Graphics Environment." In: *Computing in Science Engineering* 9.3 (2007), pp. 90–95.

[14] Kenji Kira and Larry A. Rendell. "The Feature Selection Problem: Traditional Methods and a New Algorithm." In: *Proceedings of the Tenth National Conference on Artificial Intelligence*. AAAI'92. AAAI Press, 1992, pp. 129–134.

[15] David D. Lewis. "Feature Selection and Feature Extraction for Text Categorization." In: *In Proceedings of Speech and Natural Language Workshop*. Morgan Kaufmann, 1992, pp. 212–217.

[16] J. Li, K. Cheng, S. Wang, F. Morstatter, R. Trevino, J. Tang, and H. Liu. "Feature Selection: A Data Perspective." In: (2016).

[17] William Lidwell, Kritina Holden, and Jill Butler. *Universal Principles of Design*. Rockport Publishers, 2010.

[18] Wes McKinney. "Data Structures for Statistical Computing in Python." In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stefan van der Walt and Jarrod Millman. 2010, pp. 51 –56.

[19] Patrick E. Meyer, Colas Schretter, and Gianluca Bontempi. *Information-Theoretic Feature Selection in Microarray Data using Variable Complementarity*. 2009.

[20] Feiping Nie, Heng Huang, Xiao Cai, and Chris H. Ding. "Efficient and Robust Feature Selection via Joint l2,1-Norms Minimization." In: *Advances in Neural Information Processing Systems 23*. Curran Associates, Inc., 2010, pp. 1813–1821.

[21] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[22] Guido Rossum. *Python Reference Manual*. Tech. rep. Amsterdam, The Netherlands, 1995.

[23] César R. Souza. *Kernel Functions for Machine Learning Applications*. 2016.

[24] Ashutosh Tiwari, John Hadden, and Chris Turner. "A New Neural Network Based Customer Profiling Methodology for Churn Prediction." In: *Computational Science and Its Applications – ICCSA 2010: International Conference, Fukuoka, Japan, Proceedings, Part IV*. Springer Berlin Heidelberg, 2010, pp. 358–369.

[25] Chih-Fong Tsai and Yu-Hsin Lu. "Customer Churn Prediction by Hybrid Neural Networks." In: *Expert Systems with Applications* 36.10 (Dec. 2009), pp. 12547–12553.

[26] S. van der Walt, S. C. Colbert, and G. Varoquaux. "The NumPy Array: A Structure for Efficient Numerical Computation." In: *Computing in Science Engineering* 13.2 (2011), pp. 22–30.

[27] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

[28] Wikipedia. *Wikipedia, The Free Encyclopedia*. Online – last accesed June, 2016.

[29] Howard Hua Yang and John Moody. "Data Visualization and Feature Selection: New Algorithms for Nongaussian Data." In: *in Advances in Neural Information Processing Systems*. MIT Press, 1999, pp. 687–693.