WPI

PREDICTING CLICKS ON MOBILE ADVERTISEMENTS

A Major Qualifying Project

Submitted to the faculty of

Worcester Polytechnic Institute

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

Submitted by: Christina Webb, Alexander Gorowara

Submitted to: Project Advisor, Professor Carolina Ruiz

Date: April 30th, 2015

Acknowledgements

We have taken efforts in this project. However, it would not have been possibly successful without the kind support and help of many individuals and our sponsor. We would like to extend our sincere thanks to all of them.

We would like to express our very great appreciation to our sponsor Chitika Inc., specifically the CEO Dr. Venkat Kolluri and two project liaisons from Chitika Inc., Joseph Regan and Stijn Peeters. Their valuable and constructive suggestions during the planning and development of this project, and well as their willingness to give us their time so generously has been much appreciated.

We would like to express our deep gratitude to Professor Carolina Ruiz, our project advisor, and Ahmedul Kabir, CS Ph.D. student, for their patient guidance, enthusiastic encouragement and useful critiques of this project work.

Finally, we wish to thank all of our friends and people who helped us for their encouragement and support throughout this project.

Table of Contents

Table of Figures

Table of Tables

Abstract

We explored methods of improving upon Chitika, Inc.'s existing means of predicting which users would most probably click on an advertisement in a mobile application. We used machine learning algorithms, primarily Naive Bayes, that trained on demographic and behavioral information supplied by the user and his/her mobile device. After an exploratory phase, we gathered performance data using the AUC metric on twenty-eight different experimental conditions. When compared to the control condition, in which no preprocessing was performed on the data before being given to the unmodified Naive Bayes algorithm, we found only minor improvements in AUC.

Executive Summary

Chitika, Inc. is an online advertising company that connects ad providers (companies that want consumers to see their advertisements) with content providers (companies that have consumer-visible website space to rent). Chitika's revenue is derived from the efficient arbitrage of ad space. The task they gave us was to seek out methods which could improve their already successful predictions of which users would or would not click on ads.

In order to improve these predictions, we worked mainly with the Naïve Bayes machine learning algorithm, which uses Bayes' Rule and observed probabilities to calculate the probability of a given datum belonging to a previously defined group. We also relied heavily on clustering algorithms, which group data into different sets without any previously existing labels or definitions of those sets.

We conducted experiments using these and other tools on a data set with over 4.5 million user impressions, collected across a week of activity on a single mobile application. This data was naturally and heavily skewed towards non-clickers, who made up 99.55% of the sample. Our experiments on this data covered four different modification conditions to the data set itself, and seven experimental methods, for a total of twenty-seven different experimental conditions.

Overall we found at best minor improvements relative to the control condition of an unmodified data set which received no treatment before being run through Naïve Bayes. However, our experiments with the method using the Expectation-Maximization clustering algorithm were sufficiently unusual and high-performing to deserve further inspection. We are

optimistic that the approaches which we have documented will be of use to Chitika as they consider trade-offs in speed, complexity, and performance.

1 Introduction

1.1 Problem Description

Chitika, Inc. is an online advertising company that connects ad providers (companies that want consumers to see their advertisements) with content providers (companies that have consumer-visible website space to rent). Chitika's revenue is derived from the efficient arbitrage of ad space.

One mechanism by which such companies operate is called real-time bidding (RTB). When a user visits a content provider's website, the content provider announces this user's arrival on an ad exchange. The user's visit (called an impression) comes with some information: data such as location, browser version, device version, and much more. Based on this information, networks such as Chitika bid for the ad space for this particular impression - the right to show a single ad to that single user. If the ad is successful (which, in most cases, means that the user clicks on the ad), the network is then paid. In this model, which must happen sufficiently fast for the user to have no noticeable delay, the ad network takes the risk that the user will not click, but reaps the reward if he/she does. The problem posed by Chitika to us appeared very simple: they wanted to improve their prediction of users' likelihood to click on ads. . This meant that they wanted to explore new ways to quantify the probability that a given user, distinguished by a limited amount of demographic, technical, and behavioral information, would find their ad enticing enough to click on it.

Our greatest asset in this project was the amount of data with which Chitika supplied us, full of anonymized user records including (where available) demographic information and behavioral history, coupled of course with the all-important target attribute: whether or not they

had clicked on the ad. A more detailed description of this data can be found in Chapter 33 Methodology.

We defined our goal for the project as the search for a well-tuned classification algorithm that a) performed well on classifying entirely unfamiliar users and b) could do so quickly enough to avoid delaying the user. For the first condition, we relied almost exclusively on the AUC, area under the (receiver operating characteristic) curve, metric to evaluate classifiers, and for the second, we found in the course of our experimentation that a large and unmistakable gap existed between those algorithms that scaled well for our purposes and those that did not, removing the need to experiment more rigorously.

By the end of our project, we had explored multiple classification algorithms and proposed variations on each, with special focus on variations on the Naïve Bayes classification algorithm. We also experimented with a variety of data pre-processing techniques, mostly involving clustering, and dimensionality reduction of the data available to us. Compared to our starting point, in which we used an entirely unmodified algorithm, we found only minor improvements relative to the control condition of declining to preprocess the data set or modify the machine learning algorithm used, and were able to increase the AUC metric for our best performing classifiers from 0.57 to 0.59.

2 Background

2.1 Machine Learning Overview

2.1.1 Performance Metrics

In order to deliver a solution which suited the needs of the sponsor, it was necessary to utilize unambiguous and objective performance metrics. We were fortunate to be inheriting a problem which the sponsor had already examined extensively, and as such they had a baseline performance measurement on a scale that we could easily use to assess our own results.

2.1.1.1 Area Under the ROC Curve (AUC)

This metric is known as area under the receiver operating characteristic curve, Area Under the ROC Curve, or just AUC. AUC can be applied to any classifier which judges test instances to be positive or negative by assigning them a probability of being positive. With such a classifier, in order to obtain a binary prediction of positive or negative, one would have to select a threshold such that all test instances with a probability above it would be positive, and all below would be negative. The AUC calculations are then done on the resulting pairs of probability and actual class. That is, for each test instance, the pair under consideration is of the form (p, class) where p is the classifier's predicted probability that the instance is positive and class is whether the instance is actually positive or negative. For example, (0.6, *positive*)denotes an instance which the classifier predicts is positive with 60% probability and actually is positive; (0.1, *negative*) denotes an instance which the classifier predicts is positive with only 10% probability and is actually negative; and (0.7, *negative*) denotes an instance which the classifier predicts is positive with 70% probability and is actually negative.

The ROC curve is then created by calculating the false positive rate and true positive rate for various thresholds, and then plotting the value pair (*false positive rate*, *true positive rate*) for

each of the threshold. The false positive rate (also known as 1-*specificity*) is the number of false positives divided by the total number of negative test instances (regardless of how they were actually classified), and the true positive rate (also known as *sensitivity*) is the number of true positives divided by the total number of positive test instances (regardless of how they were actually classified). Table 11 below gives an example of true and false positive rates.

Table 1: An example of true and false

	Predicted Class		
Actual Class		NEGATIVE	POSITIVE
	NEGATIVE	300	100
	POSITIVE	200	400

The number cells show how many instances with each real class label are given each predicted class label. The true positive rate is given as a function of:

$$TPR = \frac{True\ Positives}{Total\ Positives} = \frac{400}{200+400}$$

And the false positive rate is given as a function of:

$$FPR = \frac{False\ Positives}{Total\ Negatives} = \frac{100}{300+100}$$

The resulting ROC curve will span the domain (*FPR,TPR* = (0,0),(1,1)) from the lower left corner to the upper right corner. In the lower left are the points generated by high thresholds, which are strict enough to eliminate many potential false positives but also too strict to allow many true positives to be heard. In the upper right are the points generated by low thresholds, which are permissive enough to admit most true positives (even if they are not very well-

supported) but too permissive to filter out the majority of false positives. The AUC, then, is a threshold-independent measure of the classifier's performance, unlike other metrics such as accuracy or precision.

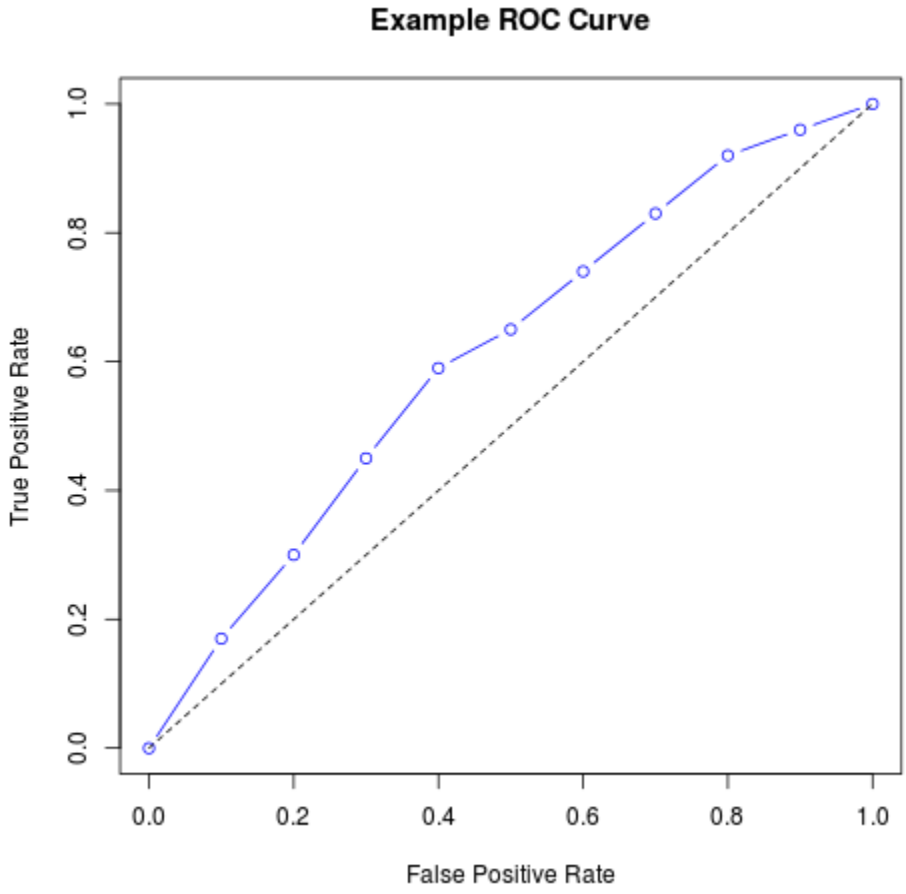


Figure 1: Sample ROC Curve

Figure 1 above gives an example of a sample ROC curve plotted from eleven different thresholds. While the Area Under the Curve, AUC, can range from 0 to 1, in practice it ranges from 0.5 to 1. The baseline curve of TPR = FPR show in Figure 1 above is the practical minimum performance for a classifier; it is achievable by a classifier which blindly assigns a random probability to each instance, discarding all available information. The area under this curve is 0.5, so any classifier which has an AUC value of less than 0.5 at some point would be worse than

random guessing. In fact, one could invert a classifier which scores less than 0.5 (transform all probabilities *p* that it outputs to 1-*p*) and thus create a better classifier. If a classifier were good enough to be constantly incorrect, always giving a test instance the opposite of its true label, then simply by labeling each test instance the opposite of the classifier's labels one could classify them perfectly.

2.1.1.2 Runtime Performance

Additionally, while we did not measure this characteristic rigorously, the classifier needed to be fast. We were not provided with an estimate of how fast it needed to be, but thankfully there seemed to be a natural divide among candidate procedures (combinations of algorithm and preprocessing) between the acceptably fast and the very, very slow. (As a note, our focus was not on the time it took the algorithm to learn; merely how much time it took to classify new instances, which was the step that would need to be done in real time).

Algorithms such as k-nearest-neighbors, which is a "lazy" learner that calculates the "distance" between the test instance and each training instance every time a classification is made, increase their classification time with the number of training instances, although the rate of increase can be slowed but not stopped. "Eager algorithms" such as logistic regression, on the other hand, are more time-intensive and resource-intensive when constructing the model that will later be used to classify a new instance. In exchange, once this step is done, they make predictions very quickly. Since the step of building the model is done infrequently and can be done offline long in advance of the need to make a prediction, we limited our search to "eager" learning algorithms only and quickly dismissed "lazy" learners.

2.1.2 Machine Learning Algorithms

One of the algorithms we investigated was logistic regression. It assigns probabilities by first transforming a test instance into a numeric vector (by rules explained below), then passing that vector to a logistic curve, which is bounded between 0 and 1. The output of that function is the predicted probability for the test instance.

The logistic curve equation has the form:

$$p(\vec{x}) = 1 / (1 + e^{-\vec{b} \cdot \vec{x}})$$

where p is the probability that some instance (represented by the vector $\vec{x}$) is a member of the positive class, and $\vec{b}$ is a constant coefficient vector. The vector $\vec{x}$ has at least as many elements as the number of attributes, and if all of the attributes are numeric, it has exactly as many; the first attribute becomes the first element, and so on. If there are any nominal attributes, however, these must first be converted into numeric attributes.

A nominal attribute may have a number of distinct values; for example, an attribute color may have the values *red*, *green*, or *blue*. To convert the attribute *color* into a numeric attribute, it is replaced by three binary attributes: *color = red*, *color = green*, and *color=blue*, each of which can have the value 0 or 1. This process is demonstrated in Table 2 below.

Table 2: The conversion process from nominal to numeric

size	*color*		*size*	*color=red*	*color=green*	*color=blue*
10	red	conversion	10	1	0	0
15	blue		15	0	0	1

12	red		12	1	0	0
17	green		17	0	1	0

The vector $\vec{b}$ is found by an iterative process (such as Newton's method), as there is not a general solution for finding the coefficients as there is for least-squares linear regression (Scott, 2002).

Once the coefficient vector is found, classifying a new instance is very quick, as there are only two significant steps: first, convert the instance into a vector; and second, give that vector as an input to the function $p(\overrightarrow{x})$. For conversion, numerical values are simply transcribed; nominal values are handled in a way that is only slightly more complex. Each nominal attribute, which may have a name such as *color* and may have valid values such as *red*, *blue*, and *green*, is converted into a series binary attributes, which may each have a value of 0 or 1. There is one binary attribute for each possible value, so the attributes may be *color_is_red*, *color_is_blue*, and *color_is_green*. (This process is also done when training the model; there are different coefficients for each binary attribute rather than one for the original nominal attribute). Once the appropriate vector is created, it is passed as input to $p(\overrightarrow{x})$and its output is then the probability that the new instance belongs to the target class.

Another probabilistic classifier which we examined is called the Naive Bayes model, named because it applies Bayesian probability calculations with the naive assumption that all attributes are independent of each other. It deals in *hypotheses* (such as *target=0* or *target=1*), which are possible target values, and *evidence* (such as *color=green* or *age=44*), which is data from attributes. The foundation of Naive Bayes is Bayes' rule, which calculates the *posterior*

probability (denoted as $p(h|e)$, the estimated probability that some hypothesis *h* is true after updating based on evidence *e*) as the function:

$$p(h|e) = \frac{p(e|h)}{p(e)} p(h).$$

Terms of note are *prior probability* (denoted as $p(h)$,the probability that some hypothesis *h* is true without any other data given) and the *likelihood ratio* or *Bayes factor*] (denoted as $\frac{p(e|h)}{p(e)}$, the ratio of the probability of evidence *e* occurring given that hypothesis *h* is true and the probability of evidence *e* occurring regardless of the truth of *h*) (Evidence-Based Diagnosis, n.d).

The likelihood ratio is how many times more likely *e* is to occur given *h* than on its own; if it is very large (and greater than one), then *e* is strong evidence for *h*; if it is very small (and less than one), then *e* is strong evidence against *h*. As a note, the likelihood ratio is bounded between 0 and $\frac{1}{p(h)}$; *e* can never occur when *h* is true (in which case $p(e|h) = 0$) or it can occur if and only if *h* is true (in which case $p(e|h) = 1$ and $p(e) = p(h)$).

In order to use Naive Bayes as a classifier, one tests multiple competing, mutually exclusive hypotheses, all of which are statements about the value of the target class, such as *target=0* or *target=1*. Bayesian calculations are done for each hypothesis, and at the end one has the values $p(target = 0|e)$ and $p(target = 1|e)$.

When instance to be classified has more than one non-target attribute, the likelihood ratios of each attribute's value are multiplied together to obtain the posterior probability, like so:

$$p(h|e \;\&\&\; f) = \frac{p(e|h)}{p(e)} \frac{p(f|h)}{p(f)} p(h).$$

This is the independence assumption that gives Naive Bayes its name.

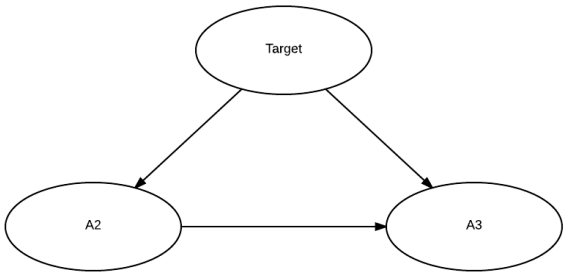


Figure 2: A Bayesian network

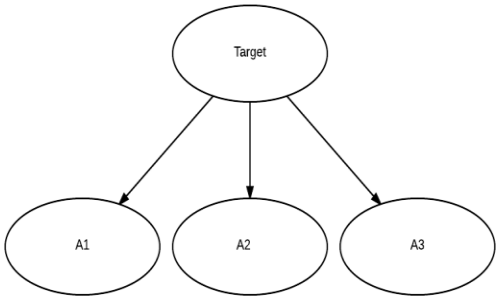


Figure 3: A Naive Bayes network

If the two events *e* and *f* are not actually independent (for example. their likelihood ratios are both $\frac{1}{p(h)}$, meaning that they occur if and only if *h* is true), the calculated posterior probability $p(h|e \;\&\&\; f)$may be greater than 1, which is clearly not a legal value for a probability. For this reason and others, most Naive Bayes algorithms calculate the posterior probability as the function:

$$p(h|e \;\&\&\; f) \;=\; p(e|h)p(f|h)p(h)$$

It then normalizes the probabilities such that the sum of all posterior probabilities is 1. Since the denominators of the likelihood ratios (the prior probabilities of *e* and *f*) are not dependent on *h*, they are implicitly included in normalization. However, the prior probabilities of the evidence will be relevant later in our experimentation.

The probabilities used in likelihood ratios are generated, for the most part, from a simple counts of the number of times that an event *e* occurs and the number of times that it occurs accompanied by the hypothesis *h*. This makes Naive Bayes incredibly fast to train; a single pass of the data, involving arithmetic more complex than can be performed by a four-function calculator, is sufficient to gather all the necessary information.

However, there are two scenarios in which a simple count is unsuitable. The first is the case in which there is an event which is possible, but sufficiently rare that it does not occur in the training data (which may be a limitation of its size). For such an event, the observed $p(e)$ and $p(e|h)$ are both zero. To avoid multiplication (or division, if the prior probability of *e* is explicitly included) by zero, Naive Bayes may use Laplace smoothing (Peng,2004), calculating the conditional probability as the function:

$$p(e|h) = \frac{\square(e \,\&\&\, h) \;+\; \alpha}{n(h) \;+\; \alpha d}$$

n(*event*) is the number of times that some *event* occurs, *d* is the number of different possible values of *e* (for example, if the *color* of an object can be either *red*, *blue*, or *green*, then d is 3), and α is a parameter (generally 1). This can be conceptualized as having a "virtual datum" corresponding to each possible event like *e && h*, so that no probability ever has a numerator of zero. As the size of the training set increases but α and *d* remain the same, this Laplacian smoothing becomes less and less significant, and eventually approaches total irrelevance.

The other case in which the conditional probabilities are not simple counts is when numeric attributes are involved. Some numeric fields, such as the hour of the day (as an integer), can be appropriately approximated as nominal attributes, because they have a manageably small

and distinct number of possible values, much like nominal attributes. But other numeric fields, such as an individual's height in centimeters, cannot be fairly approximated as nominal; there is a large number of possible values, and it is unintuitive to say that a person who is 183 cm in height should be considered in an entirely separate category from one who is only 182 cm.

One solution is to discretize all numeric attributes before running them through Naive Bayes. The 183 cm person falls into the same category of "160 cm - 190cm" as the 182 cm person, and the number of possible values becomes manageable. Another solution is to use a numeric estimator, which assumes a certain probability distribution (such as Gaussian) and, given a value (such as 183 cm), calculates the probability of that value without involving counts at all. Each hypothesis (such as *target=0* and *target=1*) has its own estimator for each attribute; the height of the population for which *target=0* is true may have a different mean and standard deviation from the population for which *target=1* is true. The conditional probabilities, then, for the 183 cm person are slightly different than the ones for the 182 cm person, but only slightly. The available distributions differ from implementation to implementation, and may be the same for each numeric attribute, regardless of the goodness of fit.

While Naive Bayes is incredibly swift and powerful, its independence assumption leaves a lot of territory to be explored.

2.1.3 Clustering Algorithms

In the course of our experimentation, we relied at times on *clustering algorithms* - algorithms which, given a data set, cluster them into a (parameterized or automatically determined) number of different groups depending on their similarity, often with no knowledge of the target attribute. Clustering algorithms can identify and act on similarities or memberships which are not explicit in the data, but which still have the potential to be useful. We used

clustering algorithms mostly to separate data into different training and test sets, and the two algorithms we relied on were k-means and expectation-maximization (EM).

Before continuing, it is important to explain the concept of *feature space*. In a previous section, we explained how a data instance can be represented as a vector; feature space is the space in which this vector can exist. It has a finite number of continuous dimensions equal to the number of attributes (where each nominal attribute is replaced by a set of binary attributes), and one can apply a variety of distance metrics to any two points within it. These distances are used to calculate the similarity of two non-identical points; distant points are very dissimilar, nearby points very similar.

The k-means algorithm randomly places *k* different points in this feature space (often by randomly selecting *k* different members of the training set), called *centroids*. Any point in feature space belongs to the cluster of the centroid to which it is closest, as seen in Figure 4.

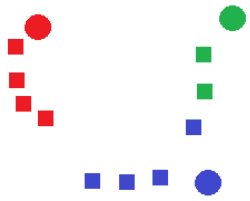


Figure 4: A freshly initialized k-means algorithm

Once the membership of every training instance is determined, the centroids shift to the center of their cluster, which is the mean position of the population of instances in the cluster. Once the centroids have moved, cluster membership is recalculated; each centroid may leave some instances behind or incorporate new instances into its cluster. This process is repeated until one of the following occurs: a set number of iterations is completed, the movement of the centroids becomes very small, or the number of training instances which change clusters in an iteration

becomes very small. The parameters of the algorithm may be set to accept only one stopping condition, or it may have multiple potential stopping conditions.

One limiting feature of k-means is the fact that it is a discrete clusterer; while there are meaningful metrics relating each instance to each cluster (distance to centroid), there are no guidelines in the algorithm itself which can meaningfully rate each instance-cluster relation other than ordinally. The algorithm does not lend itself to saying that an instance has a quantifiable mix of memberships. For this reason (among others), k-means is very sensitive to its initial conditions; two random selections of initial centroids can have very different effects.

The other clustering algorithm we used, expectation-maximization (E-M), does not suffer from this limit; instead, it classifies instances proportionally, assigning a proportion of each instance's cluster membership to each cluster.

The algorithm works by, as k-means, starting with random points somewhere in feature space and subsequently iterating until the clustering is stable. However, unlike k-means, E-M does not blindly shift its center based on its membership. Rather, it shifts based on the goodness of the fit, which is determined not discretely but by assigning each cluster a standard deviation in each feature dimension. The multi-Gaussian, multi-dimensional distribution so constructed can cluster instances probabilistically: an instance in some given position has such-and-such probability of being in this cluster, and so-and-so probability of being in that cluster, et cetera. Additionally, the implementation with which we worked had built-in cross-validation to correctly select the number of clusters based on goodness of fit, so there was no need to test various k-parameters until an appropriate one was found.

2.2 Software Tools

2.2.1 The R Project

The programming language R, which is designed to swiftly and efficiently handle vector and matrix data, has been our go-to tool for the vast majority of our data manipulation (The R Project for Statistical Computing, n.d). While initially we did attempt to use its machine learning packages for classification, we found that other packages better suited our needs and were significantly easier to work with.

R proved its value, however, with an easy-to-use command-line interpreter, quick scripting capabilities with little setup, and well-documented libraries for nearly any need (such as reading from and writing to ARFF files - a format developed for a single software tool). It may help that R is a non-commercial project frequently patronized by academia; perhaps in adapting it for their own needs, they conveniently gave us the same benefits.

2.2.2 Weka

The data analysis tool Weka is a collection of machine learning algorithms for data mining tasks, and it has been the main tool for our experimentation (Hall, 2009). We use Weka to apply Naive Bayes, Logistic Regression, and the K-Means clustering algorithm to our data and to get the ROC value from our experiments.

2.2.3 Python

We used Python to test some ideas in the first phase of our methodology when it was more practical to write our own algorithms from scratch instead of modifying Weka's. However, we did not have the resources and skills to optimize our Python code to work with larger data sets in a feasible timeframe, so we did not use it beyond the smaller experiments.

3 Methodology

Our methodology had two phases. In Phase One, the exploratory phase, we used a small data set to rapidly test a large number of approaches in a very short time. The Phase One data set came from a single day of observations from a single app on the Android platform, and had about twenty thousand instances (user impressions) and twenty-nine non-trivial attributes. We excluded any attributes which are either totally uniform across the data set (such as the application ID) or have a number of distinct categorical values on the order of the size of the data set itself (such as a user identification code). The Phase Two data set came from a series of six days of observations from an entirely different app (albeit one with many of the same user-submitted fields), encompassing over four and a half million instances and twenty-seven non-trivial attributes, many of which were identical to those in the Phase One data set.

Phase One's best-performing and most feasible analysis approaches were examined in Phase Two, which took up the tail end of the project.

3.1 Data Description

The data set that we used for Phase One included thirty different fields, or attributes, including the target *is_click*, which was a simple yes or no. These non-target fields ranged from simple numeric attributes, such as the hour of the day, to large categorical attributes with many possible values, such as in which of the 1700+ cities represented in the data the user could be found. Most of the attributes were categorical (though the term nominal is used to describe them in some machine learning literature), and many of them had an overabundance of values. The data set we used for Phase Two included twenty-seven different attributes including the target. Figure 6 below illustrates the distribution of users who clicked and did not click in our Phase two data set.

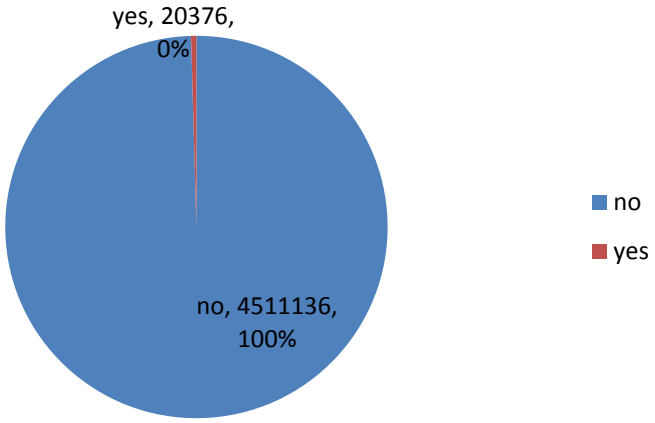


Figure 5: Distribution of target attribute: users who clicked vs users who did not click

The large number of values was not an insignificant roadblock - each attribute multiplied the number of possible combinations of values, leading to a configuration space far too large for the available data to cover even a small corner of the realm of possibilities. Many attributes were riddled with missing data. For example, data instances representing users who had turned off their GPS were missing longitude and latitude data. Figure 6 below shows a scatterplot of the location of GPS enabled users in the United States from our Phase One data set, which was 71% of the total users.

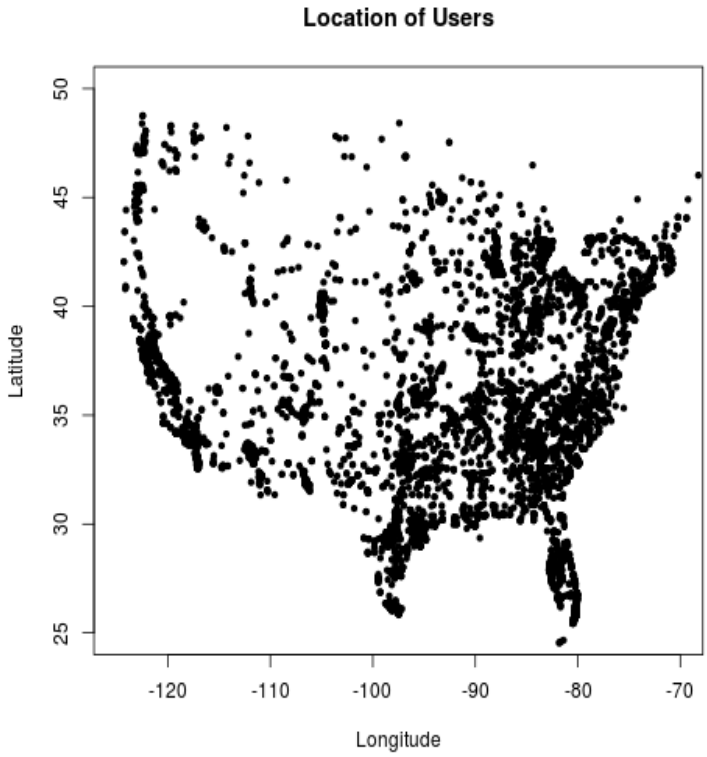


Figure 6: Location of Users based on Latitude and Longitude data

Some attributes were based on user input and were either missing or clearly incorrect, and still others were missing for reasons we do not know. User age, for example, has an unusually large number of people who claim the earliest allowable birth year (1900). A histogram of this data is shown below in Figure 7. Notice that the large spikes are at ages 18 and 94, and the smallest spike is at 114, the earliest allowable birth year.

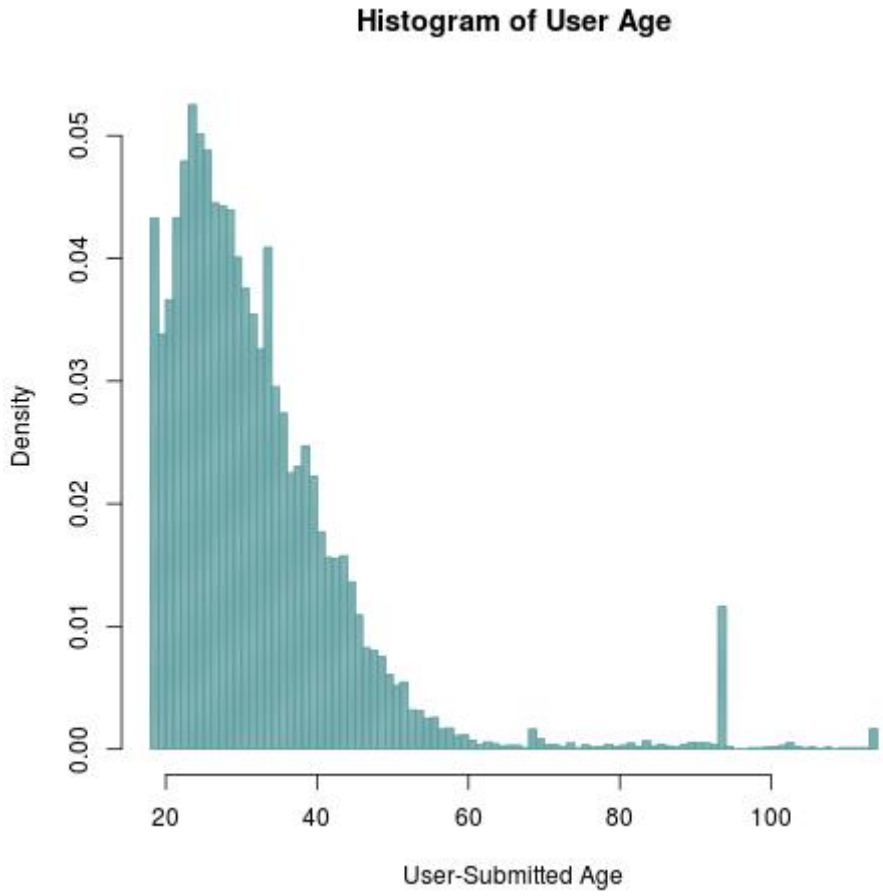


Figure 7: User-Submitted Data

Another significant issue with the data was the natural imbalance between the two values of the target attribute - the non-clickers outnumbered the clickers by a factor greater than fifteen in the Phase One data set, and composed 99.55% of the Phase Two data set, which can be seen in Figure 5. This meant that once the data was broken down into the subgroups we really cared about, one of those subgroups covered far less of the space of possible data than the combined data set.

3.2 Phase One

In Phase One, we took the "shotgun" approach: try a variety of approaches and note anything that looked promising. Phase One took the majority of the project, ranging from our choice of classification algorithm to eventual modifications and deep involvement with clustering algorithms. This phase was not rigorous in its analysis; we relied primarily on the AUC from ten-fold cross-validation to inform us which methods were worth further pursuit. For comparisons, we used a baseline AUC of 0.57 obtained from simple experiments with logistic regression, the sponsor's original algorithm of choice.

Later on, we kept a consistent training and testing set on which we could test methods that did not lend themselves to automatic cross-validation; the data separation section expands on a number of these and what difficulties came up. While our approaches were not uniformly consistent across the entirety of Phase One, we believe that with the use of baseline measurements in many later experiments, we secured a number of potential-laden novel methods to take into Phase Two.

In this section, we give a general overview of the types of approaches that we used. A more detailed record of our experimental protocols and their results can be found in Appendix A of this report.

3.2.1 Classifier Choice

Our sponsor had previously used the logistic regression classification algorithm, which is mentioned in the introduction. The challenge of logistic regression is numerically solving for the vector of constants in the exponent, and in our particular case the issue was far larger than it seemed at first - the wealth of categorical variables in the data caused the size of the necessary vector to grow to over four thousand values.

This was first noticed as a workflow issue - training a classifier on the twenty-thousand-strong Phase One data set took several minutes, and therefore the resulting cross-validation took upwards of an hour for each experiment. While not fatally damaging to the project, this delay made it difficult to quickly modify an experiment upon noticing an error or an opportunity in the output.

Further, if the classifier dragged with only twenty thousand instances, we were concerned that it would become impractical to experiment on a data set two orders of magnitude larger. Some preliminary experiments verified that the relationship between the number of instances and the time to train was such that even on a moderately sized compute cluster, experimentation would not be feasible. Instead, we sought a replacement algorithm which would be swift but similarly performant and we found the Naive Bayes algorithm. Not only did it manage to train classifiers within tenths of seconds - a hundredfold improvement over logistic regression - but it also gave us a moderate boost in AUC, from 0.57 to 0.59.

3.2.2 Data Separation

In the data separation experiments of Phase One, we explored the idea that treating the data as a monolithic whole to be judged by a single classifier could be improved upon. Behind our experiments was the idea that the same evidence could mean different things in different contexts - someone searching for “firearms” in New York City is more likely to be purchasing them for self-defense than someone making the exact same search in rural Alabama, who may well be more concerned about wild animals. It would, then, make sense to train different models on, in this example, urban and rural folk, lest the distinct meanings of the evidence be lost to the average.

Instead of taking pre-selected categories of people and separating them into different data sets, one variant of this approach is to let a clustering algorithm decide. The clustering algorithm would be trained on the training set, and then applied to both the training set and the test set, then each set would be separated into single-cluster sets. Each cluster's training set would then be used to create a classifier to be used to evaluate the appropriate cluster's test set. This is where the use of automatic cross-fold validation became impossible - the separation of the data sets was manual, and automating it as part of the classification algorithm would be non-trivial. For many of these experiments, we instead compared the weighted performance of each classifier to the performance of a similar classifier trained on an undivided training set.

As an alternative to the above approach, we also explored the use of a probabilistic, rather than discrete, clusterer. Where a clustering algorithm such as k-means will assign one and only one label - there is one and only one closest centroid for each datum - some algorithms, like E-M, have a probability distribution over the feature space. One instance may be considered to belong to a specific cluster if that is the strongest cluster at its location, but it is still reasonable to say, for example, that an instance is 75% in cluster A and 25% in cluster B.

When using this algorithm, we constructed training sets in a variety of ways, some which included all instances in the training set at least once, some which resampled the training set to adjust the class balance. When the classifiers were built and it came time to output a prediction, the probability that a given test instance belonged to the target class was calculated as the sum of the products of cluster membership (the proportion to which it belongs to the cluster) and the prediction from that cluster (the output from its classifier). The parameters for separating the data in these experiments can be found in Appendix A.

We did perform some a priori separations, divisions of training and test sets made from domain knowledge rather than clustering algorithms. Our most significant experiment in this direction was the use of geography to construct different classifiers based on census region. However, algorithmically informed separations composed the majority of our efforts in this category.

3.2.3 Feature and Value Selection

In some cases, too much information can be misleading - whether the information is inaccurate or simply unimportant, small chance patterns can imply structure where there is none, and an algorithm (or a human, for that matter) is never perfect at telling order from certain kinds of very lucky chaos. In addition, demanding information which may be unnecessary may make a classifier a burden on systems which then must continue collecting data which have long been abandoned in the name of backwards compatibility.

Our first major attempt with feature selection, or attribute removal, struck gold: by chopping off nineteen of the twenty-nine initial attributes in Phase One, we obtained a data set that was not only more comprehensible than the original but also had better performance with the same classification algorithm, and this is reflected in Table 3 below.

Table 3: The Manually Discovered Ten Attribute Data set

Hour	*Os_version*
Browser_family	*Browser_version*
Device	*Num_clicks_30*
Recency	*Frequency_hour*
Vendor	*Is_click*

This was an accident of a brute-force approach; we literally went down the list of attributes, removing each one and adding it back if its removal hurt performance. (That was how we stumbled on Naive Bayes; our problems with the long training time of logistic regression became far more apparent when we needed to judge dozens of different attributes). Our other experiments with correlation-based feature selection had a great deal of overlap with the results that led us to this ten-attribute set.

Subsequent attempts operated under the assumption that it was possible for individual values to be misleading; Naive Bayes has no way of differentiating between a strongly supported likelihood ratio with many observations and a weakly supported likelihood ratio with few. We considered the possibility that a single categorical value, such as a specific, possibly rare operating system version or a residence in a sparsely populated city, might occur in the training set once or twice and subsequently skew the results on the test set. We performed experiments for each categorical value in each attribute, replacing the target value with a missing one, and while we found some minute improvement, we found that the specific values we removed did not confer any performance benefit to the test set after being selected through cross-validation on the training set - there was no reliable predictor of performance for single-value removal.

While feature and value removal proved a mixed bag, the stroke of luck that was manual feature selection ended up increasing our AUC from 0.59 to 0.60, not to mention making future experiments worlds easier and more comprehensible. In Phase Two, we did further research on additional feature selection methods, and included the ten-attribute data set.

3.2.4 Feature Creation

In this portion of Phase One, we either combined or created attributes in the hopes that we would end up with a data set which better reflected reality in a way that Naive Bayes could, in its simple way, benefit from.

One avenue we explored was a brute-force method similar to the value removal mentioned in the previous section. In the ten-attribute set, we made each possible combination of the nominal, non-target attributes, and included that unified attribute in place of its components. For example, a browser's family and its version, normally two separate attributes, could be combined to create a single family/version attribute. We speculated that for some attributes, this could cut down on the skew created by highly correlated attributes while preserving the valuable information contained in them. It was even a return to a purer form of Bayesian reasoning; given enough data, a Bayesian agent can do better by combining all the available evidence into one "attribute", representing the state of the entire observable universe (or feature space). Taking this ideal to its extreme is unreasonable - there simply are not enough observations in the training set to provide reliable numbers for every possible combination of values. But we thought it might be useful for some subspaces of the feature space, places where every possible family/version combination, for example, is spoken for by a multitude of instances.

We found instead that while some attribute combinations produced promising gains in cross-validation performed on the training set, when we attempted to apply the same reasoning to the test set, we found that their performance there to be only weakly connected. In particular, distinctly poor performance on the training set appeared to indicate similarly poor performance

on the test set, but we could not reliably select the best performers on the training set and be certain that they would perform above average on the test set.

An alternative that we tried focused instead on the target attribute by creating a replacement. Instead of simply positive (1) or negative (0), we tried different subcategories within the positive and negative groups, carved out by the k-means clustering algorithm. In this experiment, the target attribute could be 0-1, 0-2, 0-3, 1-1, or 1-2, for example, instead of just 0 or 1. (We experimented with different numbers of clusters, as further documented in Appendix A). The logic behind this was that it was asking too much of Naive Bayes to decide which of two nebulous, poorly bounded masses in feature space a given instance could belong. Instead, by asking it to decide amongst more clearly restricted clusters, we would be giving it more specific targets to hit. The classifier would be trained on data with (as above) five possible class values rather than two, and it would assign a probability to each during testing. Later, the probability assignments would be combined, and the posterior probabilities for all the negative clusters and all the positive clusters would be summed separately and given as a single prediction between positive and negative. While this method did not yield any massive leaps in performance, it was interesting enough that we included a version of it - in which only negative instances are divided into subclasses - in Phase Two.

3.2.5 Algorithm Modification

Some of our most ambitious experiments were modifications to the Naive Bayes algorithm itself. While we did not deviate from the fundamentals of Bayesian reasoning, we did alter how we interpreted different pieces of evidence and different probabilities.

The first modification that we made concerned the Laplace smoothing mentioned in the introduction which prevents any probabilities from being equal to zero (Russell, 2010). In

artificially inflating the counts for each “event”, Naive Bayes introduces something of an egalitarian bias: a quantitative tendency towards treating all events as equally probable. We originally attempted to strengthen this bias, under the assumption that it would prevent poorly supported probabilities (such as one for an event which goes one way three out of four times, but has only four supporting observations) from becoming significant. We introduced a tunable parameter which would allow us to control the magnitude of the Laplace smoothing. Though we considered this idea independently of any existing research, we later found that the idea of tunable smoothing was already known in the literature as "m-estimation", for the parameter m (Tan, 2005). While we assumed that we would need to increase this parameter to reduce the outsized effects of poorly represented events, we instead found that lowering it by three orders of magnitude produced better performance. We called this modified algorithm "damped" Naive Bayes, according to our early efforts to "dampen" the impact of unreliable evidence.

. Instead of imposing equality on improperly supported events, we benefitted from letting the data speak for itself with fewer assumptions. This is the standard Naïve Bayes equation with Laplace smoothing, where X is a nominal attribute and |X| is the number of possible values of X:

$$p(X = x) = \frac{n(X = x) + 1}{n + |X|}$$

The “damped” Naïve Bayes that we introduced has *m* as a user-adjustable value is shown in the equation:

$$\square(X = x) = \frac{n(X = x) + m}{n + m|X|}$$

Another assumption that we found can hurt Naive Bayes' performance is the assumption that numeric attributes are normally distributed. Some implementations of the algorithm, including Weka's, have few options for how to handle numeric attributes, but they are limited to a choice of which distribution to select. Instead of letting Naive Bayes keep only the mean and standard deviation from an attribute, we decided to treat numeric attributes like nominal ones: each probability would be determined by counting how many instances matched a value.

For numeric attributes with few possible values - such as the hour of the day or even an individual's age - it is feasible to simply count how many of each value there are. There is enough data for every hour or every age to be well-represented. But this loses some information - it makes the assumption that 44-year-olds and 45-year-olds must be considered as differently as 18-year-olds and 90-year-olds - and just does not work for numeric attributes with more possible values. So instead, we considered every value that fell within some range of a target a match; 44-year-olds were now lumped with ages 39 to 49, 45-year olds with 40 to 50. There was overlap, and it was distinct from merely discretizing the attribute; each value had a slightly different probability, and the change was gradual - no sharp borders between groups. For a range, we used the standard deviation of the attribute multiplied by some constant (on which we experimented later), according to the following formula:

$$p(X = x) = \frac{n(x - c\sigma < X < x + c\sigma)}{n}$$

This treatment of numeric attributes allowed us to pursue one of our stranger experiments: event combination. Instead of treating each attribute as independent, we could instead test different combinations of values at the moment of classification, allowing us to unite some values into a single event while keeping others separate. Internet Explorer would be

considered separately from its version of 6.0, for example, but the combination of Firefox and version 35.0.1 could be used to get a more accurate picture. Again, the Bayesian ideal is one in which all events are combined into a single observation about the state of the observable universe. Unlike the simple combination of attributes, this would allow the classifier to use the probabilities of combined events if and only if they were well-supported, and would not prevent it from doing so if similar combinations happened to be only sparsely supported.

Ultimately, this approach and the treatment of numeric attributes described above were interesting, but did not make it into Phase Two because they were written outside of Weka - while tunable Laplace smoothing could be easily worked into the existing code base, the other two had to be written from scratch. For this reason, they were slow and unwieldy, and did not output results in a format consistent with Weka; when it came time to move on to large-scale testing, we believe that we made the right decision in leaving them behind.

3.3 Phase Two

In Phase Two, we conducted more rigorous experiments on a significantly larger data set from a different source in the same domain. This data, which consisted of twenty-seven non-trivial attributes and four and a half million instances, was taken from an iOS application during six days in February 2015. Using some of the methods which showed promise in Phase One, we designed five experiments, each of which was repeated on five unique data sets derived from the original.

3.3.1 Data Set Variants

In addition to the original data set (referred to as the Base data set), with twenty-seven attributes including the target, we attempted to prepare four other data sets. First, a variation which replaced the existing geographical attributes with a region attribute, placing each data

instance in the Northeast, South, Midwest, or West, which we refer to as the Domain data set. Second, a minimal subset of attributes selected by Weka's CFS Subset Evaluation algorithm (referred to as the CFS data set) - browser family and the number of recent clicks were the only two which made the cut (Witten, 2011). Third, we attempted to get some useful results of applying the Principal Components Analysis algorithm in Weka to condense the available data into a lower-dimensional feature space (Tan, 2005).This approach failed due to the prohibitive memory demands of running Principal Components Analysis with a large number of categorical attributes with large numbers of values. Finally, we created a version with the same manual selection of ten attributes that we found by dumb luck in Phase One (referred to as the Manual data set).This manual selection of attributes can be seen in Table 3 in section 3.2.3 Feature and Value Selection.

3.3.2 Experiments

We initially planned for five experiments one each data set. We began with one unmodified Naive Bayes run (the Baseline experiment) and one Naive Bayes run with Laplace smoothing turned down to one percent (the Damping experiment); these were the simple runs. Two involved the separation of the training and test sets into different data sets; one with a k-means discrete clustering for k = 3 (the K-Means experiment), and one with probabilistic clustering supported by the E-M algorithm. We also attempted a division of the negative instances into a trio of subclasses (the Subclass experiment), as covered in the section 3.2.4 Feature Creation.

Finally, as a pair of last-minute additions, we added experiments which dealt with the day of the week (the Day experiment) and the data carrier of each instance (the Carrier experiment).

In the former, we partitioned each training and test set into weekday and weekend groups, and in the latter, we divided each training and test set into ten groups –eight for the top eight carriers, one for the missing values, and one for everyone else.

4 Results

We used the results from Phase One to guide our experimental design for Phase Two. In this section we choose to present and analyze only the results from Phase Two.

In Phase Two, we focused on four summary statistics: AUC, recall, precision, and the harmonic mean of precision and recall, called the F1-score. The results for AUC are shown in Table 43 below:

Table 4: AUC Results

		Data Set			
		Base	CFS	Domain	Manual
Experiment	Baseline	0.585	0.531	0.587	0.586
	Carrier	0.560	0.558	0.566	0.579
	Damping	0.583	0.531	0.590	0.586
	Day	0.579	0.539	0.580	0.585
	EM	0.587	0.530	0.591	0.565
	K-Means	0.576	0.530	0.576	0.585
	Subclass	0.561	0.531	0.562	0.559

Additionally, we recorded the three threshold-dependent summary statistics - recall, precision, and F1-Score - for one thousand evenly spaced threshold for each experiment and data set.

The summary statistics for the base data set, which included all 27 attributes from Phase Two, have a few distinctive features. First, as demonstrated by the F1-Score and precision graphs in Figure 8, the EM experiment is an outlier, with very different behavior from the other

experiments. Second, recall dropped dramatically at a very low threshold, which is common to all data sets. It would appear that most instances were assigned a low probability of being positive, even those which were actually positive.

Third, precision leveled off fairly early, indicating that the mixture of true and false positives was relatively stable at all but the lowest thresholds. This excludes the EM experiment, which was an odd entity.

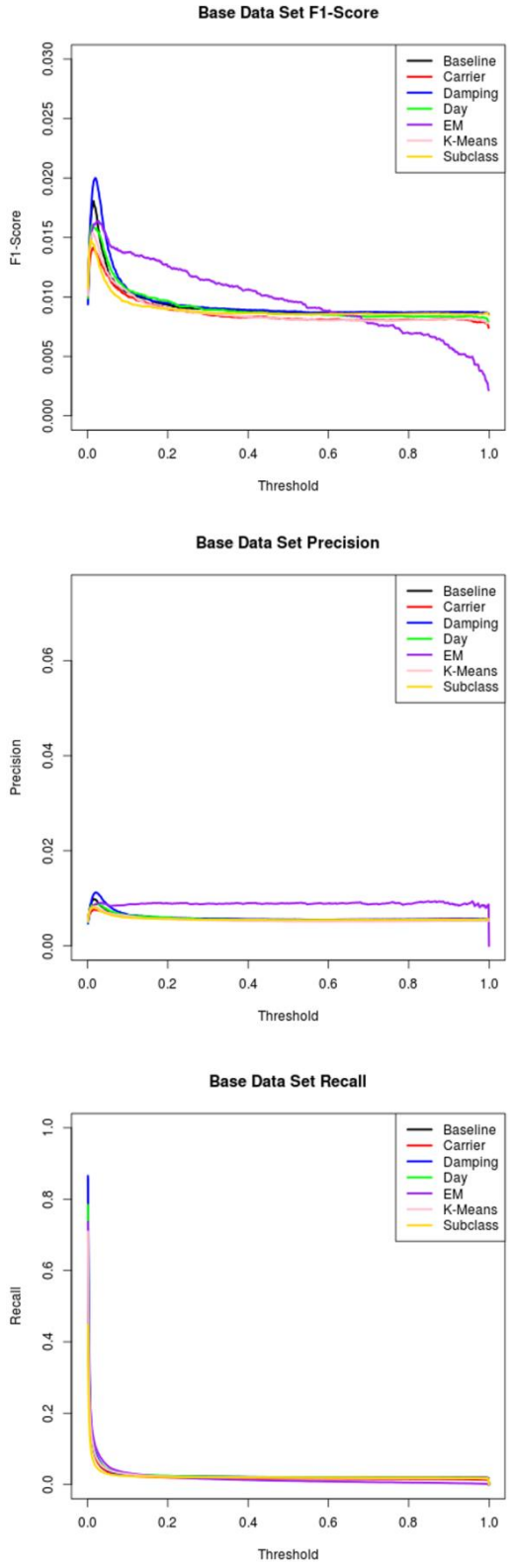


Figure 8: F1 Score, Precision, and Recall from the Base Data Set

Graphs of the summary statistics for the data set with attributes selected by CFS are blocky and full of corners, as seen in Figure 9 below. This is due to the fact that the feature space, which included one nominal and one numeric attribute, each with few possible values, was very small, and the limited number of possible instances in turn limited the number of possible probability outputs.

These experiments tended to be stricter than those performed on the base data set, with higher precision but lower recall. Once again, the EM experiment stands out, but not to the same extent. Of note is the K-Means experiment, which maintains an F1-Score around 0.010 into higher thresholds than any of its peers, which appears to be largely due to a solid maximum precision that is the highest of the entire data set. However, F1-Scores for the data set as a whole ultimately underperform compared to other data sets, with smaller upper and lower bounds.

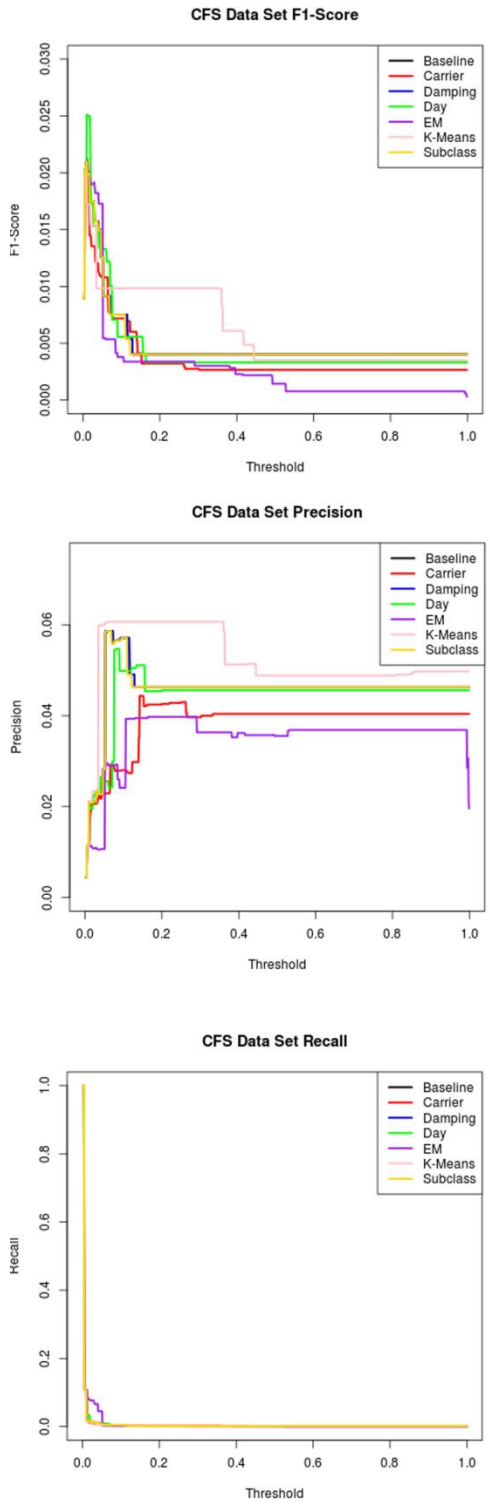


Figure 9: F1 Score, Precision, and Recall from the CFS Data Set

While the data set which included domain knowledge about geography in some cases outperformed the base data set (as is shown in the tables later in this section), the shapes of its curves were, for the most part, not notably different from those of the base data set, as can be seen in Figure 10. This is unsurprising; the domain data set differed least from the base data set, as only five attributes were removed and one added.

However, we do see some interesting behavior from our friendly neighborhood outlier, the EM experiment. Where in the base data set its precision remained relatively high for most thresholds and then dropped, here it seems to grow slowly and then increase suddenly at the end. This is encouraging behavior; it implies that the actual positives are systematically being assigned higher probability at a greater rate than in other experiments. While this may sound like a trivial achievement, it does stand out from its peers.

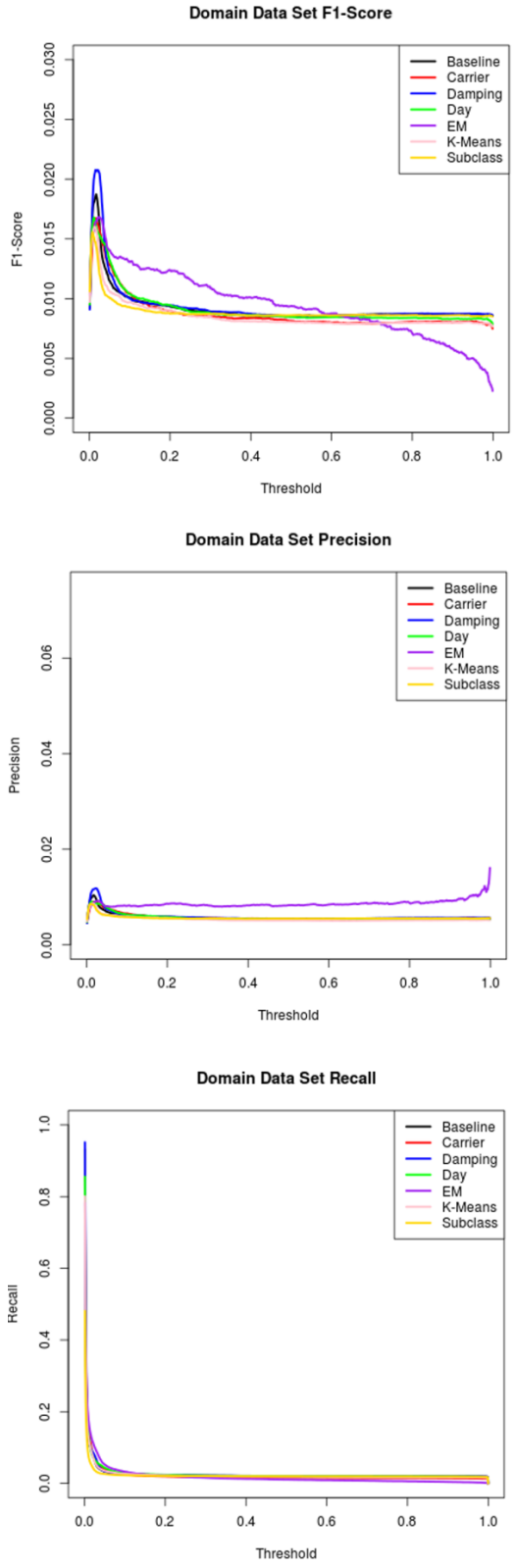


Figure 10: F1 Score, Precision, and Recall from the Domain Data Set

For the manually trimmed data set, which consisted of only ten attributes including the target, once again EM was the experiment to watch in Figure 11 below. As in the domain data set, it maintained a higher precision than its peers with a sharp rise at the end, though its behavior was not quite as steady. It did have a more impressive precision advantage over its peers, however, which is reflected in the graph of F1-Score graph, where the EM experiment achieves the highest maximum of any experiment in any data set and clearly outperforms its peers for a wide range of thresholds.

Both the Subclass experiment and the Damping experiment joined EM's sharp rise in precision at high thresholds, though this this rise was accompanied by a drop in recall sufficient to make the F1-Scores fairly ordinary.

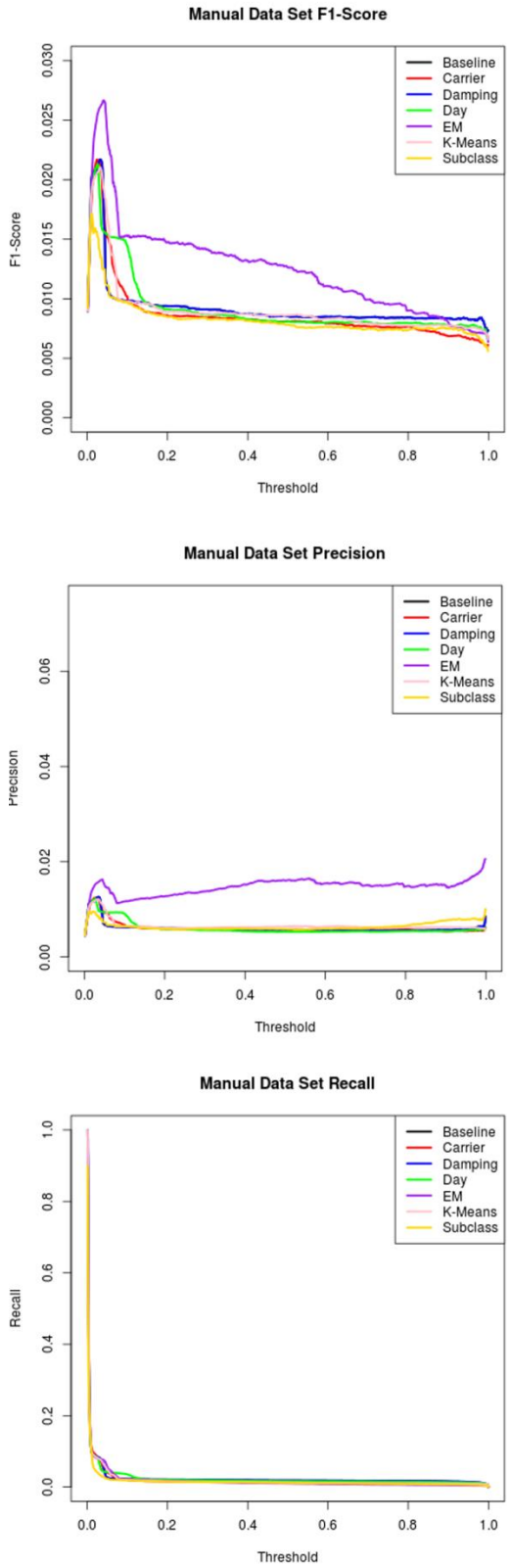


Figure 11: F1 Score, Precision, and Recall from the Manual Data Set

Finally, we examined the behavior of these three metrics at certain thresholds: those thresholds which maximized precision, and those thresholds which maximized F1-Score. (We considered including those thresholds which maximized recall, but were reminded that recall is trivially easy to maximize by setting the threshold to 0.0, which is not useful). The thresholds which do so may differ from experiment to experiment, and as such are included here in Table 54 below; however, the "goodness" of these thresholds is not a point of our study.

We note that as the positive instances composed only 0.45% of the total, a large random sample would have a precision of 0.0045. If this sample included the entire data set (in which case the recall would trivially be 1), it would have an F1-Score of 0.0090. These are the most natural, albeit unambitious, baselines for comparison.

Table 5: Data Sets and Threshold Statistics

Data Set/Experiment	Threshold which maximizes Precision	Precision at Threshold	Recall at Threshold	F1-Score at Threshold
Base/Baseline	0.015	0.010	0.101	0.018
Base/Carrier	0.018	0.008	0.084	0.014
Base/Damping	0.021	0.011	0.087	0.020
Base/Day	0.023	0.009	0.075	0.016
Base/EM	0.849	0.009	0.005	0.007
Base/K-Means	0.018	0.008	0.089	0.015
Base/Subclass	0.016	0.008	0.062	0.014
CFS/Baseline	0.058	0.057	0.005	0.009
CFS/Carrier	0.142	0.044	0.002	0.004
CFS/Damping	0.053	0.059	0.005	0.009
CFS/Day	0.080	0.055	0.004	0.007

CFS/EM	0.163	0.040	0.002	0.003
CFS/K-Means	0.058	0.061	0.005	0.010
CFS/Subclass	0.056	0.059	0.005	0.009
Domain/Baseline	0.017	0.010	0.095	0.019
Domain/Carrier	0.017	0.009	0.100	0.017
Domain/Damping	0.022	0.012	0.085	0.021
Domain/Day	0.012	0.009	0.117	0.017
Domain/EM	0.999	0.016	0.001	0.002
Domain/K-Means	0.014	0.009	0.103	0.016
Domain/Subclass	0.009	0.009	0.084	0.015
Manual/Baseline	0.035	0.013	0.076	0.022
Manual/Carrier	0.031	0.012	0.076	0.021
Manual/Damping	0.035	0.013	0.076	0.022
Manual/Day	0.027	0.012	0.080	0.021
Manual/EM	0.998	0.021	0.004	0.007
Manual/K-Means	0.033	0.012	0.077	0.021
Manual/Subclass	0.999	0.010	0.004	0.006

As shown above in Table 54 of F1-Score, the EM experiment on the Manual data set had the highest maximum among the experiments. The majority of maximum F1-scores were more than twice as large as the F1-Score of a random sample, which would be 0.009.

5 Conclusion

We found and described a variety of approaches, not all of which were successful, but many of which were worth examining further. In Phase One, we documented a series of unusual ideas in the hopes that, even if they did not suit our purposes, they might provide inspiration later. In Phase Two, we were able to present a number of options to suit a number of potential trade-offs between precision and recall.

Some experiments were able to, at their most discriminating points, improve precision values more than ten-fold over random sampling, though their recall values at such thresholds were discouraging. In light of these results, it may be more appropriate to frame our approaches as methods of reliably identifying a promising select few true positives rather than being able to correctly identify a large proportion of true positives.

Finally, though the vast majority of our Phase One experiments did not bear desired increases in AUC, and some of our Phase Two results were less than impressive, we are hopeful that our explorations into the territory of machine learning experimentation on a data set of 4.5 million data instances with a drastically skewed target attribute will help guide future research into this application, if in no way other than telling them where not to go. We are also proud to present our sponsor with a comprehensive array of approaches, choices and trade-offs, and robust evaluation of each one of them..

Bibliography

Peng, F., Schuurmans, D., & Wang, S. (2004). Augmenting naive Bayes classifiers with statistical language models. *Information Retrieval, 7*(3-4), 317-345.

Menard, Scott W. (2002). *Applied Logistic Regression* (2nd ed.). SAGE. ISBN 978-0-7619-2208-7.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009); The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1.

Witten, I., & Frank, E. (2011).*Data mining practical machine learning tools and techniques* (3rd ed.). Burlington, MA: Morgan Kaufmann.

Tan, P, Steinbach, M., & Kumar. V. (2005). *Introduction to Data mining* (1st ed.). Harlow: Addison-Wesley.

The R Project for Statistical Computing. (n.d.). Retrieved April 30, 2015, from http://www.r-project.org/

Evidence-Based Diagnosis. (n.d.). Retrieved April 30, 2015, from http://omerad.msu.edu/ebm/Diagnosis/Diagnosis6.html

Russell, Stuart; Norvig, Peter (2010). *Artificial Intelligence: A Modern Approach* (2nd ed.). Pearson Education, Inc. p. 863.

Appendix A: Phase One Experiments

Experiment 1:

- 65% train/35% test split
- Training set used to create EM clusters
- Cluster labels applied to instances of test set
- Training and test sets split based on cluster label
- Training sets used to generate logistic regression models tested on appropriate test sets

Results:

Cluster	Training Set Size	Test Set Size	AUC
1	273	186	0.517
2	3334	1776	0.504
3	1432	821	0.538
4	8871	4708	0.517
All (Weighted)	13910	7491	0.516

Experiment 2:

- 65% train/35% test split
- Training set used to create EM clusters
- Cluster labels applied to instance of test set
- Training and test sets split based on cluster label
- Training sets used to generate naive Bayes models tested on appropriate test sets

Results:

Cluster	Training Set Size	Test Set Size	AUC
1	273	186	0.572
2	3334	1776	0.543
3	1432	821	0.465
4	8871	4708	0.541
All (Weighted)	13910	7491	0.534

Experiment 3:

Training:

- 65% train/35% test split
- Training set used to create a Naive Bayes classifier (Model M0) which targets class membership in “is_click”
- Predictions made from M0 on training set
- New attribute added to training set based on error from M0 predictions (error = {yes, no})
- Attribute “is_click” temporarily removed
- Training set used to create a Naive Bayes classifier (Model M1) which targets class membership in “error”
- Predictions made from M1 on training set
- Training set split into two sets: “classification=yes” and “classification=no” (predicted error, not actual)
- Attribute “is_click” restored
- Training set “error=yes” used to create a Naive Bayes classifier (Model M2) which targets class membership in “is_click”

Testing:

- Test set is run through M1 and split into two sets: “classification=yes” and “classification=no” (predicted error, not actual
- Test set “classification=no” is evaluated in M0 and test set “classification=yes” is evaluated in M2

Results:

Model	Test Set Size	AUC
M0	7174	0.566
M1 (not terminal)	7491	0.825
M2	319	0.532
Total (weighted M0 + M2)	7493	0.565

Note: due to instances in the test set which were identical except for their class values, two additional testing instances were created in merging the “is_click” attribute back into the test set. Additionally, it may be worthwhile to redo this experiment to guard against error.

Constructed model M0 on training set

Added “error” attribute from M0 error to training set

Removed “is_click” attribute from training set

Constructed model M1 on training set w/”error” attribute and w/o “is_click” attribute

Restored “is_click” attribute to training set

Split training set into “error=yes” and “error=no” (actual error, not predicted)

Constructed model M2 on training set “error=no”, constructed model M3 on training set “error=yes”

Ran test set through M0 and obtained AUC value

Added “error” attribute from M0 error to test set

Removed “is_click” attribute from test set

Ran test set through M1 and obtained AUC value

Added “classification” attribute from M1 to test set

Restored “is_click” attribute to test set

Split test set into “classification=yes” and “classification=no” (predicted error, not actual)

Ran test set “classification=no” through M2 and obtained AUC value

Ran test set “classification=yes” through M3 and obtained AUC value

Results:

Model	Target	Terminal Model(s)?	Test Set	AUC
M0	is_click	false	7491 (all)	0.573
M1	error (from M0)	false	7491 (all)	0.825
M2	is_click	true	7174 (M1 predicted no)	0.558
M3	is_click	true	317 (M1 predicted	0.500

			yes)	
Total (weighted M2 + M3)	is_click	true	7491 (all)	0.556

Note: Based on the fact that AUC(M0) > AUC(Total), we conclude that this particular method of classification (which does not include preprocessing) is not suitable for this data. While further iterations might increase the AUC of the instances currently handled by M3, the performance of M2, which handles the majority of instances, restricts the maximum weighted AUC to 0.576, which is not significantly better than M0.

Experiment 4:

- As above, but with 10-attribute training and test sets instead

Results:

Model	Target	Terminal Model(s)?	Test Set	AUC
M0	is_click	false	7491 (all)	0.597*
M1	error (from M0)	false	7491 (all)	0.669
M2	is_click	true	7243 (M1 predicted no)	0.575
M3	is_click	true	248 (M1 predicted yes)	0.489
Total (weighted M2 + M3)	is_click	true	7491 (all)	0.572

* Model M0 performed with an AUC of 0.595 on data instances which were tested on M2.

Experiment 5:

- 65% training, 35% testing
- removed Zipcode
- Ran Correlation attribute evaluator on data set
- took out any attributes with a correlation less than .005
    - removed was Lat, area_code. day. day_of_week, data, keyword, user_age, os_family, lng, user_gender
- removed “is_click” attribute
- Ran EM, default parameters
- restored “is_click” attribute
- added cluster to data set
- separated clusters in R

Results:

Cluster	Training Set Size	Test Set Size	AUC
1	334	172	0.613
2	351	180	0.526
3	8199	4415	0.513
4	5026	2724	0.501
Total	13910	7491	0.511

Experiment 6:

- Remove attributes one by one, progressing down the list and testing whether or not the AUC increases without said attribute (testing on the full data set rather than a train/test split)
- Conduct 10-fold cross-validation with a Naive Bayes classifier

Results:

Attributes	AUC
hour, os_version, browser_family, browser_version, device, vendor, num_clicks_30, recency, frequency_hour, is_click	0.6

Experiment 7:

- Use the ten attributes gathered by testing the effect of attribute selection on Naive Bayes (hour, os_version, browser_family, browser_version, device, vendor, num_clicks_30, recency, frequency_hour, is_click)
- Split the attribute-selected data into a training set and a test set
- Run the EM algorithm on the training set and save the cluster assignment model
- Assign cluster labels to instances in both the training set and the test set
- Split the training and test sets by cluster labels
- Train one Naive Bayes model on each cluster-separated training subset, and test the models on the cluster-separated test subset with the corresponding cluster label

Results:

Cluster	Train Size	Test Size	AUC
No Clustering (Baseline)	13910	7491	0.610
1	5062	2725	0.564
2	4318	2278	0.557
3	4350	2448	0.548
All (Weighted)	13910	7491	0.557

Experiment 8:

- Use the ten attributes gathered by testing the effect of attribute selection on Naive Bayes (hour, os_version, browser_family, browser_version, device, vendor, num_clicks_30, recency, frequency_hour, is_click)
- Split the attribute-selected data into a training set and a test set
- Determine the effect of the size of the training set on the AUC by using a supervised Resample filter

Results:

Size of Training Set	AUC	Mean AUC	Standard Deviation AUC
13910	0.610	0.610	NA
12159	0.599	0.600	0.0082
12159	0.590		
12159	0.602		
12159	0.610		
11128	0.608	0.601	0.0127
11128	0.614		
11128	0.585		
11128	0.598		
9737	0.596	0.601	0.0078
9737	0.611		
9737	0.594		

9737	0.604		
5564	0.597	0.586	0.0087
5564	0.577		
5564	0.589		
5564	0.582		
1391	0.556	0.568	0.0156
1391	0.568		
1391	0.558		
1391	0.590		

Experiment 9:

- Use the ten attributes gathered by testing the effect of attribute selection on Naive Bayes (hour, os_version, browser_family, browser_version, device, vendor, num_clicks_30, recency, frequency_hour, is_click)
- Split the attribute-selected data into a training set and a test set
- Run the FarthestFirst clustering algorithm on the training set and save the cluster assignment model
- Assign cluster labels to instances in both the training set and the test set
- Split the training and test sets by cluster labels
- Train one Naive Bayes model on each cluster-separated training subset, and test the models on the cluster-separated test subset with the corresponding cluster label

Results:

Cluster	Train Size	Test Size	AUC
No Clustering (Baseline)	13910	7491	0.597
Train 1, Test 1	7887	4241	0.547
Train 2, Test 2	6023	3250	0.565
Train 1&2, Test 1	13910	4241	0.557
Train 1&2, Test 1	13910	3250	0.583
Train 1, Test 1 and Train 2, Test 2 (weighted)	13910	7491	0.555
Train 1&2, Test 1 and Train 1&2, Test 2 (weighted)	13910	13910	0.568

Note: the weighted average AUC of the model trained on the entire training set but tested on two disjoint test sets is significantly below the AUC of the same model tested on the entire test set.

Experiment 10:

- Use the train_10.arff and test_10.arff data sets (a 65%/35% train/test split of the original data reduced to ten attributes: hour, os_version, browser_family, browser_version, device, vendor, num_clicks_30, recency, frequency_hour, is_click)
- Use R to randomly split the test_10.arff data set into two equally sized subsets, test_10_1.arff and test_10_2.arff
- Train a Naive Bayes classifier on train_10.arff and test it on both subsets

Results:

Test Set	Set Size	AUC
1	3745	0.601
2	3746	0.593
1 & 2 (Weighted Average)	7491	0.597
1 & 2 (Single Set)	7491	0.597

Experiment 11:

- Implement the AddClusterProbabilities filter, which takes a clustering algorithm or model and creates new attributes (one for each cluster) which reflect the probability of belonging to a specific cluster.
- Train an EM model on the training set (less the target attribute).
- Apply the AddClusterProbabilities filter to both the training set and the test set, using the model generated on the training set.
- Create 4n new training sets, 4 for each of the n clusters generated by the EM model, according to the following rules (using the training set with cluster probabilities):
    - First training set: add each instance to the training set a number of times equal to floor(cluster probability * 10); this is the non-inclusive set
    - Second training set: add each instance to the training set a number of times equal to ceiling(cluster probability * 10); this is the all-inclusive set (it contains each instance at least once)
    - Third training set: add each instance to the training set if and only if it the current cluster's probability is greater than that of another cluster (i.e. if that instance would be assigned to that cluster); this is the simple set
    - Fourth training set: as the first training set, but after applying the supervised Resample filter with replacement and with a bias towards a uniform class (at 100% sample size); this is the all-inclusive resampled set
- Train a Naive Bayes classifier on each training set and name the file accordingly (e.g. cluster1_all.model, cluster3_non.model, cluster2_simple.model)
- Generate a probability distribution from each model on every test instance
- For each type of training set, calculate the predicted probability of membership in is_click = 1 of each instance in two ways:
    - $p(is-click\ =\ 1)\ =\ \Sigma p(cluster\ =\ i)\ *\ p(is-click_{predict\_on-i})$ takes into account the probability that the given instance belongs to a particular cluster and the confidence with which that particular cluster's model has predicted that they will have an is_click value of 1; this yields the fine prediction
    - $p(is-click\ =\ 1)\ =\ \Sigma p(cluster\ =i)\ *\ (p(is-click_{prediction-i})\ >\ 0.5)$ may dampen noise which would otherwise be introduced by precisely following the predicted probabilities; this yields the coarse prediction
- Calculate the AUC for each training type and prediction type

Results:

Training Type	AUC (coarse)	AUC (fine)
All-Inclusive	0.505	0.602

Non-Inclusive	0.520	0.571
Simple	0.529	0.573
All-Inclusive (Resampled)	0.583	0.599

Experiment 12:

- Use Weka's logistic regression to build a classifier on the train_10 data set.
- Test it on the test_10 data set.
- Build another logistic regression classifier on the train_10 data set after applying the supervised Resample filter with a bias towards a uniform class, replacement in sampling, and a 200% sample size.

Results:

Condition	AUC
No Resampling	0.590
Resampling	0.592

Note: From these results, we can conclude that the limited set of attributes selected by trial and error on Naive Bayes is also beneficial to Logistic Regression, though Naive Bayes still has superior performance.

Experiment 13:

- Construct Damped Naive Bayes class, which introduces virtual instances to each estimator in order to stabilize probability estimates, especially in cases in which there are few observations per symbol
- Run Damped Naive Bayes on the train/test split with a variety of values of M, the damping parameter, D, the supervised discretization flag, and U, the uniformity flag
    - When U is set, each symbol in an attribute receives a virtual instance of weight M; when U is not set, this weight of M is distributed among all symbols in the attribute (for a given attribute/class combination)

Results:

M	U	D	AUC
1.0	True	True	0.584
1.0	True	False	0.597
1.0	False	True	0.596
1.0	False	False	0.605
2.0	True	True	0.579
2.0	True	False	0.593
2.0	False	True	0.596
2.0	False	False	0.605
0.5	True	True	0.588
0.5	True	False	0.600
0.5	False	True	0.596
0.5	False	False	0.606

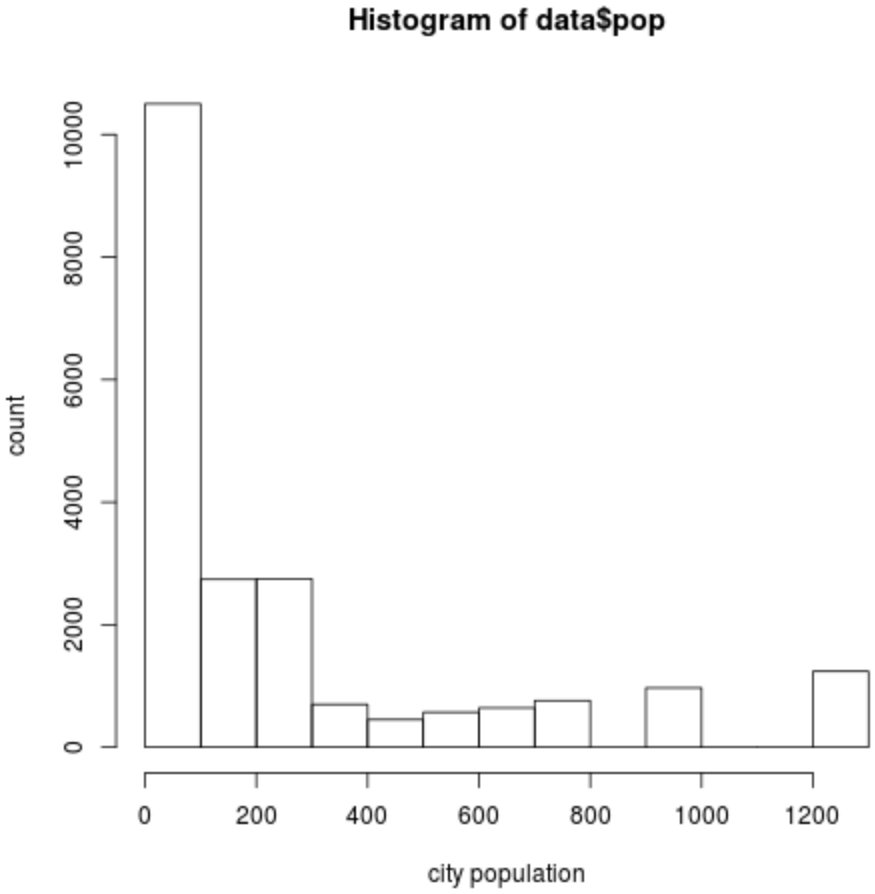
4.0	True	True	0.569
4.0	True	False	0.584
4.0	False	True	0.595
4.0	False	False	0.604
0.25	True	True	0.591
0.25	True	False	0.601
0.25	False	True	0.596
0.25	False	False	0.606
8.0	True	True	0.559
8.0	True	False	0.575
8.0	False	True	0.594
8.0	False	False	0.603
0.125	True	True	0.594
0.125	True	False	0.603
0.125	False	True	0.596
0.125	False	False	0.606

Experiment 14:

- Begin with the original 29-attribute data set
- For each city in the "city" attribute, count how many times that city occurs in the data
- Add a new attribute, "pop", which counts how many times a given instance's city occurs in the data
- Remove all attributes other than the "pop" attribute and the 10-attribute set found in previous experiments
- Perform cross-validation on the new data under different experimental conditions using Naive Bayes

Results:

Experimental Condition	AUC
Baseline (no pop attribute)	0.600
Pop as numeric attribute	0.599
Pop as supervised-discretized attribute (note: Naive Bayes supervised discretization created only one bin)	0.599
Pop as unsupervised-discretized attribute with 10 bins	0.599
Pop as unsupervised-discretized attribute with 7 bins (number found by equal-width specification)	0.600


Histogram of data$pop
count
0
2000
4000
6000
8000
10000
0
200
400
600
800
1000
1200
city population

Experiment 15 [FAILURE]:

- Using the same population counts added in the above experiment, split the data into subsets based on population size. Keep the ten attributes discovered in previous experiments, and possibly the pop attribute, if desired.
- Include instances with no population count (because they have NA values for city) in all subsets. They account for less than one-half of one percent of the original data set.
- Test each subset in its own cross-validated Naive Bayes model.
- Experiment with different splits and record the weighted-average AUC values, as well as the individual AUC values.
- **EXPERIMENT VOID; NA VALUES REPLICATED**

Results:

~~Experiment~~	~~Include pop as attribute?~~	~~Range~~	~~Bin Size~~	~~Bin AUC~~	~~Weighted Average AUC~~
~~1~~	~~Yes~~	~~1 Inf~~	~~21401~~	~~0.599~~	~~0.599~~
~~2~~	~~No~~	~~1 Inf~~	~~21401~~	~~0.600~~	~~0.600~~
~~3~~	~~Yes~~	~~1 100~~	~~10571~~	~~0.607~~	~~0.597~~
		~~101 Inf~~	~~10899~~	~~0.588~~	
~~4~~	~~No~~	~~1 100~~	~~10571~~	~~0.608~~	~~0.598~~
		~~101 Inf~~	~~10899~~	~~0.589~~	
~~5~~	~~Yes~~	~~1 50~~	~~7906~~	~~0.604~~	~~0.593~~
		~~51 Inf~~	~~13564~~	~~0.587~~	
~~6~~	~~No~~	~~1 50~~	~~7906~~	~~0.606~~	~~0.594~~
		~~51 Inf~~	~~13564~~	~~0.587~~	
~~7~~	~~Yes~~	~~1 100~~	~~10571~~	~~0.607~~	~~0.603~~

		101-300	5560	0.595	
		300-Inf	5408	0.603	
8	No	1-100	10571	0.608	0.604
		101-300	5560	0.595	
		300-Inf	5408	0.604	
9	Yes	1-100	10571	0.607	0.607
		101-200	2809	0.598	
		201-300	2820	0.620	
		301-Inf	5408	0.603	
10	No	1-100	10571	0.608	0.608
		101-200	2809	0.601	
		201-300	2820	0.619	
		301-Inf	5408	0.604	
11	Yes	1-50	7906	0.602	0.611
		51-100	2734	0.653	
		101-200	2809	0.598	
		201-300	2820	0.620	
		301-Inf	5408	0.603	
12	No	1-50	7906	0.603	0.611

		~~51-100~~	~~2734~~	~~0.651~~	
		~~101-200~~	~~2809~~	~~0.601~~	
		~~201-300~~	~~2820~~	~~0.619~~	
		~~301-Inf~~	~~5408~~	~~0.604~~	

Note: I noticed around the four-bin mark that the AUC values for the different classes seemed to start diverging - there could be a difference of nearly 0.10 between the majority and minority class. All AUC values reported here are weighted averages reported by Weka, though I am mystified as to how a two-class classifier can have different AUC values for each class. ADDENDUM: I now suspect that this is related to the inclusion of NA values, which somehow acquired NA values for every attribute, including class.

Experiment 16:

- Using the same population counts added in the above experiment, split the data into subsets based on population size. Keep the ten attributes discovered in previous experiments, and possibly the pop attribute, if desired.
- Take instances with no population count (because they have NA values for city) into its own subset. They account for less than one-half of one percent of the original data set.
- Test each subset in its own cross-validated Naive Bayes model.
- Experiment with different splits and record the weighted-average AUC values, as well as the individual AUC values.

Results:

Experiment	Include pop as attribute?	Split	Bin Size	Bin AUC	Weighted Average AUC
1	Yes	1-Inf	21332	0.597	0.596
		NA	69	0.217	
2	No	1-Inf	21332	0.598	0.597
		NA	69	0.217	
3	Yes	1-100	10502	0.604	0.588
		101-Inf	10830	0.574	
		NA	69	0.217	
4	No	1-100	10502	0.605	0.588
		101-Inf	10830	0.574	
		NA	69	0.217	
5	Yes	1-50	7837	0.615	0.590

		51-Inf	13495	0.578	
		NA	69	0.217	
6	No	1-50	7837	0.616	0.590
		51-Inf	13495	0.577	
		NA	69	0.217	
7	Yes	1-50	7837	0.615	0.591
		51-200	5405	0.576	
		201-Inf	8090	0.582	
		NA	69	0.217	
8	No	1-50	7837	0.616	0.592
		51-200	5405	0.576	
		201-Inf	8090	0.583	
		NA	69	0.217	
9	Yes	1-50	7837	0.615	0.588
		51-200	5405	0.576	
		201-300	2751	0.578	
		301-Inf	5339	0.569	
		NA	69	0.217	
10	No	1-50	7837	0.616	0.588

		51-200	5405	0.576	
		201-300	2751	0.578	
		301-Inf	5339	0.570	
		NA	69	0.217	
11	Yes	1-50	7837	0.615	0.581
		51-100	2665	0.570	
		101-200	2740	0.529	
		201-300	2751	0.578	
		301-Inf	5339	0.569	
		NA	69	0.217	
12	No	1-50	7837	0.616	0.582
		51-100	2665	0.571	
		101-200	2740	0.531	
		201-300	2751	0.578	
		301-Inf	5339	0.570	
		NA	69	0.217	

Conclusion:

Binning instances based on the frequencies of their cities is not a useful method of separating instances.

Experiment 17:

- Using US Census Bureau regions and divisions data (as found at http://www.census.gov/geo/maps-data/maps/pdfs/reference/us_regdiv.pdf), assign data instances to regions and divisions based on US State (extranational data will be assigned to region "Foreign" and division "Foreign")
- Test a variety of conditions to determine whether keeping the region and/or division attributes (or dividing into subsets based on those attributes) is useful, using the ten-attribute data set as a base
- Use Naive Bayes with ten-fold cross-validation

Results:

Condition	Keep Region?	Keep Division?	Split	Size	Split AUC	Weighted Total AUC
Baseline	No	No	None	21401	NA	0.600
R, D as attributes	Yes	Yes	None	21401	NA	0.600
R as attribute	Yes	No	None	21401	NA	0.599
D as attribute	No	Yes	None	21401	NA	0.600
Split into subsets based on Region	No	Yes	Northeast	1533	0.507	0.584
			Midwest	2891	0.556	
			West	5864	0.589	
			South	11041	0.601	
			NA	72	0.199	
Split into	No	No	Northeast	1533	0.511	0.582

subsets based on Region			Midwest	2891	0.551	
			West	5864	0.588	
			South	11041	0.600	
			NA	72	0.199	
Split into subsets based on Division	No	No	East North Central	2303	0.555	0.570
			East South Central	1100	0.549	
			Middle Atlantic	1379	0.486	
			Mountain	1835	0.544	
			New England	154	0.631	
			Pacific	4029	0.590	
			South Atlantic	4832	0.581	
			West North Central	588	0.503	
			West South Central	5109	0.599	
			NA	72	0.199	

Conclusion: geographical splitting at the granularity of region or division is counter-effective.

Experiment 18:

- Use a 65%/35% train/test split on the ten-attribute data.
- Declare Weka's Naive Bayes algorithm as a baseline, but construct a Naive Bayes algorithm which uses a different estimator for numeric attributes as follows: p(attribute = value) = n(attribute - c * stddev <= value <= attribute + c * stddev)/N, where c is some adjustable parameter which is consistent within each classifier.
- Train and test the resulting models with the same train and test subsets used for the baseline, varying the parameter c to maximize AUC.

Results:

Model Type	c-parameter	AUC
Weka	N/A	0.597
Homebrew	0.001	0.599
Homebrew	0.01	0.602
Homebrew	0.0625	0.603
Homebrew	0.125	0.601
Homebrew	0.25	0.602
Homebrew	0.5	0.600
Homebrew	0.75	0.596
Homebrew	1.0	0.596
Homebrew	1.5	0.594
Homebrew	2.0	0.594

Notes:

- The current design as of 1/12/15 uses lazy cached learning - each of 7490 test instances takes an average of 1/20th of a second to classify, compared to approximately ½ of a second without caching. This classification time may be improved upon later.

- Subsequent experiments with a small m-parameter (0.01 or 0.0001) for m-estimation (as attempted in previous experiments) with a c-parameter of 0.0625 yielded an AUC value of 0.610.
- The current design as of 1/13/15 rounds numeric values to the nearest one-hundredth of a standard deviation, which actually increases AUC very slightly and decreases average prediction time to about 1/100th of a second.

Experiment 19:

- Use the homebrew python Naive Bayes algorithm on a 65%/35% train/test split.
- Set the parameter for m-estimation to 1 (normal for Naive Bayes) and the parameter for numeric range inclusion to 0.0625.
- When predicting the class of a test datum, combine events as follows: find the pair of events A = a, B = b such that the support (number of occurrences) for their intersection A = a & B = b is greater than any other pair. Remove A = a, B = b from the list of events and replace it with A = a & B = b. Stop if no intersection A = a & B = b has a support count greater than some minimum support.
- When evaluating candidate events for combination, do not bother to collect the support count for the intersection if either of the candidate events have a support count less than the minimum.

Results:

Minimum Support	AUC
Baseline (no combination)	0.602
100	0.595
200	0.597
500	0.594
1000	0.580
1500	0.581
2000	0.592
2500	0.596
3000	0.595

Experiment 20:

- Use the homebrew python Naive Bayes algorithm on a 65%/35% train/test split.
- Set the parameter for m-estimation to 1 (normal for Naive Bayes) and the parameter for numeric range inclusion to 0.01.
- When predicting the class of a test datum, combine events as follows: find the pair of events A = a, B = b such that the proportion between the product of the likelihood ratios for the separate events and the likelihood ratio of the intersection A = a & B = b is most extreme. Merge those events.
- Remove A = a, B = b from the list of events and replace it with A = a & B = b. Stop if no intersection A = a & B = b has a support count greater than some minimum support.
- When evaluating candidate events for combination, do not bother to collect the support count for the intersection if either of the candidate events have a support count less than the minimum.

Results:

Minimum Support	AUC
Baseline (no combination)	0.602
100	0.601
200	0.597
300	0.596
400	0.596
500	0.588
600	0.583
700	0.581
800	0.582
900	0.581

1000	0.580
1200	0.582
1400	0.575
1600	0.584
1800	0.584
2000	0.586
2500	0.584
3000	0.583
3500	0.588
4000	0.589

Experiment 21:

- Use the homebrew python Naive Bayes algorithm on a 65%/35% train/test split.
- Set the parameter for m-estimation to 1 (normal for Naive Bayes) and the parameter for numeric range inclusion to 0.01.
- When predicting the class of a test datum, combine events as follows: find the pair of events A = a, B = b such that the proportion between the product of the likelihood ratios for the separate events and the likelihood ratio of the intersection A = a & B = b is most extreme. Merge those events.
- Remove A = a, B = b from the list of events and replace it with A = a & B = b. Stop after some number of iterations, or if no intersection A = a & B = b has a support count greater than 10.
- When evaluating candidate events for combination, do not bother to collect the support count for the intersection if either of the candidate events have a support count less than the minimum.

Results:

Maximum Iterations	AUC
Baseline (no combination)	0.602
1	0.596
2	0.596
3	0.591
4	0.587

Experiment 22[DISREGARD]:

- Repeat the experiment below for the 29-attribute and the 10-attribute data sets.
- Split the data set into positive and negative instances.
- Build clustering models on the positive and negative sets and save them. Record the algorithms and parameters used for these models, which may be different for each set.
- Apply the model built on the positive set to the positive set, and apply the model built on the negative set to the negative set. Give the cluster labels names which reflect whether they are built from positive or negative instances.
- Rejoin the positive and negative sets. Remove the class variables, letting the cluster labels (which have the old class information in their names) become the class.
- Train and test a Naive Bayes model using leave-one-out cross-validation.
- Record all AUC values and, if necessary, distributions calculated by the model, as alternative methods of calculating AUC may be necessary.

Results:

Experiment	Original Class Label	Algorithm	Cluster Class Label	Cluster AUC	Cluster Size	Weighted Total AUC
Base.10	Positive	None	Positive	0.598	1228	0.598
	Negative	None	Negative	0.598	20173	
Base.29	Positive	None	Positive	0.573	1228	0.573
	Negative	None	Negative	0.573	20173	
K-2-1.10	Positive	K-Means, with k=2	Positive-1	0.776	578	0.597
			Positive-2	0.741	650	
	Negative	None	Negative	0.587	20173	
K-2-1.29	Positive	K-Means, with k=2	Positive-1	0.758	412	0.568
			Positive-2	0.635	816	
	Negative	None	Negative	0.561	20173	

K-1-2.10	Positive	None	Positive	0.579	1228	0.943
	Negative	K-Means, with k=2	Negative-1	0.951	12750	
			Negative-2	0.989	7603	
K-1-2.29	Positive	None	Positive	0.578	1228	0.745
	Negative	K-Means, with k=2	Negative-1	0.780	6158	
			Negative-2	0.745	14015	

Predicted		Confusion Matrix Base.10	
Positive	Negative		
40	1188	Positive	Actual
294	19879	Negative	

Predicted		Confusion Matrix Base.29	
Positive	Negative		
65	1163	Positive	Actual
666	19507	Negative	

Predicted	Confusion Matrix K-2-1.10

Positive-1	Positive-2	Negative		
17	4	557	Positive-1	Actual
0	38	612	Positive-2	
160	280	19733	Negative	

Predicted		Equivalent Confusion Matrix K-2-1.10	
Positive	Negative		
59	1169	Positive	Actual
440	19733	Negative	

Predicted			Confusion Matrix K-2-1.29	
Positive-1	Positive-2	Negative		
25	6	381	Positive-1	Actual
3	49	764	Positive-2	
262	509	19402	Negative	

Predicted		Equivalent Confusion Matrix K-2-1.29	
Positive	Negative		
83	1145	Positive	Actual

771	19402	Negative	

Predicted			Confusion Matrix K-1-2.10	
Positive	Negative-1	Negative-2		
14	906	308	Positive	Actual
126	12404	40	Negative-1	
134	0	7469	Negative-2	

Predicted		Equivalent Confusion Matrix K-1-2.10	
Positive	Negative		
14	1214	Positive	Actual
134	19913	Negative	

Predicted			Confusion Matrix K-1-2.29	
Positive	Negative-1	Negative-2		
55	273	900	Positive	Actual
157	3568	2433	Negative-1	
376	1817	11822	Negative-2	

Predicted		Equivalent Confusion Matrix K-1-2.29	
Positive	Negative		
55	1173	Positive	Actual
533	19640	Negative	

Note: it was observed at this point that the AUC values exhibited significant differences based on clustering without showing any real improvement in the confusion matrices. It became clear that part of the AUC value was based on discovering distinctions within the "real" classes (e.g. distinguishing Negative-1 from Negative-2), which were not actually useful. This error became far more pronounced when the Negative class was divided into clusters, as the Negative clusters formed the majority of the instances. The decision was then made to alter the experimental protocol. **ALL DATA ABOVE SHOULD BE KEPT, BUT DISREGARDED.**

Experiment 23:

- Repeat the experiment below for the 29-attribute and the 10-attribute data sets.
- Split the data set into a training set (65%) and a test set (35%).
- Split the training set and test set into subsets of positive and negative instances, designated positive-train, negative-train, positive-test, and negative-test.
- Build clustering models on the positive-train and negative-train sets and save them. Record the algorithms and parameters used for these models, which may be different for each set.
- Apply the model built on the positive-train set to the positive-train and positive-test sets, and do similarly for the negative model and sets. Give the cluster labels names which reflect whether they are built from positive or negative instances.
- Reunite the positive-train and negative-train sets, and the positive-test and negative-test sets. Remove the class variables, letting the cluster labels (which have the old class information in their names) become the class.
- Train a Naive Bayes model on the reunited training set, and save it.
- Record the predicted class distribution of the model on the reunited test set. Compute the probability that each instance will belong to the "real" positive class as the sum of the probabilities that it will belong to each of the positive clusters.
- Calculate and record the AUC value for each experimental configuration.

Results:

Experiment	Original Class Label	Algorithm	Cluster Class Label	Cluster Size in Test Set	AUC	AUC with Resampling (Biased, 500% w/ replace)
base.10	Positive	None	Positive	430	0.597	0.604
	Negative	None	Negative	7061		
base.29	Positive	None	Positive	430	0.573	0.573
	Negative	None	Negative	7061		
em.10	Positive	EM	Positive-1	181	0.566	0.568

			Positive-2	109		
			Positive-3	140		
	Negative	EM	Negative-1	1404		
			Negative-2	1190		
			Negative-3	1943		
			Negative-4	2524		
k-equal.10	Positive	None	Positive	430	0.564	0.555
	Negative	K-means, with k=16 (approx. negative-to-positive ratio)	Negative-1	406		
			Negative-2	185		
			Negative-3	281		
			Negative-4	321		
			Negative-5	294		
			Negative-6	446		
			Negative-7	149		
			Negative-8	282		
			Negative-9	445		
			Negative-10	1332		
			Negative-11	1086		
			Negative-12	950		
			Negative-13	173		
			Negative-14	299		

			Negative-15	105		
			Negative-16	307		
em-plus.10	Positive	EM	Positive-1	181	0.520	0.522
			Positive-2	109		
			Positive-3	140		
	Negative	None	Negative	7061		
em-minus.10	Positive	None	Positive	430	0.580	0.582
	Negative	EM	Negative-1	1404		
			Negative-2	1190		
			Negative-3	1943		
			Negative-4	2524		

Note: it was determined after experiment em.10 that two different methods of calculating the probabilities for use in the ROC curve were practically equivalent, owing most likely to the fact that most predictions were very strong, with a typical instance being 95% certain of its belonging to a particular cluster. The two methods were as follows: P(Positive) = sum(P(Positive-i)), and P(Positive) = max(P(Positive-i))/(max(P(Positive-i)) + max(P(Negative-i))). The two methods returned AUC values for experiment em.10 which were identical to the least significant digit (0.001).

Note: following observation of the experiments em-plus.10 and em-minus.10, in which it was observed that individual clusters had high AUC values, it was hypothesized that the closeness of the two AUC values as calculated in the note above was due to the easy separability of the two clusters – there is no real danger of an instance being ambiguously between Positive-1 and Positive-2, for example. With this in mind, it would be wise to resume testing by multiple methods if the AUC of clusters should happen to go down below some rather obscene value, especially if non-globular clusters are used, which would make predictions among different clusters under the same "real" target more difficult.

Experiment 24:

- Remove a single value from a single nominal attribute in the ten-attribute data set, other than the class attribute.
- Perform 100-fold cross-validation on the resulting data set.
- Record the resulting AUC value.

Results:

Experiments performed: 354 single-value removals, 1 multiple-value removal

AUC Value	Attribute-Value
0.598	(Not replicated for brevity)
0.599	(Not replicated for brevity)
0.600	device-one, device-p769, device-sph-l900, browser_version-4.1, vendor-alcatel, device-lg730, os_version-2.3.3, device-vs410pp, device-sgh-t999, device-sgh-i437z, vendor-lg, device-sph-l300, device-ms840, device-sph-l710, browser_version-11.0.696.34, device-ls720, os_version-2.3.7, device-sgh-t399, device-other, device-l38c, browser_version-4.0, device-sm-g730a, device-sch-i110, device-sgh-t989, device-ls970, device-sch-r820, device-sgh-i337, device-sch-r530m, vendor-other
0.601	browser_version-30.0.0.0
...	...
0.608	(All Attribute-Value combinations named in the above cells)

Note: leave-one-out cross-validation gave an AUC value of 0.607, compared to 0.598 for the otherwise unmodified ten-attribute data set.

Attribute-Value	AUC
device-as680	0.599

device-sm-s765c	0.599
browser_family-chrome	0.599
device-sgh-i747	0.599
device-electrify	0.599
device-sm-n900a	0.599
device-x500	0.599
device-desire	0.599
device-galaxy nexus	0.599
device-droid x2	0.599
device-sgh-t759	0.599
device-a851l	0.599
device-sgh-i407	0.599
browser_family-silk	0.599
device-sgh-i997	0.599
device-one	0.6
device-sm-c105a	0.599
device-m660	0.599
device-sm-g870a	0.599
device-sch-r530c	0.599
browser_version-30.0.1599.105	0.599
device-sgh-t879	0.599
device-sgh-t599	0.599
device-gt-i8190	0.599
device-sm-n900t	0.599

device-r800xhttp	0.599
device-sgh-s959g	0.599
device-p870	0.599
device-sgh-m919n	0.599
device-sph-m820	0.599
device-ls995	0.599
device-sgh-t769	0.599
device-ls980	0.599
device-p769	0.6
device-as695	0.599
device-vs700	0.599
device-p659	0.599
device-sch-r890	0.599
browser_version-35.0.1916.141	0.599
device-u8651	0.599
device-h866c	0.599
device-u8687	0.599
device-sgh-i537	0.599
browser_version-0.0	0.599
device-sch-s720c	0.599
device-amaze 4g	0.599
vendor-zte	0.599
device-sgh-t599n	0.598
device-sph-m840	0.599

device-kftt	0.599
device-droid razr 4g	0.599
device-gt-i9505	0.599
os_version-2.3.6	0.599
device-sph-l900	0.6
os_version-2.2.1	0.599
device-sch-s738c	0.599
os_version-2.3.4	0.599
device-sch-i500	0.599
device-ms770	0.599
browser_version-37.0.2062.94	0.599
os_version-4.1.2	0.599
device-sch-r970c	0.599
device-sgh-i847	0.599
device-glacier	0.599
vendor-amazon	0.599
device-gt-n7105	0.599
device-sgh-m819n	0.599
device-sgh-t699	0.599
device-gt-s5301l	0.599
device-z995	0.599
browser_version-36.0.1985.135	0.598
device-h3000c	0.599
browser_version-30.0.1599.92	0.599

device-0p9o110	0.599
device-d801	0.599
device-sch-r720	0.599
device-sm-g386t	0.599
browser_version-36.0.1985.131	0.599
browser_version-37.0.2062.76	0.599
device-vs450pp	0.599
browser_version-4.1	0.6
device-apa9292kt	0.599
vendor-alcatel	0.6
device-x515c	0.599
device-d950	0.599
device-acquire	0.599
device-lg730	0.6
device-droidx	0.599
browser_version-31.0	0.599
device-sch-r960	0.599
os_version-2.3.1	0.599
device-y301a1	0.599
device-sch-i510	0.599
vendor-samsung	0.598
os_version-2.3.3	0.6
device-ls840	0.599
device-sgh-i547	0.599

device-sgh-i257	0.599
device-z990g	0.599
device-m866	0.599
device-y301a2	0.599
device-vs410pp	0.6
device-a9192	0.599
device-p999	0.599
device-sch-l710	0.599
device-sgh-t999	0.6
device-u8730	0.599
device-ls855	0.599
device-ls860	0.599
device-droid pro	0.599
vendor-htc	0.599
device-d500	0.599
browser_version-33.0.1750.166	0.599
device-sgh-i437z	0.6
device-m865	0.599
device-sgh-i437	0.599
device-sm-g900az	0.599
device-sch-i200	0.599
device-c715c	0.599
vendor-lg	0.6
device-one touch 995s	0.599

device-xt615	0.599
device-d800	0.599
device-sph-m580bst	0.599
device-sph-l300	0.6
device-one touch 988	0.599
device-sch-r830	0.599
browser_family-mobile safari	0.599
browser_version-30.0.1599.103	0.599
browser_family-firefox	0.599
device-u8686	0.599
device-sph-d710bst	0.599
device-ms840	0.6
browser_version-30.0.0.0	0.601
os_version-4.4	0.599
browser_version-28.0.1500.94	0.598
device-sch-r740c	0.598
device-sgh-t959v	0.599
vendor-sprint	0.599
device-droid4	0.599
device-vle u	0.599
device-vm696	0.599
device-milestone	0.599
device-m886	0.599
device-sch-r680	0.599

device-gt-i9152	0.599
device-first	0.599
os_version-4.4.2	0.598
device-pn072	0.599
browser_version-18.0.1025.308	0.599
os_version-2.3.5	0.599
browser_version-30.0	0.599
device-z992	0.599
device-m931	0.599
os_version-4.4.4	0.599
device-ls696	0.599
device-h868c	0.599
device-sph-m830	0.599
device-sch-s968c	0.599
vendor-huawei	0.599
Device-nexus 5	0.599
device-sch-r760x	0.599
device-edownload	0.599
device-ph39100	0.599
device-lw690	0.599
device-lg855	0.599
device-apx515ckt	0.599
device-ms690	0.599
device-sgh-s730g	0.599

device-sch-r760	0.599
device-sph-l710	0.6
device-gt-i9500	0.599
vendor-motorola	0.599
device-sph-m930bst	0.599
device-sph-l720t	0.599
os_version-4.1.1	0.599
os_version-4.0.4	0.598
browser_version-33.0.1750.136	0.599
browser_version-34.0.1847.114	0.599
device-z998	0.599
device-kfjwi	0.599
os_version-0.0	0.599
browser_version-33.0.1750.517	0.599
device-u8680	0.599
device-970.0	0.599
device-l40g	0.599
device-980.0	0.599
browser_version-35.0.1916.138	0.599
device-z777	0.599
device-sch-s960l	0.599
device-sm-g900t1	0.599
device-gt-i9295	0.599
device-gt-n7000	0.599

browser_version-4.2.2	0.599
device-one m8	0.599
device-c525c	0.599
device-n910	0.599
device-sch-i605	0.599
device-sch-r920	0.599
device-739.0	0.599
device-sch-r970x	0.599
device-gt-s7270l	0.599
device-sgh-i777	0.599
browser_version-11.0.696.34	0.599
device-d850	0.599
device-one s	0.599
device-m865c	0.599
device-sgh-t999v	0.599
device-z740g	0.599
device-sph-m580	0.599
device-pn07120	0.599
device-lg870	0.599
device-sm-g900t	0.599
device-gt-i9200	0.599
device-droid bionic	0.599
device-sch-r730	0.599
device-d415	0.599

browser_version-36.0.1985.128	0.599
device-ls720	0.6
device-sph-d710vmub	0.599
device-us730	0.599
device-wx445	0.599
device-ms695	0.599
device-gt-i8190n	0.599
device-m881	0.599
device-sgh-t999l	0.599
device-gt-n7100	0.599
device-d321	0.599
os_version-2.2.2	0.599
device-sch-i535	0.599
device-c800	0.599
device-gt-i9300	0.599
browser_version-33.0.0.0	0.599
os_version-4.2.1	0.599
device-sph-l710t	0.599
device-sch-r930	0.599
device-kfot	0.599
os_version-2.3.7	0.6
browser_version-4.2	0.599
device-kfsowi	0.599
device-sgh-i827	0.599

browser_version-30.0.1599.82	0.599
device-sgh-i717	0.599
device-sgh-i337m	0.599
browser_version-32.0.1700.99	0.599
device-sgh-t399	0.6
device-sm-g900p	0.599
device-ms870	0.598
os_version-4.0	0.599
device-other	0.6
browser_version-22.0.1485.81203	0.599
device-sgh-i437p	0.599
device-l38c	0.6
browser_version-4.0	0.6
device-z990	0.599
device-h881c	0.599
device-c729	0.599
device-gt-i9082l	0.599
browser_family-opera	0.599
os_version-x86_64	0.599
device-sgh-i527	0.599
browser_version-18.0.1025.166	0.599
device-sch-i405	0.599
device-sm-g730a	0.6
device-sm-n900	0.599

device-us780	0.599
device-sph-d600	0.599
device-sch-i200pp	0.599
browser_version-35.0.1916.122	0.599
device-sph-l520	0.599
device-padfone	0.599
device-sch-i110	0.6
device-sph-l600	0.599
device-d959	0.599
device-sph-d700	0.599
device-0p6b130	0.599
device-sch-r830c	0.599
vendor-sonyericsson	0.599
device-sgh-t589	0.599
device-z993	0.599
device-sgh-t679	0.599
device-sgh-t989	0.6
device-sch-i535pp	0.599
device-sgh-i337z	0.599
device-sph-d710	0.598
device-m920	0.599
device-sgh-i727	0.599
Device-nexus 4	0.599
device-sch-r970	0.599

device-u8665	0.599
device-sch-i545	0.599
device-sgh-i927	0.599
os_version-4.0.3	0.598
device-droid razr	0.599
device-g510-0251	0.599
device-h1000c	0.599
device-sm-n900p	0.599
os_version-4.3	0.599
device-sgh-i747m	0.599
device-sgh-i317	0.599
browser_family-maxthon	0.599
device-sch-r940	0.599
device-p505	0.599
device-sch-m828c	0.599
device-sgh-m919	0.599
browser_version-3.23	0.599
device-sm-g900a	0.599
device-apc715ckt	0.599
device-as730	0.599
device-droid3	0.599
device-ms910	0.599
device-sch-r530u	0.599
device-sgh-t999n	0.599

device-one touch 909b	0.599
device-d520	0.599
browser_version-31.0.1650.59	0.599
device-sm-n900v	0.599
device-ls970	0.6
device-sph-m930	0.599
device-pn071	0.599
device-sch-i800	0.599
device-lw770	0.599
device-sgh-t959	0.599
device-sch-r820	0.6
device-sgh-i337	0.6
device-d851	0.599
device-gt-i9080l	0.599
device-sensation	0.599
device-sm-g900r4	0.599
device-sph-m950	0.599
vendor-asus	0.599
device-sch-r530x	0.599
device-sgh-i577	0.599
os_version-2.1	0.599
device-u8652	0.599
device-sch-r950	0.599
device-gt-i8190l	0.599

device-sgh-t889	0.599
device-sch-i435	0.599
device-sch-r530m	0.6
device-sph-l720	0.598
vendor-other	0.6
device-a510c	0.599
device-onetouch	0.599
os_version-4.4.3	0.599
os_version-4.2.2	0.599
device-p925	0.599
browser_version-26.0.1410.58	0.599
device-droid2	0.599
device-h867g	0.599
device-z740	0.599
device-p930	0.599
device-sch-i415	0.599
device-m835	0.599

Note: since there were no distinct training and test sets, even with cross-validation this may be a case of overfitting. The experiment is replicated below with only a training set used for intermediate results, and a test set used for "final" results:

Results with cross-validation on training set:

Attribute	Value	AUC
browser_version	28.0.1500.94	0.592

browser_family	chrome	0.593
	mobile_safari	
browser_version	18.0.1025.166	
	36.0.1985.135	
device	sgh-t599n	
os_version	4.4.2	
vendor	samsung	
browser_version	33.0.0.0	0.594
device	apa9292kt	
	ms870	
	other	
	p870	
	sch-i800	
	sch-r530x	
	sch-r740c	
	sch-s968c	
	sph-d710	
	sph-l720	
	z995	
os_version	2.3.6	
	4.0.3	
	4.1.2	

	4.4.4	
browser_family	firefox	0.595
	maxthon	
	opera	
	silk	
browser_version	0	
	11.0.696.34	
	18.0.1025.308	
	22.0.1485.81203	
	26.0.1410.58	
	3.23	
	30	
	30.0.1599.103	
	30.0.1599.105	
	30.0.1599.82	
	30.0.1599.92	
	31	
	31.0.1650.59	
	32.0.1700.99	
	33.0.1750.136	
	33.0.1750.517	
	34.0.1847.114	
	35.0.1916.122	

	35.0.1916.138
	35.0.1916.141
	36.0.1985.128
	36.0.1985.131
	37.0.2062.76
	37.0.2062.94
	4
	4.1
	4.2
	4.2.2
	0p6b130
	0p9o110
	739
	970
	980
	a510c
	a851l
	acquire
	amaze_4g
	apc715ckt
	apx515ckt
	as695
	as730
device	c525c

c715c
c729
c800
d321
d415
d500
d520
d800
d801
d850
d851
d950
d959
desire
droid2
droid3
droid4
droid_bionic
droid_razr
droid_razr_4g
droid_x2
droidx
electrify
first

galaxy_nexus
glacier
gt-i8190l
gt-i9080l
gt-i9082l
gt-i9152
gt-i9200
gt-i9295
gt-i9300
gt-i9500
gt-i9505
gt-n7100
gt-s5301l
gt-s7270l
h1000c
h3000c
h866c
h867g
h868c
h881c
kfot
kfsowi
kftt
l38c

l40g
lg730
lg855
lg870
ls696
ls720
ls840
ls855
ls860
ls970
ls980
ls995
lw690
lw770
m660
m835
m865
m865c
m866
m886
m920
m931
milestone
ms695

ms770
ms840
ms910
n910
nexus_4
nexus_5
one
one_m8
one_s
one_touch_909b
one_touch_988
one_touch_995s
onetouch
p505
p659
p769
p925
p999
padfone
ph39100
pn071
pn07120
r800xhttp
sch-i110

sch-i200
sch-i200pp
sch-i405
sch-i415
sch-i435
sch-i500
sch-i510
sch-i535
sch-i535pp
sch-i545
sch-i605
sch-l710
sch-m828c
sch-r530c
sch-r530m
sch-r530u
sch-r720
sch-r730
sch-r760
sch-r760x
sch-r820
sch-r830
sch-r830c
sch-r890

sch-r920
sch-r950
sch-r970
sch-r970c
sch-r970x
sch-s720c
sch-s738c
sch-s960l
sensation
sgh-i257
sgh-i317
sgh-i337m
sgh-i337z
sgh-i407
sgh-i437
sgh-i437p
sgh-i437z
sgh-i527
sgh-i537
sgh-i547
sgh-i577
sgh-i717
sgh-i727
sgh-i747

sgh-i777
sgh-i827
sgh-i847
sgh-i927
sgh-i997
sgh-m819n
sgh-m919
sgh-m919n
sgh-s730g
sgh-s959g
sgh-t399
sgh-t589
sgh-t599
sgh-t679
sgh-t699
sgh-t769
sgh-t879
sgh-t889
sgh-t959
sgh-t959v
sgh-t999
sgh-t999l
sgh-t999n
sgh-t999v

sm-c105a
sm-g386t
sm-g730a
sm-g870a
sm-g900a
sm-g900az
sm-g900p
sm-g900r4
sm-g900t
sm-g900t1
sm-n900
sm-n900a
sm-n900p
sm-n900t
sm-n900v
sm-s765c
sph-d700
sph-d710bst
sph-d710vmub
sph-l300
sph-l520
sph-l600
sph-l710
sph-l710t

sph-l720t
sph-l900
sph-m580
sph-m580bst
sph-m820
sph-m830
sph-m840
sph-m930
sph-m930bst
sph-m950
u8651
u8652
u8665
u8680
u8686
u8687
u8730
us730
us780
vle_u
vm696
vs410pp
vs450pp
vs700

	wx445
	x500
	x515c
	xt615
	y301a1
	y301a2
	z740
	z740g
	z777
	z990
	z990g
	z992
	z993
	z998
os_version	0
	2.1
	2.2.1
	2.2.2
	2.3.1
	2.3.3
	2.3.5
	2.3.7
	4
	4.0.4

	4.1.1	
	4.2.1	
	4.2.2	
	4.3	
	4.4	
	4.4.3	
	x86_64	
vendor	alcatel	
	amazon	
	asus	
	htc	
	huawei	
	lg	
	motorola	
	other	
	sonyericsson	
	sprint	
	zte	
browser_version	30.0.0.0	0.596
device	sgh-i337	
	sgh-t989	
os_version	2.3.4	

Results on Train/Test Split:

Attribute	Value	AUC
None	None	0.597
browser_version	30.0.0.0	0.529
device	sgh-i337	0.595
device	sgh-t989	0.595
os_version	2.3.4	0.595

Conclusion: greedy single-value removal is not useful, as it leads to overfitting.

Experiment 25:

- Take the ten-attribute data set and discretize all numeric attributes. Create a 65%/35% train/test split.
- Create nine choose two (36) duplicate train/test pairs, with each one corresponding to a possible set of two out of the nine non-target attributes. Each train/test pair will have the two chosen attributes replaced by a single attribute which is the concatenation of the two chosen attributes (e.g. "device" and "vendor" being replaced by "device/vendor").
- Run 100-fold cross-validation using a Naive Bayes classifier on the training set of each of the duplicates and the original, recording the AUC of each selection.
- Evaluate the Naive Bayes model on the corresponding test set, recording the AUC of each selection.
- Compare the AUC of the two sets to determine if the training data can be used to accurately predict whether or not the combination of attributes will be helpful for test data.

Results:

Attribute	Attribute	Cross-Validation AUC	Train/Test Split AUC
None	None	0.578	0.609
1	2	0.575	0.605
1	3	0.575	0.61
1	4	0.579	0.614
1	5	0.562	0.588
1	6	0.565	0.61
1	7	0.582	0.608
1	8	0.575	0.608
1	9	0.574	0.606
2	3	0.572	0.607
2	4	0.578	0.616
2	5	0.566	0.59

2	6	0.57	0.602
2	7	0.584	0.609
2	8	0.575	0.608
2	9	0.576	0.605
3	4	0.579	0.616
3	5	0.564	0.592
3	6	0.566	0.604
3	7	0.584	0.608
3	8	0.575	0.608
3	9	0.576	0.606
4	5	0.559	0.591
4	6	0.564	0.605
4	7	0.583	0.607
4	8	0.575	0.608
4	9	0.572	0.606
5	6	0.569	0.602
5	7	0.581	0.609
5	8	0.575	0.608
5	9	0.579	0.616
6	7	0.582	0.61
6	8	0.575	0.608
6	9	0.577	0.605
7	8	0.575	0.608
7	9	0.569	0.603

8	9	0.578	0.609

Note: The value of R, the correlation coefficient, was found to be 0.7295. However, despite the fact that this indicated a moderate positive correlation, the best combinations for each choice of first attribute on average underperformed the baseline of no combination.