



MASTERS OF SCIENCE  
HIGH PERFORMANCE COMPUTING  
WITH DATA SCIENCE

---

# Analysing Players in Online Games

---

BRIAN FLYNN

UNIVERSITY OF EDINBURGH

September, 2019

## ABSTRACT

---

This project serves as an investigation into how predictive analytics can be used by our industrial partner, deltaDNA, as a valuable service for their clients. We aimed to make predictions on players early in their life cycle on two grounds: whether they will spend in-game at some point, and whether they will play for more than some defined number of days.

To evaluate the usefulness of predictive services, we construct a model building framework which is used to generate a new logistic regression model based on the parameters chosen by a user. Variable parameters for this framework include which features are used for modelling, what the definition of a long term player is, and which time frames should be considered for training and testing models.

We find that in principle, predictive analytics can be shown to be useful for game player data. We consider the specific case of generating and running predictive models within a Vertica database, as employed by deltaDNA, and find that the newly added suite of machine learning functions to the Vertica system are overly restrictive, and hence immediate implementation is not practical or useful.

Ultimately, we recommend adoption of an online/offline hybrid where data is taken from the database, processed offline, and uploaded to present to clients via the deltaDNA platform, accepting the inherent performance costs this invokes, and dismissing several less favourable options.

## ACKNOWLEDGEMENTS

---

This project would not have been possible without the access to data and unwavering support offered by the staff of deltaDNA. In particular, I am especially thankful for the insightful suggestions and advice offered by Dr. Isaac Roseboom.

I would like to thank Dr. Adam Carter in EPCC for his immense help and patience throughout this project. His guidance has been crucial to my progress at all times, and I am extremely grateful for his input.

# CONTENTS

---

I	INTRODUCTION, CONTEXTUAL REVIEW AND GOALS	1
1	INTRODUCTION	2
1.1	Dissertation Overview	2
1.2	Goals	3
2	DELTADNA AND GAMES ANALYTICS	4
2.1	Services	4
2.1.1	Player Segmentation	4
2.1.2	Retention and Lifetime Value Predictors	6
2.2	Data Format	7
2.2.1	Single Game Data	8
2.2.2	Encoding Categorical Variables	8
2.3	Programming Practices at deltaDNA	9
2.4	Intrinsic Challenges of Gaming data	9
3	DATA SCIENCE	11
3.1	Machine Learning	11
3.1.1	Unsupervised Learning	12
3.1.2	Supervised Learning	12
3.2	Accuracy Metrics	13
3.3	Case Study	15
3.3.1	Uplift	17
3.4	Choosing a Metric	18
4	ALGORITHMS	20
4.1	Regression	20
4.1.1	Null Hypothesis and P-value	21
4.2	Linear Regression	21
4.3	Logistic Regression	23
4.3.1	Mathematical Formulation	23
4.4	Suitability of Algorithms	26
5	GOALS	27
5.1	Overarching Goal	27
5.2	Build a model	27
5.3	Learn by applying predictive models	28
5.4	Extend modelling to address other questions	29
5.5	Assess Models	30
5.6	Generalising model	31

5.7	Deviation from Earlier Plan	32
II	METHOD	33
6	GENERAL APPROACH	34
6.1	Mapping Goals to Methodology	34
6.2	CRISP-DM	34
6.3	Junctions and Decisions	36
7	INITIAL DATA INSPECTION	38
7.1	Python Model	38
7.1.1	Data cleaning	39
7.2	Unsuitability of static data	39
8	LOGISTIC REGRESSION	41
8.1	Vertica SQL Model	41
8.1.1	Vertica Machine Learning Functions	41
8.1.2	Limitations	45
8.2	Dynamic Queries for feature selection	45
8.2.1	Limitations and Next Steps	47
9	EXTENSION TO FRAMEWORK	49
9.1	General Linear Model	49
9.1.1	Running a General Linear Model	50
9.1.2	Factorisation	51
9.1.3	Switching Between Model Types	51
9.2	Variable Parameters	53
9.2.1	Predictions Based on Early Behaviour	53
9.2.2	Normalisation	54
9.3	Formatting results	54
9.3.1	Vertica Results	54
9.3.2	Hybrid Results	55
III	RESULTS AND ANALYSIS	57
10	BUILDING A MODEL	58
10.1	Vertica Functions	58
11	USING THE FRAMEWORK	60
11.1	Feature Selection	60
11.1.1	Eligible Features	61
11.1.2	Basic Numeric Features	63
11.1.3	Impact of Categorical Features	64
11.2	Choosing Feature Sets	65
11.2.1	Spender Model	66
11.2.2	Churn Model	67

11.2.3	Selecting a Feature Set	67
11.3	Long-Term And Early Behaviour Cutoff Points	69
11.3.1	Spender Models	69
11.3.2	Churn Models	70
11.4	Chosen Parameters Summary	72
12	EXTEND THE MODEL	73
12.1	Weighting Samples	73
12.2	Time Based Model	75
12.3	Other games	76
12.3.1	Data Quality	77
12.3.2	Churn Models	78
12.3.3	Spender Models	79
12.3.4	Conclusion	79
13	ASSESS FRAMEWORK	80
13.1	Feasibility	80
13.1.1	Performance	80
13.2	Vertica Only Model	81
13.2.1	Conclusion on Vertica's Applicability	82
IV	CONCLUSIONS AND RECOMMENDATIONS	83
14	GENERIC FRAMEWORK	84
14.1	Framework Evaluation	84
14.1.1	Goal 1: Build a Model. Chapter 10.	84
14.1.2	Goal 2: Learn by Applying Predictive Models. Chapter 11.	84
14.1.3	Goal 3: Extend Modelling to Address Other Questions. Chapter 11.	85
14.1.4	Goal 4: Assessing the Model. Chapter 13.	86
14.1.5	Goal 5: Generalising to Framework. Chapter 14.	86
15	RECOMMENDATIONS FOR FUTURE DEVELOPMENT	87
15.1	Improvements to Framework	88
15.1.1	Pipeline for Analysis	88
15.1.2	Automated Parameter Selection	89
15.1.3	Answer More Useful Questions	89
15.2	User Interface and Platform Integration	90
16	CONCLUSIONS	91
16.1	Final Conclusion	92
16.2	Final Recommendation	92

## LIST OF TABLES

---

Table 2.1	Number of Players by Gender and Age	5
Table 2.2	Number of Spenders by Age and Gender	5
Table 2.3	Percentage of Spenders by Age and G gender	5
Table 3.1	Players Registered in the First Release of a Game's Cycle	15
Table 3.2	Players Registered in the Second Release of a Game's Cycle	15
Table 3.3	True/False Positives/Negatives from Random Prediction	16
Table 3.4	Overall Results from Random Prediction	16
Table 3.5	Metrics of Random Prediction	16
Table 3.6	True/False Positives/Negatives from Trained Prediction	17
Table 3.7	Metrics of Trained Prediction	17
Table 3.8	F1-Scores Example	19
Table 4.1	Probability and Odds Ratio Example	24
Table 8.1	Sample Output from Vertica Logistic Regression	44
Table 8.2	Example Accuracy Metrics by Feature Set	47
Table 10.1	Results of Random and Generated Predictions	58
Table 10.2	Accuracy Metrics of Random and Generated Predictions	59
Table 11.1	Eligible Features	62
Table 11.2	Feature Sets	63
Table 11.3	F1 Scores for Purely Numeric Feature Sets	64
Table 11.4	F1 Scores for Categorical and Basic Numeric Features	65
Table 11.5	Losses of extending longterm cutoff by one day	71
Table 12.1	Results of Weighting Positive Spending Samples	74
Table 12.2	Timeline of Training/Test Timeframes	75
Table 12.3	F1-Scores for Time based models	76
Table 13.1	Timings for Model Types (in seconds)	81

## LIST OF FIGURES

---

Figure 2.1	Sample Retention Predictor	6
35figure.caption.19		
Figure 11.1	F1-Score Vs Probability Threshold for Spender Models	66
Figure 11.2	F1-Score Vs Probability Threshold for Churn Models	67
Figure 11.3	Maximum Observed F1-Score from Feature Set Vs Long-term cutoff	68
Figure 11.4	F1-Score Vs Early Behaviour Cutoff Day	69
Figure 11.5	F1-Score Vs Early Behaviour Cutoff By Long-term cutoff	70
Figure 12.1	F1-Score Vs Probability Threshold for Varying Positive Weights	74
Figure 12.2	F1-Score Vs Probability Threshold for Churn Models on Other Games	78
Figure 12.3	F1-Score Vs Probability Threshold for Spender Models on Other Games	79



## Part I

### INTRODUCTION, CONTEXTUAL REVIEW AND GOALS

## INTRODUCTION

---

### 1.1 DISSERTATION OVERVIEW

This project is a first step towards the introduction of predictive analysis tools to the online services offered by our industrial partner, the gaming analytics company deltaDNA. We must understand *why* this might be considered useful, before we can begin to question *how* we can proceed.

We therefore offer a brief description of deltaDNA, including a discussion on the type and quantity of data available, to establish a set of goals by which to determine whether our output may be deemed successful. Ultimately, any such industrial partnership aims to result in a product which the partner could develop into a valuable service for their clients in the long term.

With a firm grasp on the context of the project, we can then turn our attention to the mechanics of achieving those objectives, namely through employing data science techniques. A vital stage of any data science project is to identify which approaches potentially match the problem at hand; we will study a number of algorithms, and assess them in terms of suitability to our goals.

Then we may proceed with the project body: the steps towards achieving the goals we set out. This involves several distinct stages, which we may consider as initial learning; project groundwork; essential model formulation; extension of that model into a robust framework, and application of that framework to test a number of propositions.

When we are satisfied with the generated model, we must evaluate it under a number of viewpoints. We will identify quantifiable metrics by which we can deem our model, and its subsequent iterations, successful or unsuccessful.

These determinations are the driving factor in deciding what the next logical step of the project ought to be: i.e. whether we can continue development along the current trajectory, or we should reconsider the goals given the evidence of the most recent stage.

Finally, we offer a discussion on interesting aspects of the project which were not examined fully. In doing so we present proposals for the continuation of this study, with the ultimate goal of developing a ready-to-go product for deltaDNA to implement.

## 1.2 GOALS

We cannot fully declare goals until the context of the project is understood. Therefore a thorough discussion on the aims of the project, including means to measure success, are removed until after a description of deltaDNA, their motivations in undertaking this project, and the data science techniques available. We will revisit goals in Chapter 5.

## DELTADNA AND GAMES ANALYTICS

---

As described in the *Project Preparation* report submitted prior to this thesis, deltaDNA, [1], are a gaming analytics company offering services to game developers on mobile and console platforms (this section reiterates some of the descriptions given in said report). Within this capacity, they aim to help clients understand the player base using their game, such that advertising campaigns and in-game offers can be specialised to individual players in order to optimise *Average Revenue Per User*, (ARPU).

In particular, clients are generally interested in two cases: players who spend money in-game, and those that consistently play the game over an extended period. Our primary concerns, therefore, are *revenue*, the real currency spent by players, and *churn*, whether a player will leave the game.

A player is deemed to have *churned* when they leave the game and do not return. In effect, this is used to separate players who may be considered as *long-term* from *short-term*. That is, a short term player churns considerably earlier than a long term player. Obviously, long-term players are of much greater value to game developers. Even if their increased playing time does not result in them spending in-game, their presence lets the developer present ads to them, and can thus increase the revenue generated from that player either way. Therefore, models which aim to predict churn are built by determining whether a player will become a long-term player or not. As such, we require a *long term cutoff*: the minimum number of days a player must play for to be considered a long term player. We find that the value chosen for this cutoff is of significance in modelling, so do not declare one here. We will demonstrate how this parameter can be chosen in §11.3. Throughout this text, we will interchangeably refer to models built for churn and finding long term players; note that these are equivalent.

### 2.1 SERVICES

#### 2.1.1 Player Segmentation

One core analysis provided by deltaDNA drills down players by their characteristics. This can be of interest to clients to understand who their users are, how they interact with the game, and which type of players are more lucrative. For instance, if a game has a player base of 10,000 players, of whom 150 have ever paid, we could establish the perspective seen in tables 2.1 to 2.3

Age	Number of males	Number of females	Total number of players by age
15-20	3000	2000	5000
20-25	2000	1500	3500
25-30	750	300	1050
30+	250	200	450
Total	6000	4000	10000

Table 2.1: Number of Players by Gender and Age

Age	Number of male spenders	Number of female spenders	Total spenders by age
15-20	20	10	30
20-25	35	35	70
25-30	10	15	25
30+	15	10	25
Total	80	70	150

Table 2.2: Number of Spenders by Age and Gender

Age	% male spenders	% female spenders	% spenders by age
15-20	0.7	0.5	0.6
20-25	1.8	2.3	2
25-30	1.3	5	2.4
30+	6	5	5.6
Total	1.3	1.8	1.5

Table 2.3: Percentage of Spenders by Age and G gender

This simplistic example demonstrates how misleading data can be: a first glance would suggest that the majority of revenue derives from males aged 15-20, since this segment has the highest number of players. When we break the information down we see that this segment is in fact among the worst *spenders*, with only 0.7% of these actually spending. In fact, the highest proportion of spenders is males over the age of 30 (6%), followed by females over the age of 25 (5%), both of which are considerably higher than the overall average spend-rate of 1.5%. This allows the client to focus a campaign on these users, since they are the most likely to result in increased revenue.

Clearly the above example is overly simplistic, though it demonstrates the process by which deltaDNA can tell clients who their most valuable players are. Moreover, clients can define user segments and track all metrics pertaining to only those players over time, for instance to

track the impact made by an advertising campaign targeted as 17-19 year olds using Android devices in the United Kingdom.

### 2.1.2 Retention and Lifetime Value Predictors

Two of the key forward-looking services provided by deltaDNA are their retention and lifetime value (LTV) predictors.

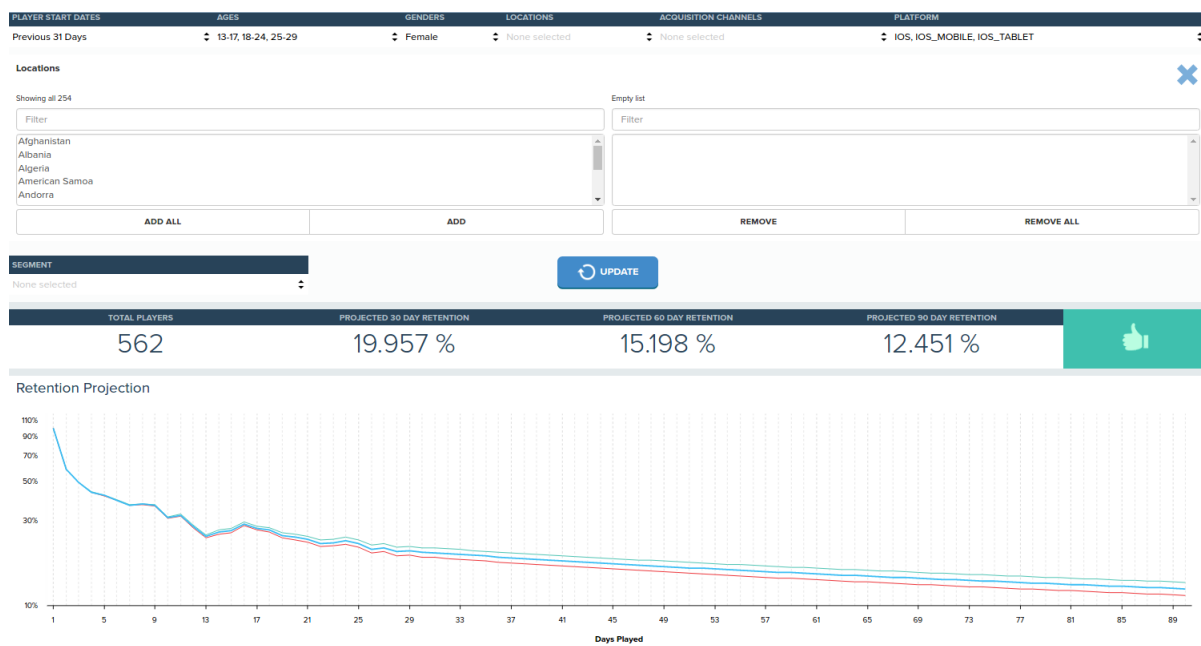


Figure 2.1: Sample Retention Predictor

These allow the user to input a set of parameters about the segments of users they wish to examine, and find out the rate at which that demographic is expected to return to the game, or spend in-game. For example, as in Figure 2.1, the client could inspect female players aged 13-29 who've signed up in the previous month on iOS platforms only. This results in 562 eligible players. The predictor shows that 19.957% are given as *projected 30 day retention*; of which 15.198% are *projected 60 day retention*, and 12.451 *projected 90 day retention*. Likewise, the LTV predictor shows the projected number of spenders within a segment, their average spend and the total revenue generated.

We note here that these are *projections*, as opposed to *predictions*. These tools provide analysis by projecting trends, rather than explicitly considering players as individuals. For instance, a projection *assumes* that 2% of males buy in-game, following historical trends, while only 1.5% of females do. It also *assumes* that 2.5% of 18-25 year olds spend, while 1% of 12-15 year olds

do. By compounding these assumptions based on the set of parameters chosen by the client, it derives an overall *rate* of spending among that demographic.

These services then, taken together with segmentation, §2.1.1, provides context for our project's scope: we aim to reduce the "guesswork" whereby we can only establish that a certain percentage of a certain demographic may follow a particular behaviour. Instead, we aim to be able to make statements on *each individual* player, based on their specific combination of characteristics and previous behaviour. That is, rather than saying

"This player belong to segment number 1, and therefore has a 2% probability of expenditure"  
we would prefer to say:

"This player's characteristics correspond to a probability of 20% that they will spend."

This specificity on a player-by-player basis would enable developers to target only the most likely candidates for their purpose, and would therefore be a valuable addition to the arsenal of services offered by deltaDNA.

## 2.2 DATA FORMAT

deltaDNA store data about players in the column oriented relational database management system Vertica, offered by Hewlett Packard, [2]. Clients of deltaDNA alter their source code to feed players' data to the platform's services. As such there is an abundance of user data available for analysis. There are two key types of data available for each player

- Demographic data, e.g.
  - age
  - nationality
  - gender
- Behavioural data, e.g.
  - number of levels completed
  - time played
  - currency spent

A large number of game-specific metrics are collected for each game, for instance a game which has several types of points/coins achievable by players will store data for each such field;

naturally the significance of such features will vary by game, and it is not universal to claim that any variation of points may be important to all models. Arguments of this kind extend to most types of data available, so we conclude that the fields which should be considered must be decided individually on a game-by-game basis, as discussed in §11.1.

### 2.2.1 *Single Game Data*

For the purpose of explanation, it suffices to consider only one game's data so that we may understand what is and isn't available, and therefore what is and isn't possible. In particular, we take data from a popular match-three game tracked by deltaDNA, henceforth referred to as the training game, Game A.

deltaDNA operate a data hierarchy: all events are stored in a wide events table. From this, metrics and session level aggregates, such as number of levels completed in a single session, are drawn which are then used to drive common charts and analyses. All actions are stored in summary by the session they occurred in a table called `fact_user_sessions_day`. Lifetime summaries, such as total number of missions played since signing up for the game, are held in a table called `user_metrics`.

As such, there is an abundance of data available for Game A. It is clear from inspection that many of the fields duplicate information, for example fields such as platform have multiple fields, `fieldPlatformFirst` and `fieldPlatformLast`. In some cases players change platform, though it is likely to have little positive impact to include both of these features. Such deliberate considerations form the basis for large scale feature elimination. We scrutinise which features are eligible for inclusion in modelling in §11.1, when we have a stronger grasp on how our proposed model will work. This is the first step in a process which is typically referred to as *Feature Selection*, where we choose a small number of features known about all samples to make predictions about those samples.

### 2.2.2 *Encoding Categorical Variables*

In some cases, variables are recorded in categories, such as gender, nationality etc. We may expect some of these fields to be useful, however learning algorithms invariably accept only numerical input in order that an equation may be generated. We consider then, the possibility of *encoding* categorical variables. For our purposes, there are two main methods of encoding:

**BINARY** If the variable is binary, it can be encoded with the addition of a single column. For instance, if we are trying to predict which players become spenders from historical data,



we have a field `spender`, which registers 1 for spenders (positive cases) and 0 for non-spenders (negative cases). That is, we represent expenditure as a binary variable.

**ONEHOT ENCONDING** For a category with  $m$  possibilities,  $m$  new fields are added, each corresponding to one of the original categories. In the new field which the record *does* match, that value is turned to one, while in the  $m - 1$  other fields, it registers as 0. For example, rather than having a field `region`, we introduce three new fields: `Europe` (1 or 0), `Asia` (1 or 0) or `America` (1 or 0).

## 2.3 PROGRAMMING PRACTICES AT DELTADNA

The services provided by deltaDNA extract data from the Vertica database and present view-points on that data via the web based platform, e.g. Figure 2.1. During development, we are mindful that a tool would be most useful for actual implementation if it can perform all of its tasks remotely on the Vertica database, i.e. without having to extract, manipulate, analyse and upload results to the deltaDNA platform. This concept is absorbed in our goals, §5, and will be addressed in our analysis, Part III.

In order to query the Vertica database, we submit SQL queries. This can be done directly using a connection to the database, e.g. using a database client such as DBeaver, [3], or else using a driver such as RJDBC for R, [4], allowing us to write R scripts and push SQL queries through those. This matches the approach used by data science staff within the firm, so is suitable for building products like those envisioned here.

Vertica offers a predictive analysis toolkit which can directly run machine learning algorithms on data held in its database, [5]. This is a relatively new feature, however, and therefore has limited ability. In particular, only three algorithms are currently available:

- Linear Regression (supervised method)
- Logistic Regression (supervised method)
- k-means clustering (unsupervised method)

In §3.1, we will discuss how these relate to our data and nominate one to proceed with.

## 2.4 INTRINSIC CHALLENGES OF GAMING DATA

Every type of data has inherent challenges which must be understood and overcome before we can expect to profit from their analysis. In our case, we are dealing with data about game

players. Consequently, our data has limitations such as players many withholding information such as gender and age, which may have been useful in predictive analysis.

A more prominent issue in dealing with gaming data is the cyclical nature of the gaming industry: upon release, games experience a high level of sign-ups, which plateau within a number of days. Typically, games release updates periodically, inducing surges of sign-ups for a few days thereafter. This scattered data frequency means that our model must be sensitive to these fluctuations: players change over time, so we are effectively chasing a moving target, so our model can not be statically trained and assessed, but rather must evolve alongside the player base.

Suppose new releases are denoted  $r_i$ , corresponding to a six-week period,  $t_i$ . One potential solution is to make predictions on players who sign up in  $t_i$  based on a model which was trained using all player data from  $t_{i-1}$ . In doing so, our model *will be* based on fundamentally different players than those it assesses, but the two sets should be sufficiently similar to allow the model a fair level of accuracy. We will test this *trailing* model concept in Aim 3.2.

Another considerable option is to train a set of models, say one for each day after sign-up. i.e. one model attempts to predict behaviour on sign-up; if the player returns for a second day, another model predicts behaviour using updated values, and so forth. This would aim to answer questions such as:

- If a player pays within the first four sessions, how likely are they to spend again?
- If a player plays for four consecutive days without spending in-game, how likely are they to ever spend?
- If a player plays for three days, then does not play for two days, how likely are they to return?

This option would seem to offer more specific predictive ability and are certainly of interest to developers, who can devise actions such as push notifications or specific offers based on the results. However, training and updating such a large spectrum of models - across a number of games - would take considerable time and resources, and so performance becomes an issue. We do not examine these possibilities in this dissertation, though it is an interesting area of future development, so is included in our recommendations for future work, Chapter 15.

## DATA SCIENCE

---

*Data Science* refers to the process of extracting actionable knowledge from available data. That is, applying computational techniques to *learn* from a dataset, by determining the relationship between a set of characteristics. This is often done through the process of *Machine Learning*. All data science topics relevant for our purposes fall under machine learning, so we do not study other aspects of the field here. A broad understanding of data science, and more specifically our basis for machine learning definitions, can be found in many standard textbooks, such as *Doing Data Science*, [6], which was consulted throughout this project.

### 3.1 MACHINE LEARNING

Machine learning is usually applied as one of two types: *supervised* or *unsupervised*.

**SUPERVISED ALGORITHMS** Methods where an outcome is known in the case of each instance within the training data. That is, the model aims to determine the relationship between a set of characteristics - *independent variables* - and a target characteristic - *dependent variable*.

**UNSUPERVISED ALGORITHMS** Methods without a target variable. The algorithm aims to find structure among the input characteristics, which can then be seen to categorise, or *classify*, the data into distinct types.

These two types of algorithms are suitable to different data sets and objectives, and it is important to ensure that the most appropriate method has been chosen before applying it. In our case, we have data pertaining to game players, and we wish to predict their future behaviour; i.e. whether they are likely to become spenders or long term players. In generating a training set, we will be looking at relatively old players, and will therefore *know* whether a set of characteristics correspond to a particular case. In other words, we have known values for a target variable. Therefore, we can automatically eliminate unsupervised methods and declare that our problem is better suited to supervised algorithms. We will briefly describe some basic concepts of unsupervised methods, but our main focus in this chapter will be to clarify the role of supervised learning and see how it can be applied to our general goals.

### 3.1.1 *Unsupervised Learning*

*Unsupervised* machine learning methods are those which seek to determine structure within the input space of unlabelled data, i.e. data which is not definitively belonging to one of a number of distinct categories. Unsupervised methods therefore find similar instances and group them together. This is known as *clustering*.

One such method imposes that a dataset consists of a predetermined number of groups. Each new instance passed to the algorithm is then determined to belong to the group whose average it lies nearest to, called k-means clustering.

### 3.1.2 *Supervised Learning*

*Supervised* machine learning refers to methods where the algorithm aims to arrange data into known categories. That is, to map a set of input features to one of finite possible outcomes, by inferring a relationship between said input characteristics.

This is useful in cases where we wish to *classify* instances of data where a value is known for each such characteristic. For example, consider attempting to determine whether a patient has some heart infection: in all cases tested to date, we would have data pertaining to each patient - e.g. age, weight, height, eating habits etc. - as well as whether or not that patient is found to have the condition. Suppose we have the case history of ten thousand patients who have been tested specifically for this condition, and that two hundred test positive.

A model is *trained* on a subset of instances for which the target feature is known, i.e. a *training set*. That is, a relationship is derived between the characteristic features of the patients and their association to the condition. Such a relationship could, for instance, indicate that patients who smoke are highly likely to test positive, meaning they have *high correlation*. Moreover, there may be no apparent *correlation* between a patient's height and their condition. While training the model, it is expected that indicators of greater significance will naturally become more decisive by virtue of the coefficient assigned to them. We will go into greater detail about these details in §4.1.

Having built a predictive model, it remains to verify that model's effectiveness. This can be done by applying the resultant equation to instances of data which were not included in the training of the model, but for which the true target feature is known. This is called a *test set*. We can simply compare the output of the model with the real value to establish the accuracy of the model. Some more specific metrics exist for assessing particular aspects of supervised methods, discussed in §3.2.

### 3.2 ACCURACY METRICS

In order to evaluate the effectiveness of a machine learning algorithm, we introduce a number of standard metrics which are commonplace in data mining, [7]. We first introduce the concept of *true/false positives/negatives*, which are terms to describe a prediction made by a classification algorithm on a sample when compared with its known value.

**TRUE POSITIVES (TP)** Cases where a sample *does* have the characteristic we are testing for, and the model predicts that it does.

**FALSE POSITIVES (FP)** Cases where a sample *does not* have the characteristic of interest, but the model predicts that it does.

**TRUE NEGATIVES (TN)** Cases where a sample *does not* have the characteristic, and the model predicts that it does not.

**FALSE NEGATIVES (FN)** Cases where a sample *does* have the characteristic, but the model predicts that it does not.

We can now apply these concepts to find the *rates* at which they occur when a given model is applied.

**ACCURACY** The rate at which the algorithm predicts the correct result.

$$\frac{TP + TN}{TP + TN + FN + FP} \quad (1a)$$

**PRECISION** Positive predictive rate. Of those predicted to test positive for our condition, what percentage *actually have* the condition.

$$\frac{TP}{TP + FP} \quad (1b)$$

**SENSITIVITY** True positive rate; also known as *Recall*. Of those which *are* positive of the condition, what percentage are found to be positive.

$$\frac{TP}{TP + FN} \quad (1c)$$

**SPECIFICITY** True negative rate. Of those which *do not have* the condition of interest, how many are predicted not to have the condition.

$$\frac{TN}{TN + FP} \quad (1d)$$

These metrics become meaningful only in the context of the model being built; it is senseless to suggest that a good algorithm must have an accuracy over a certain percentile, as this is a declaration that depends entirely on the requirements of the client. Consider the case of detecting cancer: an algorithm may be devised which is accurate in 90% of cases, but all of the 10% of incorrect diagnoses are false negatives; that is, they tell ill patients they are healthy. Alternatively, consider where the 10% of inaccurate diagnoses are false positives, telling healthy patients they are ill. This will be less severe since they will be brought for subsequent testing and the incorrect prediction discovered, whereas the false negative may be let go. In this case, a low false negative rate is more important than a low false positive rate, so we would opt for an algorithm with high sensitivity over one with high specificity.

In many cases, and indeed in ours (as we will see in §3.4), none of these rates are sufficient to assess a model one. We instead aim to find a *balance*, which represent a model's effectiveness for more than one criteria. There are a set of derived metrics which allow us to quantify the interaction between these metrics. Of potential interest to us are:

**UPLIFT** The percentage improvement, with respect to some dedicated metric, gained by using the developed method over the alternative/previous method. This differs from the standard definition where uplift is used to measure impact of an action compared with the case where no action is taken. Note that uplift is quoted in absolute percentage points: a shift from 10% to 20% accuracy constitutes an uplift of 10%, as opposed to quoting uplift as a percentage improvement over the initial model, where the same shift would be a 100% uplift.

**F-SCORE** or *F-measure*, is the harmonic mean of precision and sensitivity, [8]. This demonstrates the weighting given to one aspect over the other. In the case where precision and sensitivity are considered equally important, the  $F_1$ -score is given by Equation 2a.

$$F_1 = \frac{2 \times \text{precision} \times \text{sensitivity}}{\text{precision} + \text{sensitivity}} \quad (2a)$$

In the case where sensitivity is considered some arbitrary factor  $\beta$  times as important as precision (note  $\beta$  can be less than one), we have the generic  $F_\beta$ -score, Equation 2b

$$F_\beta = (1 + \beta^2) \frac{\text{precision} \times \text{sensitivity}}{(\beta^2 \times \text{precision}) + \text{sensitivity}} \quad (2b)$$

**ROC CURVE** *Receiver Operating Characteristic Curve* is a plot which shows how sensitivity and specificity are coupled by plotting sensitivity against (1 - specificity), [9].

**AUROC CURVE** *Area Under ROC curve*, is a numerical representation of the balance between specificity and sensitivity for a model, calculated as the area underneath the ROC plot. An ideal model would achieve sensitivity of 1 (or 100%), and specificity of 1(100%), so

the only points on the ROC plot would be at the coordinates  $(0,0)$ ,  $(0,1)$ ,  $(1,0)$  and  $(1,1)$ , giving an AUROC of 1. Comparatively higher AUROC indicates that a model has found more true positives per false negative.

### 3.3 CASE STUDY

To illustrate the meaning of accuracy metrics, consider a newly released game. Suppose that within six weeks of its initial release, 25,000 players register. Within that six weeks, 500 of those players spend money, while 5,000 play for at least ten days. That is, 2% spenders; 80% *churn* (i.e. leave), as in Table 3.1.

	Number of players
New sign-ups	25,000
Long-term players	5,000
Short-term players	20,000
Spenders	500
Non-Spenders	24,500

Table 3.1: Players Registered in the First Release of a Game's Cycle

Following an update, over the following six weeks 10,000 new players register. During this second cycle, 250 of those new sign-ups spend money, and 1,500 play for more than ten days. By blindly applying the above statistics, when a player signs up, they are deemed as "likely spenders" with probability 2%, and with 20% probability they are deemed "likely long-term players", resulting in the outlook of Table 3.2. Significantly, these are effectively random guesses.

	Flagged	Actual
Total New sign-ups		10,000
Long-term players	2,000	1,500
Short-term players	8,000	8,500
Spenders	200	250
Non-Spenders	9,800	9,750

Table 3.2: Players Registered in the Second Release of a Game's Cycle

Given the random nature of these predictions, we expect these predictions to have a very poor success rate. In effect we have a random sample of 200 users dubbed as likely spenders. The entire new sign-up group has a spending rate of 2.5% - meaning realistically roughly 5 of those "likely spenders" will turn out to be actual spenders. Likewise, 20% of the "likely long-term

players" will prove to be - meaning roughly 300 are correctly predicted. Taking these values as representative of the random predictions applied, we find the accuracy metrics (recall from §3.2) found in Table 3.5.

	True Positives	False Negatives	True Negatives	False Positives
Long-term	300	1,200	6,800	1,700
Spenders	5	245	9,555	195

Actual spenders/long-term
Non-spenders/short-term

Table 3.3: True/False Positives/Negatives from Random Prediction

	Correct	Incorrect
Long-term	7,100	2,900
Spenders	9,560	440

Table 3.4: Overall Results from Random Prediction

Metric (Equation)	Long-term Players (%)	Spenders (%)
Accuracy (1a)	71	95.6
Precision (1b)	15	2.5
Sensitivity (1c)	20	2
Specificity (1d)	80	98

Table 3.5: Metrics of Random Prediction

These results enable several notes of interest:

- High accuracy and specificity can be achieved with a very poor model
  - In cases where the negative case dominates, a model which simply calls nearly all cases as negative will be correct most of the time. This is known as *over-fitting*.
  - But that model will fail to identify a useful number of positive cases.
- Predicting spenders is a much more intricate task than predicting long term players
  - Even random prediction is much more likely to be successful for long term play.
  - In training real models, the relative shortage of spenders will lead to similar difficulty in predicting spenders.




### 3.3.1 Uplift


These base-line metrics from a random predictive model give us something to compare any subsequent model against. We have mentioned *uplift* as a means of comparison: the improvement observed between models. We will compute the uplift for each of the named metrics in relation to a hypothetical model.

Suppose we build a predictive model on the same game, which registers the results in Table 3.6, and therefore the metrics given in Table 3.7.

	True Positives	False Negatives	True Negatives	False Positives
Long-term	9000	600	7,650	850
Spenders	225	25	8,850	900



Actual spenders/long-term



Non-spenders/short-term

Table 3.6: True/False Positives/Negatives from Trained Prediction

Metric	Long-term Players (%)	Uplift	Spenders (%)	Uplift
Accuracy	86	15	91	-5
Precision	51	36	20	17.5
Sensitivity	60	40	90	88
Specificity	90	10	91	-8

Table 3.7: Metrics of Trained Prediction

By comparing Tables 3.5 and 3.7, and the values for uplift, we can draw important insights:

- Accuracy is not necessarily the most important metric
  - In the random case for spenders, accuracy was high but only five spenders were identified.
  - In the trained case, accuracy dropped by 5%, but 90% of spenders were identified.
- Precision, sensitivity and specificity may only be judged in proper context
  - 20% precision for spenders would seem poor, but this depends on the requirements of the user: they may prefer to identify as many actual spenders as possible, and allow for a large rate of false positives. In other cases, they may prefer to miss out

on a large number of actual spenders to ensure that those which are flagged have a very high chance of actually being so - that is, the model is very precise.

- Likewise, high specificity or sensitivity are not essential for a model to be successful, depending on the preference of the client.

It remains to define criteria by which we can definitively declare one model as being better than another, which we address in §3.4.

### 3.4 CHOOSING A METRIC

There are a number of options for what is most important in a model. Targeting individual metrics has drawbacks.

- **Accuracy:** Accuracy can be extremely misleading. e.g. a model which finds 9,000 true negatives and 1,000 false negatives will give 90% accuracy, despite not having found a single positive sample.
- **Sensitivity:** We can achieve high sensitivity by over-fitting to positive cases. The results will show a high rate of true positives but also a high rate of false positives.
- **Precision:** High precision can occur where a model is extremely selective. It predicts a low rate of positive cases, but a high proportion of those are true. The absolute number found, however, is relatively low.
- **Specificity:** Most negative samples will be identified by models which are overly likely to predict as negative. As a result, a high rate of false negatives also occurs.

Ultimately, the decision lies with the client whether they are willing to accept a high number of false positives so long as most true positives are found (high sensitivity), for example.

For our purposes, however, we opt to use only one metric when comparing different models. This is not exhaustive in that we will overlook some potential advantages/disadvantages between models, but we are justified in doing so due to the simplicity afforded by direct comparison of only one metric. We must therefore choose one value which reflects the aspects of predictive quality which are most important to us.

We assume the role of the client, and state that it is mostly important to find as many true positives as possible, i.e. high sensitivity. Nevertheless, we want the majority of positive predictions to be true, so we need to reach a basic level of precision also. Specificity is less directly important since the absolute number of true negatives (and therefore false positives) is less important than keeping the number of absolute false positives *lower* than the number of true positives, which is reflected by precision.

It follows that F-Score is the natural choice to evaluate our models. F-Score can be seen to measure the balance between precision and sensitivity, compared with AUROC which gives a measure of the coupling of sensitivity with specificity. In this report we specifically consider F1-Score, though it should be noted that results could be improved by optimising models using other values of  $\beta$ . We also note that all plots of F1-Score hereafter are scaled  $[0, 100]$ , rather than the usual  $[0, 1]$ . This is a consequence of calculating precision/sensitivity as percentages and submitting those directly into Equation 2a, rather than passing them as fractions of 1. This should be rectified in future development, but is accepted here provided we maintain consistency.

TP	FN	FP	Precision	Sensitivity	F1-Score
1000	0	0	100	100	$(\frac{2 \times 100 \times 100}{100 + 100}) = 100$
1000	0	1000	50	100	$(\frac{2 \times 50 \times 100}{50 + 100}) = 67$
500	500	500	50	50	$(\frac{2 \times 50 \times 50}{50 + 50}) = 50$
500	500	1000	33	50	$(\frac{2 \times 33 \times 50}{33 + 50}) = 37$

Table 3.8: F1-Scores Example

Table 3.8 gives an example where a model is looking for 1000 positives within test data. It demonstrates that higher F1-Scores clearly correspond to models which are more effective at both finding true positives, and finding more true positives per false positive. We can conclusively elect F1-Score as our primary metric for evaluating models for the remainder of this text.

## ALGORITHMS

---

There are a large number of algorithms available in the world of supervised machine learning, which we have elected to complete this project with in §3.1. We must nominate only a few, however, which may be applicable to our problem. We have seen in §2.3 that Vertica offer only linear and logistic regression at present, so it is sensible to restrict ourselves to these options, so that our end product is essentially implementable.

### 4.1 REGRESSION

*Regression* is a highly developed and widely available supervised machine learning algorithm, [10]. It relates numerical characteristics of a set of observations to a single feature of interest (common to those observations), by developing an equation which matches each characteristic with a multiplicative coefficient, and then summing them all together to find an output value which represents the inputs' similarity to the case that they exhibit the condition being investigated.

Regression generates an equation

$$Y \approx f(x_i) = \beta_0 + \sum_{i=1}^n \beta_i \cdot x_i \quad (3)$$

where  $Y$  is the feature under investigation, e.g. weight of an animal, and  $x_i$  are the *independent variables* expected to indicate the result, e.g. height, age, etc. This is done by hypothesizing a function to relate the independent variables to the dependent, or *target* variable ( $Y$ ). Initially, a tentative equation is postulated to represent the relation. The parameters  $\beta_i$  are then altered, and the improvement achieved from doing this is gauged by comparison between the generated  $Y$  and the known true value. This process is repeated iteratively across the number of training samples provided, until the improvement gotten from a refinement of the parameters is minute. At this point, the algorithm is said to have converged.

The method of finding this linear model dictates how  $\beta_i$  are varied, and how the resultant  $Y$  is interpreted. In particular, in linear regression  $Y$  simply represents the actual value of the sample. In logistic regression, however,  $Y$  is the *log-odds*, which may be manipulated to achieve the probability that the sample belongs to a certain class. A full explanation of the mathematical formulation of logistic regression is given in §4.3.1.

### 4.1.1 Null Hypothesis and P-value

In general, there are more available independent features than can or should be included in fitting models. In some cases, values are invalid for reasons related to the individual model, as we will see in §11.1. More generally, it is important only to include features which can be shown to be relevant.

In particular, we require that a feature can overcome the *Null Hypothesis* in predicting spend/churn. The null hypothesis is the claim that a feature has no bearing on an outcome, e.g. that age does not affect whether a player spends.

A standard procedure to statistically validate a feature for inclusion is *Pure Significance Testing*, [11]. This is done by finding the *p-value*, which roughly gives the probability that results gotten by using the independent feature were effectively random. It is standard practice to accept a feature to have defeated the null hypothesis if it returns a p-value less than 5%, [12]. We follow this well accepted guideline, and impose that features ought to be discarded if they fail to show p-values less than this threshold.

## 4.2 LINEAR REGRESSION

*Linear Regression* simply returns the numerical value achieved by the regression function when the set of input variables is passed for each instance. As such, this leads to a continuous spectrum of possible results. Use cases for linear regression range widely, though simple cases can be understood intuitively.

For example, consider a study of employee salary,  $S$ , where known variables for each employee are their rank,  $R$ , within the company (on a scale 1-10, for example), their highest level of qualification,  $Q$ , (e.g. level 8 for a Bachelor's degree), and which floor they sit on,  $F$ .

We might expect:

- Internal company rank correlates highly with salary, e.g. CEO (level 10) earns more than an intern (level 1).
- Education correlates well with salary, though not as highly with rank.
- The floor number has no bearing on seniority or salary.

We postulate a relationship:

$$S = \beta_0 + (\beta_1 \cdot \text{Rank}) + (\beta_2 \cdot \text{Qualification}) + (\beta_3 \cdot \text{Floor}) \quad (4)$$

Suppose we run a linear regression on training data and find:

- A very low p-value for rank, around 0.001
- A p-value of around 0.04 for qualification
- A p-value of 0.5 for floor number
- $\beta_0 = 20,000$ ;  $\beta_1 = 10,000$ ;  $\beta_2 = 1,000$ ;  $\beta_3 = -500$

This tells us that rank and qualification contribute to the success of the algorithm, since both have p-values within the stated threshold of 0.05. Floor number, however, has insufficient evidence to defeat the null hypothesis, i.e. we can not trust its inclusion to improve the model's ability to unearth the true relationship between the target and predictive features.

Now, consider two cases:

- An executive (rank 8), with a BSc (level 8), who sits on the eight floor, who earns \$100,000

$$- R = 8; \quad Q = 8; \quad F = 8$$

$$\Rightarrow S = 20,000 + (10,000 \cdot 8) + (1,000 \cdot 8) + (-500 \cdot 8)$$

$$\Rightarrow S = \$104,000 \quad (4\% \text{ over-estimated})$$

- A graduate salesperson (rank 2) with a MSc (level 9) who sits on the fourth floor, earning \$50,000

$$- R = 2; \quad Q = 9; \quad F = 4$$

$$\Rightarrow S = 20,000 + (10,000 \cdot 2) + (1,000 \cdot 9) + (-500 \cdot 4)$$

$$\Rightarrow S = \$47,000 \quad (6\% \text{ under-estimated})$$

Now, suppose we had built a model which did not consider floor number, we would find:

- Executive :  $S = 20,000 + (10,000 \cdot 8) + (1,000 \cdot 8)$

$$\Rightarrow S = \$108,000 \quad (8\% \text{ over-estimated})$$

- Graduate :  $S = 20,000 + (10,000 \cdot 2) + (1,000 \cdot 9)$

$$\Rightarrow S = \$49,000 \quad (2\% \text{ under-estimated})$$

Notice how the floor number impacted the model: when it was included, it caused the graduate's approximation to be far too low, but served to "correct" the executive's estimate somewhat. When it was removed, the executive's estimate weakened, though the graduate's improved. This is a typical signature of a feature which does not defeat the null hypothesis: it changes the outcome of the model effectively at random.

We may be inclined to include such features which *appear* to give us improvement in some places, but we must understand that any seeming improvement is based on random chance. Predictions based on chance defeat the very purpose of machine learning as a means to derive understanding and structure within data. This underlines the importance of including *only* features with acceptable p-value (i.e. those which disprove the null hypothesis). This also shows that validation of a feature's usefulness to a model through metrics such as p-value is an essential step of the feature selection process.

While an imperfect case, the point of linear regression is well expressed here: to get a general *approximation* for a numeric value, which belongs to a continuous spectrum, and which is believed to depend on a number of known factors.

## 4.3 LOGISTIC REGRESSION

*Logistic Regression* is a supervised machine learning algorithm which formulates the probability of an input set of characteristics as belonging to one of the known possible categories. For example, a set where ten numerical fields correspond to players which use either Android, iOS, or web-based platforms on which to play games. By training a logistic regression model on cases where information about the platform is known, we can then apply the model on instances where the platform is not known.

This is achieved by calculating the probability that the instance belongs to each individual category, and then choosing the category which returns the highest probability. Probabilities are found as outlined in §4.3.1.

### 4.3.1 Mathematical Formulation

As in linear regression, an equation is postulated to represent the relationship between a dependent variable, and a set of independent features, Equation 3. The equation returns values on a continuous spectrum, as in linear regression, though the meaning of  $Y$  here is different. In order to understand how to interpret  $Y$  we must first introduce a number of values.

**TRUE COUNT** Number of samples which do have the attribute under investigation.

**FALSE COUNT** Number of samples which do not have the attribute under investigation.

**TOTAL COUNT** Total number of samples considered.

**PROBABILITY,  $P$** , the ratio of the case under inspection being true to all cases

$$P = \frac{\text{True Count}}{\text{Total Count}} \quad (5a)$$

**ODDS RATIO,  $\alpha$** , the ratio of the case under inspection being true, against all forms of it being false

$$\alpha = \frac{\text{True Count}}{\text{False Count}} \quad (5b)$$

**LOG-ODDS,  $\Theta$** , Natural logarithm of the odds ratio.

$$\Theta = \log_e(\alpha) \quad (5c)$$

The values output as  $Y$  are the *log-odds*: the natural logarithm of the odds ratio of a case. Distinct from probability,  $P$ , which quantifies the chance of the case being investigated being true, the *odds ratio*,  $\Theta$ , is the ratio of the case being investigated being true, against all forms of it being false. This is an important distinction when there are more than two possible classifications.

For example, if there are ten balls in a hat, one blue, three red and six green, we would have the values shown in Table 4.1.

Colour	True Count	False Count	Probability	Odds Ratio	Log-odds
Blue	1	9	$\frac{1}{10} = 0.1$	$\frac{1}{9} = 0.11$	-2.2
Red	3	7	$\frac{3}{10} = 0.3$	$\frac{3}{7} = 0.43$	-0.84
Green	6	4	$\frac{6}{10} = 0.6$	$\frac{6}{4} = 1.5$	0.4

Table 4.1: Probability and Odds Ratio Example

Note that the probability is bounded by zero and one, while the odds ratio can exceed one but is always positive, and the log-odds is on a continuous spectrum.

$$P \in [0, 1]$$

$$\alpha \in [0, \infty)$$

$$\Theta \in \mathbb{R}$$

We can also relate the odds ratio to probability:

$$P = \frac{\alpha}{1 + \alpha} \quad (6a)$$



$$\alpha = \frac{P}{1 - P} \quad (6b)$$

Now recall the operation performed by regression: iteratively improving an equation which represents a relationship between features and a target variable, Equation 3. This implicitly demands that the relationship being modelled is linear. Logistic regression aims to distinguish between discrete cases, i.e. classify into categories. In training the model, samples will be passed with a vector  $X$  of features and a discrete value for the target variable. The model will fit the regression equation to the log odds of the system.

Recall our example of predicting salaries from §4.2. Suppose that instead of aiming to predict the salary precisely, we wanted to predict whether the employee had a high or low salary, where they are split by \$60,000. The model would generate an equation for log-odds,  $\Theta$  for the sample under scrutiny. We can determine the corresponding probability of that employee having a high salary as follows:

$$\Theta = \beta_0 + \beta_1 \cdot R + \beta_2 \cdot Q$$

$$\alpha = \exp(\Theta)$$

$$P = \frac{\alpha}{1 + \alpha}$$

Again, consider our two examples - a graduate and an executive. Suppose, when modelling high-earners, the model finds

$$\beta_0 = 0.033; \quad \beta_1 = 0.33; \quad \beta_2 = -0.2$$

Then,

- Executive:  $R = 8; \quad Q = 8$

$$\Rightarrow \Theta = 0.033 + (0.33 \cdot 8) + (-0.2 \cdot 8) = 1.1$$

$$\Rightarrow \alpha = \exp(1.1) = 3$$

$$\Rightarrow P_{exec} = \frac{3}{1 + 3}$$

$$\Rightarrow P_{exec} = 0.75$$

- Graduate:  $R = 2; \quad Q = 9$

$$\Rightarrow \Theta = 0.033 + (0.33 \cdot 2) + (-0.2 \cdot 9) = -1.1$$

$$\Rightarrow \alpha = \exp(-1.1) = 0.33$$

$$\begin{aligned}\Rightarrow P_{grad} &= \frac{0.33}{1 + 0.33} \\ \Rightarrow P_{grad} &= 0.25\end{aligned}$$

The model therefore predicts that, with 75% probability, the executive will be a high-earner. Likewise, with 25% probability the graduate will be a high-earner, and therefore with 75% probability will be a low-earner. These allow us simply to classify the executive and graduate as high and low earners respectively. Note if we were not predicting a binary outcome - say there  $n$  possible outcomes - we would have performed the same procedure for each possible category, and found  $n$  probabilities for the sample, and chosen the highest as the classification. For example, we would have generated one equation for high earners, one for middle-earners, and one for low-earners; the executive and graduate would be assigned probabilities for each of the three classes, and the highest likelihood is chosen as the classification output from the model. This demonstrates the core principles of calculating the log odds using the regression equation, and thereafter transforming this to represent probability of the sample belonging to the class examined.

#### 4.4 SUITABILITY OF ALGORITHMS

We are now in a position to decide which supervised method is most suitable for our needs.

Recall that we are trying to find whether players will be spenders/churners. These can be seen as classifications, and hence that our problems are suited to logistic regression.

We could attempt to run linear regression to predict the precise number of days a player will play for, or how much they will spend, but these have less immediately applicable consequences. We are not especially interested to know whether a player will leave after thirteen or fourteen days; it is more useful to state whether they will last for more or less than, say, ten days. This is because over specificity does not afford us any true actionable insight, whereas merely knowing they will play for long enough to run an entire campaign on is more useful. Likewise, given the relative shortage of spenders, it is most useful to find those who have attributes corresponding to spenders, and then trying to get them to spend as much as possible, rather than finding that they should, for instance, spend twice.

Overall it is most sensible to adopt logistic regression as our prime candidate from here on. Future development may aim to find greater specificity as outlined here, though we deem it sufficient to group players into classes instead.

## GOALS

---

With a firm grasp on the context of the project, we are now in a position to commit a set of goals.

### 5.1 OVERARCHING GOAL

We have one ultimate aim in pursuing this project: to establish whether or not predictive analysis holds the potential to bring useful, accurate services to the clients of deltaDNA. Based on our findings, we aim to offer suggestions as to whether it is worth further time and investment to develop the end product of this project to full scale implementation on their website in future. In the case that we can prove it worthwhile, the framework built here should serve as a helpful starting point for professional software development.

Clearly this goal is qualitative, and the output will only be a suggestion. To help us reach a conclusion on the matter, we formulate a list of *Goals* to build and evaluate predictive analysis models. These goals can be broken down into intermediate *Aims*, which must all be objectively measurable. Therefore each aim has associated *Metrics* by which to determine their success.

### 5.2 BUILD A MODEL

First we must build a predictive model on our data using a machine learning algorithm.

**Goal 1.** *Apply a machine learning algorithm to game player data.*

Proof of concept that a supervised method can be trained on the data. We have seen reasons to believe logistic regression is the most suitable candidate for application here, so we first aim to build a logistic regression model.

**Aim 1.1.** *Build a logistic regression model.*

Assemble a functional logistic regression prediction algorithm which operates on data taken from deltaDNA's database.

**Metric 1.1.1.** *Prove the model is working correctly.*

Show that the model behaves as expected, by making predictions based on a learned equation rather than effectively guessing.

**Aim 1.2.** *Build model on Vertica.*

Use inbuilt machine learning algorithm on Vertica to construct a functional logistic regression model acting directly on the database.

**Metric 1.2.1.** *Prove the model works correctly.*

Confirm that the logistic model performs properly and gives the expected output for a test case. Proof is a somewhat arbitrary concept here; we stipulate that a true predictive model would achieve significant improvement in accuracy compared with a random prediction. As we have decided that F1-Score is our primary metric, a model can be deemed to work correctly if it offers non-negligible uplift in F1-Score compared against a random predictive method.

**Metric 1.2.2.** *Prove that Vertica machine learning functions perform identically to R.*

We wish to verify that the machine learning functions added to the Vertica system achieve the same results as logistic regression models built using other languages/tools, namely R.

### 5.3 LEARN BY APPLYING PREDICTIVE MODELS

When we have such a model in place, we can use it to derive insight into the players/data being studied.

**Goal 2.** *Demonstrate the usefulness of predictive modelling, and use it to learn best values for model parameters.*

Vary model parameters to examine numerous aspects of the data and enable decisions for what should form the basis of a real-time model, such as feature sets and the distinction between long and short term players.

**Aim 2.1.** *Find a feature set which can successfully predict churn/spending.*

Test a number of subsets of the available player features and find which set gives the best F1-Score.

**Metric 2.1.1.** *Sets are considered successful if they achieve an uplift in F1-Score of at least 10 percentage points compared with random predictions.*

If we can not produce significant uplift compared with random prediction, we must conclude that the data is not well suited to predictive analysis.

**Metric 2.1.2.** *Highest F1-Score of all feature sets.*

Whichever feature set corresponds to the highest F1-Score observed will be taken as the *best known feature set* for that model type (churn/spend).

**Aim 2.2.** *Show impact of using categorical variables in addition to numerical.*

Run a number of models using a variation of numerical and categorical features to gauge the uplift (or lack thereof) of including categorical variables (since the Vertica models do not yet

include factorisation to encode categorical variables and therefore the vertica model is restricted to numerical features).

**Metric 2.2.1.** *Appreciable uplift in F1-Score by adding categorical variables*

Find the immediate impact of individual and groups of categorical features when added to the model. Non-negligible impact leads to the conclusion that restricting ourselves to numerical features, as required to run directly in Vertica, is detrimental to the potential value of predictive analysis.

**Aim 2.3.** *Find distinction between long and short term players.*

Vary the total number of days played at which a player is taken as a long-term user, and inspect how this parameter affects the model. Find which cutoff point (number of days played) gives the best predictions.

**Metric 2.3.1.** *High F1-Score, with consideration of how useful such a value is.*

We do not simply take the highest F1-Score obtained, but must balance F1-Score with usefulness. For instance, it is of little use to have a model which can successfully predict whether players remain for three days. It is more useful to have a less precise model which can find those players who play for over ten days. Therefore this parameter can only be chosen in context, as in §11.3.

## 5.4 EXTEND MODELLING TO ADDRESS OTHER QUESTIONS

When we have gained basic knowledge from building logistic regression models to find which parameters give best results, we can extend our view to more general applications.

**Goal 3.** *Extend the model to more general applications.*

Ensure that models can be equally effective when is used for other purposes, such as to make predictions on other games. Additionally, build models to investigate hypotheses which are believed to be true, but have no foundation currently.

**Aim 3.1.** *Test usefulness of weighting by positive cases.*

Investigate the effect of weighting on the model. In the case of isolating spenders, they occur at such a low rate, weighting the model to consider them more significant than non-spenders *may* help to identify more, though would then also lead to more false positives.

**Metric 3.1.1.** *Improved F1-Score compared with the case where weightings were not included.*

If uplift is observed for F1-Score, this can be recommended for implementation.

**Aim 3.2.** *Build a model which to test whether player bases evolve over time.*

Investigate whether the theory stated in §2.4 is credible.

**Metric 3.2.1.** *Observable, non-negligible uplift when training data is more recent to test data.*

If training on data more recent to the test players is shown to improve a model, our final recommendation can include that models should be updated periodically. If not, it suffices to build models once for each game and maintain that model indefinitely.

**Aim 3.3.** *Make predictions on other games.*

Using a common feature set, build models on a number of games to test that our model building process is appropriate.

**Metric 3.3.1.** *Achieve similar metrics for new games as were seen for the game which was initially dealt with.*

The process of building models can be deemed effective if it achieves acceptable accuracy metrics for a number of games, indicating that it is not overly fitted to one use case.

## 5.5 ASSESS MODELS

Additionally, when the model has been scrutinised with respect to correctness and accuracy, we also must consider factors which would influence its applicability and potential for implementation.

**Goal 4.** *Assess model with a view to its implementation.*

When the model achieves success on the fundamental grounds of functionality and accuracy, we can turn to assessing its performance in other aspects, and considering optimisations which may improve the overall quality of the end product.

**Aim 4.1.** *Measure performance.*

Examine the time taken to build and run predictive models.

**Metric 4.1.1.** *If the model can not be run on demand in reasonable time, it is ineligible for further development without optimisation.*

The concept of *reasonable* timing is a matter of context. If it is seen to take too long, we must weigh the time required to improve performance against the expected benefit. If it takes too long to do so, we might expect clients to be too impatient to wait, and thus the models will not be built or used.

**Aim 4.2.** *Investigate feasibility of Vertica only model*

Rather than pulling and uploading data for each query, it would be preferable to run the entire model directly on Vertica. We must investigate whether this is possible, and if so what limitations are incurred. We must determine if it is worthwhile to work only within the limitations of Vertica.

**Metric 4.2.1.** *F1-Score within ten percent negative uplift of the best model built on any platform.*

We expect a significant disparity between models built using different languages/tools. Due to the restrictions of Vertica, we initially suspect it to give poor results. If we can generate

results *almost* as good as exporting data and processing it elsewhere, the sacrifice in F1-Score is accounted for by an uplift in performance (time to build the model), and is accepted.

## 5.6 GENERALISING MODEL

Each of the goals above require individual models to be built in order that we can fulfil the intermediate aims. We notice that it is possible to extend our model building process into a more robust *framework*, where model parameters are easily changeable variables. In achieving such a framework, it would be relatively straightforward to generate new models for each aim listed, allowing us to analyse our project from a greater number of viewpoints than would have been possible by manually building new models each time.

**Goal 5.** *Generalise to a model testing framework.*

Extend our model to a framework where it is straightforward to build models to test different hypotheses. In order to achieve all the aims listed here, such a framework has a number of requirements.

- Goal 1
  - Build a new logistic regression model each time.
  - Have an option to build directly on Vertica using built-in machine learning functions.

The framework should then allow *variable inputs* for the following parameters.

- Goal 2
  - Feature selection
    - \* Should allow for categorical variables to be included.
  - Distinction between short and long term players
- Goal 3
  - Basis game to train on.
  - Training and test time frames.
  - Weightings of target variable.

**Aim 5.1.** *Enable straightforward variable switches and model generation.*

We aim to build a framework in which it is trivial to change parameters, and have a new model built using those, and have it evaluated automatically.

**Metric 5.1.1.** *All previous aims can be satisfied by using the framework to build a model easily.*

If we can straightforwardly build new models to investigate *all* of the aims of Goals 1 to 4, we deem our framework to be sufficiently generic and reusable. This corresponds to a high degree of user input/control, as was initially envisioned for this project.

## 5.7 DEVIATION FROM EARLIER PLAN

We note a deviation from the goals outlined in *Project Preparation*, and a corresponding shift in the method used to achieve those goals as compared with the plan laid out previously. This owes to a number of factors, but primarily is due to machine learning methods becoming available directly on Vertica, in effect allowing us to "skip the middle man", and focus on direct application to the data at hand, rather than extracting data and exploring it elsewhere.

Hence our project fundamentally shifted goals early on, and all subsequent development was therefore on a different path from the plan laid out in *Project Preparation*.



## Part II

### METHOD

## GENERAL APPROACH

---

In order to achieve the goals laid out in Chapter 5, we must adopt a strategy which encompasses planning, execution and evaluation.

An important note here is that, at all pivot points during development of a model such as this, the next step is determined by the success of the previous stage. For example, if logistic regression is seen not to offer any discernible improvement over random chance, we would alter our course of action rather than continue to build an effectively useless regression model. Thus, our method and analysis are somewhat intertwined, for the purpose of justifying why certain decisions were taken. This Part, however, will focus primarily on methodology, only briefly analysing whether an outcome is acceptable. Thorough analysis is reserved for full discussion in Part III.

### 6.1 MAPPING GOALS TO METHODOLOGY

We have set a number of aims in §5. Goal 5 effectively ties all of these into a single deliverable: a framework which can be easily modified to generate new models with variable parameters, and automatically return the results achieved by that model. Our steps were shaped by this end goal: the early stages of the project focused on building a single functional logistic regression model. When this was in place, we introduced some freedom into the model so that it could be reused, and ultimately moved to abstracting that model into a more robust framework which would allow simple switches to build distinct individual models. Later stages were then based on verifying that the framework performed as expected, and analysing the differences between models to understand how to get the most out of the data.

### 6.2 CRISP-DM

In a wider context, our project is not unusual in the world of data mining. To be confident that we are following a sensible path, we project our method onto the most typically used data mining infrastructure, the Cross Industry Standard Process for Data Mining, CRISP-DM, [13].

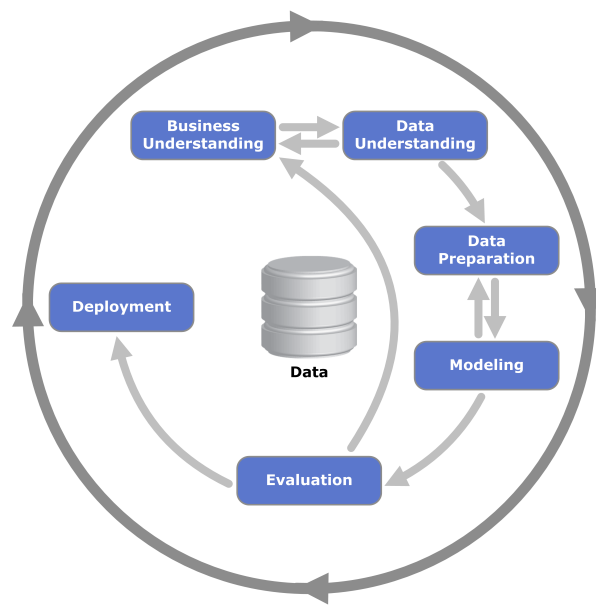


Figure 6.1: CRISP-DM Diagram\*

CRISP-DM is formulated as six stages, as in Figure 6.1. We relate our project to those headings.

1. Business Understanding: What do deltaDNA need to get out of this project?  
 ⇒ Their long term goal is to bring a predictive analytics service to their clients. The immediate goal we address is to prove that such a service can be useful. The two initial use cases of interest are whether newly signed up players are likely to spend in-game or remain as players for longer than a certain threshold.
2. Data Understanding: What data is available, and what is relevant for our project?  
 ⇒ We must first devote time to exploring the data and deciphering what could and could not be used in a real implementation on the deltaDNA platform.
3. Data Preparation: How must the data be formatted so that it can be used in our models?  
 ⇒ We must address inconsistencies and other problems in the data before using it for predictions.
4. Modelling: How can we apply machine learning to gain insight into the data?  
 ⇒ We will apply logistic regression to generate an equation to represent the probability a player exhibits the behaviour of interest.
5. Evaluation: How can we determine if our model is correct and successful?  
 ⇒ A model can be seen to have predictive power if it demonstrates appreciable uplift

---

\* Image from Wikipedia, [14]

compared to random prediction. Models are evaluated against each other in terms of relative uplift. We have chosen F1-Score as the most appropriate metric to represent our requirements for a model's success.

6. Deployment: How will the model(s) be used in the short and long term?  
 $\Rightarrow$  In the long term, this should form the basis for a predictive analytics toolkit to be introduced on the deltaDNA platform. In the short term, we will use it to learn about the data, and to find which parameters give optimum predictive ability, such as feature set and long term cutoff.

### 6.3 JUNCTIONS AND DECISIONS

Throughout the project, progression has depended on the success or failure of the most recent approach. This has carved the path followed, so here we state the steps taken, the problems arising, and the decisions taken to overcome those problems. Each of these stages correspond to a chapter in Part III, where complete descriptions of the issues and decisions are given. Here we merely state the high level decisions taken for brevity, so that the logic behind our methodology can be understood.

- Analyse data and possibility of modelling on static data using scikit-learn in Python, §7.
  - Required downloading and storing a large data set with the specific features required, leading to very low generality and reusability, and high overheads to generate a new model.
  - Examined the data from the perspective of what we ultimately aim to achieve §7.2.
    - \* Concluded the data set/features being used were insufficient and largely unsuitable.
    - \* Decided to build model in Vertica, where we can more easily join tables to access more data, and attempt to identify useful features there.
- Build logistic regression model via SQL queries on Vertica using in-built machine learning functions, §8.1.
  - The resultant accuracy metrics validate that this stage met the metrics of all aims of Goal 1.
  - Not dynamic: to change which features were used, much of the script had to be rewritten.

- \* Decided to move to a more flexible structure, §8.1.2.
- Prepare queries in R by pasting together dynamic inputs, then submit the queries to Vertica, §8.2 .
  - Obtained acceptable results.
  - Still somewhat intricate to change the model on other grounds than feature selection (e.g. which game is modelled upon). §8.2.1.
- \* Sensible next step was to generalise the model as far as possible.
- Develop the model into a framework which allows the user to choose the significant model parameters easily, §9.
  - This enabled straightforward and fast remodelling which facilitated testing new hypotheses quickly.
  - This was deemed a satisfactory end product since it directly satisfies Goal 5, and therefore allows us to iteratively build models to fulfil Goals 1 and 4.

## INITIAL DATA INSPECTION

---

### 7.1 PYTHON MODEL

The early stages of this project were devoted to gaining a thorough understanding of the data available to us. This was done under the premise of building any functional predictive model using Python. In our attempt to do so, we found places where the data was not fit for modelling, and therefore built some data preparation functionality. Ultimately, we realised that the static data we had been using was not appropriate to our goals, and elected to abandon Python in favour of acting directly on the Vertica database.

Our first attempt was to build a simple, functional logistic regression model; Python is a natural choice to begin with due to its simplicity and multitude of dedicated data science libraries. In particular, the library `scikit-learn`, [15], is generally believed to be a leading toolkit for analysis, [16].

In order that we can work on data using `scikit-learn`, the data must be locally stored and loaded into Python. We therefore must download and operate on a *static* data set. This causes a number of problems, since we now must choose between

- Downloading and storing all features from one table in case they may be useful.
  - Holding a large file and reading it into Python repeatedly, causing poor performance.
  - The file may be too large to download.
- Selecting a subset of features to retain a reasonably sized data set.
  - Can not check impact of all features easily.
- Selecting a subset of samples to retain a reasonably sized data set.
  - The subset may miss out on a significant portion of samples and therefore the model will not be fairly representative of the data set as a whole.
- Downloading a new data set for each model.
  - Cannot generically extend to other games since new data must be downloaded from Vertica each time.

The data held in our static set comes only from the `user_metrics` table. This was judged not to be ready for immediate analysis, e.g. due to the presence of non-numeric values such as `NaN`. Several functions were built to process the data: the data was passed through those functions in a pipeline, in a bid to prepare it for use in the logistic regression functions provided in `scikit-learn`.

### 7.1.1 Data cleaning

In order that our static data set could be modelled upon, it was necessary first to *clean* the data through a series of functions.

- Removing invalid values (e.g. `NAs`) from the data.
- Removing categorical features.
- Eliminating fields which have very low variance across all samples.

Functions were also generated to normalise the features available in the `user_metrics` table. These aimed to reduce total summary values, such as total number of missions played over a player's lifetime, to meaningful averages, such as average number of missions per session played. This was intended to overcome the bias of fitting a model using both old and new players.

Upon consideration of our model's aim, i.e. that it can make predictions on new players, it is apparent that operating only on this data set is insufficient. This is because the `user_metrics` data set had over three hundred features, but almost all of them relate to summarising a player's behaviour over their lifetime. Such information simply is not available in the first few days of a player's life span, and therefore such fields are ineligible for what we are trying to achieve. Even normalisation of total summaries will give misleading data which is not be admissible. Only data available within the first few days is valid; this data is contained in a table which stores player events per session, `fact_user_sessions_day`. We must alter our modelling to take only *valid* numerical features, as well unchanging categorical features from `user_metrics`, and join those with numerical and categorical features from the sessions table, from which all data generated in a players first few days can be used.

## 7.2 UNSUITABILITY OF STATIC DATA

Thus, even before modelling we can conclude that modelling on static data using Python is not appropriate here, and hence we decide to change at this early juncture, to model in Vertica instead so that tables can be joined as needed. Note that we are not concluding that Python,

and specifically scikit-learn - are not applicable whatsoever. Later in the project, we begin to model locally in R on data generated dynamically in Vertica, §9.1. This could have been done using Python instead; R was chosen for convenience. We have not applied machine learning techniques in Python fully enough to draw conclusions about its usefulness for our needs.



## LOGISTIC REGRESSION

---

Building a logistic regression model was approached in a number of stages. We begin by building an elementary model on the Vertica database. This leads to a hard-coded, cumbersome, but functional model.

Operating on data stored in Vertica can be achieved in a number of ways. The most direct method taken was to initiate a connection by providing valid credentials in a database tool, here DBeaver was used, [3]. From here we can post queries to the database we have connected to (a schema on Vertica), where SQL queries can be used to organise the data as we need it.

An alternative access method is to use a Vertica driver and send queries via the Java Database Connectivity (JDBC) API. This was successfully implemented in R using the `RJDBC` package, [4]. That is, queries are passed to Vertica from R scripts. This allows us to prepare queries independently and submit them from R. `RJDBC` can send queries specifically to update the database to which it is connected, or to return data from the queries passed. As such, this lets us write new tables as needed, and also pull data from Vertica into the local R session. This newly pulled data can be manipulated and modelled in R, for instance to be fit to a general linear model, [17]. This is performed in §9.1.

### 8.1 VERTICA SQL MODEL

Recall that to achieve Goals 2 (using models to learn about data and parameter optimisation), 3 (extending models to address other questions) and 4 (assessing our product), we aim to achieve to broader Goal 5, i.e. constructing a model-building framework.

The first step towards doing so is to build *any* functional logistic regression model, which can then be abstracted. We aim to simultaneously satisfy Aims 1.1 and 1.2 by assembling an initial model using Vertica's machine learning functions, which can obtain accuracy metrics above reasonably low expectations of appreciable uplift compared with random chance predictions.

#### 8.1.1 Vertica Machine Learning Functions

We have mentioned several times that Vertica offer machine learning functions which can be applied directly on the data held therein, [5].

These are invoked by calling on the functions as laid out in the documentation, [18]. We apply the logistic regression function to our data in Listing 8.1\*. (Note here we have deliberately taken a minimal example; the below model is not intended to produce reliable predictions using these features).

```
1 SELECT
2     v_ml.logisticReg(
3         'spender_log_reg_model',
4         'training_data',
5         'spender',
6         'fieldFacebookFriendsCount, number_of_events',
7         '--epsilon=0.00001 --max_iterations=100'
8     )
9 ;
```

Listing 8.1: Vertica Logistic Regression Functions

In Listing 8.1:

- `v_ml` is a schema on Vertica where the machine learning toolkit is enabled (since it is not preinstalled on all schemas).
- `spender_log_reg_model` is the name of the model we are building.
- `training_data` is the table which holds the data on which we want to train our model.
- `spender` is the target (dependent) variable.
- `fieldFacebookFriendsCount, number_of_events` are the independent variables, i.e. those which are used to predict the value of `spender`.
- `epsilon` is the tolerance threshold required to accept that the model has converged, i.e. when the improvement from the most recent iteration is less than this value, the regression equation is deemed to have converged.
- `max_iterations` is the maximum number of iterations which the model will attempt; at this number it will terminate refining the coefficients of the regression if it has not already converged (it will terminate upon convergence otherwise).

In order that we can construct such a model, we must prepare a single table, `training_data`, which holds the numerical fields, `fieldFacebookFriendsCount` and `number_of_events`.

---

\* Snippets included in this report are pseudo-code, and overlook some technical intricacies so therefore would not work independent of other functionality.

We also require a value for `spender`, which we have set to be a binary case: 1 for spending players, 0 for non-spenders.

A single sample (in this case sample is a player), should be held in one row so that corresponding dependent and independent features can be unambiguously linked. In most cases, as in our case, relevant data pertaining to a single instance (player) is held across numerous table in a relational database. Therefore the data must be unified into a single table first.

In the unprocessed database, there is no explicit field named `spender`.

`fieldFacebookFriendsCount` is held in the `user_metrics` table, which also holds a field for `totalRealCurrencySpent`.

`number_of_events` is held in the `factUserSessionsDay` table.

We therefore need to create a table based on a union of the two primary tables we are working with. Note that Listing 8.2 must be run first so that the training table is available for Listing 8.1.

```

1 CREATE training_data AS (
2     SELECT
3         u.user_id ,
4         u.fieldFacebookFriendsCount ,
5         SUM(f.number_of_events) ,
6         CASE
7             WHEN u.totalRealCurrencySpent > 0 THEN 1
8             ELSE 0
9         END AS spender
10    FROM
11        user_metrics AS u
12    LEFT JOIN
13        factUserSessionsDay as f
14    ON
15        u.user_id = f.user_id
16    GROUP BY
17        user_id , fieldFacebookFriendsCount
18 )

```

Listing 8.2: Creating `training_data`

Listings 8.1 to 8.2 suffice to build a logistic regression model. We can extract the summary of the model, i.e. the coefficients, p-value etc. found for each feature included, by following the protocol, [18], as in Listing 8.3.

```

1 SELECT

```

```

2      v_ml.summarylogisticReg(
3          USING PARAMETERS
4              model_name = 'spender_log_reg_model',
5              owner = 'owner_name'
6      )
7 ;

```

Listing 8.3: Summarising a logistic regression model built on Vertica

We can then apply the model for testing on other data:

```

1  SELECT
2      user_id ,
3      spender ,
4      v_ml.predictLogisticReg(
5          fieldFacebookFriendsCount ,
6          number_of_events
7          USING PARAMETERS
8              model_name = 'spender_log_reg_model',
9              owner='owner_name'
10     ) as predicted_spender
11 FROM
12     test_data
13 ;

```

Listing 8.4: Applying a logistic regression model in Vertica

The results of Listing 8.4 will resemble Table 8.1. Running the model on a test set, i.e. data where spender is also known, we can compare the spender column with the prediction column to determine whether the sample returns a TP, FP, TN or FN (recall accuracy metrics, §3.2).

User ID	Spender	Prediction	Classification
A00001	0	0	True Negative
A00002	0	1	False Positive
A00003	1	1	True Positie
⋮	⋮	⋮	⋮

Table 8.1: Sample Output from Vertica Logistic Regression

We can then count the occurrence of each of these classification types, and calculate the rates at which they occur, i.e. the accuracy, precision, sensitivity and specificity, and by extension, F1-Score, our primary means of assessing a model's quality.

This demonstrates how each new model will give a distinct set of accuracy metrics, and as discussed in §3.2, these can be used to evaluate the quality of the model.

### 8.1.2 Limitations

This preliminary model serves as a proof of concept, but ultimately the machine learning methods provided by Vertica are undermined by the rigid mechanism by which the model shown in Listing 8.1 is built. For instance, suppose we wanted to alter the model only slightly, to include one extra feature. This would involve changing how the unifying table is built, how the regression function is called, and how the prediction function is called. Optimising such models in terms of feature sets is clearly therefore a cumbersome task. It is a sensible progression, then, to prepare the queries *outside* the data base utility (DBeaver), in a more dynamic fashion. These generated queries can then be submitted to Vertica in such a manner as to build and evaluate a model based on new features with ease.

## 8.2 DYNAMIC QUERIES FOR FEATURE SELECTION

As mentioned, Vertica is accessed by submitting queries via clients such as RJDBC in R, and equivalents in Python and other languages. We opt to develop in R in keeping with the programming preferences of staff in deltaDNA.

In order to simplify the task of building a model using a new feature set, we wish to be able to declare a feature set and have queries generated which build a table, build a model on that table, and predict results.

The concept we may apply here is to simply paste the input features into the appropriate section of a query using R, and to then send that query to Vertica via RJDBC. Take the case in Listing 8.5:

```
1 features <- "fieldFacebookFriendsCount , number_of_events"
2
3 query <- paste("
4     SELECT
5         v_ml.logisticReg(
6             'spender_log_reg_model',
7             'training_data',
```

```

8          'spender' ,
9          " , features , "
10         '--epsilon=0.00001 --max_iterations=100'
11     )
12     ;" ,
13     sep="")
14
15 dbSendUpdate(connection , query)

```

Listing 8.5: Pasting input features into queries

By directly pasting the value held in `features`, we regenerate the query used in Listing 8.1, which is then passed using the `dbSendUpdate` functionality of the RJDBC API.

The same concept is used to prepare a query to build the `training_data` table in the first place.

Note one issue here: the value of `features` was set to be a string equal to the features of interest separated by commas. In more intricate areas of the model, this is insufficient. For example, when forming the `training_data` table, we need to reproduce the body of the query seen in Listing 8.2, so the features from each individual table (`user_metrics` and `factUserSessionsDay`) must be preceded by the table alias. Consider the substitute code of Listing 8.6

```

1 user_features <- c("fieldFacebookFriendsCount", "
   fieldUserLevelMin")
2
3 u_user_features <- ""
4
5 for (feature in user_features){
6     u_user_features <- paste(u_user_features , " 'u.", feature , "
   ' , " , sep="")
7 }

```

Listing 8.6: Replacement code to enable generic feature selection

This results in a value `u_user_features` which holds

```
('u.fieldFacebookFriendsCount', 'u.fieldUserLevelMin')
```

which can be used in a query to join the user features table with another. We apply this concept repeatedly when preparing queries to be used in constructing regression models.

The end goal here is that we can state a list of features for each table, which are then joined into a single table, and build a model using those features as the independent variables. The result of this dynamic feature selection is thus a set of values for precision, sensitivity etc. corresponding to each feature set. Then, we may form outlooks such as that seen in Table 8.2

Features Used	Sensitivity	Precision	F1-Score
fieldFacebookFriendsCount, number_of_events	60	70	65
number_of_events	55	65	60

Table 8.2: Example Accuracy Metrics by Feature Set

### 8.2.1 *Limitations and Next Steps*

This method, too, however is limited by what Vertica's functions can do. Namely, we are restricted to numerical features when building a model, since Vertica does not offer encoding categorical variables as a standard procedure. This means that, if we wish to use a feature such as gender in our predictions, we would have to manually encode it ourselves (recall one-hot encoding, §2.2.2). While this may be worthwhile for obvious fields such as gender, it bounds the generality of the script: we can not simply add a categorical feature to inspect its impact, as would be preferable.

An equivalent model in R would offer factorisation of categorical variables. Reworking our model to feed data into R instead of building directly on Vertica hence offers the benefit of including categorical features, but betrays one of our fundamental requirements, since we had imposed that the entire model ought to take place on Vertica for performance reasons. However, this gives rise to an important consideration: how does the Vertica toolkit compare to more advanced regression algorithms like those in R and Python? Moreover, having models in both Vertica and R allows us to contextualise the results achieved with respect to the data available, and what else is possible from using that data. If the inclusion of a few basic categorical features dramatically improves the accuracy metrics, we must then decide whether it is worthwhile to proceed using such sub-optimal methods as can be reached by Vertica.

Such a realisation would invite us to decide how to move forward. We may be able to identify a few significant features, such as gender or geographical region, and encode them by hand in every model. This action is unfavourable however as it involves a large overhead to develop new models. Alternatively, we may conclude that Vertica can not achieve the results we desire, and that we must find a way to pull the data from Vertica, and build the model elsewhere. Such a conclusion would have a drastic effect on the feasibility of implementation, which would then have to be reevaluated. Either way, it is a worthwhile diversion to develop twin models in Vertica and R so that we can determine whether the Vertica version is acceptable. Additionally,

this can be used to verify that the results of the Vertica model are reproducible elsewhere, confirming that the functions work as expected and are valid to use on our data.

This is one of the key reasons to direct the model towards a reusable framework rather than a hard coded optimal model. §9.1 details how the model is altered to build models in either Vertica or R by switching only one character.



## EXTENSION TO FRAMEWORK

---

So far we have developed a model which builds a logistic regression model on Vertica by preparing SQL queries in R and submitting those queries through RJDBC.

The success of separating feature selection from the model construction led to the observation that we could incorporate a degree of flexibility, so that the model can be reused under different conditions, and therefore be used to test a series of suppositions. This chapter thus aims to generalise the model we have built so far into a robust testing framework, as laid out in Goal 3. This is done by considering each aspect of the model to determine whether it would be useful to reserve as a pillar, i.e. to be included in all new models, or whether it should be altered to become a variable parameter which the user can easily change. The framework will then take the set of parameters provided by the user, build a logistic regression model on the specified platform (Vertica/R) for the specified target feature (spender/long-term player). It will automatically evaluate the model and produce summaries for accuracy metrics, including F1-Score.

### 9.1 GENERAL LINEAR MODEL

A major overhaul of the code is to change from building the regression model on Vertica, to extracting data from Vertica and building the model in R instead. Note here that other languages could have equally have been applied, though since we are already preparing queries in R it is sensible to keep the entire project in the same language.

R offers *General Linear Models* (GLM), [17]. These can be used to build a range of linear models, including logistic regression by nominating the binomial family type. The algorithm models the linear relationship between log odds of a set of characteristics; we have seen how this can be used to interpret the probability that an instance belongs to a given class, §4.3.1. A significant advantage of R is that we can vary the probability at which we predict a positive outcome, whereas in Vertica this is effectively fixed at 50%. By predicting as a likely spender a player who only returns 25% probability of being a spender, we expect to increase the number of true positives, but accept a higher number of false positives. This leads to a range of sensitivity and precision values, giving a value for F1-Score for each probability threshold. We can then plot F1-Score against probability threshold: the overall shape of this fit indicates the quality of the candidate model. This form of analysis is undertaken, for instance, in §11.1, to differentiate between which feature sets are most useful.

To run a GLM, we need to hold a complete data set locally which can be passed to the function. The data required is the same as that which is passed to `v_ml.logisticReg` in Listing 8.1. It must be prepared as in Listing 8.2, and then downloaded. From end-to-end then, this model will prepare queries in R, submit them to Vertica to build new tables, download the minimum needed information from Vertica, and fit a model in R. For this reason, we refer to this approach as *hybrid modelling*.

Another option provided by R which is not available in Vertica is assigning *weights* to each sample. By weighting a particular sample say twice as much as all others, it will be counted twice when fitting the data, marking it as more important to learn from than other cases. This is useful in cases where one class is heavily outweighed by others. In our case, the rate of expenditure is quite low - of order 1%. Then, it is likely that the actual spenders share many traits with non-spenders. In order to differentiate, a fit could focus on the positive cases, by assuming that a deviation in some observable which corresponds to a player spending is of more significance than an equivalent deviation corresponding to a non-spender. Thus, by counting the spenders' values twice (or more), the resultant linear model is more likely to pick up on such deviations. This hypothesis is tested in §12.1.

### 9.1.1 Running a General Linear Model

Suppose again we generate a training data table in Vertica using the procedure of Listing 8.2. We must query Vertica to pull this data to R, and then fit a GLM to it as in Listing 9.1

```

1 # Pull data from Vertica
2 r_training <- dbGetQuery(
3     connection ,
4     "SELECT * FROM training_data"
5 )
6
7 # Weight spenders by factor of k
8 weights_vector <- ifelse(
9     r_training['spender']==1,
10    k,
11    1)
12
13 # Build logistic regression model
14 my_logistic_regression_model <- glm(
15     spender ~
16     fieldFacebookFriendsCount + number_of_events ,
17     data=r_training ,

```

```

18         family=binomial(link="logit")
19         weights=weights_vector
20     )
21
22
23 # Apply model to training data
24 predictions <- predict(
25     my_logistic_regrssion_model,
26     newdata= r_test
27 )

```

Listing 9.1: Fitting a GLM in R

Again, in the test data we ought to know the true result for each player as well as the predictions, so we can evaluate the model using the standard accuracy metrics.

### 9.1.2 Factorisation

Now suppose our training data has another, non numeric field to represent the region the player is from, i.e. Europe, North America etc. We can include region as a predictive feature by encoding it (recall one hot encoding, §2.2.2), using factorisation in R, [19].

As with numerical features, any which we wish to include in a new model must be taken from the user metrics and sessions tables in Vertica. Since these must be processed differently, however, we compose our queries from four separate user-generated lists:

1. Numeric user metrics features
2. Categorical user metrics features
3. Numeric session features
4. Categorical session features

These four feature types can be varied in the `feature_selection.R` script, which feeds them into scripts designed to manipulate them until they are eligible for input into SQL queries.

### 9.1.3 Switching Between Model Types

Consider the pipeline of building a model:

1. User writes lists of features to use.
2. Feature lists get manipulated into queries to send to Vertica.
3. Those queries generate two tables in Vertica which have the exact same features - one to be used for training and one to be used to test the resultant model.
4. Pipeline branches depending on where the model will be built:
  - (a) If in Vertica:
    - i. Machine learning functions called directly on training table.
    - ii. Model evaluated on test data.
  - (b) If in R:
    - i. Download entire training and test tables into data frames.
    - ii. Declare weightings to be applied.
    - iii. Factorise categorical variables by adding columns to the data frames.
    - iv. Run GLM on training data.
    - v. Evaluate GLM on test data

Evidently both models share the early steps involved so our framework begins by calling the same functions regardless of which type is being used. We can introduce a single variable, `model_type`, to simplify choosing which model ought to be followed. Depending on what this value is set to, a conditional is instigated which calls the necessary R functions and interacts with Vertica as needed.

Recall also that we have two primary use cases for predictive analysis in deltaDNA: expenditure and churn. We therefore have four basic model types enabled:

1. Vertica Spender
2. Vertica Churn
3. Hybrid Spender
4. Hybrid Churn

## 9.2 VARIABLE PARAMETERS

A truly generic framework should allow us to test for all of the aims set out in Goals 1 to 4. Our generalisation therefore must make it straightforward to follow the following aspects of the model it builds.

1. Feature selection
2. Longterm cutoff day number
3. Basis game
4. Training/Testing time frames
5. Weighting for positive samples

These are all made into global parameters which the user need only change in one place in order to build a model with.

### 9.2.1 *Predictions Based on Early Behaviour*

In addition to making predictions on the available data fields, it is usually necessary to manipulate the data in some way so that it represents a useful interpretation. For instance, we have mentioned that one main source of data is a table which gathers information for each session played. As such, a single player will have multiple entries in that table.

Consider how this type of logic must be applied to our models. We wish to make predictions about what a player will *go on to do*. This type of prediction is most useful early in the player's life cycle; we would ideally be able to distinguish a spender from non-spenders upon sign-up, or within the first few days, so that the game developer can optimise revenue potential. This is equally useful in identifying those who do not exhibit behaviour like other spenders, since the developer can instead sell third party ads to be shown to that user, rather than displaying their own ads for game play purchases. This furthers our motivation to be able to make a confident prediction about a player at a very early stage.

With this in mind, we introduce new functionality to our logistic regression. As mentioned, actions are stored in summary of each session in the table `facts_user_sessions_day`. Each player has a record stored therein for each session they play. Those records contain details such as number of events made by the player during that session only.

Recall also that users have the power to choose which features from the sessions table are used to fit a model. We expect that a player's first day ought to be a strong indicator of their

future behaviour, as well as the cumulative numbers for their first  $n$  days; we add another variable parameter to represent this, `early_behaviour_cutoff`. That is, the user can define how many days at the start of a player's life cycle to take into account when trying to predict whether they will spend or churn. We can then test to find which `early_behaviour_cutoff` facilitates the best results. A concern here, however, is the time spent waiting before a predictive model can be relied upon: if a model uses the first three days of play to make its predictions, then the model cannot be applied until at least three days after the player has signed up. In this time we may have missed vital opportunities to benefit from that player. With this in mind, it may be more sensible to accept a sub-optimal model which uses a lower early cutoff but can be applied much sooner than superior models. We illustrate how an `early_behaviour_cutoff` can be chosen in §11.3.

### 9.2.2 Normalisation

Another often mandatory data cleaning step is that of normalisation: averaging numerical values by some sensible scale. For example, rather than including total number of events generated by a player in their lifetime, instead including the total number of events they generated per session. This eliminates the natural bias that would occur towards players who have simply played for longer or for more sessions, and would reveal actual, meaningful game play behaviour, for example how many levels they play for before terminating the session.

Normalisation is *not* required where we are taking regard of session features. This is because we are interested in the cumulative early behaviour of players: we do not take any values which span the total life time of players, so need not normalise in this respect.

## 9.3 FORMATTING RESULTS

We may now assume a functional logistic regression testing framework, and now consider how we can evaluate the models it produces. Throughout this discussion, we have concerned ourselves with the accuracy metrics described in §3.2. We must manipulate the predictions output by the model to count the instances of each outcome type (e.g. true positives). Obviously, this formatting depends on the type of model built.

### 9.3.1 Vertica Results

Applying a model to test data is done via Listing 8.4. We construct a number of tables, again held in Vertica:

- Predictions table: columns for true value and predicted value.
- Accuracy table: columns for the outcome types.
- Metrics table: columns for each of the accuracy metrics. This table has one row, holding all the accuracy metrics for the most recent model built.

Then all the results we need are held in the metrics table, so we export that from Vertica and analyse the results locally.

### 9.3.2 Hybrid Results

When the model is constructed in R, we have more freedom over the results we find. Recall that the GLM returns a value for the log-odds corresponding to the input sample, which we can manipulate to represent a probability between 0 and 1 that the sample is a positive case. We can easily construct a data frame which has a column for the true value (e.g. `spender = 1`), and a column for the corresponding probability. It is then straightforward to "accept" samples with variable probability. Having done so, we map each sample to a binary prediction. We can then compute whether that binary prediction is true/false and positive/negative, and count the occurrences of each. This results in a set of values for the accuracy metrics, corresponding to the probability threshold elected.

Now, we can change that probability threshold, and therefore achieve another set of accuracy metrics. Significantly (from a performance perspective), we need not regenerate models for each new probability threshold, since the new probability is calculated once, and it is the *acceptance* of that probability which changes each time. By calculating accuracy metrics for a range of probability thresholds between 0 and 100%, we have sufficient information to plot F1-Score against probability threshold, which we can use as a measure of improvement between models.

Models built in Vertica always give an acceptance threshold of 0.5, because Vertica effectively calculates the probability of the two cases, e.g. the case of spender, and the case of non-spender, separately. These two will sum to 100% probability, so only in the case that spender results in probability above 50% will it be taken as a likely spender; all others will be deemed non-spenders.

We can show that R and Vertica build the same regression equations. Then, we can carry out development and testing using GLM in R alone, mainly so that we can assess different feature sets in terms of F1-Score Vs. Probability Threshold plots, assured that by simply switching back to Vertica, we would see the same results as along the 50% probability threshold.

Additionally, this lets us find the optimum probability threshold per feature set, and therefore to determine whether the Vertica (unchangeable) default is naive; another crucial evaluation of the Vertica machine learning toolkit.



## Part III

# RESULTS AND ANALYSIS

BUILDING A MODEL

---

The most fundamental requirement we have is that we can build *functional predictive* models. Goal 1 is to apply machine learning to game player data, and is met by *proving* that the predictive models generated are effective. This requires us to declare criteria by which we can accept a model as predictive. A reasonable criterion to introduce is to claim that, if a model has demonstrable uplift over random chance, that the regression equation represents a true relationship between the target and independent variables. That is, we suppose that the accuracy achieved is a result of an underlying relationship, since otherwise it would not be expected to out-perform a random prediction.

## 10.1 VERTICA FUNCTIONS

We use our framework to assemble a model directly on Vertica using the in-built machine learning functions. Our purpose here is not to find the optimum model available, which is addressed hereafter, but rather to build any simple model as a proof of concept. We therefore use a single numerical feature, `number_of_events` - for players' first three days - to predict whether a new sign up will play for over ten days.

The overall rate of players going on to play for over ten days is 34%. A random prediction can be made which classifies a player as long-term with probability 34%. For comparison, we can also simply make a prediction with probability 50%, equivalent to flipping a coin to make a prediction about a player.

Table 10.1 shows the results of predictions made at the rate of occurrence (34%), a 50/50 guess, and the basic logistic regression model built using a single variable. Table 10.2 derives accuracy metrics based on these.

Model	True Positives	False Positives	False Negatives	True Negatives
Logistic Regression	2,668	982	3,442	10,397
Random at occurrence rate	2,069	3,864	4,045	7,522
Random at 50/50	3,051	5,698	3,063	5,688

Table 10.1: Results of Random and Generated Predictions

Model	Accuracy	Sensitivity	Precision	F1-Score
Logistic Regression	75	44	73	55
Random at occurrence rate	55	34	35	34
Random at 50/50	50	50	35	41

Table 10.2: Accuracy Metrics of Random and Generated Predictions

We observe that, while the absolute number of true positives uncovered by logistic regression is of the same order as the random predictions, the number of false positives is much lower, and true negatives much higher. The accuracy metrics show significant uplift compared with both random predictions for all categories. In particular, we have elected to take F1-Score as our primary measure of the effectiveness of a model. We can see that a random prediction yields an F1-Score of around 34. The single-feature model provides an uplift of 21 points compared to this random result. This is too high to be attributed to random chance; we infer that this uplift indicates that the model has at least some predictive power, and so has worked as expected.

In summary, we have built a logistic regression model using Vertica machine learning functions, and shown such a model to offer non-negligible uplift compared with random predictions, proving that the model works as expected. This satisfies Metrics 1.1.1 and, 1.2.1, so (by also satisfying Metric 1.2.2 in §11.1.2), we can consider all of Goal 1 to have been met.

## USING THE FRAMEWORK

---

Having established that the framework built can be used to generate functional predictive models, we can now turn our attention to applying that framework towards answering useful questions. This follows Goal 2, which was to "Demonstrate the usefulness of predictive modelling, and use it to learn best values for model parameters".

We perform a sample feature selection to find a set of features which can be used in later models as a constant parameter to inspect the affect of other changes. Another model parameter we aim to optimise is the definition of a long-term player, i.e. at what number of days does it most advantageous to classify a player as "long term".

A final parameter to find is the number of days at the start of a player's life cycle to be considered when building a model, i.e. at which day it is best to make a prediction about a player. Ideally, as soon as a player signs up we could predict whether they are likely spenders, however this would be based almost entirely on categorical variables such as region and gender. A more advanced prediction can be made when observations on the individual's playing behaviour can be taken into account. However, there is a balance to be reached between making accurate predictions and waiting too long so that only a few players remain, and the opportunity to generate revenue has passed its peak.

### 11.1 FEATURE SELECTION

Feature selection is a dominant concern in most machine learning projects such as ours. We use our framework to generate accuracy metrics for different feature sets with ease, so that comparisons can be drawn and we can nominate a set of features to use hereafter. We note that feature selection is not our principal goal in this project: we do not claim that the feature sets investigated are definitively the optimal combinations possible. Instead we take sensible subsets of the available features to test a series of hypotheses relating to which features may be useful and why. Thus, we are rigorous in our examination of the chosen subsets, but inexhaustive in compiling subsets to examine.

### 11.1.1 Eligible Features

We must first consider the data available in the context of what we're trying to achieve, so that we can determine which features are pertinent to our predictions, but more importantly, which are *eligible* for inclusion. By eligible, we mean that such features would be available in the same form when such models would be applied in real time. As discussed in §7.2, since we are dealing with historical data, many of the available fields correspond to *total* summations of a player's behaviour, e.g. total minutes played. Recall that our framework is wired to make predictions within a few days of the player signing up. As such, the only data that would be available to the model are unchanging features, such as region and device\*, and the session metrics from those early days, such as time played, events generated and missions completed.

This automatically eliminates the vast majority of features available from the user metrics table. That table holds numeric values such as level reached, but includes several values pertaining to the same feature: max, min, sum, first, last. Since these values accumulate over time, we must not include them in our models; their presence may be seen to have markable impact, but it would lend false credibility to the predictive power of our framework, since the maximum level reached by a player may be reached outside their first few days of play, for instance.

We need not list all the user features since there are so many. Some examples of ineligible features in that table are the level reached by a player, the missions played and the in game currency (e.g. gold coins) won. The only eligible features from the user metrics table are listed in Table 11.1.

We may, however, include all features recorded in the user sessions table. An example of a feature available in that table is the number of events generated in a session. As each of these have a relation to a date, and the player's signup date is recorded, we can unambiguously sum all the events generated in single sessions, over all the sessions played on the first day, and also sum them for all the sessions within the first  $n$  days.

We must also be mindful of including *signatures* of the target behaviour, i.e. a value which correlates highly with the target value for a known, trivial reason. For example, the number of devices used by the player is not excluded immediately in the same way total number of events would be. When we think about the device count, however, it is obvious that players who have played for longer are implicitly more likely to do so on a range of devices. An example of a non-trivial relationship, i.e. what we are trying to uncover, is the total number of events generated in a players first day being highly correlated with players who spend money eventually. Each numeric variable which does not directly represent a summary value, then, must be considered individually and removed if it is considered a signature.

---

\* Some features, like device type, may change during the player's life cycle, though we do not take account of this relatively infrequent case, and instead only consider the device the signed up on, for example.

After discarding ineligible features, we must now determine the *validity* of those remaining. Each eligible feature is entered as the sole predictor to fit models for spending and long term play. We then inspect the summaries of the resultant models: if the feature is found to have a p-value from which we cannot defeat the null hypothesis, we consider that feature invalid, §4.1.1. This process of elimination leaves us with only a small number of features, which are available early in a player's life cycle, and which have been shown to be related to the target variable in a nontrivial manner.

We can view our elimination process as a funnel, resulting in Table 11.1:

- Number of Available features
  - User Table: 280
  - Session Table: 30
- Eligible features
  - User Table: 6
  - Session Table: 11
- Valid
  - User Features: 4
  - Session Table: 6

		User Table	Sessions Table
Invalid	Numerical		Transactions Invites accepted Invites sent Missions aborted Social events
	Categorical	Operating System	Gender
Valid	Numerical	Facebook friends Videos watched	Number of events Time played Missions Started Missions completed Missions failed
	Categorical	Device Type Platform	Region

Table 11.1: Eligible Features

We now endeavour to find which combination of valid features can achieve the best predictive results. We do this by designing feature sets to test different hypotheses, in line with some of our goals. Table 11.2 states the lists of features used for modelling. Note the order of the sets is meaningless.

Set	Features	Test
1	Number of events; time played (seconds)	These are expected to be the most significant features available - a <i>basic numeric features</i> set.
2	Events; time played; missions started/completed/failed	Impact of including extraneous on top of basic numeric features.
3	Events; time played; region	Impact of adding geographical data on top of basic numeric features.
4	Events; time played; device type; platform	Impact of including categorical variables on top of basic numeric features.
5	All valid features	See if using all features helps or over-complicates the regression equation.
6	Events	Use a single, presumed-useful feature to check whether other features are actually detracting from predictive power.
7	Time played	Same test as set 6.
8	Events; time; region; platform; device type	Presumed useful numerical and categorical features, plus region to test direct impact of region on set 4.
9	Events; time; region; device type	Exclude feature (platform) which gives low p-value when used individually, but high p-value when included with other features.
10	Events; time played; missions started	Include only one feature related to mission count

Table 11.2: Feature Sets

### 11.1.2 Basic Numeric Features

Feature sets which use exclusively numerical values can be modelled directly on Vertica. We have seen why this is favourable for our purposes, §2.3. Feature sets 1, 2, 6 and 7 (defined in Table 11.2) are therefore our only potential Vertica-only models. Table 11.3 shows the F1-Scores reached by models built on Vertica using these sets. This is a preliminary screening, however, since the feature sets chosen to proceed with are elected later on, in §11.2. These values merely serve as a bench mark: since these can be achieved straightforwardly using only numerical

values by directly applying Vertica's machine learning tools, any feature set which achieves worse results can be automatically dismissed. Note that Vertica's logistic regression functions calculate the log odds (and therefore probability) of the sample testing as true, and therefore the predictive models built in Vertica are equivalent to those in R which take a probability threshold of 50%. Note also that the models used to find these values were trained using constant long term cutoff (8 days), early behaviour cutoff (3 days) and time frames, on data from one game only. Varying these parameters can be expected to alter the success of the model, though for the purposes of setting a bottom line, it suffices to hold them constant.

Feature Set	F1-Score Spender	F1-Score Churn
1	1	62
2	N/A	56
6	0.3	61
7	1.5	61

Table 11.3: F1 Scores for Purely Numeric Feature Sets

These values provide ballpark ideas of what is achievable for both predictive goals: predicting churn can be done at a much more successful rate than finding potential spenders. We can see from these that an F1-Score of order 5 would be considered a large improvement for a spender model, while we aim for an F1-Score of 65 or higher to be satisfied with a churn predictor.

Set 2 shows N/A F1-Score for the spender predictions, since it does not find any true positives, and other spender model's results have disappointing F1-Scores. We immediately flag these as poor sets to use, though reserve full classification till §11.2. The rigid probability threshold can be seen to be a major downfall of the Vertica machine learning suite: we will see in §11.2 that the same feature sets can obtain higher F1-Scores simply by tuning the model to accept different output log-odds from the logistic regression equation.

One positive observation, for churn models at least, from Table 11.3 is that the basic numeric feature set does achieve reasonably good results, confirming our belief that a player's sheer level of interaction in their early days can be used to predict future behaviour.

### 11.1.3 Impact of Categorical Features

As mentioned, R allows for factorisation of categorical variables, and therefore their inclusion when constructing predictive models. This is expected to result in notable uplift when compared with models built on purely numerical data. Feature set 3 adds region to the basic numeric feature set (which was proven as acceptably effective in 11.1.2), set 4 adds the player's device type and platform (e.g Android/iOS), and set 8 includes all of the above.



For fairness in comparison to Table 11.3, we quote the F1-Score of these feature sets when a probability threshold of 50% is used, in Table 11.4.

Feature Set	Spender	Uplift <sup>†</sup>	Churn	Uplift
3	3	2	61	-1
4	0.5	-0.5	68	6
8	1.5	0.5	64	3
9	2	1	62	1

Table 11.4: F1 Scores for Categorical and Basic Numeric Features

As we expected, the addition of categorical features has been almost universally beneficial. Again, we do not claim this analysis as absolute, and do not base our choice of optimal feature set on these results, but include them as a demonstration of how careful feature selection can enhance a model. We can also note a fundamental shortcoming of our project here: without explicitly testing each subset of valid features, and exploring the space of potentially valid features from other data sets available to deltaDNA, we may have overlooked the most suitable feature combination for our needs. We accept this limitation, however, since our project was not focused purely on feature selection, but rather focused on constructing a means to simplify feature selection and model modification, by building a framework.

In finding positive uplift in almost all of these simplistic test cases, we pronounce that categorical features give appreciable improvement compared with purely numeric feature sets, and thus we have met Metric 2.2.1, fulfilling Aim 2.2. This is pronounced further by inspecting the shape of the F1 Vs Probability Threshold for categorical sets compared with numerical sets. These plots are shown in §11.1.

## 11.2 CHOOSING FEATURE SETS

The usefulness of a feature set is relative to alternative feature sets in the context of the prediction being made. We use F1-Score to differentiate between possible sets. As seen in Tables 11.3 and 11.4, however, F1-Scores vary wildly with the target prediction. The probability threshold at which a sample is accepted also influences F1-Score; since we can easily tweak this parameter (in R), it is sensible to judge feature sets by the shape of the plot F1-Score against probability threshold, as proposed in §9.3.2.

<sup>†</sup> Note here that uplift is quoted with respect to the basic numeric feature set, set 1, in terms of absolute percentage points.

We wish to build two predictive models, one for a player's likelihood to spend, and one for their likelihood to churn (leave), and we have ten example feature sets which can be used to generate these predictions, Table 11.2.

### 11.2.1 *Spender Model*

The F1-Scores achieved by each feature set at each probability threshold are shown in Figure 11.1. Four sets are seen as superior to the others: sets 3, 5, 8 and 9. An important observation here is that all models have trailed to well below their peak at the probability threshold 50% (the vertical black line), which corresponds to the classical logistic regression classification, as offered by Vertica.

One result of Figure 11.1 (and Figure 11.2) is that, for feature sets 3, 4, 8 and 9, the F1-Score found at 50% probability threshold is the same as it was found to be in Vertica, Table 11.4. This allows us to definitively conclude that Vertica and R construct identical models when using the same data, i.e. we fulfil Metric 1.2.2.

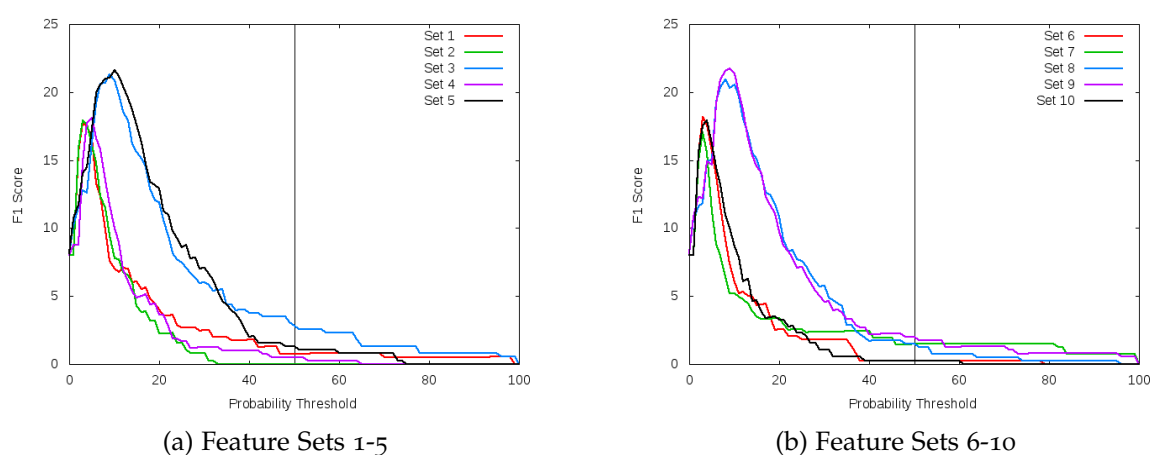


Figure 11.1: F1-Score Vs Probability Threshold for Spender Models

The F1-Scores peak at 22, in feature set 9 with a probability threshold of 10%, which comes from a sensitivity of 33% (one in three actual spenders were detected); and a precision of 16% (one in six predicted spenders were actually spenders). In this set, 4% of samples are spenders. A random prediction gives sensitivity of 2% and precision of 3%. Clearly, then, even though these values appear rather low, in the context of trying to isolate spenders, these sets - coupled with careful choice of probability threshold - offer significant uplift against the most basic alternative of predicting at the rate of occurrence.

### 11.2.2 Churn Model

Likewise for churn models, F1-Score is plotted against probability threshold for each feature set in Figure 11.2.

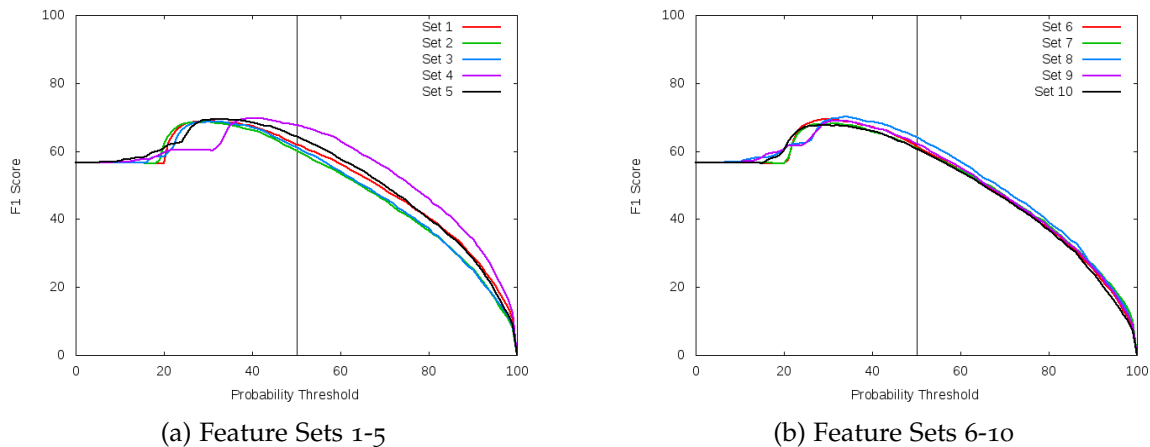


Figure 11.2: F1-Score Vs Probability Threshold for Churn Models

All sets trace a similar pattern apart from set 4: above the 40% threshold set 4 shows the highest F1-Scores, but beforehand it lags below the other sets. The peak reached by 4 is in line with those of other sets however, so we are cautious that the visual may be misleading; it may be that feature set 4 reaches many of the same metrics as all others, just slightly later along the probability threshold spectrum. We therefore do not declare it as the best available for future work set without further evidence, but nominate set 4 for further inspection, performed in §11.2.3. Additionally, set 8 is visible apart from the other feature sets here.

Again we note that the highest F-Scores are recorded away from the 50% probability threshold, so declare our conclusion that, while Vertica does not allow manipulation of the log-odds to be accepted, the suite of machine learning functions offered are of little benefit to deltaDNA.

### 11.2.3 Selecting a Feature Set

In deciding a feature set for further use there are other considerations which are reflected as parameters in our framework. The framework was explicitly designed to allow the definition of a long-term player to change, so we can perform further inspection of the feature sets, with respect to the churn model only. We do so by plotting the *maximum* F1-Score obtained by each feature set, against the number of days taken as long-term cutoff, Figure 11.3. For visibility,

we plot only a subset of seemingly stronger feature combinations: set 4 since it showed most prominently for sets 1-5 for the churn model; set 8 since it was most prominent from sets 6-10, and sets 5 and 6 since they too are more visible than the remaining sets, though by a smaller margin, in Figure 11.2.

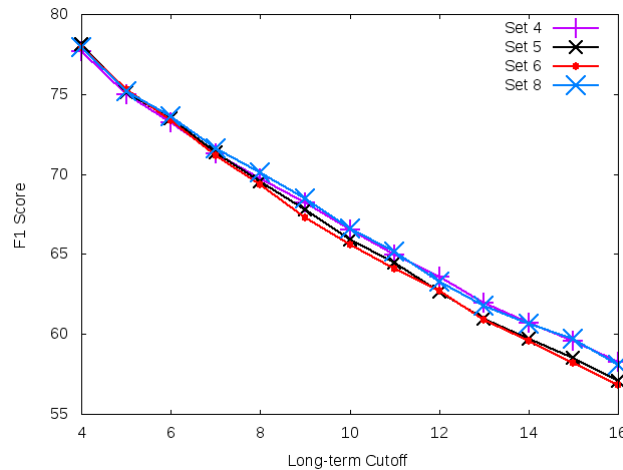


Figure 11.3: Maximum Observed F1-Score from Feature Set Vs Long-term cutoff

Figure 11.3 shows a discrepancy between sets 4 and 8 from sets 5 and 6 (note also that all other feature sets show very similar behaviour to sets 5 and 6 but are omitted for visibility here). We can discard all feature sets apart from 4 and 8 based on the higher F1-Scores attainable when using one of these for long term cutoffs above 7.

We are now left with a choice of two feature sets. Set 4 uses the number of events generated in the first three days of play, the total number of seconds in the first three days of play, as well as the device and platform the player uses. Set 8 uses *the same* features, with the addition of what region they are playing in. These seem like reasonably intuitive feature sets for predicting player behaviour: a player who plays a lot during their early days, i.e. spends a lot of time and/or generates a lot of events, is intuitively likely to be a more serious player, and therefore play for longer. The device and platform on which they play tells us a lot about their demographic, and potentially their propensity to spend money in-game. For example, we can infer that, since iOS devices are owned more densely in developed countries, that the rate of expenditure on iOS should be higher than on low end Android devices. Moreover, the region of a player directly provides further demographic information to a model, though this field may suffer from over-generalisation: knowing a user's country may be a stronger indicator of their economic background than knowing their continent (which region gives).

### 11.3 LONG-TERM AND EARLY BEHAVIOUR CUTOFF POINTS

It remains to determine values to distinguish between long and short term players, and the definition of early behaviour used when training models. There are a number of considerations to take into account when deciding these.

The definition of "early" behaviour influences both spender and churn models. It effectively means how long a game developer would have to wait before they can make predictions on new players. Clearly a prediction is of more value earlier in a player's lifespan. In the case of a spender model, the earlier a player is flagged as a likely spender, the client can alter their treatment of that player, e.g. instead of presenting third party ads, they can show game-related ads such as offers to buy in-game currency, which is of more direct value. For this reason, it was decided not to consider using early cutoffs of over five days, since this would be too great of an overhead for clients to have to wait, and the majority of their players would have left by then, meaning that a highly accurate model is mostly just finding true negatives.

#### 11.3.1 Spender Models

Figure 11.4 shows the maximum F1-Score achieved using feature set 8 to predict spenders by varying the early behaviour cutoff point.

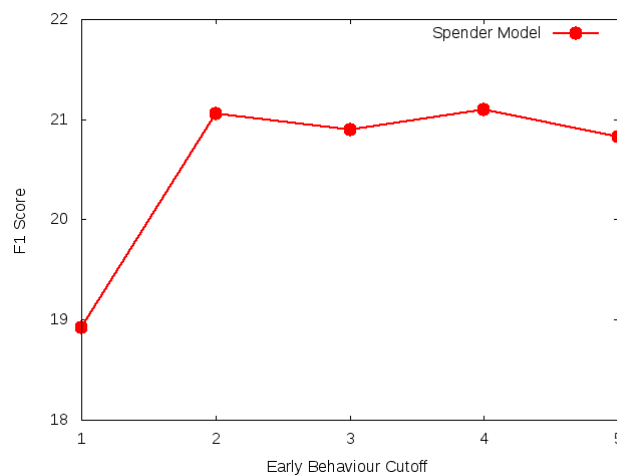


Figure 11.4: F1-Score Vs Early Behaviour Cutoff Day

The largest improvement gained by waiting one extra day before making predictions comes from switching to day 2 as the cutoff. Our choice is straightforward: an early cutoff of two should be used in later models for spenders.

### 11.3.2 Churn Models

Likewise, knowing a player is likely to play long-term allows the client to optimise their dealings with that player. There is a caveat in this case however, that the usefulness of a churn predictive model is limited by the combination of early and long term cutoffs. That is, if a model is highly successful in predicting which players will play for longer than four days, but uses the player data from the first three days to do so, it is not very useful. It would be preferable for a model to be able to predict at an early stage whether the player remains after a relatively high number of days. Thus we aim to maximise the time between early and long cutoff, so as to get the most value out of the model. Figure 11.5 shows the maximum F1-Score reached using each value for early cutoff, and is organised by the longterm cutoff used. This lets us see the benefit of waiting another day to make the predictions, as well as the benefit of lengthening our definition of a long term player.

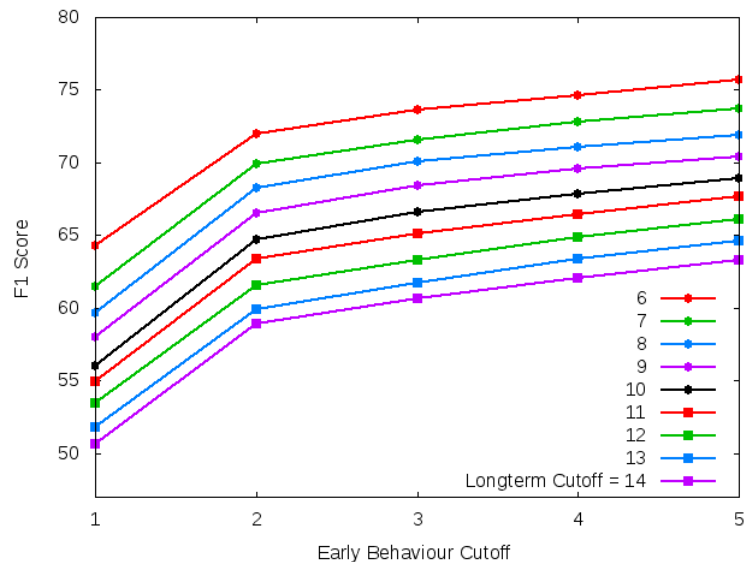


Figure 11.5: F1-Score Vs Early Behaviour Cutoff By Long-term cutoff

As in the spender model, the greatest benefit - from a single day increase in early behaviour cutoff - is from one to two days, for all values of long term cutoff. As we would expect, the models monotonically improve with early behaviour cutoff value. This is due to closing the gap between early and long term cutoffs. Like the spender model, though, it is obvious that the best choice for early cutoff is two days. If the differences had been more subtle between early cutoffs, a more rigorous method to decide on a value would be to compare the slopes from one early cutoff to the next; while the slope gets larger for each successive cutoff, the model is improving. When it begins to decrease, we take the previous value. Again we would exercise

caution, however, to ensure that the model does not incur too long of a startup period, so that it is as helpful to broader aims as well as purely raising F1-Score.

Selecting a long term cutoff to proceed with is more intricate. We must balance effectiveness of a model, gauged by F1-Score, with usefulness, more vaguely related to whether it is helpful to be able to predict whether a player will remain for over  $n$  days. Ultimately, this decision forms a major pillar of the user input, since the division will change game-to-game, so the parameter is easily changed in the framework. Additionally, the framework allows the user to choose a range of long term cutoffs; we can envision an automated process where plots such as Figures 11.3 and 11.5 are shown to the user and they can then opt for one long term cutoff.

For testing purposes, we ought to accept one long term cutoff. Since we have chosen to take an early cutoff of 2, we can consider the *loss* in F1-Score suffered by extending the longterm range, Table 11.5. Note that we take a lower bound for long term as six days, and an upper bound of fourteen days. This is because there is relatively little value in predicting that a player will remain for five or less days, and the vast majority of players have already left after fourteen, so even though accuracy may seem to improve, it is usually due to overfitting to negative cases.

Long-term shift (days)	Absolute Loss	Percentage Loss
6 to 7	2.0	2.8
7 to 8	1.7	2.5
8 to 9	1.7	2.5
9 to 10	1.8	2.7
10 to 11	1.4	2.1
11 to 12	1.8	2.8
12 to 13	1.6	2.6
13 to 14	1.0	1.7

Table 11.5: Losses of extending longterm cutoff by one day

Table 11.5 confirms what we may infer from visually inspecting Figure 11.5: the largest gaps (for an early cutoff of 2) occur between using longterm cutoffs of 6 to 7; 9 to 10 and 11 to 12. As discussed, we are weary of setting the long term cutoff too early, since it is less useful to do so, or too late, since the absolute number of positive cases available is too low. We opt to take 9 as our long term value: this is sufficiently high that it is meaningful and useful to distinguish between players who leave before or stay until then, and still achieves a relatively high F1-Score, compared with those of higher long term cutoffs. We reiterate that this is for illustrative purposes, and this decision ultimately lies with the game owner, since they will have a stronger understanding of what's important in their predictions.

## 11.4 CHOSEN PARAMETERS SUMMARY

In summary of this chapter, we have chosen model parameters to hold constant while performing other tests. Those parameters are:

- **Feature Set:** We performed a demonstrative, incomplete feature selection by building models with ten subsets of the available, valid data, Table 11.2. We found that different subsets gave the best results for the spender and churn models, but decided to proceed using a set which simultaneously achieves high results for both model types. Set 8, consisting of the number of events, time played, device type, platform and region of a player, is chosen as a basis set for future tests.
- **Early Behaviour Cutoff Point:** The models are based on how a new player behaves in their first few days, so we found how many days allowed for the greatest return. Slightly higher F1-Scores for later cutoff points were shunned since the benefit of the resultant stronger models is expected to be outweighed by the cost of having to wait longer to be able to make predictions. A cutoff of two days was chosen, which lets the client make predictions early in a new player's lifespan, but has enough knowledge of the individual's habits to make more reliable predictions than would be possible at the point of sign up.
- **Long Term Cutoff Point:** We aimed to find a number of days to use as the distinction between long and short term players which was not so low that it is of little impact (e.g. finding that players will play for over four days is less useful than finding they will play for over ten). We also require the cutoff to be low enough that a considerable portion of the players do actually play to that point, such that the absolute number of positive cases identified is significant. We chose nine days to balance these requirements.

This chapter focused on using our framework to learn about the data, as per Goal 2. §11.1 tested numerous feature sets, most of which exhibited uplift in F1-Score versus random chance, satisfying Metric 2.1.1. Set 8 was found to give one of the highest F1-Scores overall, to fulfil Metric 2.1.2, so Aim 2.1 is completed.

§11.1.3 shows that categorical features improve predictive ability, meeting Metric 2.2.1 and thereby completing Aim 2.2.

A value for long term cutoff is chosen with due consideration of how useful that value would be for real clients. Further, a value for `early_behaviour_cutoff` is determined with similar considerations. We claim that §11.3 can therefore be seen to settle Metric 2.3.1, and by extension, Aim 2.3.



## EXTEND THE MODEL

---

With a framework in place which has been proven useful for optimising model parameters, we can turn to Goal 3, which aims to use that framework to address more general concepts.

In particular, we use it to test two theories about the data. Firstly, we have discussed that weighting positive samples could be useful when training models on training sets where positive cases are relatively sparse. We investigate whether this is a valid supposition in §12.1. Secondly, it is believed that player bases change over time, and therefore models ought to be updated to reflect that. We test whether this is true in §12.2.

Finally, the overall value of our framework is assessed by testing how it performs when applied to games other than Game A, which has been solely used for all analysis to date. §12.3 seeks to determine whether the framework can accordingly be proposed as a basis for online implementation within the deltaDNA platform, or if it is too specialised to our test case.

### 12.1 WEIGHTING SAMPLES

One proposal offered to overcome the natural bias inherent to our data due to the large imbalance of negative to positive cases is to weight the positive samples when fitting a model, introduced in §9.1. The premise here is that by taking, for example an actual spender, as being some arbitrary factor  $k$  times more significant to the model than non-spenders, that the resultant equation would be more likely to find a higher number of true positives, with the accepted collateral damage of also finding many more false positives.

In all other models, this weighting has not been present, i.e.  $k = 1$ . Again, we consider the F1-Score across the spectrum of probability thresholds as a representation of how good a model is. Figure 12.1 shows the results for weighting positive cases by factors 2, 5 and 10. These models are generated using feature set 8, on Game A, which was used throughout Chapter 11.

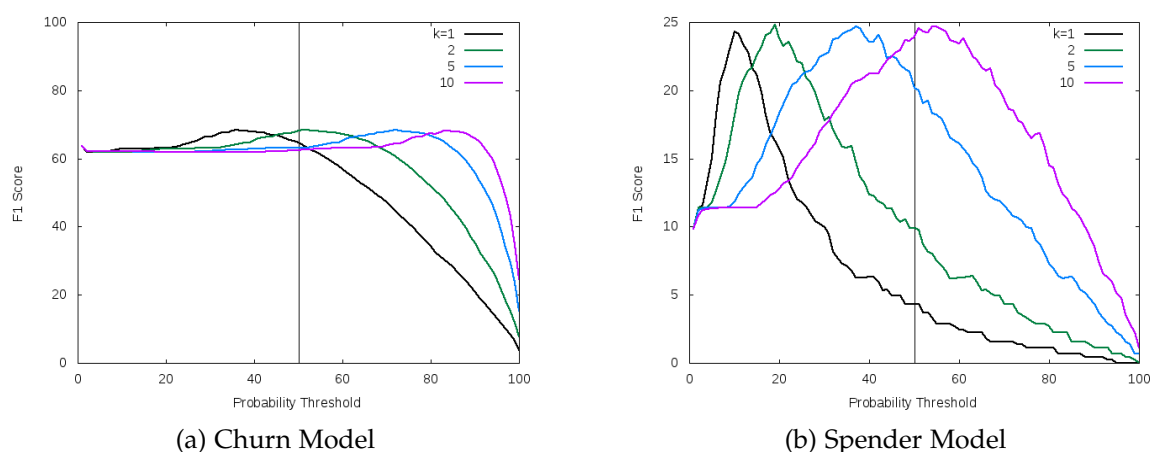


Figure 12.1: F1-Score Vs Probability Threshold for Varying Positive Weights

The effect of weighting is apparent from Figure 12.1: models using different weights find the same number of true/false positives, but at different probability thresholds. This is not surprising: weighting positive cases is equivalent to over-sampling, to have the effect that a model is more likely to predict a sample as a positive than is likely in reality. This is the exact same effect as accepting samples with lower probability. Thus, where the model is trained by over sampling, or weighting, *all* samples are given a higher probability than a true reflection of reality. Table 12.1 shows results surrounding the peaks of each model visible in Figure 12.1. We show the points for each model which achieve almost the same number of true positives. We can gauge the success of a model by how many false positives are required to find a certain number of true positives; consider the number of false positives as the cost of finding a desired level of sensitivity.

k (weighting)	Probability Threshold	True Positives	False Negative	F1-Score
1	9	371	1504	24.33
	10	329	1205	24.14
2	15	374	1978	23.31
	17	325	1489	24.21
5	31	383	1968	23.77
	35	326	1470	24.45
10	48	376	1936	23.63
	53	325	1439	24.67

Table 12.1: Results of Weighting Positive Spending Samples

Table 12.1 shows that weighting drastically increases the number of false positives found per true positive; weighting positive and negatives equally ( $k = 1$ ) in fact gives the lowest number of false positives for peak F1-Score.

We can conclude that weighting is pointless when constructing models in R, since the same results can be achieved by dropping the probability threshold, which also results in fewer false positives.

We note, however, that weighting *would* be of benefit for models built in Vertica: Figure 12.1 shows that weighting can be used to shift the peak of the F1-Score plot towards the 50% probability threshold which is used as default in Vertica. This would require more rigorous optimisation of  $k$ . However, since the in-built machine learning functions in Vertica do not allow the user to simply weight samples, this would have to be hard-coded into the model building architecture, which is a non-trivial reformulation of the current framework. Alternatively, in later releases of Vertica, we may expect improvements to the functions offered currently, such as to set the probability threshold, or to enable weighting/over-sampling. Such additions would increase the potential real-life application of predictive analysis directly on Vertica for deltaDNA.

Metric 3.1.1 is hereby not met: we have not found marked uplift by weighting training samples. We declare, as the result of Aim 3.1, that weighting is an ineffective aspect of modelling, in the context of our project.

## 12.2 TIME BASED MODEL

We have mentioned, §2.4, that gaming data is suspected of changing over time: that the player base which sign up during one release are implicitly different to those of later releases. For this reason, the framework was designed to allow the user to choose the training and testing time frames.

To test this hypothesis, we generate a number of models, maintaining the feature set, early and long cutoff points, and target game as constant. Again, we measure the effectiveness of a model by the maximum F1-Score it achieves.

We define three time periods, T1, T2 and T3 as in the timeline shown in Table 12.2.

2015				2016	
Jan - Mar	Apr - Jun	Jul - Sep	Oct - Dec	Jan-Mar	Apr - Jun
	T1			T2	T3

Table 12.2: Timeline of Training/Test Timeframes

Using these, we train three models:

- T<sub>3</sub>/T<sub>3</sub>: Train model in same period as test data. Players here should not differ, according to our hypothesis, so this should give the best results.
- T<sub>2</sub>/T<sub>3</sub>: Train model on a player base from a three month time frame, and test it on the immediately subsequent three month player base. This would be the most logistically sensible model type to build in reality, rather than building new models all the time, update them every few months (e.g. after every update), provided the model can perform well enough over that time span.
- T<sub>1</sub>/T<sub>3</sub>: Test a model on players from a year later than those it trained on. If this model shows a non-negligible deterioration compared with the others, we can attribute the result to the effect of player base evolving over time, and validate the hypothesis.

Table 12.3 lists the maximum F1-Scores for testing models trained on different time frames.

Training Time frame	Test Time frame	Spender F1-Score	Churn F1-Score
T <sub>1</sub>	T <sub>3</sub>	13.33	58.66
T <sub>2</sub>	T <sub>3</sub>	21.14	65.76
T <sub>3</sub>	T <sub>3</sub>	21.78	65.96

Table 12.3: F1-Scores for Time based models

There seems to be some credibility to the claim that the player base has changed over time: both the spender and churn models exhibit a drop in reliability when applied to players who join some time later, compared with models which are trained and tested on the same or closer time periods. It is possible that this effect has other basis, but the uplift gained from trailing the user base by only three months, rather than a whole year, is high enough to justify our belief.

A key conclusion we can draw is that predictive models cannot be simply built once and reused over the lifetime of a game, but must instead be regenerated periodically so as to ensure the model fit is as close to the actual core player base as possible as it evolves over time.

### 12.3 OTHER GAMES

One of the main tests we can perform to validate our framework is to build models on data from other games. Recall that the framework was developed entirely on one game's data, so there is a risk that the results seen so far are not generalisable, and therefore of little practical use for deltaDNA. If we can prove that acceptable results are obtainable for any game, then

our framework stands a much higher chance of forming the basis of an implementation of predictive analysis on the deltaDNA platform.

Some very important considerations are the availability, consistency, reliability and usefulness of features across different games. We cannot expect that the same features which garner satisfactory results for one game can be blindly used on another and achieve success. Each game requires individual attention to find its most effective feature set. Furthermore, all games will have differing rates of expenditure and long-term play, so the choice of long term cutoff, and indeed *all* model parameters, must be newly devised for each game.

It may be possible, however, to find a number of elementary features, common to all games, which have definite predictive power. For example, we might expect that the number of events generated in the first two days is roughly as useful on most games.

With this in mind, we must manage expectations for what may be considered successful for other games. We do not perform meticulous feature selection for our new games, and since the platform and device type are not available for all games, we accept that the results seen here do not necessarily reflect the best possible predictive analysis on all games. Feature set 1, from Table 11.2, is used since it is the simplest high performing set which is known to be available for all games without individual inspection of their user metrics tables.

### 12.3.1 *Data Quality*

Upon examination of data from other games, we can make several general observations.

Compared with Game A (the game used throughout development), the absolute number of positive samples, i.e. spenders and long-term players, is considerably lower. Game A has the highest rate of spenders/long term play seen in any game which currently employs the deltaDNA services, so models built on its data have a much higher chance of success. We can attempt to emulate a higher number of positive samples to train on by weighting the real positives, though as seen in §12.1, this does not improve the output, but merely shifts the results along the spectrum of probability threshold.

The vast majority of games available have substantially less player data than Game A. We find four games with similar quantities of sign-ups in the first six months of 2016. In keeping with §12.2, we train models for each game on sign-ups from the January to March 2016, and evaluate those models on sign-ups from April to June 2016. We plot F1-Score against probability threshold in Figures 12.2 and 12.3. Recall that we have anonymised the games we have access to, so refer to them as games A, B, C, D and E, where Game A was used during development.

## 12.3.2 Churn Models

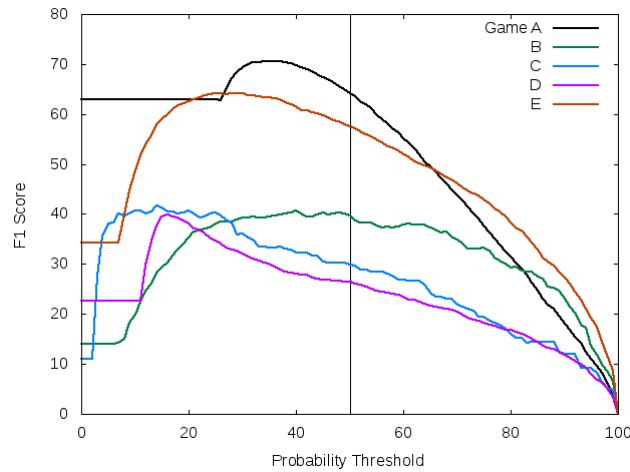


Figure 12.2: F1-Score Vs Probability Threshold for Churn Models on Other Games

The results of churn models on other games, shown in Figure 12.2, are clearly disappointing. In most cases, F1-Scores peak at around 40, well below the peak of Game A. The result of Game E, however, can be seen to show that it is possible to build useful models on other games, and that it is not mere coincidence that we trained on a uniquely suitable game. More scrupulous feature selection on a game-by-game basis should improve the results for all games, though presumably not to the extent that they could surpass the F1-Scores of Game A. For churn models, then, other games simply do not seem to exhibit the same level of success as that seen for Game A. As mentioned, however, Game A has especially high rates of long term play, and therefore the logistic regression can expose telling indicators more precisely. We conclude that we must lower our expectations to reflect that most games are not as naturally suited to high quality churn predictions, due to their having significantly lower rates of long-term play.

### 12.3.3 Spender Models

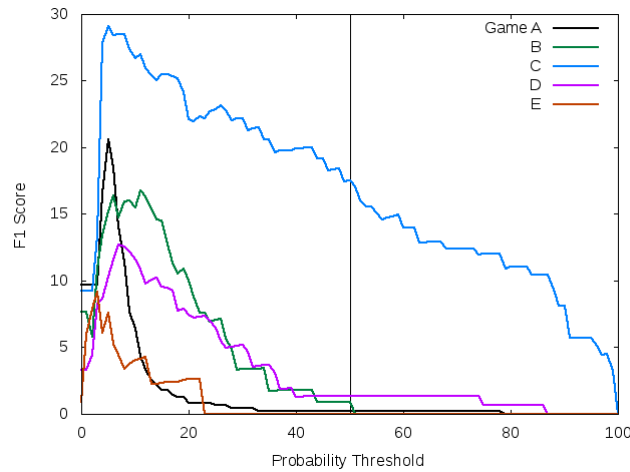


Figure 12.3: F1-Score Vs Probability Threshold for Spender Models on Other Games

The results for spender models on other games, Figure 12.3, are more promising. Most prominently, Game A is eclipsed by Game C, and Game B almost matches the curve of Game A. Taken together with Figure 12.2, where Game E is the most successful, we can verify that the framework, when applied to numerous games other than the one it was developed with, can achieve basic measures of success.

### 12.3.4 Conclusion

While the results do not unanimously indicate high quality predictive analysis, we can at least conclude that there is sufficient evidence to ratify the framework's applicability across player data for most games, albeit in most cases with F1-Scores lower than the high F1-Scores seen for Game A.

In reaching this conclusion, we slightly modify our expectations of success, though overall can claim to meet Aim 3.3, since we have demonstrated the framework's usefulness for building models on a wider selection of games. We notice a limitation however, that in order for a highly-functional predictive model to work, the base game needs to have a large number of players, and a reasonably high rate of spenders/long term players among those.

## ASSESS FRAMEWORK

---

### 13.1 FEASIBILITY

A major concern with regard to the integration prospects of our framework is how quickly users can get predictions from it in practice. It is essential that clients do not have to wait for very long to view predictions when requested, since they have high standing expectations of deltaDNA which we do not wish to tarnish. This leads to a two leading possible implementation strategies.

1. Models are built automatically for all games and updated periodically. Then, when a new player signs up, and after the initial overhead to observe their behaviour (e.g. two days), predictions are made about that player without the explicit request of the game owner. This seems like a sensible solution since the owner does not have to wait while the model is trained upon request, but incurs the cost of building models for all games. This would require either the client to spend some time personally optimising their individual model, or else adopting a standard set of parameters for use on all games (which would not be optimal), or else developing the framework to find the best features automatically, which would require a lot of development time to perfect.
2. Train model on request. It may save considerable resources, as deltaDNA expand to house hundreds, and later thousands of games, to only train at the request of a client. In doing so, they could have the choice of all model parameters. We could, for instance, provide suggestions for how to build the best model, e.g. to train a model on recent data since player bases change over time. This approach gives total freedom to the client, but has the drawbacks that there would be a time delay before they get access to predictions on new signups, and in many cases the parameters chosen will not reach the full potential available in the data, unless the client commits to exhaustive feature selection processing, which most will not, due to the time and data science understanding required.

#### 13.1.1 *Performance*

Feasibility is gauged in terms of the speed of the framework. Table 13.1 shows the range of sample timings achieved for each model type.



	Churn	Spender
R	11.2 - 14.7	11.8 - 17.6
Vertica	12.5 - 26.2	10.6 - 65.3

Table 13.1: Timings for Model Types (in seconds)

The minimum timing most accurately reflects how the framework performs when unhindered, i.e. when the network connection to Vertica is fast and the local machine running R is not dedicated to other tasks. There is much higher variance on runtimes for models run on Vertica. This is due to the overhead in connecting to the Vertica database, which fluctuates depending on the business of the network.

The lower end of the timings are generally acceptable: it is not unreasonable that building a predictive model would take up to a minute. Aim 4.1 discussed building models in *reasonable* time, such that clients would be willing to accept the delay in obtaining predictions. We deem Metric 4.1.1 met since the entire framework is usually run in less than thirty seconds, failing delays beyond our control in connecting to Vertica.

### Optimisation

It is possible to refine the model-building process in order to optimise performance. This would require more detailed profiling of the framework to establish the most time-intensive sections, and then investing further time to study and improve them.

Performance was not our primary concern during this project, so we expect that the overall code is not as efficient as it could be. The code, written in R, is designed for simplicity. Identifying areas for performance enhancement is intricate, and since R is highly optimised, the cost of time and resources for redevelopment would probably prove unjustified compared to the performance benefit achieved. Nevertheless, this may be taken as a point of future development along the path towards full scale implementation.

## 13.2 VERTICA ONLY MODEL

Aim 4.2 was to investigate the feasibility of a Vertica only model, i.e. one which would not require downloading data to another platform to perform analysis there, which we have done using R. While R may even seem a faster solution going by Table 13.1, we are mindful that the deltaDNA platform is *already* optimised for interacting with Vertica. Adding a layer of R, or indeed any other platform, in between Vertica and deltaDNA will undoubtedly add a significant lag for the user. This is why we are so intent on working as much as possible within Vertica.

As discussed, Vertica's machine learning functionality at present is subject to strict limitations. The most troubling of these are that only numerical features can be used for fitting a model (apart from manually encoding features, which is not widely practical), and that the probability threshold at which samples are predicted as positive is fixed at 50%.

Recalling Figures 11.1 and 11.2, and in particular feature sets 1, 6, 7 and 10, we can see that purely numeric feature sets are not necessarily much worse than mixed (numerical and categorical) sets in all cases, and surmise that more meticulous feature selection *could* lead to a numeric set almost as good as, or better than, the best mixed set observed, fulfilling Metric 4.2.1.

The issue of Vertica's overly rigid probability threshold too can be overcome, albeit with a significant amount of developer effort. This could be done by adding columns to the training data to represent log-odds and building a linear regression model on that; then feeding the predictions through steps outlined in §4.3 to calculate a probability, and finally making prediction based on this derived value. Alternatively, §12.1 shows that weighting positive cases more heavily than negatives can side-step this issue by increasing the positive probability predicted on all samples. We can introduce weightings in Vertica, though it is not a trivial addition. We also saw how this is equivalent to over sampling. Over sampling would also demand a major overhaul to the current iteration of the Vertica model building architecture.

Evidently, in order for Vertica to achieve the same functionality as is available immediately in R, a large amount of additional development is required. We must conclude that the current machine learning offering in Vertica is too restrictive to perform profitable predictive analysis on game player data.

### 13.2.1 Conclusion on Vertica's Applicability

Thus, Aim 4.2 results in one of the pivotal findings of this project: Vertica, at present, is not suitable for deltaDNA to integrate predictive analysis. To proceed, they must opt for one of three core options. The first is to accept some level of interaction with another platform such as R and accept the performance overhead this invokes. The second option is to build a more complex model construction framework to include weighting/over-sampling and offer factorisation of categorical features as standard. The third option is to accept sub-optimal results available by simply applying Vertica machine learning functions without additional functionality. Finally, they can simply choose to wait and request that later releases of Vertica's machine learning functionality make it more useful, for instance by enabling categorical features to be passed to regression models.

## Part IV

# CONCLUSIONS AND RECOMMENDATIONS

## GENERIC FRAMEWORK

---

In Chapter 5, we laid out a number of goals to drive our project. In Part II, we showed how we developed our workings to result in a generic framework for building and evaluating logistic regression models. Part III then considered all of the aims which comprise Goals 1 to 4. It remains to evaluate the framework itself, the subject of Goal 5. This is reserved as a conclusion, rather than included in Results and Analysis, since the analysis given in this Chapter ultimately sums up the entire output of our project. We recall the aims and metrics of each goal, and assess how the framework was applied to explore those aims. This leaves us in a position to declare whether our framework may be considered a successful final product for this dissertation or not.

### 14.1 FRAMEWORK EVALUATION

Metric 5.1.1 states that the generic framework can be considered a success if it can be used to build models to test all other metrics. Here we will recall the work of Part III to assess whether we can deem this to be the case. To analyse this, consider all the goals, aims and metrics laid out in §5.

#### 14.1.1 *Goal 1: Build a Model. Chapter 10.*

Here we aimed to build a model using numerical features in Vertica. This was done by specifying a single character in a `global_variables` file, which tells the framework to build in Vertica rather than R, and choosing features in a `feature_selection` file. By building the model, and proving that it works correctly, we satisfied Metrics 1.1.1 to 1.2.1. Further, in §11.2.1 we show that Vertica gives identical results to R, meeting 1.2.2.

#### 14.1.2 *Goal 2: Learn by Applying Predictive Models. Chapter 11.*

Here we aimed to learn from the data, by manipulating the framework, to optimise various aspects of the predictive models.

Metric 2.1.1 is met by Feature Set 8 (from Table 11.2) achieving high levels of success for our test game, as in §11.2.3. Feature sets are chosen once in the `feature_selection` file. We added a `set_id` for each of the defined feature sets, and looped over them to generate plots such as Figure 11.2.

Metric 2.2.1 is addressed in §11.1.3, where we prove that, in general, models benefit from being able to consider categorical variables as standard. This is another result of the simplicity of feature set variation.

Metric 2.3.1 is met in §11.3 where we determine the most effective values for long and early cutoffs, and demonstrate that varying these values does have a noticeable impact on models. Again, these can be looped over to pull data for plots such as Figure 11.5. Further, the framework has a parameter for long term low and high cutoff points; the default functionality is to loop over these and return results for all resultant models, which is intended to simplify choosing a long term cutoff for later users.

#### 14.1.3 Goal 3: Extend Modelling to Address Other Questions. Chapter 11.

By extending the model, we aimed to accomplish the aims which constitute Goal 3.

To investigate weighting positive samples, our framework has a value,  $k$ , which can be looped over, resulting in Figure 12.1. Metric 3.1.1 is defeated since weighting is shown not to be useful for our cause. We do gain knowledge that weighting is equivalent to changing the probability threshold at which a player is deemed as a positive prediction. Note also that probability threshold is easily changed by the user, and it has the default behaviour to calculate all results between a lower and upper bound set by the user.

Aim 3.2 was to investigate whether there was foundation for the belief that player bases change over time. Metric 3.2.1 was satisfied by the uplift seen by training a model in the same time frame as that in which it is tested, compared with training on players who signed up a year earlier. Our framework has parameters `training_start_date`, `training_end_date`, `test_start_date` and `test_end_date`, so this was easily tested.

Metric 3.3.1 demands similar levels of success, as seen for the original development game, when the framework is used on other games. In §12.3, again by changing a single value in the `global_variables` file - `environment_id`. We find that, in some cases, respectable results were possible, though we found unsatisfactory results for many games. We concluded that not all games are automatically suitable for predictive analysis, and that no feature set can universally reap high results, meaning that each game will require individual attention in order to benefit from machine learning.

#### 14.1.4 *Goal 4: Assessing the Model. Chapter 13.*

Goal 4 was to assess the model with a view to how it may be implemented on the deltaDNA platform.

We first needed to show that the framework could build models in reasonable time (Aim 4.1), which was done by adding timing calls into the framework. Metric 4.1.1 was met, declaring that the time taken to build models is low enough that the cost of optimising performance would likely outweigh any benefit.

Aim 4.2, to investigate whether a Vertica only model could be useful, was carried out by changing the `model_type` parameter in the `global_variables` file to compile models directly on Vertica. §13.2 concludes that the current suite of machine learning functions on Vertica are insufficient to provide fully operational predictive analysis to the clients of deltaDNA, without either significant investment in development, or extracting data to R (or another platform), and modelling there.

#### 14.1.5 *Goal 5: Generalising to Framework. Chapter 14.*

Finally, as evidenced by §14.1.1 to 14.1.4, we were able to test for Goals 1 to 4 through simple switches to our final product, the framework built by extending the ideas used when building individual models. As such, we can claim to have satisfied Metric 5.1.1, and so have reached Goal 5.

## RECOMMENDATIONS FOR FUTURE DEVELOPMENT

---

This project was the first step in the long process of introducing a new predictive analysis service to the deltaDNA gaming analytics platform. As such, there are several major stages to undertake following on from the framework we have developed.

A very high level overview of the stages involved in realising a new service on a real-time platform is:

**PROOF OF CONCEPT** To show that the considered application can be fruitful.

**FEASIBILITY STUDY** To show that the considered application is achievable within the time frames and resources available, and works fast enough to be considered useful.

**FRAMEWORK** Develop a system which provides the service at a low level.

**TESTING** Verify that the low level implementation works in all known cases.

**USER INTERFACE** Build on the implementation for user friendliness and simplicity.

**INTEGRATION** Offer the service to a subset of clients to gauge its practicality.

**REITERATION** Improve the framework and user interface based on feedback and analysis of the previous version.

**FULL SCALE IMPLEMENTATION** Integrate the service within a major update of the platform.

Our framework addresses some of these stages partially, but does not attempt to encompass all of these steps.

1. We provide full proof of concept that predictive analysis *can* be useful on game players, when performed carefully.
2. We perform a brief feasibility study to show that predictive tools need not be overly cumbersome and can be built and applied in short enough time periods as not to discourage clients.
3. We have developed a framework which can be used on any game to test its suitability for predictive analysis. A shortcoming of the framework in its current form is that it does not find optimum parameters such as feature set and long term cutoff, but rather depends on the user to make such decisions.

4. We have performed some basic testing to be confident that our framework works across a number of games, to some elementary measures of success. While the model construction works, it often results in unsatisfactory results such as F1-Score. Further testing is needed to identify useful features across in all games.
5. Our framework represents a first iteration of a low-level platform. It can be used by future developers to improve upon, towards a program which is ready to implement.
6. We have not considered user interface, or integration on the web platform for deltaDNA.

Thus there are two classes of further development required: improvements to the underlying predictive analysis; and usability/implementation. We are not majorly concerned with how our tool may be implemented, since this is a more a software engineering task than data science application. We briefly summarise requirements for such a development in §15.2. We can offer more insightful suggestions for how to improve the framework towards one which could more realistically be called on in such an implementation, §15.1.

## 15.1 IMPROVEMENTS TO FRAMEWORK

Our current model has a number of key shortcomings. Most notably, it cannot be generically applied to a new game and automatically build the best model available on its data. Instead, it requires human attention to consider and decide between results that the framework generates.

### 15.1.1 *Pipeline for Analysis*

A relatively simple extension to our current model is to create a single click process which generates, analyses and plots results. At present, the framework is run, and separate scripts are run in R and Gnuplot to produce plots such as Figure 11.5. A script to call the appropriate scripts in the correct order would speed up the process visualising the results of the candidate model, and hence speed up choosing between available parameters. However, there is no repeated formula which could be used to produce all plots which may be desired, so extra consideration is needed to determine how to most elegantly achieve these plots when varying different parameters each run. For instance, we could add another switch in `global_variables` which instructs the framework on what should be analysed from the output. For example there could be one flag to make the framework produce a plot of F1-Score against Probability Threshold, to gauge the quality of feature sets.



### 15.1.2 *Automated Parameter Selection*

The natural next step for our framework is to enable it to analyse and find the parameters which would achieve optimum results, with minimum human input. In particular, the following seem within reaching distance of the current infrastructure.

**FEATURE SELECTION** Have a user pass a list of all *eligible* features, and have the framework identify which of those are *valid* by eliminating features on the grounds of p-value, as done above. Compose a number of subsets of valid features and run models using those to find data such as that which underlies Figure 11.1 and 11.3. Choose whichever feature set yields the highest F1-Score.

**PROBABILITY THRESHOLD** Nominate the probability threshold at which the peak F1-Score is found as the value which should be used in the real-life model.

**LONGTERM AND EARLY BEHAVIOUR CUTOFF** Having elected a feature set to use for a particular game, build a series of models by varying long term and early behaviour cutoffs, e.g. Figure 11.5. Then, by analysing the uplift offered by delaying running models by one day, determine which early cutoff ought to be used. Then compare the negative uplift for using a longterm cutoff one day higher, and find the day which is sufficiently high to be useful, but still achieves high results. These decisions will be more difficult to impose computationally, since they are largely decisions of context and require human interpretation. Defining unambiguous criteria to make these decisions is thus one of the main next steps for this project.

### 15.1.3 *Answer More Useful Questions*

Another aspect which was not given due consideration during this project is using predictive analysis to answer interesting questions, which account for particular behaviour such as early spending. For example, questions such as

- "If a player pays within their first two days playing, how likely are they to ever pay again?"
- "If a player plays for three days and then does not play for three days, how likely are they to return?"

Such questions were never our aim so we have not designed our framework towards answering them. We expect that this could be a promising area for future development.

## 15.2 USER INTERFACE AND PLATFORM INTEGRATION

The eventual online implementation of this concept will depend on the future development of our framework. In the case that parameters can be automatically optimised, as envisaged in §15.1.2, then models can be built in the background, and predictions simply made available to clients without their input.

Alternatively, since user input was a cornerstone of our project's foundations, an online implementation may be built to preserve the user's ability to define the model themselves. In that case, a user interface would have to have sections where the user can input:

- Model Type (Churn/Spender)
- Features
- Longterm cutoff
- Early Behaviour cutoff
- Training/Testing start and end dates
- Game (though most users will only have access to one game)
- Weighting for positive cases
- $\beta$  (for use in calculating  $F_\beta$ -Score)

Ideally, some combination of these two possibilities would be composed, whereby the framework finds what it believes to be the optimum parameters, but ultimate power remains with the user to allow them to test their own hypotheses, as was desired at the beginning of the project.

## CONCLUSIONS

---

This project was driven by an overarching goal: to establish whether or not predictive analytics has the potential to bring useful, accurate services to the clients of our industrial partner, deltaDNA. In particular, clients of deltaDNA typically care about whether their players spend over their life span, and whether they play for more than a certain number of days. These form our two use cases: spend and churn modelling.

To answer this question, we examined the data available to deltaDNA, and determined that logistic regression was the most likely candidate to realise such predictive analysis. We then set out to build a logistic regression model which would outperform randomly making predictions based on the rate of occurrence in the training data. Since the Vertica database is employed by deltaDNA, we first restricted ourselves to applying Vertica's newly available machine learning functions directly on the Vertica database. We showed that such models can defeat random chance, but do not achieve especially high accuracy metrics.

We then considered how our model could be manipulated to learn about the data, so as to optimise the models built and therefore the outputs achieved. Ultimately this led to a complete generalisation from building individual models, to constructing a framework used to build models, where the user has control over all model parameters, such as what features are used to make predictions, and what the definition of a long term player is. This framework allows for models to be built in either Vertica or R, so that we can contextualise the results we would obtain by restricting ourselves to Vertica's inbuilt functions, with respect to what can be achieved using the same data in R.

With this framework in place, it was relatively straightforward to test a series of hypothesis, and to generate data to optimise the variable parameters. We use the framework to choose a feature set, long term and early behaviour cutoffs, and probability thresholds which give the best model fits. We prove that the disparity in time between when a model is trained and assessed is of importance to how that model performs, and so models cannot be built once and maintained indefinitely for a single game, but rather must be updated periodically. We also prove that weighting positive samples during model training is not helpful, and that redefining the probability threshold for a non-biased model is of more benefit.

We test the generality of our framework by using a constant feature set to build models on a number of games. We found that the framework can be used to successfully model numerous games, as was hoped and expected, but that many games have insufficient data or too low rates of expenditure/long-term play to model well.

## 16.1 FINAL CONCLUSION

In final answer to the question of whether predictive analysis can be of use to deltaDNA, we conclude that it can. However, at present it is considered unfeasible to extract and analyse data offline in R, where most of our promising results were found. The preference would be to build and analyse models directly on Vertica. The current offering of machine learning functions on Vertica, we find, is too rigid to be of any real use.

There are therefore three potential avenues by which deltaDNA can further the introduction of predictive analytics as a core service following directly from our project.

The first is to invest more time and resources to develop a method which is efficient enough to feed data from Vertica, process it offline, then upload and present it to clients via the on-line platform. This would allow the most straightforward development on top of our current framework, and requires less development before it is viable for implementation.

The second option is to redevelop our framework to enable weighting/over-sampling so that the peak F1-Score can be seen at 50% probability threshold and hence Vertica can achieve high results. This development could extend further to a generalisation of encoding categorical variables for inclusion in modelling. Failing this, we simply have to restrict ourselves to numerical features, which is not ideal. This option may be seen as favourable for its simplicity in implementing (since there is already infrastructure in place for Vertica queries to drive parts of the deltaDNA platform). We do not recommend this, however, since it would require significant development to achieve, and would result in sub-optimal, restricted models.

The third option is to simply accept the sub-optimal methods currently available in Vertica. As seen throughout this text, however, this would ultimately result in a poor predictive service, which is of little addition to the high standards expected of deltaDNA's services at present.

A final alternative is to postpone adding this service until the Vertica machine learning toolkit is more advanced, and in particular allows for the inclusion of categorical variables, and for the user to set the probability threshold (in a binary prediction) above which a sample is accepted as a positive prediction.

## 16.2 FINAL RECOMMENDATION

We recommend the first option as the most straightforward choice following directly from this project. We accept that this is out of sync with the desire to keep all working within Vertica, but believe that, given the evidence presented here, it is of more benefit to facilitate an offline framework.

## BIBLIOGRAPHY

---

- [1] deltaDNA. <https://www.deltadna.net>. Accessed: 01-08-2016.
- [2] Vertica, Hewlett Packard. <http://www8.hp.com/uk/en/software-solutions/advanced-sql-big-data-analytics/>. Accessed: 16-08-2016.
- [3] Dbeaver. <http://dbeaver.jkiss.org/about/>. Accessed: 01-08-2016.
- [4] Rjdbc. <https://www.rforge.net/RJDBC/>. Accessed: 16-08-2016.
- [5] Predictive analysis functions in vertica. <https://my.vertica.com/get-started-vertica/machine-learning-predictive-analytics/>. Accessed: 01-08-2016.
- [6] Rachel Schutt and Cathy O'Neil. *Doing data science: Straight talk from the frontline*. "O'Reilly Media, Inc.", 2013.
- [7] Jin Huang. *Performance measures of machine learning*. University of Western Ontario, 2006.
- [8] F1-score. [https://en.wikipedia.org/wiki/F1\\_score](https://en.wikipedia.org/wiki/F1_score). Accessed: 16-08-2016.
- [9] Receiver operator characteristic curve. [https://en.wikipedia.org/wiki/Receiver\\_operating\\_characteristic](https://en.wikipedia.org/wiki/Receiver_operating_characteristic). Accessed: 16-08-2016.
- [10] Samprit Chatterjee and Ali S Hadi. *Regression Analysis by Example*. John Wiley & Sons, 2015.
- [11] Gianluca Bontempi. Statistical foundations of machine learning. Accessed 16-08-2016.
- [12] P-value significance threshold. <http://www.jerrydallal.com/lhsp/p05.htm>. Accessed: 16-08-2016.
- [13] Colin Shearer. The crisp-dm model: the new blueprint for data mining. *Journal of data warehousing*, 5(4):13–22, 2000.
- [14] CRISP-DM, wikipedia entry. [https://en.wikipedia.org/wiki/Cross\\_Industry\\_Standard\\_Process\\_for\\_Data\\_Mining](https://en.wikipedia.org/wiki/Cross_Industry_Standard_Process_for_Data_Mining). Accessed: 16-08-2016. Image submitted by Kenneth Jensen.
- [15] Scikit-learn. <http://scikit-learn.org/stable/>. Accessed: 01-08-2016.
- [16] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al.

Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

- [17] R generalised linear model. <https://stat.ethz.ch/R-manual/R-devel/library/stats/html/glm.html>. Accessed: 01-08-2016.
- [18] Vertica logistic regression functionality. <https://my.vertica.com/get-started-vertica/machine-learning-predictive-analytics/#Logistic>. Accessed: 01-08-2016.
- [19] Factorisation in R. <https://stat.ethz.ch/R-manual/R-devel/library/base/html/factor.html>. Accessed: 16-08-2016.