Improving Tictoc with Machine Learning

David Cordoba

Introduction

Over the summer, a friend and I created an event discovery application after we were in San Francisco one night and found it was too difficult to discover cool local events happening around us at that specific time. The kinds of events we wanted were things like live music at a coffee shop, mariachi night at a Mexican restaurant, or a happy hour at a bar. These are events that are not easy to find because there is no one centralized source that compiles all of them. You'd have to go to the websites of each business near you to find out if any of them have cool events happening at the moment. We scrapedevents.sfgate.com since they had pretty good events for the Bay Area and we scraped a happy hours website. They both had well-structured data, so we were able to place each event into one of the following categories: sports, culture, music, food, and happy hours.

Categorizing events with poor metadata

A problem we have run into while working on Tictoc is that a lot of events out there don't have great metainfo. Our events have very few fields, a title, description, time range, location, and category. Most new sources we find have all but the category, so we need something to assign to the proper category. Fortunately, we have many events with good category info, so we have a pretty good training set to use for classifying unknown events. The first challenge to get this to work is to find out which features to use for classification.

Preparing the data

The data is a bit awkward to use because it is currently stored in a google app engine project online. Because of processing constraints on app engine, bulk downloading of the data is a bottleneck for my progress. What I am downloading, though, is a CSV file that contains the text for a description, the text for the title, and then the category. I will need to take a look at the words that are found and extract any information that is not valid. Much like how the data was preprocessed in problem set 2 for the naive bayes excercise, I will need to select some set of distinct tokens, then create a file that could be loaded into Matlab to create a matrix of token appearance counts, where the i,jth element is the number of appearances of token j in the training example j.

There are some events that are counted in multiple categories. For our purposes, we don't want to determine which set of categories it could fit in, just the best one. For that reason, I will randomly select a category for events with multiple categories to be placed in. This prevents us from being penalized when two versions of the same event with different categories end up in the testing set.

Feature Selection

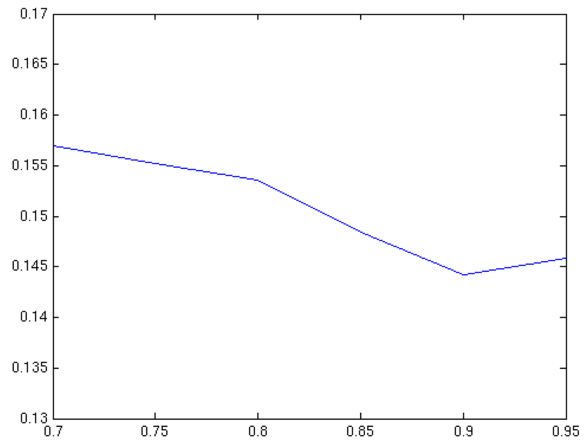
My first attempt at classification is to use text classification on the title and description of the event. I'll try adding in the location afterward to see if that has a positive impact on the classification success rate. My very first basic attempt will be to treat the title and description as a bag of words that I will then use for classification.

Of course, the list of tokens being used could be modified to be a bit more descriptive of the categories themselves. One method for doing this is selecting k words from each category that have the highest mutual information score. Mutual information determines which tokens give the most information about a particular category. For example, after calculating the mutual information scores for each of the categories, I found that the most relevant words in the *happy hours* category included “drink” and “appetizer”, while the most relevant words in the *sports* category included “football” and “basketball.”

Training And Classification

No Feature Selection

The data set includes over 10,000 event items. Before implementing feature selection I decided to test out my naïve Bayes Classifier to get an estimation of how much feature selection really helps. In this training example, we are only tokenizing by spaces, and including each word available. Testing is done by holding out a percentage of the total data set for classification, then comparing the predicted value to the actual value.

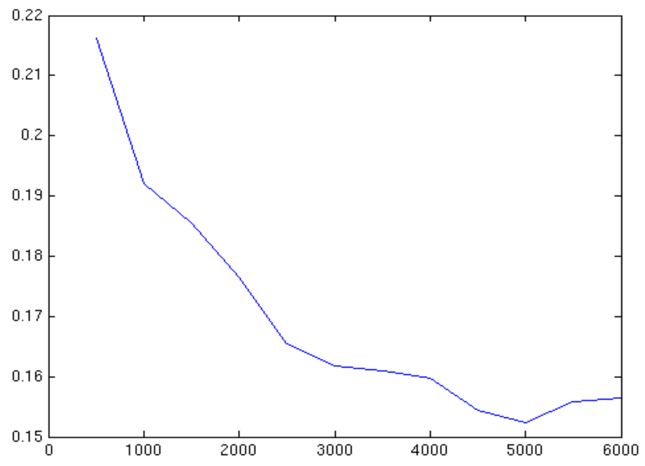


Classification Error as a function of training set size(ratio of data set used for training)

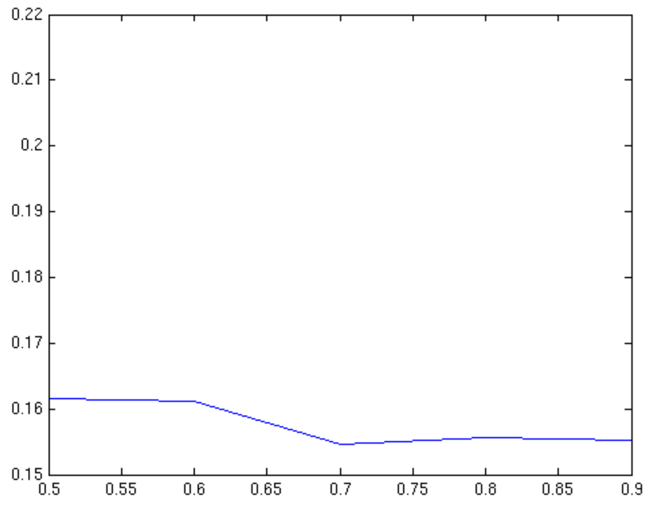
The results I got were not great, but they are a good start. This tells me that this training is a lot better than it would be if it were to trivially select the most likely category based on the training set. About 30% of the events belong to the “culture” category, so we are getting a better error rate than the 70% rate we would average if we just guessed.

With Feature Selection

Now we need to select the best features (based on the mutual information score) and truncate the dictionary of features so that we only use those in the top k for each category. There are well over 6,500 unique words (on average) in each category. In order for this to work we need to train on only a subset of those words. There are two approaches, we could take a lump sum from each category and add that to our dictionary, or we could take a percentage. Testing is done just as before, except this time we are holding the training set size constant at 70%, and testing on the remaining 30% of the examples. We will vary the feature set size instead.



Training error as a function of number of features taken from each category



Training error as a function of percentage of category vocab used as features

Without feature selection, the average error rate for training on 70% of the examples was about .157. We only eclipsed that when we selected about 4500 words from each category, or about 70% of the vocab of a category. This is not great performance, and it shows that for this particular data set, feature selection by mutual information does not do much to improve performance.

Recommending Events

One of the major problems that we encountered while developing this application was that there were simply too many events out there, and we had no way of displaying relevant ones only. Our best bet was to hand-pick a few of them and display those. We had created the ability for users to bookmark certain events that they like, but we didn't know how we could use that information to curate the events for us.

Acquiring the Data

Unlike in category classification, this time I did not have the luxury of too many data points. Bookmarking, as it turns out, is not something people generally like to do. Additionally, we only allow you to bookmark, and there is no opposite action to show your distaste for a certain event. I set up a website where people could go vote on random events in order to gather more data, and used that information for the recommendations.

Different Recommendation Engines

There are two general approaches that I could have used to make recommendations, content based filtering and collaborative filtering. Content based filtering requires knowledge of how to model the different events, whereas collaborative filtering figures it out based on the similarity of ratings from users. For this project I decided to use collaborative filtering, but since I already had this naïve bayes classifier for events, I decided to try that out on the recommendations to see how well it did.

Collaborative Filtering

I implemented this portion using Mahout's collaborative filtering library. After formatting the data in a way that the library would accept it, I fed it in and used the built in slope one recommender. This provided me with the top recommendations for each user, and the projected rating for the events that a user had not yet rated.

Evaluating the results

My first idea for evaluating the results was to hold out a certain amount of the ratings data for testing, while training on the rest. I would use the predicted value and use it to compare to the user's 0 or 1 rating. With this evaluation, the correct rating was given 57.28% of the time. The average rating for all of the data points was about 0.51, so this was only slightly better than guessing would have been. Collaborative filtering is not great for projecting scores, but excels when you are asking for the top recommendations. I took the top 10 recommendations given by the algorithm and asked users to rate those events (among other filler events). Of the events that were given as recommendations, almost 70% received favorable ratings from the users. Collaborative filtering is more useful when there are a lot of users rating a lot of different items. Unfortunately, I was not able to collect enough data to provide extremely relevant recommendations, but the modest results I got give me hope that we might some day be able to implement this a lot better.

Just for Kicks: Naïve Bayes recommendations

Since naïve bayes performed decently well when choosing the right category for an event, I decided to see how well it would do when I ask it to choose if an event would be something I would like or something I wouldn't like. I had rated 114 events; I gave 68 likes and 46 dislikes. By purely guessing that I would "like" an event, I should expect about a 40% error rate. After running naïve bayes, that is exactly what I got. I decided to check out the most informative words for each category, based on the mutual event score, to see if there are some key words that determine whether or not I like an event. Turns out there aren't. The words were mostly random, and didn't really make sense. Maybe it's just that there wasn't enough data, but the text content of the event does not necessarily tell us much about whether an event would seem good to me or not. In order to make the recommendations better, we just need a lot more user ratings, so we need to emphasize that in our application.