Predictive analytics and data mining

Article · May 2010

CITATIONS
23

READS
4,846

1 author:

Charles Elkan
University of California, San Diego
158 PUBLICATIONS **23,299** CITATIONS

SEE PROFILE

Predictive analytics and data mining

Charles Elkan
elkan@cs.ucsd.edu

April 29, 2010

Contents

Chapter 1

Introduction

There are many definitions of data mining. We shall take it to mean the application of learning algorithms and statistical methods to real-world datasets. There are numerous data mining applications in science, engineering, and business. We shall focus on applications that are related to business, but the methods that are useful are mostly the same for applications in science or engineering.

The focus will be on methods for making predictions. For example, the available data may be a customer database, along with labels indicating which ones failed to pay their bills. The goal will then be to predict which customers might fail to pay in the future. In general, analytics is a newer name for data mining. Predictive analytics indicates a focus on making predictions.

The main alternative to predictive analytics can be called descriptive analytics. This area is often also called "knowledge discovery in data" or KDD. It is fascinating and often highly useful, but it is harder to obtain direct benefit from descriptive analytics. For example, it may be noteworthy and interesting that customers of Whole Foods tend to be liberal and wealthy, but what can Whole Foods do with that information? In contrast, predictions can be used directly to make decisions that maximize benefit to the decision-maker. For example, customers who are more likely not to pay in the future can have their credit limit reduced now. It is important to understand the difference between a prediction and a decision. Data mining lets us make predictions, but predictions are useful to us only if they lead to decisions that have better outcomes.

1.1 Limitations of predictive analytics

It is important to understand the limitations of predictive analytics. First, in general, one cannot make progress without a dataset for training of adequate size and quality.

Second, it is crucial to have a clear definition of the concept that is to be predicted, and to have historical examples of the concept. Consider for example this extract from an article in the London Financial Times dated May 13, 2009:

> Fico, the company behind the credit score, recently launched a service that pre-qualifies borrowers for modification programmes using their in-house scoring data. Lenders pay a small fee for Fico to refer potential candidates for modifications that have already been vetted for inclusion in the programme. Fico can also help lenders find borrowers that will best respond to modifications and learn how to get in touch with them.

It is hard to see how this could be a successful application of data mining, because it is hard to see how a useful labeled training set could exist. The target concept is "borrowers that will best respond to modifications." From a lender's perspective (and Fico works for lenders not borrowers) such a borrower is one who would not pay under his current contract, but who would pay if given a modified contract. Especially in 2009, lenders had no long historical experience with offering modifications to borrowers, so FICO did not have relevant data. Moreover, the definition of the target is based on a counterfactual, that is on reading the minds of borrowers. Data mining cannot read minds.

For a successful data mining application, the actions to be taken based on predictions need to be defined clearly and to have reliable profit consequences. The difference between a regular payment and a modified payment is often small, for example $200 in the case described in the newspaper article. It is not clear that giving people modifications will really change their behavior dramatically.

For a successful data mining application also, actions must not have major unintended consequences. Here, modifications may change behavior in undesired ways. A person requesting a modification is already thinking of not paying. Those who get modifications may be motivated to return and request further concessions.

Additionally, for predictive analytics to be successful, the training data must be representative of the test data. Typically, the training data come from the past, while the test data arise in the future. If the phenomenon to be predicted is not stable over time, then predictions are likely not to be useful. Here, changes in the general economy, in the price level of houses, and in social attitudes towards foreclosures, are all likely to change the behavior of borrowers in the future, in systematic but unknown ways.

Last but not least, for a successful application it helps if the consequences of actions are essentially independent for different examples. This may be not the case here. Rational borrowers who hear about others getting modifications will try to make themselves appear to be candidates for a modification. So each modification

generates a cost that is not restricted to the loss incurred with respect to the individual getting the modification.

Another example of an application of predictive analytics that is unlikely to succeed is trying to learn a model to predict which persons will commit a terrorist act. There are so few positive training examples that statistically reliable patterns cannot be learned. Moreover, intelligent terrorists will take steps not to fit in with patterns exhibited by previous terrorists [Jonas and Harper, 2006].

1.2 Overview

In this course we shall only look at methods that have state-of-the-art accuracy, that are sufficiently simple and fast to be easy to use, and that have well-documented successful applications. We will tread a middle ground between focusing on theory at the expense of applications, and understanding methods only at a cookbook level. We shall not look at multiple methods for the same task, when there is one method that is at least as good as all the others from most points of view. In particular, for classifier learning, we will look at support vector machines (SVMs) in detail. We will not examine alternative classifier learning methods such as decision trees, random forests, neural networks, boosting, and bagging. All these methods are excellent, but there are no clearly identified important scenarios in which they are definitely superior to SVMs. (The exception may be random forests, which are widely used in commercial applications nowadays.)

Chapter 2

Predictive analytics in general

This chapter explains supervised learning, linear regression, and data cleaning and recoding.

2.1 Supervised learning

The goal of a supervised learning algorithm is to obtain a classifier by learning from training examples. A classifier is something that can be used to make predictions on test examples. This type of learning is called "supervised" because of the metaphor that a teacher (i.e. a supervisor) has provided the true label of each training example.

Each training and test example is represented in the same way, as a row vector of fixed length p. Each element in the vector representing an example is called a feature value. It may be real number or a value of any other type. A training set is a set of vectors with known label values. It is essentially the same thing as a table in a relational database, and an example is one row in such a table. Row, tuple, and vector are essentially synonyms. A column in such a table is often called a feature, or an attribute, in data mining. Sometimes it is important to distinguish between a feature, which is an entire column, and a feature value.

The label y for a test example is unknown. The output of the classifier is a conjecture about y, i.e. a predicted y value. Often each label value y is a real number. In this case, supervised learning is called "regression" and the classifier is called a "regression model." The word "classifier" is usually reserved for the case where label values are discrete. In the simplest but most common case, there are just two label values. These may be called -1 and +1, or 0 and 1, or no and yes, or negative and positive.

With n training examples, and with each example consisting of values for p different features, the training data are a matrix with n rows and p columns, along with

a column vector of y values. The cardinality of the training set is n, while its dimensionality is p. We use the notation x_{ij} for the value of feature number j of example number i. The label of example i is y_i. True labels are known for training examples, but not for test examples.

2.2 Data cleaning and recoding

In real-world data, there is a lot of variability and complexity in features. Some features are real-valued. Other features are numerical but not real-valued, for example integers or monetary amounts. Many features are categorical, e.g. for a student the feature "year" may have values freshman, sophomore, junior, and senior. Usually the names used for the different values of a categorical feature make no difference to data mining algorithms, but are critical for human understanding. Sometimes categorical features have names that look numerical, e.g. zip codes, and/or have an unwieldy number of different values. Dealing with these is difficult. It is also difficult to deal with features that are really only meaningful in conjunction with other features, such as "day" in the combination "day month year."

Even with all the complexity of features, many aspects are typically ignored in data mining. Usually, units such as dollars and dimensions such as kilograms are omitted. Difficulty in determining feature values is also ignored: for example, how does one define the year of a student who has transferred from a different college, or who is part-time?

A very common difficulty is that the value of a given feature is missing for some training and/or test examples. Often, missing values are indicated by question marks. However, often missing values are indicated by strings that look like valid known values, such as 0 (zero). It is important not to treat a value that means missing inadvertently as a valid regular value.

Some training algorithms can handle missingness internally. If not, the simplest approach is just to discard all examples (rows) with missing values. Keeping only examples without any missing values is called "complete case analysis." An equally simple, but different, approach is to discard all features (columns) with missing values, However, in many applications, both these approaches eliminate too much useful training data.

Also, the fact that a particular feature is missing may itself be a useful predictor. Therefore, it is often beneficial to create an additional binary feature that is 0 for missing and 1 for present. If a feature with missing values is retained, then it is reasonable to replace each missing value by the mean or mode of the non-missing values. This process is called imputation. More sophisticated imputation procedures exist, but they are not always better.

Some training algorithms can only handle categorical features. For these, features that are numerical can be discretized. The range of the numerical values is partitioned into a fixed number of intervals that are called bins. The word "partitioned" means that the bins are exhaustive and mutually exclusive, i.e. non-overlapping. One can set boundaries for the bins so that each bin has equal width, i.e. the boundaries are regularly spaced, or one can set boundaries so that each bin contains approximately the same number of training examples, i.e. the bins are "equal count." Each bin is given an arbitrary name. Each numerical value is then replaced by the name of the bin in which the value lies. It often works well to use ten bins.

Other training algorithms can only handle real-valued features. For these, categorical features must be made numerical. The values of a binary feature can be recoded as 0.0 or 1.0. It is conventional to code "false" or "no" as 0.0, and "true" or "yes" as 1.0. Usually, the best way to recode a feature that has k different categorical values is to use k real-valued features. For the jth categorical value, set the jth of these features equal to 1.0 and set all $k-1$ others equal to 0.0.[1]

Categorical features with many values (say, over 20) are often difficult to deal with. Typically, human intervention is needed to recode them intelligently. For example, zipcodes may be recoded as just their first one or two letters, since these indicate meaningful regions of the United States. If one has a large dataset with many examples from each of the 50 states, then it may be reasonable to leave this as a categorical feature with 50 values. Otherwise, it may be useful to group small states together in sensible ways, for example to create a New England group for MA, CT, VT, ME, NH.

An intelligent way to recode discrete predictors is to replace each discrete value by the mean of the target conditioned on that discrete value. For example, if the average label value is 20 for men versus 16 for women, these values could replace the male and female values of a variable for gender. This idea is especially useful as a way to convert a discrete feature with many values, for example the 50 U.S. states, into a useful single numerical feature.

However, as just explained, the standard way to recode a discrete feature with m values is to introduce $m-1$ binary features. With this standard approach, the training algorithm can learn a coefficient for each new feature that corresponds to an optimal numerical value for the corresponding discrete value. Conditional means are likely to be meaningful and useful, but they will not yield better predictions than the coefficients learned in the standard approach. A major advantage of the conditional-

[1] The ordering of the values, i.e. which value is associated with $j=1$, etc., is arbitrary. Mathematically it is preferable to use only $k-1$ real-valued features. For the last categorical value, set all $k-1$ features equal to 0.0. For the jth categorical value where $j<k$, set the jth feature value to 1.0 and set all $k-1$ others equal to 0.0.

means approach is that it avoids an explosion in the dimensionality of training and test examples.

Mixed types.

Sparse data.

Normalization. After conditional-mean new values have been created, they can be scaled to have zero mean and unit variance in the same way as other features.

2.3 Linear regression

Let x be an instance and let y be its real-valued label. For linear regression, x must be a vector of real numbers of fixed length. Remember that this length p is often called the dimension, or dimensionality, of x. Write $x = \langle x_1, x_2, \ldots, x_p \rangle$. The linear regression model is

$$y = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_p x_p.$$

The righthand side above is called a linear function of x. The linear function is defined by its coefficients b_0 to b_p. The coefficient b_0 is called the intercept. It is the value of y according to the model, if $x_i = 0$ for all i. The coefficient b_i is the amount by which the predicted y value increases if x_i increases by 1. For example, suppose x_i is a binary feature where $x_i = 0$ means female and $x_i = 1$ means male, and suppose $b_i = -2.5$. Then the predicted y value for males is lower by 2.5, everything else being held constant.

Suppose that the training set has cardinality n, i.e. it consists of n examples of the form $\langle x_i, y_i \rangle$, where $x_i = \langle x_{i1}, \ldots, x_{ip} \rangle$. Let b be any set of coefficients. The predicted value for x_i is

$$\hat{y}_i = f(x_i, b) = b_0 + \sum_{j=1}^{p} b_j x_{ij}.$$

If we define $x_{i0} = 1$ for every i, then we can write

$$\hat{y}_i = \sum_{j=0}^{p} b_j x_{ij}.$$

Finding good values for the coefficients b_0 to b_p is the job of the training algorithm. The standard approach is to choose values that minimize the sum of errors on the training set, where the error on training example i is defined to be the square $(y_i - \hat{y}_i)^2$. The training algorithm then finds

$$\hat{b} = \text{argmin}_b \sum_{i=1}^{n} (y_i - \sum_{j=0}^{p} b_j x_{ij})^2.$$

The objective function $\sum_i (y_i - \sum_j b_j x_{ij})^2$ is called the sum of squared errors, or SSE for short.

The optimal coefficient values $\hat{b}$ are not defined uniquely if the number n of training examples is less than the number p of features. Even if $n > p$ is true, the optimal coefficients can have multiple equivalent values if some features are themselves related linearly. For an intuitive example, suppose features 1 and 2 are height and weight respectively. Suppose that $x_2 = 120 + 5(x_1 - 60) = -180 + 5x_1$ approximately, where the units of measurement are pounds and inches. Then the same model can be written in many different ways:

- $y = b_0 + b_1 x_1 + b_2 x_2$
- $y = b_0 + b_1 x_1 + b_2(-180 + 5x_1) = [b_0 - 180 b_2] + [b_1(1 + 5b_2)]x_1 + 0x_2$

and more. In the extreme, suppose $x_1 = x_2$. Then all models $y = b_0 + b_1 x_1 + b_2 x_2$ are equivalent for which $b_1 + b_2$ equals a constant.

When two or more features are approximately related linearly, then the true values of the coefficients of those features are not well determined. The coefficients obtained by training will be strongly influenced by randomness in the training data. Regularization is a way to reduce the influence of this type of randomness. Consider all models $y = b_0 + b_1 x_1 + b_2 x_2$ for which $b_1 + b_2 = c$. Among these models, there is a unique one that minimizes the function $b_1^2 + b_2^2$. This model has $b_1 = b_2 = c/2$. We can obtain it by setting the objective function for training to be the sum of squared errors (SSE) plus a function that penalizes large values of the coefficients. A simple penalty function of this type is $\sum_{j=1}^{p} b_j^2$. A parameter λ can control the relative importance of the two objectives, namely SSE and penalty:

$$\hat{b} = \text{argmin}_b \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} b_j^2.$$

If $\lambda = 0$ then one gets the standard least-squares linear regression solution. As λ gets larger, the penalty on large coefficients gets stronger, and the typical values of coefficients get smaller. The parameter λ is often called the strength of regularization.

The penalty function $\sum_{j=1}^{p} b_j^2$ is only sensible if the typical magnitude is similar for the values of each feature. This an important motivation for data normalization. Note that in the formula $\sum_{j=1}^{p} b_j^2$ the sum excludes the intercept coefficient b_0. One reason for doing this is that the target y values are typically not normalized.

2.4 Interpreting coefficients of a linear model

It is common to desire a data mining model that is interpretable, that is one that can be used not only to make predictions, but also to understand mechanisms in the

phenomenon that is being studied. Linear models, whether for regression or for classification as described in later chapters, do appear interpretable at first sight. However, much caution is needed when attempting to derive conclusions from numerical coefficients.

Consider the following linear regression model for predicting high-density cholesterol (HDL) levels.[2]

predictor	coefficient	std.error	Tstat	p-value
intercept	1.16448	0.28804	4.04	<.0001
BMI	-0.01205	0.00295	-4.08	<.0001
LCHOL	0.31109	0.10936	2.84	0.0051
GLUM	-0.00046	0.00018	-2.50	0.0135
DIAST	0.00255	0.00103	2.47	0.0147
BLC	0.05055	0.02215	2.28	0.0239
PRSSY	-0.00041	0.00044	-0.95	0.3436
SKINF	0.00147	0.00183	0.81	0.4221
AGE	-0.00092	0.00125	-0.74	0.4602

From most to least statistically significant, the predictors are body mass index, the log of total cholesterol, diastolic blood pressures, vitamin C level in blood, systolic blood pressure, skinfold thickness, and age in years. (It is not clear what GLUM is.)

The example illustrates at least two crucial issues. First, if predictors are collinear, then one may appear significant and the other not, when in fact both are significant or both are not. Above, diastolic blood pressure is statistically significant, but systolic is not. This may possibly be true for some physiological reason. But it may also be an artifact of collinearity.

Second, a predictor may be practically important, and statistically significant, but still useless for interventions. This happens if the predictor and the outcome have a common cause, or if the outcome causes the predictor. Above, vitamin C is statistically significant. But it may be that vitamin C is simply an indicator of a generally healthy diet high in fruits and vegetables. If this is true, then merely taking a vitamin C supplement will cause an increase in HDL level.

A third crucial issue is that a correlation may disagree with other knowledge and assumptions. For example, vitamin C is generally considered beneficial or neutral.

[2] Note that HDL cholesterol is considered beneficial and sometimes called "good" cholesterol. Source: `http://www.jerrydallal.com/LHSP/importnt.htm`. Predictors have been reordered here from most to least statistically significant, as measured by p-value.

If lower vitamin C was associated with higher HDL, one would be cautious about believing this relationship, even if the association was statistically significant.

2.5 Evaluating performance

In a real-world application of supervised learning, we have a training set of examples with labels, and a test set of examples with unknown labels. The whole point is to make predictions for the test examples.

However, in research or experimentation we want to measure the performance achieved by a learning algorithm. To do this we use a test set consisting of examples with known labels. We train the classifier on the training set, apply it to the test set, and then measure performance by comparing the predicted labels with the true labels (which were not available to the training algorithm).

It is absolutely vital to measure the performance of a classifier on an independent test set. Every training algorithm looks for patterns in the training data, i.e. correlations between the features and the class. Some of the patterns discovered may be spurious, i.e. they are valid in the training data due to randomness in how the training data was selected from the population, but they are not valid, or not as strong, in the whole population. A classifier that relies on these spurious patterns will have higher accuracy on the training examples than it will on the whole population. Only accuracy measured on an independent test set is a fair estimate of accuracy on the whole population. The phenomenon of relying on patterns that are strong only in the training data is called overfitting. In practice it is an omnipresent danger.

Most training algorithms have some settings that the user can choose between. For naive Bayes these algorithm parameters include the degree of smoothing λ and the number of bins to use when discretizing continuous features. It is natural to run the algorithm multiple times and to measure the accuracy of the classifier learned with different settings. A set of labeled examples used in this way to pick settings for an algorithm is called a validation set. If you use a validation set, it is important to have a final test set that is independent of both the training set and the validation set.

Dividing the available data into training, validation, and test sets should be done randomly, in order to guarantee that each set is a random sample from the same distribution.

Quiz question

Suppose you are building a model to predict how many dollars someone will spend at Sears. You know the gender of each customer, male or female. Since you are using linear regression, you must recode this discrete feature as continuous. You decide to use two real-valued features, x_{11} and x_{12}. The coding is a standard "one of n" scheme, as follows:

gender	x_{11}	x_{12}
male	1	0
female	0	1

Learning from a large training set yields the model

$$y = \ldots + 15x_{11} + 75x_{12} + \ldots$$

Dr. Roebuck says "Aha! The average woman spends $75, but the average man spends only $15."

Write your name below, and then answer the following three parts with one or two sentences each:

(a) Explain why Dr. Roebuck's conclusion is not valid. *The model only predicts spending of $75 for a woman if all other features have value zero. This may not be true for the average woman. Indeed it will not be true for any woman, if features such as "age" are not normalized.*

(b) Explain what conclusion can actually be drawn from the numbers 15 and 75. *The conclusion is that if everything else is held constant, then on average a woman will spend $60 more than a man. Note that if some feature values are systematically different for men and women, then even this conclusion is not useful, because it is not reasonable to hold all other feature values constant.*

(c) Explain a desirable way to simplify the model. *The two features x_{11} and x_{12} are linearly related. Hence, they make the optimal model be undefined, in the absence of regularization. It would be good to eliminate one of these two features. The expressiveness of the model would be unchanged.*

Quiz for April 6, 2010

Your name:

Suppose that you are training a model to predict how many transactions a credit card customer will make. You know the education level of each customer. Since you are using linear regression, you recode this discrete feature as continuous. You decide to use two real-valued features, x_{37} and x_{38}. The coding is a "one of two" scheme, as follows:

	x_{37}	x_{38}
college grad	1	0
not college grad	0	1

Learning from a large training set yields the model

$$y = \ldots + 5.5x_{37} + 3.2x_{38} + \ldots$$

(a) Dr. Goldman concludes that the average college graduate makes 5.5 transactions. Explain why Dr. Goldman's conclusion is likely to be false.

The model only predicts 5.5 transactions if all other features, including the intercept, have value zero. This may not be true for the average college grad. It will certainly be false if features such as "age" are not normalized.

(b) Dr. Sachs concludes that being a college graduate causes a person to make 2.3 more transactions, on average. Explain why Dr. Sachs' conclusion is likely to be false also.

First, if any other feature have different values on average for men and women, for example income, then $5.5 - 3.2 = 2.3$ is not the average difference in predicted y value between groups. Said another way, it is unlikely to be reasonable to hold all other feature values constant when comparing groups.

Second, even if 2.3 is the average difference, one cannot say that this difference is caused by being a college graduate. There may be some other unknown common cause, for example.

Linear regression assignment

This assignment is due at the start of class on Tuesday April 13. You should work in a team of two. Choose a partner who has a different background from you.

Download the file `cup98lrn.zip` from from `http://archive.ics.uci.edu/ml/databases/kddcup98/kddcup98.html`. Read the associated documentation. Load the data into Rapidminer. Select the 4843 records that have feature `TARGET_B=1`. Save these as a native-format Rapidminer example set.

Now, build a linear regression model to predict the field `TARGET_D` as accurately as possible. Use mean squared error (MSE) as the definition of error, and use ten-fold cross-validation to measure MSE. Do a combination of the following:

- Recode non-numerical features as numerical.
- Discard useless features.
- Transform features to improve their usefulness.
- Compare different strengths of regularization.

Do the steps above repeatedly in order to explore alternative ways of using the data. The outcome should be the best possible model that you can find that uses 30 or fewer of the original features.

To make your work more efficient, be sure to save the 4843 records in a format that Rapidminer can load quickly. You can use three-fold cross-validation during development to save time also. You will likely find that the built-in Rapidminer methods for selecting subsets of features are too slow to be usable. If you normalize all input features, and you use strong regularization (ridge parameter 10^7 perhaps), then the regression coefficients will indicate the relative importance of features.

The deliverable is a brief report of about two pages, formatted similarly to this assignment description. Describe what you did that worked, and your results. Explain any assumptions that you made, and any limitations on the validity or reliability of your results. Include a printout of your final Rapidminer process, and of your final regression model (not included in the two pages). Do not speculate about future work, and do not explain ideas that did not work. Write in the present tense. Organize your report logically, not chronologically.

Comments on the regression assignment

It is typically useful to rescale predictors to have mean zero and variance one. However, it loses interpretability to rescale the target variable. Note that if all predictors have mean zero, then the intercept of a linear regression model is the mean of the target, $15.6243 here, assuming that the intercept is not regularized.

The assignment specifically asks you to report mean squared error, MSE. One could also report root mean squared error, RMSE, but whichever is chosen should be used consistently. In general, do not confuse readers by switching between multiple performance measures without a good reason.

As is often the case, good performance can be achieved with a very simple model. The most informative single feature is `LASTGIFT`, the dollar amount of the person's most recent gift. A model based on just this single feature achieves RMSE of $9.98. In 2009, three of 11 teams achieved similar final RMSEs that were slightly better than $9.00. The two teams that omitted `LASTGIFT` achieved RMSE worse than $11.00. In 2010, the best believable RMSE obtained by an team was $8.04.

The assignment asks you to produce a final model based on at most 30 of the original features. Despite this directive, it is not a good idea to begin by choosing a subset of the 480 original features based on human intuition. The teams that did this all omitted features that in fact would have made their final models considerably better, including sometimes the feature `LASTGIFT`. As explained above, it is also not a good idea to eliminate automatically features with missing values.

Rapidminer has operators that search for highly predictive subsets of variables. These operators have two major problems. First, they are too slow to be used on a large initial set of variables, so it is easy for human intuition to pick an initial set that is bad. Second, these operators try a very large number of alternative subsets of variables, and pick the one that performs best on some dataset. Because of the high number of alternatives considered, this subset is likely to overfit substantially the dataset on which it is best. For more discussion of this problem, see the section on nested cross-validation in a later chapter.

Chapter 3

Introduction to Rapidminer

By default, many Java implementations allocate so little memory that Rapidminer will quickly terminate with an "out of memory" message. This is not a problem with Windows 7, but otherwise, launch Rapidminer with a command like

```
java -Xmx1g -jar rapidminer.jar
```

where `1g` means one gigabyte.

3.1 Standardization of features

The recommended procedure is as follows, in order.

- Normalize all numerical features to have mean zero and variance 1.
- Convert each nominal feature with k alternative values into k different binary features.
- Optionally, drop all binary features with fewer than 100 examples for either binary value.
- Convert each binary feature into a numerical feature with values 0.0 and 1.0.

It is not recommended to normalize numerical features obtained from binary features. The reason is that this normalization would destroy the sparsity of the dataset, and hence make some efficient training algorithms much slower. Note that destroying sparsity is not an issue when the dataset is stored as a dense matrix, which it is by default in Rapidminer.

Normalization is also questionable for variables whose distribution is highly uneven. For example, almost all donation amounts are under $50, but a few are up

to $500. No linear normalization, whether by z-scoring or by transformation to the range 0 to 1 or otherwise, will allow a linear classifier to discriminate well between different common values of the variable. For variables with uneven distributions, it is often useful to apply a nonlinear transformation that makes their distribution have more of a uniform or Gaussian shape. A common transformation that often achieves this is to take logarithms. A useful trick when zero is a frequent value of a variable x is to use the mapping $x \mapsto \log(x + 1)$. This leaves zeros unchanged and hence preserves sparsity.

Eliminating binary features for which either value has fewer than 100 training examples is a heuristic way to prevent overfitting and at the same time make training faster. It may not be necessary, or beneficial, if regularization is used during training. If regularization is not used, then at least one binary feature must be dropped from each set created by converting a multivalued feature, in order to prevent the existence of multiple equivalent models. For efficiency and interpretability, it is best to drop the binary feature corresponding to the most common value of the original multivalued feature.

If you have separate training and test sets, it is important to do all preprocessing in a perfectly consistent way on both datasets. The simplest is to concatenate both sets before preprocessing, and then split them after. However, concatenation allows information from the test set to influence the details of preprocessing, which is a form of information leakage from the test set, that is of using the test set during training, which is not legitimate.

3.2 Example of a Rapidminer process

Figure 3.2 shows a tree structure of Rapidminer operators that together perform a standard data mining task. The tree illustrates how to perform some common subtasks.

The first operator is ExampleSource. “Read training example” is a name for this particular instance of this operator. If you have two instances of the same operator, you can identify them with different names. You select an operator by clicking on it in the left pane. Then, in the right pane, the Parameters tab shows the arguments of the operator. At the top there is an icon that is a picture of either a person with a red sweater, or a person with an academic cap. Click on the icon to toggle between these. When the person in red is visible, you are in expert mode, where you can see all the parameters of each operator.

The example source operator has a parameter named attributes. Normally this is file name with extension “aml.” The corresponding file should contain a schema definition for a dataset. The actual data are in a file with the same name with exten-

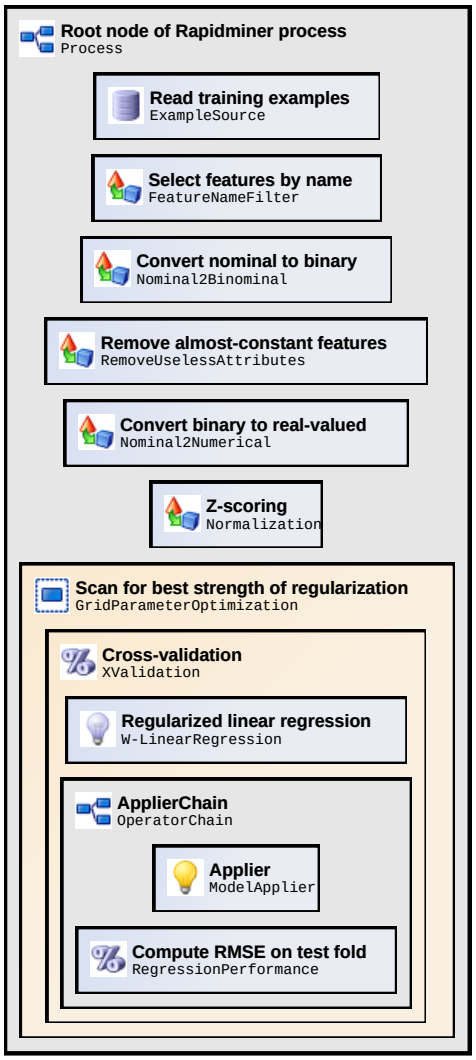

Figure 3.1: Rapidminer process for regularized linear regression.

sion "dat." The easiest way to create these files is by clicking on "Start Data Loading Wizard." The first step with this wizard is to specify the file to read data from, the character that begins comment lines, and the decimal point character. Ticking the box for "use double quotes" can avoid some error messages.

In the next panel, you specify the delimiter that divides fields within each row of data. If you choose the wrong delimiter, the data preview at the bottom will look wrong. In the next panel, tick the box to make the first row be interpreted as field names. If this is not true for your data, the easiest is to make it true outside Rapidminer, with a text editor or otherwise. When you click Next on this panel, all rows of data are loaded. Error messages may appear in the bottom pane. If there are no errors and the data file is large, then Rapidminer hangs with no visible progress. The same thing happens if you click Previous from the following panel. You can use a CPU monitor to see what Rapidminer is doing.

The next panel asks you to specify the type of each attribute. The wizard guesses this based only on the first row of data, so it often makes mistakes, which you have to fix by trial and error. The following panel asks you to say which features are special. The most common special choice is "label" which means that an attribute is a target to be predicted.

Finally, you specify a file name that is used with "aml" and "dat" extensions to save the data in Rapidminer format.

To keep just features with certain names, use the operator FeatureNameFilter. Let the argument `skip_features_with_name` be `.*` and let the argument `except_features_with_name` identify the features to keep. In our sample process, it is `(.*AMNT.*)|(.*GIFT.*)(YRS.*)|(.*MALE)|(STATE)|(PEPSTRFL)|(.*GIFT)|(MDM.*)|(RFA_2.*)`.

In order to convert a discrete feature with k different values into k real-valued 0/1 features, two operators are needed. The first is `Nominal2Binominal`, while the second is `Nominal2Numerical`. Note that the documentation of the latter operator in Rapidminer is misleading: it cannot convert a discrete feature into multiple numerical features directly. The operator `Nominal2Binominal` is quite slow. Applying it to discrete features with more than 50 alternative values is not recommended.

The simplest way to find a good value for an algorithm setting is to use the `XValidation` operator nested inside the `GridParameterOptimization` operator. The way to indicate nesting of operators is by dragging and dropping. First create the inner operator subtree. Then insert the outer operator. Then drag the root of the inner subtree, and drop it on top of the outer operator.

3.3 Other notes on Rapidminer

Reusing existing processes.

Saving datasets.

Operators from Weka versus from elsewhere.

Comments on specific operators: Nominal2Numerical, Nominal2Binominal, RemoveUselessFeatures.

Eliminating non-alphanumerical characters, using quote marks, trimming lines, trimming commas.

Chapter 4

Support vector machines

This chapter explains soft-margin support vector machines (SVMs), including linear and nonlinear kernels. It also discusses detecting overfitting via cross-validation, and preventing overfitting via regularization.

We have seen how to use linear regression to predict a real-valued label. Now we will see how to use a similar model to predict a binary label. In later chapters, where we think probabilistically, it will be convenient to assume that a binary label takes on true values 0 or 1. However, in this chapter it will be convenient to assume that the label y has true value either $+1$ or -1.

4.1 Loss functions

Let x be an instance, let y be its true label, and let $f(x)$ be a prediction. Assume that the prediction is real-valued. If we need to convert it into a binary prediction, we will threshold it at zero:

$$\hat{y} = 2 \cdot I(f(x) \geq 0) - 1$$

where $I()$ is an indicator function that is 1 if its argument is true, and 0 if its argument is false.

A so-called loss function l measures how good the prediction is. Typically loss functions do not depend directly on x, and a loss of zero corresponds to a perfect prediction, while otherwise loss is positive. The most obvious loss function is

$$l(f(x), y) = I(f(x) \neq y)$$

which is called the 0-1 loss function. The usual definition of accuracy uses this loss function. However, it has undesirable properties. First, it loses information: it does not distinguish between predictions $f(x)$ that are almost right, and those that are

very wrong. Second, mathematically, its derivative is either undefined or zero, so it is difficult to use in training algorithms that try to minimize loss via gradient descent.

A better loss function is squared error:

$$l(f(x), y) = (f(x) - y)^2$$

which is infinitely differentiable everywhere, and does not lose information when the prediction $f(x)$ is real-valued. However, this loss function says that the prediction $f(x) = 1.5$ is as undesirable as $f(x) = 0.5$ when the true label is $y = 1$. Intuitively, if the true label is $+1$, then a prediction with the correct sign that is greater than 1 should not be considered incorrect.

The following loss function, which is called hinge loss, satisfies the intuition just suggested:

$$l(f(x), y) = \max\{0, 1 - yf(x)\}.$$

The hinge loss function deserves some explanation. Suppose the true label is $y = 1$. Then the loss is zero as long as the prediction $f(x) \geq 1$. The loss is positive, but less than 1, if $0 < f(x) < 1$. The loss is large, i.e. greater than 1, if $f(x) < 0$.

Using hinge loss is the first major insight behind SVMs. An SVM classifier f is trained to minimize hinge loss. The training process aims to achieve predictions $f(x) \geq 1$ for all training instances x with true label $y = +1$, and to achieve predictions $f(x) \leq -1$ for all training instances x with $y = -1$. Overall, training seeks to classify points correctly, *and to distinguish clearly between the two classes*, but it does not seek to make predictions be exactly $+1$ or -1. In this sense, the training process intuitively aims to find the best possible classifier, without trying to satisfy any unnecessary additional objectives also.

4.2 Regularization

Given a set of training examples $\langle x_i, y_i \rangle$ for $i = 1$ to $i = n$, the total training loss (sometimes called empirical loss) is the sum

$$\sum_{i=1}^{n} l(f(x_i), y_i).$$

Suppose that the function f is selected by minimizing average training loss:

$$f = \text{argmin}_{f \in F} \frac{1}{n} \sum_{i=1}^{n} l(f(x_i), y_i)$$

where F is a space of candidate functions. If F is too flexible, and/or the training set is too small, then we run the risk of overfitting the training data. But if F is too

restricted, then we run the risk of underfitting the data. In general, we do not know in advance what the best space F is for a particular training set. A possible solution to this dilemma is to choose a flexible space F, but at the same time to impose a penalty on the complexity of f. Let $c(f)$ be some real-valued measure of complexity. The learning process then becomes to solve

$$f = \text{argmin}_{f \in F} \ \lambda c(f) + \frac{1}{n} \sum_{i=1}^{n} l(f(x_i), y_i).$$

Here, λ is a parameter that controls the relative strength of the two objectives, namely to minimize the complexity of f and to minimize training error.

Suppose that the space of candidate functions is defined by a vector $w \in \mathbb{R}^d$ of parameters, i.e. we can write $f(x) = g(x; w)$ where g is some fixed function. In this case we can define the complexity of each candidate function to be the norm of the corresponding w. Most commonly we use the square norm:

$$c(f) = ||w||^2 = \sum_{j=1}^{d} w_j^2.$$

However we could also use other norms, including the L_0 norm

$$c(f) = ||w||_0^2 = \sum_{j=1}^{d} I(w_j \neq 0)$$

or the L_1 norm

$$c(f) = ||w||_1^2 = \sum_{j=1}^{d} |w_j|.$$

The square norm is the most convenient mathematically.

4.3 Linear soft-margin SVMs

A linear classifier is a function $f(x) = g(x; w) = x \cdot w$ where g is the dot product function. Putting the ideas above together, the objective of learning is to find

$$w = \text{argmin}_{w \in \mathbb{R}^d} \ \lambda ||w||^2 + \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i(w \cdot x_i)\}.$$

A linear soft-margin SVM classifier is precisely the solution to this optimization problem. It can be proved that the solution to this minimization problem is always

unique. Moreover, the objective function is convex, so there are no local minima. Note that d is the dimensionality of x, and w has the same dimensionality.

An equivalent way of writing the same optimization problem is

$$w = \text{argmin}_{w \in \mathbb{R}^d} ||w||^2 + C \sum_{i=1}^{n} \max\{0, 1 - y_i(w \cdot x_i)\}$$

with $C = 1/(n\lambda)$. Many SVM implementations let the user specify C as opposed to λ. A small value for C corresponds to strong regularization, while a large value corresponds to weak regularization. Intuitively, everything else being equal, a smaller training set should require a smaller C value. However, useful guidelines are not known for what the best value of C might be for a given dataset. In practice, one has to try multiple values of C, and find the best value experimentally.

Mathematically, the optimization problem above is called an unconstrained primal formulation. There is an alternative formulation that is equivalent, and is useful theoretically. This so-called dual formulation is

$$\max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$
$$\text{subject to } 0 \leq \alpha_i \leq C.$$

The primal and dual formulations are different optimization problems, but they have the same unique solution. The solution to the dual problem is a coefficient α_i for each training example. Notice that the optimization is over $\mathbb{R}^n$, whereas it is over $\mathbb{R}^d$ in the primal formulation. The trained classifier is $f(x) = w \cdot x$ where the vector

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i.$$

This equation says that w is a weighted linear combination of the training instances x_i, where the weight of each instance is between 0 and C, and the sign of each instance is its label y_i. The training instances x_i such that $\alpha_i > 0$ are called support vectors. These instances are the only ones that contribute to the final classifier.

The constrained dual formulation is the basis of the training algorithms used by standard SVM implementations, but recent research has shown that the unconstrained primal formulation in fact leads to faster training algorithms, at least in the linear case as above. Moreover, the primal version is easier to understand and easier to use as a foundation for proving bounds on generalization error. However, the dual version is easier to extend to obtain nonlinear SVM classifiers. This extension is based on the idea of a kernel function.

4.4 Nonlinear kernels

Consider two instances x_i and x_j, and consider their dot product $x_i \cdot x_j$. This dot product is a measure of the similarity of the two instances: it is large if they are similar and small if they are not. Dot product similarity is closely related to Euclidean distance through the identity

$$d(x_i, x_j) = ||x_i - x_j|| = (||x_i||^2 - 2x_i \cdot x_j + ||x_j||^2)^{1/2}$$

where by definition $||x||^2 = x \cdot x$.[1] Therefore, the equation

$$f(x) = w \cdot x = \sum_{i=1}^{n} \alpha_i y_i (x_i \cdot x)$$

says that the prediction for a test example x is a weighted average of the training labels y_i, where the weight of y_i is the product of α_i and the degree to which x is similar to x_i.

Consider a re-representation of instances $x \mapsto \Phi(x)$ where the transformation Φ is a function $\mathbb{R}^d \to \mathbb{R}^{d'}$. In principle, we could use dot-product to define similarity in the new space $\mathbb{R}^{d'}$, and train an SVM classifier in this space. However, suppose we have a function $K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$. This function is all we need in order to write down the optimization problem and its solution; we do not need to know the function Φ in any explicit way. Specifically, let $k_{ij} = K(x_i, x_j)$. The learning task is to solve

$$\max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j k_{ij} \quad \text{subject to} \quad 0 \leq \alpha_i \leq C.$$

The solution is

$$f(x) = [\sum_{i=1}^{n} \alpha_i y_i \Phi(x_i)] \cdot x = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x).$$

This classifier is a weighted combination of at most n functions, one for each training instance x_i. These are called basis functions.

The result above says that in order to train a nonlinear SVM classifier, all that we need is the kernel matrix of size n by n whose entries are k_{ij}. And in order to apply

[1] If the instances have unit length, that is $||x_i|| = ||x_j|| = 1$, then Euclidean distance and dot product similarity are perfectly anticorrelated. For many applications of support vector machines, it is advantageous to normalize features to have the same mean and variance. It can be advantageous also to normalize instances so that they have unit length. However, in general one cannot have both normalizations be true at the same time.

the trained nonlinear classifier, all that we need is the kernel function K. The function Φ never needs to be known explicitly. Using K exclusively in this way instead of Φ is called the "kernel trick." Practically, the function K can be much easier to deal with than Φ, because K is just a mapping to $\mathbb{R}$, rather than to a high-dimensional space $\mathbb{R}^{d'}$.

Intuitively, regardless of which kernel K is used, that is regardless of which re-representation Φ is used, the complexity of the classifier f is limited, since it is defined by at most n coefficients α_i. The function K is fixed, so it does not increase the intuitive complexity of the classifier.

One particular kernel is especially important. The radial basis function (RBF) kernel is the function

$$K(x_i, x) = \exp(-\gamma||x_i - x||^2)$$

where $\gamma > 0$ is an adjustable parameter. Using an RBF kernel, each basis function $K(x_i, x)$ is "radial" since it is based on the Euclidean distance $||x_i - x||$ from x_i to x. With an RBF kernel, the classifier $f(x) = \sum_i \alpha_i y_i K(x_i, x)$ is similar to a nearest-neighbor classifier. Given a test instance x, its predicted label $f(x)$ is a weighted average of the labels y_i of the support vectors x_i. The support vectors that contribute non-negligibly to the predicted label are those for which the Euclidean distance $||x_i - x||$ is small.

The RBF kernel can also be written

$$K(x_i, x) = \exp(-||x_i - x||^2/\sigma^2)$$

where $\sigma^2 = 1/\gamma$. This notation emphasizes the similarity with a Gaussian distribution. A smaller value for γ, i.e. a larger value for σ^2, corresponds to basis functions that are less peaked, i.e. that are significantly non-zero for a wider range of x values. Using a larger value for σ^2 is similar to using a larger number k of neighbors in a nearest neighbor classifier.

4.5 Selecting the best SVM settings

Getting good results with support vector machines requires some care. The consensus opinion is that the best SVM classifier is typically at least as accurate as the best of any other type of classifier, but obtaining the best SVM classifier is not trivial. The following procedure is recommended as a starting point:

1. Code data in the numerical format needed for SVM training.
2. Scale each attribute to have range 0 to 1, or -1 to $+1$, or to have mean zero and unit variance.

3. Use cross-validation to find the best value for C for a linear kernel.
4. Use cross-validation to find the best values for C and γ for an RBF kernel.
5. Train on the entire available data using the parameter values found to be best via cross-validation.

It is reasonable to start with $C = 1$ and $\gamma = 1$, and to try values that are smaller and larger by factors of 2:

$$\langle C, \gamma \rangle \in \{\ldots, 2^{-3}, 2^{-2}, 2^{-1}, 1, 2, 2^2, 2^3, \ldots\}^2.$$

This process is called a grid search; it is easy to parallelize of course. It is important to try a wide enough range of values to be sure that more extreme values do not give better performance. Once you have found the best values of C and γ to within a factor of 2, doing a more fine-grained grid search around these values is sometimes beneficial, but not always.

Quiz question

(a) Draw the hinge loss function for the case where the true label $y = 1$. Label the axes clearly.

(b) Explain where the derivative is (i) zero, (ii) constant but not zero, or (iii) not defined.

(c) For each of the three cases for the derivative, explain its intuitive implications for training an SVM classifier.

Quiz for April 20, 2010

Your name:

Suppose that you have trained SVM classifiers carefully for a given learning task. You have selected the settings that yield the best linear classifier and the best RBF classifier. It turns out that both classifiers perform equally well in terms of accuracy for your task.

(a) Now you are given many times more training data. After finding optimal settings again and retraining, do you expect the linear classifier or the RBF classifier to have better accuracy? Explain very briefly.

(b) Do you expect the optimal settings to involve stronger regularization, or weaker regularization? Explain very briefly.

CSE 291 Assignment

This assignment is due at the start of class on Tuesday April 20. As before, you should work in a team of two, choosing a partner with a different background. You may keep the same partner as before, or change partners.

This project uses data published by Kevin Hillstrom, a well-known data mining consultant. You can find the data at `http://cseweb.ucsd.edu/users/elkan/250B/HillstromData.csv`. For a detailed description, see `http://minethatdata.com/blog/2008/03/minethatdata-e-mail-analytics-and-data.html`.

For this assignment, use only the data for customers who are not sent any email promotion. Your job is to train a good model to predict which customers visit the retailer's website. For now, you should ignore information about which customers make a purchase, and how much they spend.

Build support vector machine (SVM) models to predict the target label as accurately as possible. In the same general way as for linear regression, recode non-numerical features as numerical, and transform features to improve their usefulness. Train the best possible model using a linear kernel, and also the best possible model using a radial basis function (RBF) kernel. The outcome should be the two most accurate SVM classifiers that you can find, without overfitting or underfitting.

Decide thoughtfully which measure of accuracy to use, and explain your choice in your report. Use nested cross-validation carefully to find the best settings for training, and to evaluate the accuracy of the best classifiers as fairly as possible. In particular, you should identify good values of the soft-margin C parameter for both kernels, and of the width parameter for the RBF kernel.

For linear SVMs, the Rapidminer operator named FastLargeMargin is recommended. Because it is fast, you can explore models based on a large number of transformed features. Training nonlinear SVMs is much slower, but one can hope that good performance can be achieved with fewer features.

As before, the deliverable is a well-organized, well-written, and well-formatted report of about two pages. Describe what you did that worked, and your results. Explain any assumptions that you made, and any limitations on the validity or reliability of your results. Explain carefully your nested cross-validation procedure.

Include a printout of your final Rapidminer process, and a description of your two final models (not included in the two pages). Do not speculate about future work, and do not explain ideas that do not work. Write in the present tense. Organize your report logically, not chronologically.

Chapter 5

Classification with a rare class

In many data mining applications, the goal is to find needles in a haystack. That is, most examples are negative but a few examples are positive. The goal is to identify the rare positive examples, as accurately as possible. For example, most credit card transactions are legitimate, but a few are fraudulent. We have a standard binary classifier learning problem, but both the training and test sets are unbalanced. In a balanced set, the fraction of examples of each class is about the same. In an unbalanced set, some classes are rare while others are common.

5.1 Measuring performance

A major difficulty with unbalanced data is that accuracy is not a meaningful measure of performance. Suppose that 99% of credit card transactions are legitimate. Then we can get 99% accuracy by predicting trivially that every transaction is legitimate. On the other hand, suppose we somehow identify 5% of transactions for further investigation, and half of all fraudulent transactions occur among these 5%. Clearly the identification process is doing something worthwhile and not trivial. But its accuracy is only 95%.

For concreteness in further discussion, we will consider only the two class case, and we will call the rare class positive. Rather than talk about fractions or percentages of a set, we will talk about actual numbers (also called counts) of examples. It turns out that thinking about actual numbers leads to less confusion and more insight than thinking about fractions. Suppose the test set has a certain total size n, say $n = 1000$. We can represent the performance of the trivial classifier as follows:

		predicted	
		positive	negative
truth	positive	0	10
	negative	0	990

The performance of the non-trivial classifier is

		predicted	
		positive	negative
truth	positive	5	5
	negative	45	945

A table like the ones above is called a 2×2 contingency table. Above, rows corespond to actual labels, while columns correspond to predicted labels. It would be equally valid to swap the rows and columns. Unfortunately, there is no standard convention about whether rows are actual or predicted. Remember that there is a universal convention that in notation like x_{ij} the first subscript refers to rows while the second subscript refers to columns.

A table like the ones above is also called a confusion matrix. For supervised learning with discrete predictions, only a confusion matrix gives complete information about the performance of a classifier. No single number that summarizes performance, for example accuracy, can provide a full picture of the usefulness of a classifier.

The four entries in a 2×2 contingency table have standard names. They are called called true positives tp, false positives fp, true negatives tn, and false negatives fn, as follows:

		predicted	
		positive	negative
truth	positive	tp	fn
	negative	fp	tn

The terminology true positive, etc., is standard, but as mentioned above, whether columns correspond to predicted and rows to actual, or vice versa, is not standard.

As mentioned, the entries in a confusion matrix are counts, i.e. integers. The total of the four entries $tp+tn+fp+fn = n$, the number of test examples. Depending on the application, different summaries are computed from these entries. In particular, accuracy $a = (tp + tn)/n$. Assuming that n is known, three of the counts in a confusion matrix can vary independently. This is the reason why no single number

can describe completely the performance of a classifier. When writing a report, it is best to give the full confusion matrix explicitly, so that readers can calculate whatever performance measurements they are most interested in.

When accuracy is not meaningful, two summary measures that are commonly used are called precision and recall. They are defined as follows:

- precision $p = tp/(tp + fp)$, and
- recall $r = tp/(tp + fn)$.

The names "precision" and "recall" come from the field of information retrieval. In other research areas, recall is often called sensitivity, while precision is sometimes called positive predictive value.

Precision is undefined for a classifier that predicts that every test example is negative, that is when $tp + fp = 0$. Worse, precision can be misleadingly high for a classifier that predicts that only a few test examples are positive. Consider the following confusion matrix:

		predicted	
		positive	negative
truth	positive	1	9
	negative	0	990

Precision is 100% but 90% of actual positives are missed. F-measure is a widely used metric that overcomes this limitation of precision. It is the harmonic average of precision and recall:

$$F = \frac{1}{1/p + 1/r} = \frac{pr}{r + p}.$$

For the confusion matrix above $F = 1 \cdot 0.1/(1 + 0.1) = 0.09$.

Besides accuracy, precision, recall, and F-measure, many other summaries are also commonly computed from confusion matrices. Some of these are called specificity, false positive rate, false negative rate, positive and negative likelihood ratio, kappa coefficient, and more. Rather than rely on agreement and understanding of the definitions of these, it is preferable simply to report a full confusion matrix explicitly.

5.2 Thresholds and lift

A confusion matrix is always based on discrete predictions. Often, however, these predictions are obtained by thresholding a real-valued predicted score. For example, an SVM classifier yields a real-valued prediction $f(x)$ which is then compared to

the threshold zero to obtain a discrete yes/no prediction. Confusion matrices cannot represent information about the usefulness of underlying real-valued predictions. We shall return to this issue below, but first we shall consider the issue of selecting a threshold.

Setting the threshold determines the number $tp + fp$ of examples that are predicted to be positive. In some scenarios, there is a natural threshold such as zero for an SVM. However, even when a natural threshold exists, it is possible to change the threshold to achieve a target number of positive predictions. This target number is often based on a so-called budget constraint. Suppose that all examples predicted to be positive are subjected to further investigation. An external limit on the resources available will determine how many examples can be investigated. This number is a natural target for the value $fp+tp$. Of course, we want to investigate those examples that are most likely to be actual positives, so we want to investigate the examples x with the highest prediction scores $f(x)$. Therefore, the correct strategy is to choose a threshold t such that we investigate all examples x with $f(x) \geq t$ and the number of such examples is determined by the budget constraint.

Given that a fixed number of examples are predicted to be positive, a natural question is how good a classifier is at capturing the actual positives within this number. This question is answered by a measure called lift. The definition is a bit complex. First, let the fraction of examples predicted to be positive be $x = (tp + fp)/n$. Next, let the base rate of actual positive examples be $b = (tp + fn)/n$. Let the density of actual positives within the predicted positives be $d = tp/(tp + fp)$. Now, the lift at x is defined to be the ratio d/b. Intuitively, a lift of 2 means that actual positives are twice as dense among the predicted positives as they are among all examples.

Lift can be expressed as

$$\begin{aligned} \frac{d}{b} &= \frac{tp/(tp+fp)}{(tp+fn)/n} = \frac{tp \cdot n}{(tp+fp)(tp+fn)} \\ &= \frac{tp}{tp+fn}\frac{n}{tp+fp} = \frac{recall}{prediction\ rate}. \end{aligned}$$

Lift is a useful measure of success if the number of examples that should be predicted to be positive is determined by external considerations. However, budget constraints should normally be questioned. In the credit card scenario, perhaps too many transactions are being investigated and the marginal benefit of investigating some transactions is negative. Or, perhaps too few transactions are being investigated; there would be a net benefit if additional transactions were investigated. Making optimal decisions about how many examples should be predicted to be positive is discussed in the next chapter.

While a budget constraint is not normally a rational way of choosing a threshold for predictions, it is still more rational than choosing an arbitrary threshold. In par-

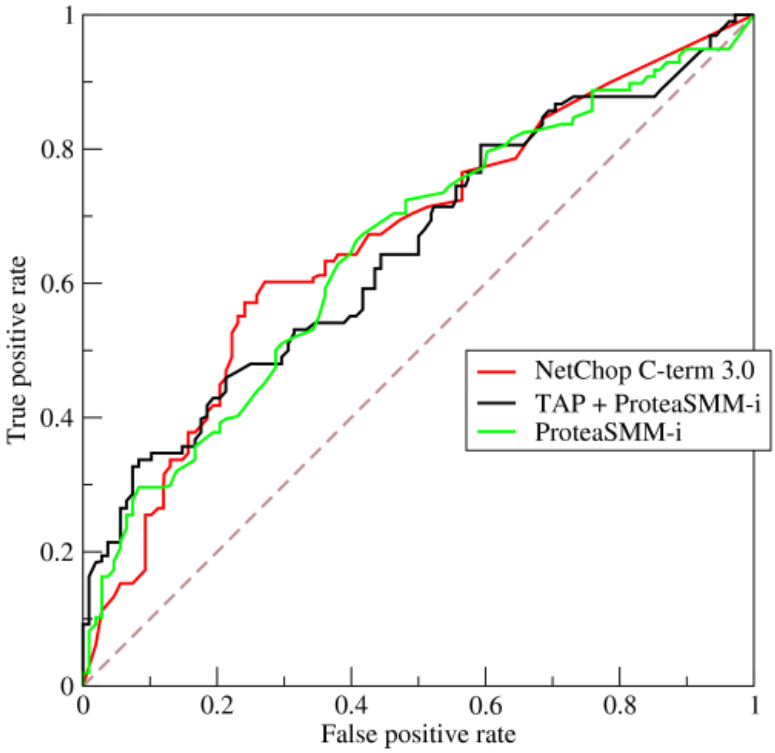

Figure 5.1: ROC curves for three alternative binary classifiers. Source: Wikipedia.

ticular, the threshold zero for an SVM classifier has some mathematical meaning but is not a rational guide for making decisions.

5.3 Ranking examples

Applying a threshold to a classifier with real-valued outputs loses information, because the distinction between examples on the same side of the threshold is lost. Moreover, at the time a classifier is trained and evaluated, it is often the case that the threshold to be used for decision-making is not known. Therefore, it is useful to compare different classifiers across the range of all possible thresholds.

Typically, it is not meaningful to use the same numerical threshold directly for different classifiers. However, it is meaningful to compare the recall achieved by different classifiers when the threshold for each one is set to make their precisions

equal. This is what an ROC curve does.[1] Concretely, an ROC curve is a plot of the performance of a classifier, where the horizontal axis measures false positive rate (fpr) and the vertical axis measures true positive rate (tpr). These are defined as

$$fpr = \frac{fp}{fp + tn}$$
$$tpr = \frac{tp}{tp + fn}$$

Note that tpr is the same as recall and is sometimes also called "hit rate."

In an ROC plot, the ideal point is at the top left. One classifier uniformly dominates another if its curve is always above the other's curve. It happens often that the ROC curves of two classifiers cross, which implies that neither one dominates the other uniformly.

ROC plots are informative, but do not provide a quantitative measure of the performance of classifiers. A natural quantity to consider is the area under the ROC curve, often abbreviated AUC. The AUC is 0.5 for a classifier whose predictions are no better than random, while it is 1 for a perfect classifier. The AUC has an intuitive meaning: it can be proved that it equals the probability that the classifier will rank correctly two randomly chosen examples where one is positive and one is negative.

One reason why AUC is widely used is that, as shown by the probabilistic meaning just mentioned, it has built into it the implicit assumption that the rare class is more important than the common class.

5.4 Conditional probabilities

Reason 1 to want probabilities: predicting expectations.

Ideal versus achievable probability estimates.

Reason 2 to want probabilities: understanding predictive power.

Definition of well-calibrated probability.

Brier score.

Converting scores into probabilities.

5.5 Isotonic regression

Let f_i be prediction scores on a dataset, and let $y_i \in \{0, 1\}$ be the corresponding true labels. Let f^j be the f_i values sorted from smallest to largest, and let y^j be the y_i

[1] ROC stands for "receiver operating characteristic." This terminology originates in the theory of detection based on electromagnetic waves.

values sorted in the same order. For each f^j we want to find an output value g_j such that the g_j values are monotonically increasing, and squared error relative to the y^j values is minimized. Formally, the optimization problem is

$$\min_{g_1,\ldots,g_n} (y^j - g_j)^2 \text{ subject to } g_j \leq g_{j+1} \text{ for } j = 1 \text{ to } j = n-1$$

It is a remarkable fact that if squared error is minimized, then the resulting predictiobns are well-calibrated probabilities.

There is an elegant algorithm called "pool adjacent violators" (PAV) that solves this problem in linear time. The algorithm is as follows, where pooling a set means replacing each member of the set by the arithmetic mean of the set.

Let $g_j = y^j$ for all j
Start with $j = 1$ and increase j until the first j such that $g_j \not\leq g_{j+1}$
Pool g_j and g_{j+1}
Move left: If $g_{j-1} \not\leq g_j$ then pool g_{j-1} to g_{j+1}
Continue to the left until monotonicity is satisfied
Proceed to the right

Given a test example x, the procedure to predict a well-calibrated probability is as follows:

Apply the classifier to obtain $f(x)$
Find j such that $f^j \leq f(x) \leq f^{j+1}$
The predicted probability is g_j.

5.6 Univariate logistic regression

The disadvantage of isotonic regression is that it creates a lookup table for converting scores into estimated probabilities. An alternative is to use a parametric model. The most common model is called univariate logistic regression. The model is

$$\log \frac{p}{1-p} = a + bf$$

where f is a prediction score and p is the corresponding estimated probability.

The equation above shows that the logistic regression model is essentially a linear model with intercept a and coefficient b. An equivalent way of writing the model is

$$p = \frac{1}{1 + e^{-(a+bf)}}.$$

As above, let f_i be prediction scores on a training set, and let $y_i \in \{0, 1\}$ be the corresponding true labels. The parameters a and b are chosen to minimize the total

loss

$$\sum_i l(\frac{1}{1+e^{-(a+bf_i)}}, y_i).$$

The precise loss function l that is typically used in the optimization is called conditional log likelihood (CLL) and is explained in Chapter ??? below. However, one could also use squared error, which would be consistent with isotonic regression.

Quiz

All questions below refer to a classifier learning task with two classes, where the base rate for the positive class $y = 1$ is 5%.

(a) Suppose that a probabilistic classifier predicts $p(y = 1|x) = c$ for some constant c, for all test examples x. Explain why $c = 0.05$ is the best value for c.

The value 0.05 is the only constant that is a well-calibrated probability. "Well-calibrated" means that $c = 0.05$ equals the average frequency of positive examples in sets of examples with predicted score c.

(b) What is the error rate of the classifier from part (a)? What is its MSE?

With a prediction threshold of 0.5, all examples are predicted to be negative, so the error rate is 5%. The MSE is

$$0.05 \cdot (1 - 0.05)^2 + 0.95 \cdot (0 - 0.05)^2 = 0.05 \cdot 0.95 \cdot 1$$

which equals 0.0475.

(c) Suppose a well-calibrated probabilistic classifier satisfies $a \leq p(y = 1|x) \leq b$ for all x. What is the maximum possible lift at 10% of this classifier?

If the upper bound b is a well-calibrated probability, then the fraction of positive examples among the highest-scoring examples is at most b. Hence, the lift is at most $b/0.05$ where 0.05 is the base rate. Note that this is the highest possible lift at any percentage, not just at 10%.

2009 Assignment

This assignment is due at the start of class on Tuesday April 28, 2009. As before, you should work in a team of two. You may change partners, or keep the same partner.

Like previous assignments, this one uses the KDD98 training set. However, you should now use the entire dataset. (Revised.) The goal is to train a classifier with real-valued outputs that identifies test examples with `TARGET_B` $= 1$. Specifically, the measure of success to optimize is lift at 10%. That is, as many positive test examples as possible should be among the 10% of test examples with highest prediction score.

You should use logistic regression first, because this is a fast and reliable method for training probabilistic classifiers. If you are using Rapidminer, then use the logistic regression option of the `FastLargeMargin` operator. As before, recode and transform features to improve their usefulness. Do feature selection to reduce the size of the training set as much as is reasonably possible.

Next, you should apply a different learning method to the same training set that you developed using logistic regression. The objective is to see whether this other method can perform better than logistic regression, using the same data code in the same way. The second learning method can be a support vector machine, a neural network, or a decision tree, for example. Apply cross-validation to find good algorithm settings.

(Deleted: When necessary, use a postprocessing method (isotonic regression or logistic regression) to obtain calibrated estimates of conditional probabilities. Investigate the probability estimates produced by your two methods. What are the minimum, mean, and maximum predicted probabilities? Discuss whether these are reasonable.

Assignment

This assignment is due at the start of class on Tuesday April 27, 2010. As before, you should work in a team of two. You may change partners, or keep the same partner.

Like the previous assignment, this one uses the e-commerce data published by Kevin Hillstrom However, the goal now is to predict who makes a purchase on the website (again, for customers who are not sent any promotion). This is a highly unbalanced classification task.

First, use logistic regression. If you are using Rapidminer, then use the logistic regression option of the `FastLargeMargin` operator. As before, recode and transform features to improve their usefulness. Investigate the probability estimates produced by your two methods. What are the minimum, mean, and maximum predicted probabilities? Discuss whether these are reasonable.

Second, use your creativity to get the best possible performance in predicting who makes a purchase. The measure of success to optimize is lift at 25%. That is, as many positive test examples as possible should be among the 25% of test examples with highest prediction score. You may apply any learning algorithms that you like. Can any method achieve better accuracy than logistic regression?

For predicting who makes a purchase, compare learning a classifier directly with learning two classifiers, the first to predict who visits the website and the second to predict which visitors make purchases. Note that mathematically

$$p(buy|x) = p(buy|x, visit)p(visit|x).$$

As before, be careful not to fool yourself about the success of your methods.

Quiz for April 27, 2010

Your name:

The current assignment is based on the equation

$$p(buy = 1|x) = p(buy = 1|x, visit = 1)p(visit = 1|x).$$

Explain clearly but briefly why this equation is true.

Assignment

The purpose of this assignment is to help develop three important abilities. The first ability is critical thinking, i.e. the skill of identifying what is most important and then identifying what may be incorrect. The second ability is understanding a new application domain quickly, in this case a task in computational biology. The third ability is presenting your opinions in a persuasive way. You must explain your arguments and conclusions crisply, concisely, and clearly.

The paper you should read is *Predicting protein-protein interactions from primary structure* by Joel Bock and David Gough, published in the journal *Bioinformatics* in 2001. The full text of this paper in PDF is supposed to be available free. If you have difficulty obtaining it, please post on the class message board.

You should figure out and describe three major flaws in the paper. The flaws concern

- how the dataset is constructed,
- how each example is represented, and
- how performance is measured and reported.

Each of the three mistakes is serious. The paper has 248 citations according to Google Scholar as of May 26, 2009, but unfortunately each flaw by itself makes the results of the paper not useful as a basis for future research. Each mistake is described sufficiently clearly in the paper: it is a sin of commission, not a sin of omission.

The second mistake, how each example is represented, is the most subtle, but at least one of the papers citing this paper does explain it clearly. It is connected with how SVMs are applied here. Remember the slogan: "If you cannot represent it then you cannot learn it."

Separately, provide a brief critique of the four benefits claimed for SVMs in the section of the paper entitled *Support vector machine learning*. Are these benefits true? Are they unique to SVMs? Does the work described in this paper take advantage of them?

Sample answers

Here is a brief summary of what I see as the most important flaws of the paper *Predicting protein-protein interactions from primary structure*.

(1) How the dataset is constructed. The problem here is that the negative examples are not pairs of genuine proteins. Instead, they are pairs of randomly generated amino acid sequences. It is quite possible that these artificial sequences could not fold into actual proteins at all. The classifiers reported in this paper may learn mainly to distinguish between real proteins and non-proteins.

The authors acknowledge this concern, but do not overcome it. They could have used pairs of genuine proteins as negative examples. It is true that one cannot be sure that any given pair really is non-interacting. However, the great majority of pairs do not interact. Moreover, if a negative example really is an interaction, that will presumably slightly reduce the apparent accuracy of a trained classifier, but not change overall conclusions.

(2) How each example is represented. This is a subtle but clear-cut and important issue, assuming that the research uses a linear classifier.

Let x_1 and x_2 be two proteins and let $f(x_1)$ and $f(x_2)$ be their representations as vectors in $\mathbb{R}^d$. The pair of proteins is represented as the concatenated vector $\langle f(x_1)\ f(x_2)\rangle \in \mathbb{R}^{2d}$. Suppose a trained linear SVM has parameter vector w. By definition $w \in \mathbb{R}^{2d}$ also. (If there is a bias coefficient, so $w \in \mathbb{R}^{2d+1}$, the conclusion is the same.)

Now suppose the first protein x_1 is fixed and consider varying the second protein x_2. Proteins x_2 will be ranked according to the numerical value of the dot product $w \cdot \langle f(x_1)\ f(x_2)\rangle$. This is equal to $w_1 \cdot f(x_1) + w_2 \cdot f(x_2)$ where the vector w is written as $\langle w_1\ w_2\rangle$. If x_1 is fixed, then the first term is constant and the second term $w_2 \cdot f(x_2)$ determines the ranking. The ranking of x_2 proteins will be the same regardless of what the x_1 protein is. This fundamental drawback of linear classifiers for predicting interactions is pointed out in [Vert and Jacob, 2008, Section 5].

With a concatenated representation of protein pairs, a linear classifier can at best learn the propensity of individual proteins to interact. Such a classifier cannot represent patterns that depend on features that are true only for specific protein pairs. This is the relevance of the slogan "If you cannot represent it then you cannot learn it."

Note: Previously I was under the impression that the authors stated that they used a linear kernel. On rereading the paper, it fails to mention at all what kernel they use. If the research uses a linear kernel, then the argument above is applicable.

(3) How performance is measured and reported. Most pairs of proteins are non-interacting. It is reasonable to use training sets where negative examples (non-interacting pairs) are undersampled. However, it is not reasonable or correct to report

performance (accuracy, precision, recall, etc.) on test sets where negative examples are under-represented, which is what is done in this paper.

As for the four claimed advantages of SVMs:

1. SVMs are nonlinear while requiring "relatively few" parameters.

   The authors do not explain what kernel they use. In any case, with a nonlinear kernel the number of parameters is the number of support vectors, which is often close to the number of training examples. It is not clear relative to what this can be considered "few."

2. SVMs have an analytic upper bound on generalization error.

   This upper bound does motivate the SVM approach to training classifiers, but it typically does not provide useful guarantees for specific training sets. In any case, a bound of this type has nothing to do with assigning confidences to individual predictions. In practice overfitting is prevented by straightforward search for the value of the C parameter that is empirically best, not by applying a theorem.

3. SVMs have fast training, which is essential for screening large datasets.

   SVM training is slow compared to many other classifier learning methods, except for linear classifiers trained by fast algorithms that were only published after 2001, when this paper was published. As mentioned above, a linear classifier is not appropriate with the representation of protein pairs used in this paper.

   In any case, what is needed for screening many test examples is fast classifier application, not fast training. Applying a linear classifier is fast, whether it is an SVM or not. Applying a nonlinear SVM typically has the same order-of-magnitude time complexity as applying a nearest-neighbor classifier, which is the slowest type of classifier in common use.

4. SVMs can be continuously updated in response to new data.

   At least one algorithm is known for updating an SVM given a new training example, but it is not cited in this paper. I do not know any algorithm for training an optimal new SVM efficiently, that is without retraining on old data. In any case, new real-world protein data arrives slowly enough that retraining from scratch is feasible.

Chapter 6

Detecting overfitting: cross-validation

6.1 Cross-validation

Usually we have a fixed database of labeled examples available, and we are faced with a dilemma: we would like to use all the examples for training, but we would also like to use many examples as an independent test set. Cross-validation is a procedure for overcoming this dilemma. It is the following algorithm.

Input: Training set S, integer constant k
Procedure:
  partition S into k disjoint equal-sized subsets $S_1, \ldots, S_k$
  for $i = 1$ to $i = k$
    let $T = S \setminus S_i$
    run learning algorithm with T as training set
    test the resulting classifier on S_i obtaining tp_i, fp_i, tn_i, fn_i
  compute $tp = \sum_i tp_i$, $fp = \sum_i fp_i$, $tn = \sum_i tn_i$, $fn = \sum_i fn_i$

The output of cross-validation is a confusion matrix based on using each labeled example as a test example exactly once. Whenever an example is used for testing a classifier, it has not been used for training that classifier. Hence, the confusion matrix obtained by cross-validation is a fair indicator of the performance of the learning algorithm on independent test examples.

If n labeled examples are available, the largest possible number of folds is $k = n$. This special case is called leave-one-out cross-validation (LOOCV). However, the time complexity of cross-validation is k times that of running the training algorithm

once, so often LOOCV is computationally infeasible. In recent research the most common choice for k is 10.

Note that cross-validation does not produce any single final classifier, and the confusion matrix it provides is not the performance of any specific single classifier. Instead, this matrix is an estimate of the average performance of a classifier learned from a training set of size $(k-1)n/k$ where n is the size of S. The common procedure is to create a final classifier by training on all of S, and then to use the confusion matrix obtained from cross-validation as an informal estimate of the performance of this classifier. This estimate is likely to be conservative in the sense that the final classifier may have slightly better performance since it is based on a slightly larger training set.

Suppose that the time to train a classifier is proportional to the number of training examples, and the time to make predictions on test examples is negligible in comparison. The time required for k-fold cross-validation is then $O(k \cdot (k-1)/k \cdot n) = O((k-1)n)$. Three-fold cross-validation is therefore twice as time-consuming as two-fold. This suggests that for preliminary experiments, two-fold is a good choice.

The results of cross-validation can be misleading. For example, if each example is duplicated in the training set and we use a nearest-neighbor classifier, then LOOCV will show a zero error rate. Cross-validation with other values of k will also yield misleadingly low error estimates.

Subsampling means omitting examples from some classes. Subsampling the common class is a standard approach to learning from unbalanced data, i.e. data where some classes are very common while others are very rare. It is reasonable to do subsampling in the training folds of cross-validation, but not in the test fold. Reported performance numbers should always be based on a set of test examples that is directly typical of a genuine test population. It is a not uncommon mistake with cross-validation to do subsampling on the test set.

6.2 Nested cross-validation

A model selection procedure is a method for choosing the best learning algorithm from a set of alternatives. Often, the alternatives are all the same algorithm but with different parameter settings, for example different C and γ values. Another common case is where the alternatives are different subsets of features. Typically, the number of alternatives is finite and the only way to evaluate an alternative is to run it explicitly on a training set. So, how should we do model selection?

A simple way to apply cross-validation for model selection is the following:

Input: dataset S, integer k, set V of alternative algorithm settings
Procedure:

partition S randomly into k disjoint subsets $S_1, \ldots, S_k$ of equal size
for each setting v in V
    for $i = 1$ to $i = k$
        let $T = S \setminus S_i$
        run the learning algorithm with setting v and T as training set
        apply the trained model to S_i obtaining performance e_i
    end for
    let $M(v)$ be the average of e_i
end for
select $\hat{v} = \text{argmax}_v \, M(v)$

The output of this model selection procedure is $\hat{v}$. The input set V of alternative settings can be a grid of parameter values.

But, any procedure for selecting parameter values is itself part of the learning algorithm. It is crucial to understand this point. The setting $\hat{v}$ is chosen to maximize $M(v)$, so $M(\hat{v})$ is not a fair estimate of the performance to be expected from $\hat{v}$ on future data. Stated another way, $\hat{v}$ is chosen to optimize performance on all of S, so $\hat{v}$ is likely to overfit S.

Notice that the division of S into subsets happens just once, and the same division is used for all settings v. This choice reduces the random variability in the evaluation of different v. A new partition of S could be created for each setting v, but this would not overcome the issue that $\hat{v}$ is chosen to optimize performance on S.

What should we do about the fact that any procedure for selecting parameter values is itself part of the learning algorithm? One answer is that this procedure should itself be evaluated using cross-validation. This process is called nested cross-validation, because one cross-validation is run inside another.

Specifically, nested cross-validation is the following process:

Input: dataset S, integers k and k', set V of alternative algorithm settings
Procedure:
    partition S randomly into k disjoint subsets $S_1, \ldots, S_k$ of equal size
    for $i = 1$ to $i = k$
        let $T = S \setminus S_i$
        partition T randomly into k' disjoint subsets $T_1, \ldots, T_{k'}$ of equal size
        for each setting v in V
            for $j = 1$ to $j = k'$
                let $U = T \setminus T_j$
                run the learning algorithm with setting v and U as training set
                apply the trained model to T_j obtaining performance e_j
            end for

```
        let M(v) be the average of e_j
    end for
    select v̂ = argmax_v M(v)
    run the learning algorithm with setting v̂ and T as training set
    apply the trained model to S_i obtaining performance e_i
end for
report the average of e_i
```

Now, the final reported average e_i is the estimated performance of the classifier obtained by running the same model selection procedure (i.e. the search over each setting v) on the whole dataset S.

Some notes:

1. Trying every setting in V explicitly can be prohibitive. It may be preferable to search in V in a more clever way, using a genetic algorithm or some other heuristic method. As mentioned above, the search for a good member of V is itself part of the learning algorithm, so it can certainly be intelligent and/or heuristic.

2. Above, the same partition of T is used for each setting v. This reduces randomness a little bit compared to using a different partition for each v, but the latter would be correct also.

Quiz for April 13, 2010

Your name:

The following cross-validation procedure is intended to find the best regularization parameter R for linear regression, and to report a fair estimate of the RMSE to be expected on future test data.

Input: dataset S, integer k, set V of alternative R values
Procedure:
  partition S randomly into k disjoint subsets $S_1, \ldots, S_k$ of equal size
  for each R value in V
    for $i = 1$ to $i = k$
      let $T = S \setminus S_i$
      train linear regression with parameter R and T as training set
      apply the trained regression equation to S_i obtaining SSE e_i
    end for
  compute $M = \sum_i e_i$
  report R and $RMSE = \sqrt{M/|S|}$
  end for

(a) Suppose that you choose the R value for which the reported RMSE is lowest. Explain why this method is likely to be overoptimistic, as an estimate of the RMSE to be expected on future data.

Each R value is being evaluated on the entire set S. The value that seems to be best is likely overfitting this set.

(b) Very briefly, suggest an improved variation of the method above.

The procedure to select a value for R is part of the learning algorithm. This whole procedure should be evaluated using a separate test set, or via cross-validation,

Additional notes: The procedure to select a value for R uses cross-validation, so if this procedure is evaluated itself using cross-validation, then the entire process is nested cross-validation.

Incorrect answers include the following:

- "The partition of S should be stratified." No; first of all, we are doing regression, so stratification is not well-defined, and second, failing to stratify increases variability but not does not cause overfitting.
- "The partition of S should be done separately for each R value, not just once." No; a different partition for each R value might increase the variability in the evaluation of each R, but it would not change the fact that the best R is being selected according to its performance on all of S.

Two basic points to remember are that it is never fair to evaluate a learning method on its training set, and that any search for settings for a learning algorithm (e.g. search for a subset of features, or for algorithmic parameters) is part of the learning method.

Chapter 7

Making optimal decisions

This chapter discusses making optimal decisions based on predictions, and maximizing the value of customers.

7.1 Predictions, decisions, and costs

Decisions and predictions are conceptually very different. For example, a prediction concerning a patient may be "allergic" or "not allergic" to aspirin, while the corresponding decision is whether or not to administer the drug. Predictions can often be probabilistic, while decisions typically cannot.

Suppose that examples are credit card transactions and the label $y = 1$ designates a legitimate transaction. Then making the decision $y = 1$ for an attempted transaction means acting as if the transaction is legitimate, i.e. approving the transaction. The essence of cost-sensitive decision-making is that it can be optimal to act as if one class is true even when some other class is more probable. For example, if the cost of approving a fraudulent transaction is proportional to the dollar amount involved, then it can be rational not to approve a large transaction, even if the transaction is most likely legitimate. Conversely, it can be rational to approve a small transaction even if there is a high probability it is fraudulent.

Mathematically, let i be the predicted class and let j be the true class. If $i = j$ then the prediction is correct, while if $i \neq j$ the prediction is incorrect. The (i, j) entry in a cost matrix c is the cost of acting as if class i is true, when in fact class j is true. Here, predicting i means acting as if i is true, so one could equally well call this deciding i.

A cost matrix c has the following structure when there are only two classes:

	actual negative	actual positive
predict negative	$c(0,0) = c_{00}$	$c(0,1) = c_{01}$
predict positive	$c(1,0) = c_{10}$	$c(1,1) = c_{11}$

The cost of a false positive is c_{10} while the cost of a false negative is c_{01}. We follow the convention that cost matrix rows correspond to alternative predicted classes, while columns correspond to actual classes. In short the convention is row/column = i/j = predicted/actual. (This convention is the opposite of the one in Section 5.1 so perhaps we should switch one of these to make the conventions similar.)

The optimal prediction for an example x is the class i that minimizes the expected cost

$$e(x,i) = \sum_j p(j|x)c(i,j). \tag{7.1}$$

For each i, $e(x,i)$ is an expectation computed by summing over the alternative possibilities for the true class of x. In this framework, the role of a learning algorithm is to produce a classifier that for any example x can estimate the probability $p(j|x)$ of each class j being the true class of x.

7.2 Cost matrix properties

Conceptually, the cost of labeling an example incorrectly should always be greater than the cost of labeling it correctly. For a 2x2 cost matrix, this means that it should always be the case that $c_{10} > c_{00}$ and $c_{01} > c_{11}$. We call these conditions the "reasonableness" conditions.

Suppose that the first reasonableness condition is violated, so $c_{00} \geq c_{10}$ but still $c_{01} > c_{11}$. In this case the optimal policy is to label all examples positive. Similarly, if $c_{10} > c_{00}$ but $c_{11} \geq c_{01}$ then it is optimal to label all examples negative. (The reader can analyze the case where both reasonableness conditions are violated.)

For some cost matrices, some class labels are never predicted by the optimal policy given by Equation (7.1). The following is a criterion for when this happens. Say that row m dominates row n in a cost matrix C if for all j, $c(m,j) \geq c(n,j)$. In this case the cost of predicting n is no greater than the cost of predicting m, regardless of what the true class j is. So it is optimal never to predict m. As a special case, the optimal prediction is always n if row n is dominated by all other rows in a cost matrix. The two reasonableness conditions for a two-class cost matrix imply that neither row in the matrix dominates the other.

Given a cost matrix, the decisions that are optimal are unchanged if each entry in the matrix is multiplied by a positive constant. This scaling corresponds to changing the unit of account for costs. Similarly, the decisions that are optimal are unchanged

if a constant is added to each entry in the matrix. This shifting corresponds to changing the baseline away from which costs are measured. By scaling and shifting entries, any two-class cost matrix

c_{00}	c_{01}
c_{10}	c_{11}

that satisfies the reasonableness conditions can be transformed into a simpler matrix that always leads to the same decisions:

0	c'_{01}
1	c'_{11}

where $c'_{01} = (c_{01} - c_{00})/(c_{10} - c_{00})$ and $c'_{11} = (c_{11} - c_{00})/(c_{10} - c_{00})$. From a matrix perspective, a 2x2 cost matrix effectively has two degrees of freedom.

7.3 The logic of costs

Costs are not necessarily monetary. A cost can also be a waste of time, or the severity of an illness, for example. Although most recent research in machine learning has used the terminology of costs, doing accounting in terms of benefits is generally preferable, because avoiding mistakes is easier, since there is a natural baseline from which to measure all benefits, whether positive or negative. This baseline is the state of the agent before it takes a decision regarding an example. After the agent has made the decision, if it is better off, its benefit is positive. Otherwise, its benefit is negative.

When thinking in terms of costs, it is easy to posit a cost matrix that is logically contradictory because not all entries in the matrix are measured from the same baseline. For example, consider the so-called German credit dataset that was published as part of the Statlog project [Michie et al., 1994]. The cost matrix given with this dataset at `http://www.sgi.com/tech/mlc/db/german.names` is as follows:

	actual bad	actual good
predict bad	0	1
predict good	5	0

Here examples are people who apply for a loan from a bank. “Actual good” means that a customer would repay a loan while “actual bad” means that the customer would default. The action associated with “predict bad” is to deny the loan. Hence, the cashflow relative to any baseline associated with this prediction is the same regardless

of whether "actual good" or "actual bad" is true. In every economically reasonable cost matrix for this domain, both entries in the "predict bad" row must be the same. If these entries are different, it is because different baselines have been chosen for each entry.

Costs or benefits can be measured against any baseline, but the baseline must be fixed. An opportunity cost is a foregone benefit, i.e. a missed opportunity rather than an actual penalty. It is easy to make the mistake of measuring different opportunity costs against different baselines. For example, the erroneous cost matrix above can be justified informally as follows: "The cost of approving a good customer is zero, and the cost of rejecting a bad customer is zero, because in both cases the correct decision has been made. If a good customer is rejected, the cost is an opportunity cost, the foregone profit of 1. If a bad customer is approved for a loan, the cost is the lost loan principal of 5."

To see concretely that the reasoning in quotes above is incorrect, suppose that the bank has one customer of each of the four types. Clearly the cost matrix above is intended to imply that the net change in the assets of the bank is then -4. Alternatively, suppose that we have four customers who receive loans and repay them. The net change in assets is then +4. Regardless of the baseline, any method of accounting should give a difference of 8 between these scenarios. But with the erroneous cost matrix above, the first scenario gives a total cost of 6, while the second scenario gives a total cost of 0.

In general the amount in some cells of a cost or benefit matrix may not be constant, and may be different for different examples. For example, consider the credit card transactions domain. Here the benefit matrix might be

	fraudulent	legitimate
refuse	\$20	$-$\$20
approve	$-x$	$0.02x$

where x is the size of the transaction in dollars. Approving a fraudulent transaction costs the amount of the transaction because the bank is liable for the expenses of fraud. Refusing a legitimate transaction has a non-trivial cost because it annoys a customer. Refusing a fraudulent transaction has a non-trivial benefit because it may prevent further fraud and lead to the arrest of a criminal.

7.4 Making optimal decisions

Given known costs for correct and incorrect predictions, a test example should be predicted to have the class that leads to the lowest expected cost. This expectation is

the predicted average cost, and is computed using the conditional probability of each class given the example.

In the two-class case, the optimal prediction is class 1 if and only if the expected cost of this prediction is less than or equal to the expected cost of predicting class 0, i.e. if and only if

$$p(y=0|x)c_{10} + p(y=1|x)c_{11} \leq p(y=0|x)c_{00} + p(y=1|x)c_{01}$$

which is equivalent to

$$(1-p)c_{10} + pc_{11} \leq (1-p)c_{00} + pc_{01}$$

given $p = p(y=1|x)$. If this inequality is in fact an equality, then predicting either class is optimal.

The threshold for making optimal decisions is p^* such that

$$(1-p^*)c_{10} + p^*c_{11} = (1-p^*)c_{00} + p^*c_{01}.$$

Assuming the reasonableness conditions $c_{10} > c_{00}$ and $c_{01} \geq c_{11}$, the optimal prediction is class 1 if and only if $p \geq p^*$. Rearranging the equation for p^* leads to

$$c_{00} - c_{10} = -p^*c_{10} + p^*c_{11} + p^*c_{00} - p^*c_{01}$$

which has the solution

$$p^* = \frac{c_{10} - c_{00}}{c_{10} - c_{00} + c_{01} - c_{11}}$$

assuming the denominator is nonzero, which is implied by the reasonableness conditions. This formula for p^* shows that any 2x2 cost matrix has essentially only one degree of freedom from a decision-making perspective, although it has two degrees of freedom from a matrix perspective. The cause of the apparent contradiction is that the optimal decision-making policy is a nonlinear function of the cost matrix.

Note that in some domains costs have more impact than probabilities on decision-making. An extreme case is that with some cost matrices the optimal decision is always the same, regardless of what the various outcome probabilities are for a given example x.[1]

Decisions and predictions may be not in 1:1 correspondence.

[1] The situation where the optimal decision is fixed is well-known in philosophy as Pascal's wager. Essentially, this is the case where the minimax decision and the minimum expected cost decision are always the same. An argument analogous to Pascal's wager provides a justification for Occam's razor in machine learning. Essentially, a PAC theorem says that if we have a fixed, limited number of training examples, and the true concept is simple, then if we pick a simple concept then we will be right. However if we pick a complex concept, there is no guarantee we will be right.

7.5 Limitations of cost-based analysis

The analysis of decision-making above assumes that the objective is to minimize expected cost. This objective is reasonable when a game is played repeatedly, and the actual class of examples is set stochastically.

The analysis may not be appropriate when the agent is risk-averse and can make only a few decisions.

Also not appropriate when the actual class of examples is set by a non-random adversary.

Costs may need to be estimated via learning.

We may need to make repeated decisions over time about the same example.

Decisions about one example may influence other examples.

When there are more than two classes, we do not get full label information on training examples.

7.6 Rules of thumb for evaluating data mining campaigns

Let q be a fraction of the test set. There is a remarkable rule of thumb that the lift attainable at q is around $\sqrt{1/q}$. For example, if $q = 0.25$ then the attainable lift is around $\sqrt{4} = 2$.

The rule of thumb is not valid for very small values of q, for two reasons. The first reason is mathematical: an upper bound on the lift is $1/t$, where t is the overall fraction of positives; t is also called the target rate, response rate, or base rate. The second reason is empirical: lifts observed in practice tend to be well below 10. Hence, the rule of thumb can reasonably be applied when $q > 0.02$ and $q > t^2$.

The fraction of all positive examples that belong to the top-ranked fraction q of the test set is $q\sqrt{1/q} = \sqrt{q}$. The lift in the bottom $1 - q$ is $(1 - \sqrt{q})/(1 - q)$. We have

$$\lim_{q \to 1} \frac{1 - \sqrt{q}}{1 - q} = 0.5.$$

This says that the examples ranked lowest are positive with probability equal to $t/2$. In other words, the lift for the lowest ranked examples cannot be driven below 0.5.

Let c be the cost of a contact and let b be the benefit of a positive. This means that the benefit matrix is

	nature simple	nature complex
predict simple	succeed	fail
predict complex	fail	fail

Here "simple" means that the hypothesis is drawn from a space with low cardinality, while complex means it is drawn from a space with high cardinality.

	actual negative	actual positive
decide negative	0	0
decide positive	$-c$	$b-c$

Let n be the size of the test set, The profit at q is the total benefit minus the total cost, which is

$$nqbt\sqrt{1/q} - nqc = nc(bt\sqrt{q}/c - q) = nc(k\sqrt{q} - q)$$

where $k = tb/c$. Profit is maximized when

$$0 = \frac{d}{dq}(kq^{0.5} - q) = k(0.5)q^{-0.5} - 1.$$

The solution is $q = (k/2)^2$.

This solution is in the range zero to one only when $k \leq 2$. When $tb/c > 2$ then maximum profit is achieved by soliciting every prospect, so data mining is pointless. The maximum profit that can be achieved is

$$nc(k(k/2) - (k/2)^2) = nck^2/4 = ncq.$$

Remarkably, this is always the same as the cost of running the campaign, which is nqc.

As k decreases, the attainable profit decreases fast, i.e. quadratically. If $k < 0.4$ then $q < 0.04$ and the attainable profit is less than $0.04nc$. Such a campaign may have high risk, because the lift attainable at small q is typically worse than suggested by the rule of thumb. Note however that a small adverse change in c, b, or t is not likely to make the campaign lose money, because the expected revenue is always twice the campaign cost.

It is interesting to consider whether data mining is beneficial or not from the point of view of society at large. Remember that c is the cost of one contact from the perspective of the initiator of the campaign. Each contact also has a cost or benefit for the recipient. Generally, if the recipient responds, one can assume that the contact was beneficial for him or her, because s/he only responds if the response is beneficial. However, if the recipient does not respond, which is the majority case, then one can assume that the contact caused a net cost to him or her, for example a loss of time.

From the point of view of the initiator of a campaign a cost or benefit for a respondent is an externality. It is not rational for the initiator to take into account these benefits or costs, unless they cause respondents to take actions that affect the initiator, such as closing accounts. However, it *is* rational for society to take into account these externalities.

The conclusion above is that the revenue from a campaign, for its initiator, is roughly twice its cost. Suppose that the benefit of responding for a respondent is λb

where b is the benefit to the initiator. Suppose also that the cost of a solicitation to a person is μc where c is the cost to the initiator. The net benefit to respondents is positive as long as $\mu < 2\lambda$.

The reasoning above clarifies why spam email campaigns are harmful to society. For these campaigns, the cost c to the initiator is tiny. However, the cost to a recipient is not tiny, so μ is large. Whatever λ is, it is likely that $\mu > 2\lambda$.

In summary, data mining is only beneficial in a narrow sweet spot, where

$$tb/2 \leq c \leq \alpha tb$$

where α is some constant greater than 1. The product tb is the average benefit of soliciting a random customer. If the cost c of solicitation is less than half of this, then it is rational to contact all potential respondents. If c is much greater than the average benefit, then the campaign is likely to have high risk for the initiator.

As an example of the reasoning above, consider the scenario of the 1998 KDD contest. Here $t = 0.05$ about, $c = \$0.68$, and the average benefit is $b = \$15$ approximately. We have $k = tb/c = 75/68 = 1.10$. The rule of thumb predicts that the optimal fraction of people to solicit is $q = 0.30$, while the achievable profit per person $cq = \$0.21$. In fact, the methods that perform best in this domain achieve profit of about $\$0.16$, while soliciting about 70% of all people.

7.7 Evaluating success

In order to evaluate whether a real-world campaign is proceeding as expected, we need to estimate a confidence interval for the profit on the test set. This should be a range $[ab]$ such that the probability is (say) 95% that the observed profit lies in this range.

Intuitively, there are two sources of uncertainty for each test person: whether s/he will donate, and if so, how much. We need to work out the logic of how to quantify these uncertainties, and then the logic of combining the uncertainties into an overall confidence interval. The central issue here is not any mathematical details such as Student distributions versus Gaussians. It is the logic of tracking where uncertainty arises, and how it is propagated.

Quiz

Suppose your lab is trying to crystallize a protein. You can try experimental conditions x that differ on temperature, salinity, etc. The label $y = 1$ means crystallization is successful, while $y = 0$ means failure. Assume (not realistically) that the results of different experiments are independent.

You have a classifier that predicts $p(y = 1|x)$, the probability of success of experiment x. The cost of doing one experiment is \$60. The value of successful crystallization is \$9000.

(a) Write down the benefit matrix involved in deciding rationally whether or not to perform a particular experiment.

The benefit matrix is

	success	*failure*
do experiment	*9000-60*	*-60*
don't	*0*	*0*

(b) What is the threshold probability of success needed in order to perform an experiment?

It is rational to do an experiment under conditions x if and only if the expected benefit is positive, that is if and only if

$$(9000 - 60) \cdot p + (-60) \cdot (1 - p) > 0.$$

where $p = p(\textit{success}|x)$. The threshold value of p is $60/9000 = 1/150$.

(c) Is it rational for your lab to take into account a budget constraint such as "we only have a technician available to do one experiment per day"?

No. A budget constraint is an alternative rule for making decisions that is less rational than maximizing expected benefit. The optimal behavior is to do all experiments that have success probability over 1/150.

Additional explanation: The \$60 cost of doing an experiment should include the expense of technician time. If many experiments are worth doing, then more technicians should be hired.

If the budget constraint is unavoidable, then experiments should be done starting with those that have highest probability of success.

If just one successful crystallization is enough, then experiments should also be done in order of declining success probability, until the first actual success.

2009 Assignment

This assignment is due at the start of class on Tuesday May 5. As before, you should work in a team of two, and you are free to change partners or not.

This assignment is the last one to use the KDD98 data. You should now train on the entire training set, and measure final success on the test set that you have not previously used. The goal is to solicit an optimal subset of the test examples. The measure of success to maximize is total donations received minus $0.68 for every solicitation.

You should train a regression function to predict donation amounts, and a classifier to predict donation probabilities. For a test example x, let the predicted donation be $a(x)$ and let the predicted donation probability be $p(x)$. You should decide to send a donation request to person x if and only if

$$p(x) \cdot a(x) \geq 0.68.$$

You should use the training set for all development work. In particular, you should use part of the training set for debugging your procedure for reading in test examples, making decisions concerning them, and tallying total profit. Only use the actual test set once, to measure the final success of your method.

Notes: The test instances are in the file `cup98val.zip` at `http://archive.ics.uci.edu/ml/databases/kddcup98/kddcup98.html`. The test set labels are in `valtargt.txt`. The labels are sorted by `CONTROLN`, unlike the test instances.

2010 Assignment

This week's assignment is to participate in the PAKDD 2010 data mining contest. Details of the contest are at `http://sede.neurotech.com.br/PAKDD2010/`. Each team of two students should register and download the files for the contest.

Your first goal should be to understand the data and submission formats, and to submit a correct set of predictions. Make sure that when you have good predictions later, you will not run into any technical difficulties.

Your second goal should be to understand the contest scenario and the differences between the training and two test datasets. Do some exploratory analysis of the three datasets. In your written report, explain your understanding of the scenario, and your general findings about the datasets.

Next, based on your general understanding, design a sensible approach for achieving the contest objective. Implement this approach and submit predictions to the contest. Of course, you may refine your approach iteratively and you may make multiple submissions. Meet the May 3 deadline for uploading your best predictions and a copy of your report. The contest rules ask each team to submit a paper of four pages.

> The manuscript should be in the scientific paper format of the conference detailing the stages of the KDD process, focusing on the aspects which make the solution innovative. The manuscript should be the basis for writing up a full paper for an eventual pos-conference proceeding (currently under negotiation). The authors of top solutions and selected manuscripts with innovative approaches will have the opportunity to submit to that forum. The quality of the manuscripts will not be used for the competition assessment, unless in the case of ties.

You can find template files for LaTeX and Word at `http://www.springer.com/computer/lncs?SGWID=0-164-6-793341-0`. Do not worry about formatting details.

Chapter 8

Learning classifiers despite missing labels

This chapter discusses how to learn two-class classifiers from nonstandard training data. Specifically, we consider three different but related scenarios where labels are missing for some but not all training examples.

8.1 Sample selection bias in general

Let x be a training instance and let y be its label. Suppose that y is observed only for some training instances. Let s be a new random variable with value $s = 1$ if and only if x is selected. This means that y is observed if and only if $s = 1$.

Formally, x, y, and s are random variables. There is some fixed unknown overall distribution $p(x, y, s)$ over triples $\langle x, y, s\rangle$. We can identify three possibilities for s. The easiest case is when $p(s = 1)$ is a constant. In this case, the labeled training examples are a fully random subset. We have fewer training examples, but otherwise we are in the usual situation. This case is called "missing completely at random" (MCAR).

A more difficult situation arises if s is correlated with x and/or with y, that is if $p(s = 1) \neq p(s = 1|x, y)$. Here, there are two subcases. First, suppose that s depends on x, but when x is fixed then s is independent of y. In this case, Bayes' rule gives that

$$p(y|x, s = 1) = \frac{p(s = 1|x, y)p(y|x)}{p(s = 1|x)} = \frac{p(s = 1|x)p(y|x)}{p(s = 1|x)} = p(y|x)$$

assuming that $p(s = 1|x) > 0$ for all x. Therefore we can learn a correct model of $p(y|x)$ from just the labeled training data, without using the unlabeled data in

any way. This case is rather misleadingly called "missing at random" (MAR). It is not the case that labels are missing completely at random, because missingness does depend on x. (Note that in this case $p(y|x, s = 0) = p(y|x)$ is true as well as $p(y|x, s = 1) = p(y|x)$.)

The real-world meaning of the assumption $p(s = 1|x) > 0$ is that label information must be available with non-zero probability for every possible instance x. Otherwise, it might be the case for some x that no labeled training examples are available from which to estimate $p(y|x, s = 1)$.

Suppose that given x, s is still correlated with y. In this case y is said to be "missing not at random" (MNAR) and inference is much more difficult. We do not discuss this case further here.

8.2 Covariate shift

Imagine that the available training data come from one hospital, but we want to learn a classifier to use at a different hospital. Let x be a patient and let y be a label such as "has swine flu." The distribution $p(x)$ of patients is different at the two hospitals, but we are willing to assume that the relationship to be learned, which is $p(y|x)$, is unchanged.

In a scenario like the one above, no unlabeled training examples are available. Labeled training examples from all classes are available, but the distribution of instances is different for the training and test sets. This scenario is called "covariate shift" where covariate means feature, and shift means change of distribution.

Following the argument in the previous section, the assumption that $p(y|x)$ is unchanged means that $p(y|x, s = 1) = p(y|x, s = 0)$ where $s = 1$ refers to the first hospital and $s = 0$ refers to the second hospital. The simple approach to covariate shift is thus train the classifier on the data $\langle x, y, s = 1\rangle$ in the usual way, and to apply it to patients from the second hospital directly.

8.3 Reject inference

Suppose that we want to build a model that predicts who will repay a loan. We need to be able to apply the model to the entire population of future applicants (sometimes called the "through the door" population). The usable training examples are people who were actually granted a loan in the past. Unfortunately, these people are not representative of the population of all future applicants. They were previously selected precisely because they were thought to be more likely to repay. The problem of "reject inference" is to learn somehow from previous applicants who were not approved, for whom we hence do not have training labels.

Another example is the task of learning to predict the benefit of a medical treatment. If doctors have historically given the treatment only to patients who were particularly likely to benefit, then the observed success rate of the treatment is likely to be higher than its success rate on "through the door" patients. Conversely, if doctors have historically used the treatment only on patients who were especially ill, then the "through the door" success rate may be higher than the observed success rate.

The "reject inference" scenario is similar to the covariate shift scenario, but with two differences. The first difference is that unlabeled training instances are available. The second difference is that the label being missing is expected to be correlated with the value of the label. The important similarity is the common assumption that missingness depends only on x, and not additionally on y. Whether y is missing may be correlated with the value of y, but this correlation disappears after conditioning on x.

A useful fact is the following. Suppose we want to estimate $E[f(z)]$ where z follows the distribution $p(z)$, but we can only draw samples of z from the distribution $q(z)$. The fact is

$$E[f(z)|z \sim p(z)] = E[f(z)\frac{p(z)}{q(z)}|z \sim q(z)],$$

assuming $q(z) > 0$ for all z. More generally the requirement is that $q(z) > 0$ whenever $p(z) > 0$, assuming the definition $0/0 = 0$. The equation above is called the importance-sampling identity.

Let the goal be to compute $E[f(x, y)|x, y \sim p(x, y)]$ for any function f. To make notation more concise, write this as $E[f]$ and write $E[f(x, y)|x, y \sim p(x, y, s = 1)]$ as $E[f|s = 1]$. We have

$$\begin{aligned} E[f] &= E[f\frac{p(x)p(y|x)}{p(x|s=1)p(y|x,s=1)}|s=1] \\ &= E[f\frac{p(x)}{p(x|s=1)}|s=1] \end{aligned}$$

since $p(y|x) = p(y|x, s = 1)$. Applying Bayes' rule to $p(x|s = 1)$ gives

$$\begin{aligned} E[f] &= E[f\frac{p(x)}{p(s=1|x)p(x)/p(s=1)}|s=1] \\ &= E[f\frac{p(s=1)}{p(s=1|x)}|s=1]. \end{aligned}$$

The constant $p(s = 1)$ can be estimated as r/n where r is the number of labeled training examples and n is the total number of training examples. Let $\hat{p}(s = 1|x)$ be

a trained model of the conditional probability $p(s = 1|x)$. The estimate of $E[f]$ is then

$$\frac{r}{n} \sum_{i=1}^{r} \frac{f(x_i, y_i)}{\hat{p}(s = 1|x_i)}.$$

This estimate is called a "plug-in" estimate because it is based on plugging the observed values of $\langle x, y \rangle$ into a formula that would be correct if based on integrating over all values of $\langle x, y \rangle$.

The weighting approach just explained is correct in the statistical sense that it is unbiased, if the propensity estimates $\hat{p}(s = 1|x)$ are correct. However, this approach typically has high variance since a few labeled examples with high values for $1/\hat{p}(s = 1|x)$ dominate the sum. Therefore an important question is whether alternative approaches exist that have lower variance. One simple heuristic is to place a ceiling on the values $1/\hat{p}(s = 1|x)$. For example the ceiling 1000 is used by [Huang et al., 2006]. However, no good method for selecting the ceiling value is known.

In medical research, $p(s = 1)/p(s = 1|x)$ is called the "inverse probability of treatment" (IPT) weight.

8.4 Positive and unlabeled examples

Suppose that only positive examples are labeled. This fact can be stated formally as the equation

$$p(s = 1|x, y = 0) = 0. \tag{8.1}$$

Without some assumption about which positive examples are labeled, it is impossible to make progress. A common assumption is that the labeled positive examples are chosen completely randomly from all positive examples. Let this be called the "selected completely at random" assumption. Stated formally, it is that

$$p(s = 1|x, y = 1) = p(s = 1|y = 1) = c. \tag{8.2}$$

Another way of stating the assumption is that s and x are conditionally independent given y.

A training set consists of two subsets, called the labeled ($s = 1$) and unlabeled ($s = 0$) sets. Suppose we provide these two sets as inputs to a standard training algorithm. This algorithm will yield a function $g(x)$ such that $g(x) = p(s = 1|x)$ approximately. The following lemma shows how to obtain a model of $p(y = 1|x)$ from $g(x)$.

Lemma 1. Suppose the "selected completely at random" assumption holds. Then $p(y = 1|x) = p(s = 1|x)/c$ where $c = p(s = 1|y = 1)$.

Proof. Remember that the assumption is $p(s = 1|y = 1, x) = p(s = 1|y = 1)$. Now consider $p(s = 1|x)$. We have that

$$\begin{aligned} p(s = 1|x) &= p(y = 1 \wedge s = 1|x) \\ &= p(y = 1|x)p(s = 1|y = 1, x) \\ &= p(y = 1|x)p(s = 1|y = 1). \end{aligned}$$

The result follows by dividing each side by $p(s = 1|y = 1)$. ■

Several consequences of the lemma are worth noting. First, f is an increasing function of g. This means that if the classifier f is only used to rank examples x according to the chance that they belong to class $y = 1$, then the classifier g can be used directly instead of f.

Second, $f = g/p(s = 1|y = 1)$ is a well-defined probability $f \leq 1$ only if $g \leq p(s = 1|y = 1)$. What this says is that $g > p(s = 1|y = 1)$ is impossible. This is reasonable because the labeled (positive) and unlabeled (negative) training sets for g are samples from overlapping regions in x space. Hence it is impossible for any example x to belong to the positive class for g with a high degree of certainty.

The value of the constant $c = p(s = 1|y = 1)$ can be estimated using a trained classifier g and a validation set of examples. Let V be such a validation set that is drawn from the overall distribution $p(x, y, s)$ in the same manner as the nontraditional training set. Let P be the subset of examples in V that are labeled (and hence positive). The estimator of $p(s = 1|y = 1)$ is the average value of $g(x)$ for x in P. Formally the estimator is $e_1 = \frac{1}{n} \sum_{x \in P} g(x)$ where n is the cardinality of P.

We shall show that $e_1 = p(s = 1|y = 1) = c$ if it is the case that $g(x) = p(s = 1|x)$ for all x. To do this, all we need to show is that $g(x) = c$ for $x \in P$. We can show this as follows:

$$\begin{aligned} g(x) &= p(s = 1|x) \\ &= p(s = 1|x, y = 1)p(y = 1|x) \\ &\quad + p(s = 1|x, y = 0)p(y = 0|x) \\ &= p(s = 1|x, y = 1) \cdot 1 + 0 \cdot 0 \text{ since } x \in P \\ &= p(s = 1|y = 1). \end{aligned}$$

Note that in principle any single example from P is sufficient to determine c, but that in practice averaging over all members of P is preferable.

There is an alternative way of using Lemma 1. Let the goal be to estimate $E_{p(x,y,s)}[h(x, y)]$ for any function h, where $p(x, y, s)$ is the overall distribution. To

make notation more concise, write this as $E[h]$. We want an estimator of $E[h]$ based on a positive-only training set of examples of the form $\langle x, s\rangle$.

Clearly $p(y = 1|x, s = 1) = 1$. Less obviously,

$$\begin{aligned} p(y = 1|x, s = 0) &= \frac{p(s = 0|x, y = 1)p(y = 1|x)}{p(s = 0|x)} \\ &= \frac{[1 - p(s = 1|x, y = 1)]p(y = 1|x)}{1 - p(s = 1|x)} \\ &= \frac{(1 - c)p(y = 1|x)}{1 - p(s = 1|x)} \\ &= \frac{(1 - c)p(s = 1|x)/c}{1 - p(s = 1|x)} \\ &= \frac{1 - c}{c}\frac{p(s = 1|x)}{1 - p(s = 1|x)}. \end{aligned}$$

By definition

$$\begin{aligned} E[h] &= \int_{x,y,s} h(x, y)p(x, y, s) \\ &= \int_x p(x) \sum_{s=0}^{1} p(s|x) \sum_{y=0}^{1} p(y|x, s)h(x, y) \\ &= \int_x p(x) \Big(p(s = 1|x)h(x, 1) \\ &\qquad + p(s = 0|x)[p(y = 1|x, s = 0)h(x, 1) \\ &\qquad\qquad + p(y = 0|x, s = 0)h(x, 0)] \Big). \end{aligned}$$

The plug-in estimate of $E[h]$ is then the empirical average

$$\frac{1}{m}\Big(\sum_{\langle x,s=1\rangle} h(x, 1) + \sum_{\langle x,s=0\rangle} w(x)h(x, 1) + (1 - w(x))h(x, 0) \Big)$$

where

$$w(x) = p(y = 1|x, s = 0) = \frac{1 - c}{c}\frac{p(s = 1|x)}{1 - p(s = 1|x)} \tag{8.3}$$

and m is the cardinality of the training set. What this says is that each labeled example is treated as a positive example with unit weight, while each unlabeled example is treated as a combination of a positive example with weight $p(y = 1|x, s = 0)$ and a negative example with complementary weight $1 - p(y = 1|x, s = 0)$. The

probability $p(s = 1|x)$ is estimated as $g(x)$ where g is the nontraditional classifier explained in the previous section.

The result above on estimating $E[h]$ can be used to modify a learning algorithm in order to make it work with positive and unlabeled training data. One method is to give training examples individual weights. Positive examples are given unit weight and unlabeled examples are duplicated; one copy of each unlabeled example is made positive with weight $p(y = 1|x, s = 0)$ and the other copy is made negative with weight $1 - p(y = 1|x, s = 0)$.

8.5 General semi-supervised learning

Spam filtering scenario.

Observational studies.

Concept drift.

Quiz

(a) Suppose you use the weighting approach to deal with reject inference. What are the minimum and maximum possible values of the weights?

Let x be a labeled example, and let its weight be $p(s = 1)/p(s = 1|x)$. Intuitively, this weight is how many copies are needed to allow the one labeled example to represent all the unlabeled examples that are similar. The conditional probability $p(s = 1|x)$ can range between 0 and 1, so the weight can range between $p(s = 1)$ and infinity.

(b) Suppose you use the weighting approach to learn from positive and unlabeled examples. What are the minimum and maximum possible values of the weights?

In this scenario, weights are assigned to unlabeled examples, not to labeled examples as above. The weights here are probabilities $p(y = 1|x, s = 0)$, so they range between 0 and 1.

(c) Explain intuitively what can go wrong if the "selected completely at random" assumption is false, when learning from positive and unlabeled examples.

The "selected completely at random" assumption says that the positive examples with known labels are perfectly representative of the positive examples with unknown labels. If this assumption is false, then there will be unlabeled examples that in fact are positive, but that we treat as negative, because they are different from the labeled positive examples. The trained model of the positive class will be too narrow.

Assignment (revised)

The goal of this assignment is to train useful classifiers using training sets with missing information of three different types: (i) covariate shift, (ii) reject inference, and (iii) no labeled negative training examples. In `http://www.cs.ucsd.edu/users/elkan/291/dmfiles.zip` you can find four datasets: one test set, and one training set for each of the three scenarios. Training set (i) has 5,164 examples, sets (ii) and (iii) have 11,305 examples, and the test set has 11,307 examples. Each example has values for 13 predictors. (Many thanks to Aditya Menon for creating these files.)

You should train a classifier separately based on each training set, and measure performance separately but on the same test set. Use accuracy as the primary measure of success, and use the logistic regression option of `FastLargeMargin` as the main training method. In each case, you should be able to achieve better than 82% accuracy.

All four datasets are derived from the so-called Adult dataset that is available at `http://archive.ics.uci.edu/ml/datasets/Adult`. Each example describes one person. The label to be predicted is whether or not the person earns over $50,000 per year. This is an interesting label to predict because it is analogous to a label describing whether or not the person is a customer that is desirable in some way. (We are not using the published weightings, so the `fnlwgt` feature has been omitted from our datasets.)

First, do cross-validation on the test set to establish the accuracy that is achievable when all training set labels are known. In your report, show graphically a learning curve, that is accuracy as a function of the number of training examples, for 1000, 2000, etc. training examples.

Training set (i) requires you to overcome covariate shift, since it does not follow the same distribution as the population of test examples. Evaluate experimentally the effectiveness of learning $p(y|x)$ from biased training sets of size 1000, 2000, etc.

Training set (ii) requires reject inference, because the training examples are a random sample from the test population, but the training label is known only for some training examples. The persons with known labels, on average, are better prospects than the ones with unknown labels. Compare learning $p(y|x)$ directly with learning $p(y|x)$ after reweighting.

In training set (iii), a random subset of positive training examples have known labels. Other training examples may be negative or positive. Use an appropriate weighting method. Explain how you estimate the constant $c = p(s = 1|y = 1)$ and discuss the accuracy of this estimate. The true value is $c = 0.4$; in a real application we would not know this, of course.

For each scenario and each training method, include in your report a learning curve figure ’ that shows accuracy as a function of the number (1000, 2000, etc.) of labeled training examples used. Discuss the extent to which each of the three missing-label scenarios reduces achievable accuracy.

Chapter 9

Recommender systems

The collaborative filtering (CF) task is to recommend items to a user that he or she is likely to like, based on ratings for different items provided by the same user and on ratings provided by other users. The general assumption is that users who give similar ratings to some items are likely also to give similar ratings to other items.

From a formal perspective, the input to a collaborative filtering algorithm is a matrix of incomplete ratings. Each row corresponds to a user, while each column corresponds to an item. If user $1 \leq i \leq m$ has rated item $1 \leq j \leq n$, the matrix entry x_{ij} is the value of this rating. Often rating values are integers between 1 and 5. If a user has not rated an item, the corresponding matrix entry is missing. Missing ratings are often represented as 0, but this should not be viewed as an actual rating value.

The output of a collaborative filtering algorithm is a prediction of the value of each missing matrix entry. Typically the predictions are not required to be integers. Given these predictions, many different real-world tasks can be performed. For example, a recommender system might suggest to a user those items for which the predicted ratings by this user are highest.

There are two main general approaches to the formal collaborative filtering task: nearest-neighbor-based and model-based. Given a user and item for which a prediction is wanted, nearest neighbor (NN) approaches use a similarity function between rows, and/or a similarity function between columns, to pick a subset of relevant other users or items. The prediction is then some sort of average of the known ratings in this subset.

Model-based approaches to collaborative filtering construct a low-complexity representation of the complete x_{ij} matrix. This representation is then used instead of the original matrix to make predictions. Typically each prediction can be computed in $O(1)$ time using only a fixed number of coefficients from the representation.

The most popular model-based collaborative filtering algorithms are based on standard matrix approximation methods such as the singular value decomposition (SVD), principal component analysis (PCA), or nonnegative matrix factorization (NNMF). Of course these methods are themselves related to each other.

From the matrix decomposition perspective, the fundamental issue in collaborative filtering is that the matrix given for training is incomplete. Many algorithms have been proposed to extend SVD, PCA, NNMF, and related methods to apply to incomplete matrices, often based on expectation-maximization (EM). Unfortunately, methods based on EM are intractably slow for large collaborative filtering tasks, because they require many iterations, each of which computes a matrix decomposition on a full m by n matrix. Here, we discuss an efficient approach to decomposing large incomplete matrices.

9.1 Applications of matrix approximation

In addition to collaborative filtering, many other applications of matrix approximation are important also. Examples include the voting records of politicians, the results of sports matches, and distances in computer networks. The largest research area with a goal similar to collaborative filtering is so-called "item response theory" (IRT), a subfield of psychometrics. The aim of IRT is to develop models of how a person answers a multiple-choice question on a test. Users and items in collaborative filtering correspond to test-takers and test questions in IRT. Missing ratings are called omitted responses in IRT.

One conceptual difference between collaborative filtering research and IRT research is that the aim in IRT is often to find a single parameter for each individual and a single parameter for each question that together predict answers as well as possible. The idea is that each individual's parameter value is then a good measure of intrinsic ability and each question's parameter value is a good measure of difficulty. In contrast, in collaborative filtering research the goal is often to find multiple parameters describing each person and each item, because the assumption is that each item has multiple relevant aspects and each person has separate preferences for each of these aspects.

9.2 Measures of performance

Given a set of predicted ratings and a matching set of true ratings, the difference between the two sets can be measured in alternative ways. The two standard measures are mean absolute error (MAE) and mean squared error (MSE). Given a probability distribution over possible values, MAE is minimized by taking the median of the dis-

tribution while MSE is minimized by taking the mean of the distribution. In general the mean and the median are different, so in general predictions that minimize MAE are different from predictions that minimize MSE.

Most matrix approximation algorithms aim to minimize MSE between the training data (a complete or incomplete matrix) and the approximation obtained from the learned low-complexity representation. Some methods can be used with equal ease to minimize MSE or MAE.

9.3 Additive models

Let us consider first models where each matrix entry is represented as the sum of two contributions, one from its row and one from its column. Formally, $a_{ij} = r_i + c_j$ where a_{ij} is the approximation of x_{ij} and r_i and c_j are scalars to be learned.

Define the training mean $\bar{x}_{i\cdot}$ of each row to be the mean of all its known values; define the the training mean of each column $\bar{x}_{\cdot j}$ similarly. The user-mean model sets $r_i = \bar{x}_{i\cdot}$ and $c_j = 0$ while the item-mean model sets $r_i = 0$ and $c_j = \bar{x}_{\cdot j}$. A slightly more sophisticated baseline is the "bimean" model: $r_i = 0.5\bar{x}_{i\cdot}$ and $c_j = 0.5\bar{x}_{\cdot j}$.

The "bias from mean" model has $r_i = \bar{x}_{i\cdot}$ and $c_j = (x_{ij} - r_i)_{\cdot j}$. Intuitively, c_j is the average amount by which users who provide a rating for item j like this item more or less than the average item that they have rated.

The optimal additive model can be computed quite straightforwardly. Let I be the set of matrix indices for which x_{ij} is known. The MSE optimal additive model is the one that minimizes

$$\sum_{\langle i,j\rangle \in I} (r_i + c_j - x_{ij})^2.$$

This optimization problem is a special case of a sparse least-squares linear regression problem: find z that minimizes $\|Az - b\|_2$ where the column vector $z = \langle r_1, \ldots, r_m, c_1, \ldots, c_n\rangle'$, b is a column vector of x_{ij} values and the corresponding row of A is all zero except for ones in positions i and $m + j$. z can be computed by many methods. The standard method uses the Moore-Penrose pseudoinverse of A: $z = (A'A)^{-1}A'b$. However this approach requires inverting an $m + n$ by $m + n$ matrix, which is computationally expensive. Section 9.4 below gives a gradient-descent method to obtain the optimal z which only requires $O(|I|)$ time and space.

The matrix A is always rank-deficient, with rank at most $m + n - 1$, because any constant can be added to all r_i and subtracted from all c_j without changing the matrix reconstruction. If some users or items have very few ratings, the rank of A may be less than $m + n - 1$. However the rank deficiency of A does not cause computational problems in practice.

9.4 Multiplicative models

Multiplicative models are similar to additive models, but the row and column values are multiplied instead of added. Specifically, x_{ij} is approximated as $a_{ij} = r_i c_j$ with r_i and c_j being scalars to be learned. Like additive models, multiplicative models have one unnecessary degree of freedom. We can fix any single value r_i or c_j to be a constant without changing the space of achievable approximations.

A multiplicative model is a rank-one matrix decomposition, since the rows and columns of the y matrix are linearly proportional. We now present a general algorithm for learning row value/column value models; additive and multiplicative models are special cases. Let x_{ij} be a matrix entry and let r_i and c_j be corresponding row and column values. We approximate x_{ij} by $a_{ij} = f(r_i, c_j)$ for some fixed function f. The approximation error is defined to be $e(f(r_i, c_j), x_{ij})$. The training error E over all known matrix entries I is the pointwise sum of errors.

To learn r_i and c_j by minimizing E by gradient descent, we evaluate

$$\begin{aligned}\frac{\partial}{\partial r_i}E &= \sum_{\langle i,j\rangle \in I} \frac{\partial}{\partial r_i} e(f(r_i, c_j), x_{ij}) \\ &= \sum_{\langle i,j\rangle \in I} \frac{\partial}{\partial f(r_i, c_j)} e(f(r_i, c_j), x_{ij}) \frac{\partial}{\partial r_i} f(r_i, c_j).\end{aligned}$$

As before, I is the set of matrix indices for which x_{ij} is known. Consider the special case where $e(u, v) = |u - v|^p$ for $p > 0$. This case is a generalization of MAE and MSE: $p = 2$ corresponds to MSE and $p = 1$ corresponds to MAE. We have

$$\begin{aligned}\frac{\partial}{\partial u} e(u, v) &= p|u - v|^{p-1} \frac{\partial}{\partial u}|u - v| \\ &= p|u - v|^{p-1} \operatorname{sgn}(u - v).\end{aligned}$$

Here $\operatorname{sgn}(a)$ is the sign of a, that is -1, 0, or 1 if a is negative, zero, or positive. For computational purposes we can ignore the non-differentiable case $u = v$. Therefore

$$\frac{\partial}{\partial r_i}E = \sum_{\langle i,j\rangle \in I} p|f(r_i, c_j) - x_{ij}|^{p-1} \operatorname{sgn}(f(r_i, c_j) - x_{ij}) \frac{\partial}{\partial r_i} f(r_i, c_j).$$

Now suppose $f(r_i, c_j) = r_i + c_j$ so $\frac{\partial}{\partial r_i} f(r_i, c_j) = 1$. We obtain

$$\frac{\partial}{\partial r_i}E = p \sum_{\langle i,j\rangle \in I} |r_i + c_j - x_{ij}|^{p-1} \operatorname{sgn}(r_i + c_j - x_{ij}).$$

Alternatively, suppose $f(r_i, c_j) = r_i c_j$ so $\frac{\partial}{\partial r_i} f(r_i, c_j) = c_j$. We get

$$\frac{\partial}{\partial r_i} E = p \sum_{\langle i,j \rangle \in I} |r_i c_j - x_{ij}|^{p-1} \operatorname{sgn}(r_i c_j - x_{ij}) c_j.$$

Given the gradients above, we apply online (stochastic) gradient descent. This means that we iterate over each triple $\langle r_i, c_j, x_{ij} \rangle$ in the training set, compute the gradient with respect to r_i and c_j based just on this one example, and perform the updates

$$r_i := r_i - \lambda \frac{\partial}{\partial r_i} e(f(r_i, c_j), x_{ij})$$

and

$$c_j := c_j - \lambda \frac{\partial}{\partial c_j} e(f(r_i, c_j), x_{ij})$$

where λ is a learning rate. Note that λ determines the step sizes for stochastic gradient descent; no separate algorithm parameter is needed for this.

After any finite number of epochs, stochastic gradient descent does not converge fully. The choice of 30 epochs is a type of early stopping that leads to good results by not overfitting the training data. For the learning rate we use a decreasing schedule: $\lambda = 0.2/e$ for additive models and $\lambda = 0.4/e$ for multiplicative models, where $1 \leq e \leq 30$ is the number of the current epoch.

Gradient descent as described above directly optimizes precisely the same objective function (given the MSE error function) that is called "incomplete data likelihood" in EM approaches to factorizing matrices with missing entries. It is sometimes forgotten that EM is just one approach to solving maximum-likelihood problems; incomplete matrix factorization is an example of a maximum-likelihood problem where an alternative solution method is superior.

9.5 Combining models by fitting residuals

Simple models can be combined into more complex models. In general, a second model is trained to fit the difference between the predictions made by the first model and the truth as specified by the training data; this difference is called the residual.

The most straightforward way to combine models is additive. Let a_{ij} be the prediction from the first model and let x_{ij} be the corresponding training value. The second model is trained to minimize error relative to the residual $x_{ij} - a_{ij}$. Let b_{ij} be the prediction made by the second model for matrix entry ij. The combined prediction is then $a_{ij} + b_{ij}$. If the first and second models are both multiplicative, then the combined model is a rank-two approximation. The extension to higher-rank approximations is obvious.

Standard principal component analysis (PCA) is a variety of mixed model. Before the first rank-one approximation is computed, the mean of each column is subtracted from that column of the original matrix. Thus the actual first approximation of the original matrix is $a_{ij} = (0 + m_j) + r_i c_j = (1 \cdot m_j) + r_i c_j$ where m_j is an initial column value and $r_i c_j$ is a multiplicative model.

The common interpretation of a combined model is that each component model represents an aspect or property of items and users. The idea is that the column value for each item is the intensity with which it possesses this property, and the row value for each user is the weighting the user places on this property. However this point of view implicitly treats each component model as equally important. It is more accurate to view each component model as an adjustment to previous models, where each model is empirically less important than each previous one, because the numerical magnitude of its contribution is smaller. The aspect of items represented by each model cannot be understood in isolation, but only in the context of previous aspects.

9.6 Further issues

One issues not discussed above is regularization: attempting to improve generalization by reducing the effective number of parameters in the trained low-complexity representation.

Another issue not discussed is any potential probabilistic interpretation of a matrix decomposition. One reason for not considering models that are explicitly probabilistic is that these are usually based on Gaussian assumptions that are incorrect by definition when the observed data are integers and/or in a fixed interval.

A third issue not discussed is any explicit model for how matrix entries come to be missing. There is no assumption or analysis concerning whether entries are missing "completely at random" (MCAR), "at random" (MAR), or "not at random" (MNAR). Intuitively, in the collaborative filtering context, missing ratings are likely to be MNAR. This means that even after accounting for other influences, whether or not a rating is missing still depends on its value, because people select which items to rate based on how much they anticipate liking them. Concretely, everything else being equal, low ratings are still more likely to be missing than high ratings. This intuition can be confirmed empirically. On the Movielens dataset, the average rating in the training set is 3.58. The average prediction from one reasonable method for ratings in the test set, i.e. ratings that are unknown but known to exist, is 3.79. The average prediction for ratings that in fact do not exist is 3.34.

Amazon: people who looked at x bought y eventually (within 24hrs usually)

Quiz

For each part below, say whether the statement in italics is true or false, and then explain your answer briefly.

(a) *For any model, the average predicted rating for unrated movies is expected to be less than the average actual rating for rated movies.*

True. People are more likely to like movies that they have actually watched, than random movies that they have not watched. Any good model should capture this fact.

In more technical language, the value of a rating is correlated with whether or not it is missing.

(b) Let the predicted value of rating x_{ij} be $r_i c_j + s_i d_j$ and suppose r_i and c_j are trained first. *For all viewers i, r_i should be positive, but for some i, s_i should be negative.*

True. Ratings x_{ij} are positive, and $x_{ij} = r_i c_j$ on average, so r_i and c_j should always be positive. (They could always both be negative, but that would be unintuitive without being more expressive.)

The term $s_i d_j$ models the difference $x_{ij} - r_i c_j$. This difference is on average zero, so it is sometimes positive and sometimes negative. In order to allow $s_i d_j$ to be negative, s_i must be negative sometimes. (Making s_i be always positive, while allowing d_j to be negative, might be possible. However in this case the expressiveness of the model $s_i d_j$ would be reduced.)

(c) We have a training set of 500,000 ratings for 10,000 viewers and 1000 movies, and we train a rank-50 unregularized factor model. *This model is likely to overfit the training data.*

True. The unregularized model has $50 \cdot (10,000 + 1000) = 550,000$ parameters, which is more than the number of data points for training. Hence, overfitting is practically certain.

Assignment

The goal of this assignment is to implement a matrix factorization method for predicting movie ratings. You should use the small MovieLens dataset available at `http://www.grouplens.org/node/73`. This has 100,000 ratings given by 943 users to 1682 movies.

You can implement any low-rank decomposition method that you like, and you can reuse existing software such as functions from Matlab or Rapidminer. However, you must understand fully the algorithms that you use, and you should explain them with mathematical clarity in your report. Whatever method you implement should handle missing values in a sensible way, so you cannot use standard SVD or PCA directly.

When reporting final results, you should do five-fold cross-validation. Note that this experimental procedure makes the task somewhat easier, because evaluation is biased towards users who have provided more ratings; it is easier to make accurate predictions for these users.

In your report, show mean absolute error graphically as a function of rank, for matrix decompositions of rank up to whatever rank provides maximum accuracy. Also provide timing information.

For extra credit, you may do any of the following:

1. Run successfully on the very large Movielens dataset recently published at `http://www.grouplens.org/node/269`.

2. Apply your method also to a fraud detection dataset, such as the ones at `http://research.cs.queensu.ca/~skill/2003873final.html`.

3. Extend your method to benefit from ancillary information such as rating dates and movie tags.

4. Design and implement a useful regularization technique to reduce overfitting.

Chapter 10

Text mining

This chapter explains how to do data mining on datasets that are collections of documents. Text mining tasks include

- classifier learning
- clustering,
- topic modeling, and
- latent semantic analysis.

Classifiers for documents are useful for many applications. Major uses for binary classifiers include spam detection and personalization of streams of news articles. Multiclass classifiers are useful for routing messages to recipients.

Most classifiers for documents are designed to categorize according to subject matter. However, it is also possible to learn to categorize according to qualitative criteria such as helpfulness for product reviews submitted by consumers.

In many applications of multiclass classification, a single document can belong to more than one category, so it is correct to predict more than one label. This task is specifically called multilabel classification. In standard multiclass classification, the classes are mutually exclusive, i.e. a special type of negative correlation is fixed in advance. In multilabel classification, it is important to learn the positive and negative correlations between classes.

10.1 The bag-of-words representation

The first question we must answer is how to represent documents. For genuine understanding of natural language one must obviously preserve the order of the words

in documents. However, for many large-scale data mining tasks, including classifying and clustering documents, it is sufficient to use a simple representation that loses all information about word order.

Given a collection of documents, the first task to perform is to identify the set of all words used a least once in at least one document. This set is called the vocabulary V. Often, it is reduced in size by keeping only words that are used in at least two documents. (Words that are found only once are often mis-spellings or other mistakes.) Although the vocabulary is a set, we fix an arbitrary ordering for it so we can refer to word 1 through word m where $m = |V|$ is the size of the vocabulary.

Once V has been fixed, each document is represented as a vector with integer entries of length m. If this vector is x then its jth component x_j is the number of appearances of word j in the document. The length of the document is $n = \sum_{j=1}^{m} x_j$. For typical documents, n is much smaller than m and $x_j = 0$ for most words j.

Many applications of text mining also eliminate from the vocabulary so-called "stop" words. These are words that are common in most documents and do not correspond to any particular subject matter. In linguistics these words are sometimes called "function" words. They include pronouns (you, he, it) connectives (and, because, however), prepositions (to, of, before), auxiliaries (have, been, can), and generic nouns (amount, part, nothing). It is important to appreciate, however, that generic words carry a lot of information for many tasks, including identifying the author of a document or detecting its genre.

A collection of documents is represented as a two-dimensional matrix where each row describes a document and each column corresponds to a word. Each entry in this matrix is an integer count; most entries are zero. It makes sense to view each column as a feature. It also makes sense to learn a low-rank approximation of the whole matrix; doing this is called latent semantic analysis (LSA) and is discussed in Section 10.8 below.

10.2 The multinomial distribution

Once we have a representation for individual documents, the natural next step is to select a model for a set of documents. This model is a probability distribution. Given a training set of documents, we will choose values for the parameters of the distribution that make the training documents have high probability.

Then, given a test document, we can evaluate its probability according to the model. The higher this probability is, the more similar the test document is to the training set.

The probability distribution that we use is the multinomial. Mathematically, this

distribution is

$$p(x;\theta) = \frac{n!}{\prod_{j=1}^{m} x_j!} \prod_{j=1}^{m} \theta_j^{x_j}.$$

where the data x are a vector of non-negative integers and the parameters θ are a real-valued vector. Both vectors have the same length m. The components of θ are non-negative and have unit sum: $\sum_{j=1}^{m} \theta_j = 1$.

Intuitively, θ_j is the probability of word j while x_j is the count of word j. Each time word j appears in the document it contributes an amount θ_j to the total probability, hence the term $\theta_j^{x_j}$.

Like any discrete distribution, a multinomial has to sum to one, where the sum is over all possible data points. Here, a data point is a document containing n words. The number of such documents is exponential in their length n: it is m^n. The probability of any individual document will therefore be very small. What is important is the relative probability of different documents. A document that mostly uses words with high probability will have higher relative probability.

At first sight, computing the probability of a document requires $O(m)$ time because of the product over j. However, if $x_j = 0$ then $\theta_j^{x_j} = 1$ so the jth factor can be omitted from the product. Similarly, $0! = 1$ so the jth factor can be omitted from $\prod_{j=1}^{m} x_j!$. Hence, computing the probability of a document needs only $O(n)$ time.

Because the probabilities of individual documents decline exponentially with length n, it is necessary to do numerical computations with log probabilities:

$$\log p(x;\theta) = \log n! - [\sum_{j=1}^{m} \log x_j!] + [\sum_{j=1}^{m} x_j \cdot \log \theta_j]$$

Given a set of training documents, the maximum-likelihood estimate of the jth parameter is

$$\theta_j = \frac{1}{T} \sum_{x} x_j$$

where the sum is over all documents x belonging to the training set. The normalizing constant is $T = \sum_x \sum_j x_j$ which is the sum of the sizes of all training documents.

If a multinomial has $\theta_j = 0$ for some j, then every document with $x_j > 0$ for this j has zero probability, regardless of any other words in the document. Probabilities that are perfectly zero are undesirable, so we want $\theta_j > 0$ for all j. Smoothing with a constant c is the way to achieve this. We set

$$\theta_j = \frac{1}{T'}(c + \sum_{x} x_j).$$

The constant c is called a pseudocount. Intuitively, it is a notional number of appearances of word j that are assumed to exist, regardless of the true number of appearances. Typically c is chosen in the range $0 < c \leq 1$. Because the equality $\sum_j \theta_j = 1$ must be preserved, the normalizing constant must be $T' = mc + T$. In order to avoid big changes in the estimated probabilities θ_j, one should have $c < T/m$.

Technically, one multinomial is a distribution over all documents of a fixed size n. Therefore, what is learned by the maximum-likelihood process just described is in fact a different distribution for each size n. These distributions, although separate, have the same parameter values.

Generative process. Sampling with replacement.

10.3 Training Bayesian classifiers

Bayesian learning is an approach to learning classifiers based on Bayes' rule. Let x be an example and let y be its class. Suppose the alternative class values are 1 to K. We can write

$$p(y = k|x) = \frac{p(x|y = k)p(y = k)}{p(x)}.$$

In order to use the expression on the right for classification, we need to learn three items from training data: $p(x|y = k)$, $p(y = k)$, and $p(x)$. We can estimate $p(y = k)$ easily as $n_k / \sum_{k=1}^{K} n_k$ where n_k is the number of training examples with class label k. The denominator $p(x)$ can be computed as the sum of numerators

$$p(x) = \sum_{k=1}^{K} p(x|y = k)p(y = k).$$

In general, the class-conditional distribution $p(x|y = k)$ can be any distribution whose parameters are estimated using the training examples that have class label k.

Note that the training process for each class uses only the examples from that class, and is separate from the training process for all other classes. Usually, each class is represented by one multinomial fitted by maximum-likelihood as described in Section 10.2. In the formula

$$\theta_j = \frac{1}{T'}(c + \sum_x x_j)$$

the sum is taken over the documents in one class. Unfortunately, the exact value of c can strongly influence classification accuracy.

When there are more than two classes, or when there are two classes and neither is rare, then it is reasonable to use accuracy to measure the success of a classifier for

documents. If there are just two classes, and one class is rare, then it is common to use the so-called F-measure instead of accuracy. This measure is the harmonic mean of precision and recall:

$$f = \frac{1}{1/p + 1/r}$$

where p and r are precision and recall for the rare class.

10.4 Burstiness

The multinomial model says that each appearance of the same word j always has the same probability θ_j. In reality, additional appearances of the same word are less surprising, i.e. they have higher probability. Consider the following excerpt from a newspaper article, for example.

> Toyota Motor Corp. is expected to announce a major overhaul. Yoshi Inaba, a former senior Toyota executive, was formally asked by Toyota this week to oversee the U.S. business. Mr. Inaba is currently head of an international airport close to Toyota's headquarters in Japan.
>
> Toyota's U.S. operations now are suffering from plunging sales. Mr. Inaba was credited with laying the groundwork for Toyota's fast growth in the U.S. before he left the company.
>
> Recently, Toyota has had to idle U.S. assembly lines, pay workers who aren't producing vehicles and offer a limited number of voluntary buyouts. Toyota now employs 36,000 in the U.S.

The multinomial distribution arises from a process of sampling words with replacement. An alternative distribution named the the Dirichlet compound multinomial (DCM) arises from an urn process that captures the authorship process better. Consider a bucket with balls of $|V|$ different colors. After a ball is selected randomly, it is not just replaced, but also one more ball of the same color is added. Each time a ball is drawn, the chance of drawing the same color again is increased. This increased probability models the phenomenon of burstiness.

Let the initial number of balls with color j be β_j. These initial values are the parameters of the DCM distribution. The DCM parameter vector β has length $|V|$, like the multinomial parameter vector, but the sum of the components of β is unconstrained. This one extra degree of freedom allows the DCM to discount multiple observations of the same word, in an adjustable way. The smaller the parameter vales β_j are, the more words are bursty.

10.5 Discriminative classification

There is a consensus that linear support vector machines (SVMs) are the best known method for learning classifiers for documents. Nonlinear SVMs are not beneficial, because the number of features is typically much larger than the number of training examples. For the same reason, choosing the strength of regularization appropriately, typically by cross-validation, is crucial.

When using a linear SVM for text classification, accuracy can be improved considerably by transforming the raw counts. Since counts do not follow Gaussian distributions, it is not sensible to make them have mean zero and variance one. Inspired by the discussion above of burstiness, it is sensible to replace each count x by $\log(1+x)$. This transformation maps 0 to 0, so it preserves sparsity. A more extreme transformation that loses information is to make each count binary, that is to replace all non-zero values by one.

One can also transform counts in a supervised way, that is in a way that uses label information. This is most straightforward when there are just two classes. Experimentally, the following transformation gives the highest accuracy:

$$x \mapsto \text{sgn}(x) \cdot |\log tp/fn - \log fp/tn|.$$

Above, tp is the number of positive training examples containing the word, and fn is the number of these examples not containing the word, while fp is the number of negative training examples containing the word, and tn is the number of these examples not containing the word. If any of these numbers is zero, we replace it by 0.5, which of course is less than any of these numbers that is genuinely non-zero.

Notice that in the formula above the positive and negative classes are treated in a perfectly symmetric way. The value $|\log tp/fn - \log fp/tn|$ is large if tp/fn and fp/tn have very different values. In this case the word is highly diagnostic for at least one of the two classes.

The transformation above is sometimes called logodds weighting. Intuitively, first each count x is transformed into a binary feature $\text{sgn}(x)$, and then these features are weighted according to their predictiveness.

10.6 Clustering documents

Suppose that we have a collection of documents, and we want to find an organization for these, i.e. we want to do unsupervised learning. The simplest variety of unsupervised learning is clustering.

Clustering can be done probabilistically by fitting a mixture distribution to the given collection of documents. Formally, a mixture distribution is a probability den-

sity function of the form

$$p(x) = \sum_{k=1}^{K} \alpha_k p(x; \theta_k).$$

Here, K is the number of components in the mixture model. For each k, $p(x; \theta_k)$ is the distribution of component number k. The scalar α_k is the proportion of component number k.

Each component is a cluster.

10.7 Topic models

Mixture models, and clusterings in general, are based on the assumption that each data point is generated by a single component model. For documents, if components are topics, it is often more plausible to assume that each document can contain words from multiple topics. Topic models are probabilistic models that make this assumption.

Latent Dirichlet allocation (LDA) is the most widely used topic model. It is based on the intuition that each document contains words from multiple topics; the proportion of each topic in each document is different, but the topics themselves are the same for all documents.

The generative process assumed by the LDA model is as follows:

Given: Dirichlet distribution with parameter vector α of length K
Given: Dirichlet distribution with parameter vector β of length V
for topic number 1 to topic number K
    draw a multinomial with parameter vector ϕ_k according to β
for document number 1 to document number M
    draw a topic distribution, i.e. a multinomial θ according to α
    for each word in the document
        draw a topic z according to θ
        draw a word w according to ϕ_z

Note that z is an integer between 1 and K for each word.

The prior distributions α and β are assumed to be fixed and known, as are the number K of topics, the number M of documents, the length N_m of each document, and the cardinality V of the vocabulary (i.e. the dictionary of all words).

For learning, the training data are the words in all documents. Learning has two goals: (i) to infer the document-specific multinomial θ for each document, and (ii) to infer the topic distribution ϕ_k for each topic. After training, φ is a vector of word probabilities for each topic indicating the content of that topic. The distribution

θ of each document is useful for classifying new documents, measuring similarity between documents, and more.

When applying LDA, it is not necessary to learn α and β. Steyvers and Griffiths recommend fixed uniform values $\alpha = 50/K$ and $\beta = .01$, where K is the number of topics.

10.8 Latent semantic analysis

10.9 Open questions

It would be interesting to work out an extension of the logodds mapping for the multiclass case. One suggestion is

$$x \mapsto \text{sgn}(x) \cdot \max_i |\log tp_i/fn_i|$$

where i ranges over the classes, tp_i is the number of training examples in class i containing the word, and fn_i is the number of these examples not containing the word.

It is also not known if combining the log transformation and logodds weighting is beneficial, as in

$$x \mapsto \log(x+1) \cdot |\log tp/fn - \log fp/tn|.$$

Quiz

(a) Explain why, with a multinomial distribution, "the probabilities of individual documents decline exponentially with length n."

The probability of document x of length n according to a multinomial distribution is

$$p(x;\theta) = \frac{n!}{\prod_{j=1}^{m} x_j!} \prod_{j=1}^{m} \theta_j^{x_j}.$$

[Rough argument.] Each θ_j value is less than 1. In total $n = \sum_j x_j$ of these values are multiplied together. Hence as n increases the product $\prod_{j=1}^{m} \theta_j^{x_j}$ decreases exponentially. Note the multinomial coefficient $n!/\prod_{j=1}^{m} x_j!$ does increase with n, but more slowly.

(b) Consider the multiclass Bayesian classifier

$$\hat{y} = \operatorname*{argmax}_k \frac{p(x|y=k)p(y=k)}{p(x)}.$$

Simplify the expression inside the argmax operator as much as possible, given that the model $p(x|y=k)$ for each class is a multinomial distribution.

The denominator $p(x)$ is the same for all k, so it does not influence which k is the argmax. Within the multinomial distributions $p(x|y=k)$, the multinomial coefficient does not depend on k, so it is constant for a single x and it can be eliminated also, giving

$$\hat{y} = \operatorname*{argmax}_k \; p(y=k) \prod_{j=1}^{m} \theta_{kj}^{x_j}.$$

where θ_{kj} is the jth parameter of the kth multinomial.

(c) Consider the classifier from part (b) and suppose that there are just two classes. Simplify the classifier further into a linear classifier.

Let the two classes be $k=0$ and $k=1$ so we can write

$$\hat{y} = 1 \;\; \textit{if and only if} \;\; p(y=1) \prod_{j=1}^{m} \theta_{1j}^{x_j} > p(y=0) \prod_{j=1}^{m} \theta_{0j}^{x_j}.$$

Taking logarithms and using indicator function notation gives

$$\hat{y} = I\left(\log p(y=1) + \sum_{j=1}^{m} x_j \log \theta_{1j} - \log p(y=0) - \sum_{j=1}^{m} x_j \log \theta_{0j} \right).$$

The classifier simplifies to

$$\hat{y} = I(c_0 + \sum_{j=1}^{m} x_j c_j)$$

where the coefficients are $c_0 = \log p(y = 1) - \log p(y = 0)$ *and* $c_j = \log \theta_{1j} - \log \theta_{0j}$. *The expression inside the indicator function is a linear function of* x.

Assignment

The purpose of this assignment is to compare two different approaches to learning a classifier for text documents. The dataset to use is called Classic400. It consists of 400 documents from three categories over a vocabulary of 6205 words. The categories are quite distinct from a human point of view, and high accuracy is achievable.

First, you should try Bayesian classification using a multinomial model for each of the three classes. Note that you may need to smooth the multinomials with pseudocounts. Second, you should train a support vector machine (SVM) classifier. You will need to select a method for adapting SVMs for multiclass classification. Since there are more features than training examples, regularization is vital.

For each of the two classifier learning methods, investigate whether you can achieve better accuracy via feature selection, feature transformation, and/or feature weighting. Because there are relatively few examples, using ten-fold cross-validation to measure accuracy is suggested. Try to evaluate the statistical significance of differences in accuracy that you find. If you allow any leakage of information from the test fold to training folds, be sure to explain this in your report.

The dataset is available at `http://www.cs.ucsd.edu/users/elkan/291/classic400.zip`, which contains three files. The main file `cl400.csv` is a comma-separated 400 by 6205 array of word counts. The file `truelabels.csv` gives the actual class of each document, while `wordlist` gives the string corresponding to the word indices 1 to 6205. These strings are not needed for training or applying classifiers, but they are useful for interpreting classifiers. The file `classic400.mat` contains the same three files in the form of Matlab matrices.

Chapter 11

Social network analytics

A social network is a graph where nodes represent individual people and edges represent relationships. Edges can be directed or undirected, and they can be of multiple types and/or weighted. Both nodes and edges can be associated with vectors of feature values.

There are two types of data mining one can do with a social network: supervised and unsupervised. The aim of unsupervised data mining is often to come up with a model that explains the statistics of the connectivity of the network. Mathematical models that explain patterns of connectivity are often time-based, e.g. the "rich get richer" idea. This type of learning can be fascinating as sociology. For example, it has revealed that nonsmokers tend to stop being friends with smokers, but thin people do not stop being friends with obese people. Models of entire networks can lead to predictions about the evolution of the network that can be useful for making decisions. For example, one can identify the most influential nodes, and make special efforts to reach them.

Discussion: closing schools because of flu.

An alternative objective is to make predictions concerning numerous individual nodes. For example, we may want to recognize which phone numbers are being used fraudulently, in a social network where an edge between persons a and b means that a calls b.

11.1 Collective classification

There are many cases where we want to predict labels for nodes in a social network. Often the training and test examples, that is the labeled and unlabeled nodes, are members of the same network, that is, they are not completely separate datasets.

The standard approach to cross-validation is not appropriate when examples are linked in a network.

In traditional classification tasks, the labels of examples are assumed to be independent. (This is not the same as assuming that the examples themselves are independent; see the discussion of sample selection bias above.) If labels are independent, then a classifier can make predictions separately for separate examples. However, if labels are not independent, then in principle a classifier can achieve higher accuracy by predicting labels for related examples simultaneously. This situation is called collective classification.

Suppose that examples are nodes in a graph, and nodes are joined by edges. Edges can have labels and/or weights, so in effect multiple graphs can be overlaid on the same examples. For example, if nodes are persons then one set of edges may represent the "same address" relationship while another set may represent the "made telephone call to" relationship.

The simplest approach to collective classification is to extend the representation of each example to include feature values computed based on its neighbors. A node can have a varying number of neighbors, but many learning algorithms require fixed-length representations. For this reason, features based on neighbors are often aggregates. Aggregation operators include "sum," "mean," "mode," "minimum," "maximum," "count," and "exists." Alternatively, one can define a kernel function (i.e. a similarity function) that takes into account neighbors.

Intuitively, the labels of neighbors are often correlated. Given a node x, we would like to use the labels of the neighbors of x as features when predicting the label of x, and vice versa. A principled approach to collective classification would find mutually consistent predicted labels for x and its neighbors. However, in general there is no guarantee that mutually consistent labels are unique, and there is no general algorithm for inferring them. Experimentally, a simple iterative algorithm often performs as well as more sophisticated methods for collective classification.

Given a node x, let $N(x)$ be the set of its neighbors. Let $S(x)$ be the bag of labels of nodes in $N(x)$, where we allow "unknown" as a special label value. Let $g(x)$ be some representation of $S(x)$, perhaps using aggregate operators. A classifier is a function $f(x, g(x))$. A training set may include examples with known labels and examples with unknown labels. Let L be the training examples with known labels. The examples that are used in training are the set

$$E = \bigcup_{x \in L} N(x).$$

Given a trained classifier $f(x, g(x))$, the algorithm for classifying test examples is the following.

Initialization: for each test node x
    compute $N(x)$, $S(x)$, and $g(x)$
    compute prediction $\hat{y} = f(x, g(x))$
repeat
    select ordering R of nodes x
    for each x according to R
        let $S(x)$ be the current predicted labels of $N(x)$
        compute prediction $\hat{y} = f(x, g(x))$
until no change in predicted labels

The algorithm above is purely heuristic, but it is sensible and often effective in practice.

11.2 Link prediction

Inferring unobserved links versus forecasting future new links.

Similar to protein-protein interaction

One can view the task as similar to collaborative filtering, and use a matrix-decomposition method. Empirically, the Enron email adjacency matrix has rank approximately 2 only

Keyword overlap is the most informative feature for predicting co-authorship. Sum of neighbors and sum of papers are next most predictive; these features measure the propensity of an individual as opposed to any interaction between individuals. Shortest-distance is the most informative graph-based feature.

11.3 Other topics

Enron email visualization: `http://jheer.org/enron/v1/enron_nice_1.png`

Network-focused versus node-focused analysis.

Cascading models of behavior.

Record linkage, alias detection.

Betweenness Centrality: Brandes 2001.

Profiles in Terror dataset.

Chapter 12

Interactive experimentation

This chapter discusses data-driven optimization of websites, in particular via A/B testing.

A/B testing can reveal "willingness to pay." Compare profit at prices x and y, based on number sold. Choose the price that yields higher profit.

Quiz

The following text is from a New York Times article dated May 30, 2009.

> Mr. Herman had run 27 ads on the Web for his client Vespa, the scooter company. Some were rectangular, some square. And the text varied: One tagline said, "Smart looks. Smarter purchase," and displayed a $0 down, 0 percent interest offer. Another read, "Pure fun. And function," and promoted a free T-shirt.
>
> Vespa's goal was to find out whether a financial offer would attract customers, and Mr. Herman's data concluded that it did. The $0 down offer attracted 71 percent more responses from one group of Web surfers than the average of all the Vespa ads, while the T-shirt offer drew 29 percent fewer.

(a) What basic principle of the scientific method did Mr. Herman not follow, according to the description above?

(b) Suppose that it is true that the financial offer does attract the highest number of purchasers. What is an important reason why the other offer might still be preferable for the company?

(c) Explain the writing mistake in the phrase "Mr. Herman's data concluded that it did." Note that the error is relatively high-level; it is not a spelling or grammar mistake.

Bibliography

[Huang et al., 2006] Huang, J., Smola, A., Gretton, A., Borgwardt, K. M., and Schölkopf, B. (2006). Correcting sample selection bias by unlabeled data. In *Proceedings of the Neural Information Processing Systems Conference (NIPS 2006)*.

[Jonas and Harper, 2006] Jonas, J. and Harper, J. (2006). Effective counterterrorism and the limited role of predictive data mining. Technical report, Cato Institute. Available at `http://www.cato.org/pub_display.php?pub_id=6784`.

[Michie et al., 1994] Michie, D., Spiegelhalter, D. J., and Taylor, C. C. (1994). *Machine Learning, Neural and Statistical Classification*. Ellis Horwood.

[Vert and Jacob, 2008] Vert, J.-P. and Jacob, L. (2008). Machine learning for in silico virtual screening and chemical genomics: New strategies. *Combinatorial Chemistry & High Throughput Screening*, 11(8):677–685(9).