CS 4780/5780 Machine Learning Due September 6, 2012

Assignment 1: Version Spaces and k-Nearest Neighbors

Instructor: Thorsten Joachims *Total Points: 100*

Course Policy:
- Assignments are due at the beginning of class on the due date in hard copy including printout of important parts of your code
- Also, submit the code via CMS.
- Write your NetIDs with submission date and time on the first page of the hard copy.
- Late assignments lose 1% of the cumulated final homework grade for each 24 hours
- No assignment will be accepted after the solution is publicized (about 4 days after due)
- Late assignments can be submitted in class, in office hours, or directly to TAs.
- All sources of material must be cited. Assignment solutions will be made available along with the graded homework solutions. The University Academic Code of Conduct will be strictly enforced, including running cheating detection software.

Problem 1: Version Spaces [25 points]

Alice is starting her Ph. D. program at Cornell this year. Since Alice is new in town, she wants to know where to get good food. While Alice tried several restaurants in Collegetown during her first week which were rated well in reviews on Opentable, she was often disappointed, and concluded that the review system does not match well with her preferences because the star-rating scale is too detailed. Opentable reviews consist of four different topics: food, ambiance, service, and noise level. Now, she wants to redesign them by *simplifying* evaluation criteria as follows:

- *Food:* Simplify to three categories such as *poor*, *average*, or *good* food.
- *Ambiance:* Simplify to three categories such as *inferior*, *normal*, or *superior* ambiance.
- *Service:* Simplify to two categories such as *bad* or *poor* service.
- *Noise-level:* Completely ignore this criterion.

She thinks noise level is too subjective to be scaled and highly sensitive to the visiting times because most restaurants have contrastive evaluations from many users. Based on these simplified criteria, Alice tries to predict whether or not she *likes* a certain restaurant. The next table shows her initial research for five different restaurants in Collegetown. Our purpose is to learn a function $F \times A \times S \rightarrow L$ using the table as a training set T of five examples.

Restaurant	Food?	Ambiance?	Service?	Like?
1	good	normal	fair	yes
2	average	superior	bad	yes
3	poor	superior	bad	no
4	good	inferior	fair	no
5	average	normal	bad	no

(a) Define attribute F, A, S, and output space L of Food, Ambiance, Service, and Like respectively. What is the size of the instance space X?

(b) Suppose that the hypothesis space consists of all possible functions on $F \times A \times S \rightarrow L$. What is the size of the hypothesis space H?

(c) Initially, Alice thinks that *she will like any restaurant having good food.* Is this hypothesis h consistent with the training set T? Why or why not?

(d) For the above training set T and hypothesis space H defined in (b), what is the size of the version space $VS_{H,T}$?

Alice realizes that the hypothesis space H under (b) is too large. She now tries to evaluate each criterion by assinging numeric scores such as ($poor = -1$, $average = 0$, $good = -1$) for Food, ($inferior = -1$, $normal = 0$, $superior = 1$) for Ambiance, ($bad = -1$, $fair = 1$) for Service, and restricts hypotheses to functions that only use the **sum of any two attribute values** to predict whether or not she would *like*. In particular, ($Like = yes$ if $sum > 0$, $Like = no$ if $sum \leq 0$). For instance, Alice now *likes* the Restaurant 4 because it achives 2 points from its *good food* and *fair service* even with *inferior ambiance*.

(e) Let H' be the new hypothesis space satisfying the above numeric formulation. Enumerate the different hypotheses in H'.

(f) For the training set T' and the hypothesis space H', what is the size of the version space $VS_{H',T'}$?

Problem 2: kNN Decision Boundaries [25 points]

Consider a binary classification problem with two real valued features x_1 and x_2. Figures 1 & 2 illustrate two different training sets S_1 and S_2. White circles denote positively labeled examples, whereas black squares denote the examples labeled as negative. In order to classify a new point, we will use unweighted k-Nearest Neighbors using Euclidean distance with various k. Thus the label for a new point will be given by the majority class (*i.e.*, positive or negative) amongst the k examples which are closest to the point to be classified.

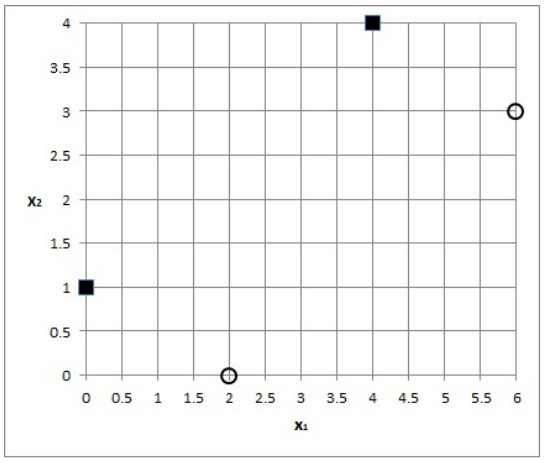

Figure 1: The training set S_1

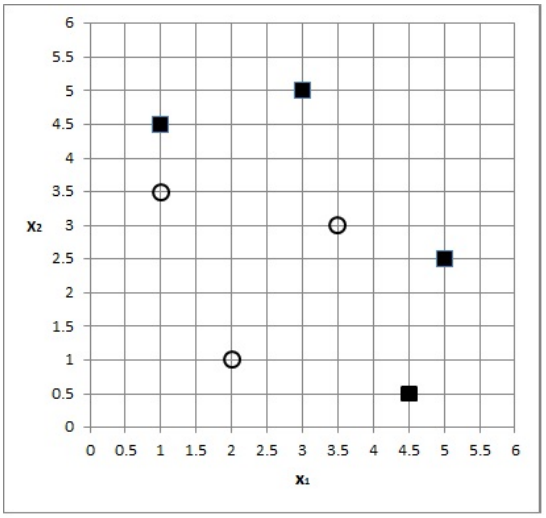

Figure 2: The training set S_2

(a) Draw the decision boundaries of S_1 and S_2 for $k = 1$.

(b) What would the labels of (3, 2) and (4, 2) be in S_1? Ties are broken by predicting the **positive** class.

(c) What would the labels of (4.5, 4) and (4, 2) be in S_2? Ties are broken by predicting the **positive** class.

When $k > 1$, a partition of spaces like the above is called the k^{th}-order Voronoi Diagram or Voronoi Tesselation. Now we are going to approximate the decision boundaries of the training set S_1 for odd $k > 1$.

(d) Draw the decision boundaries of S_1 for $k = 3$?

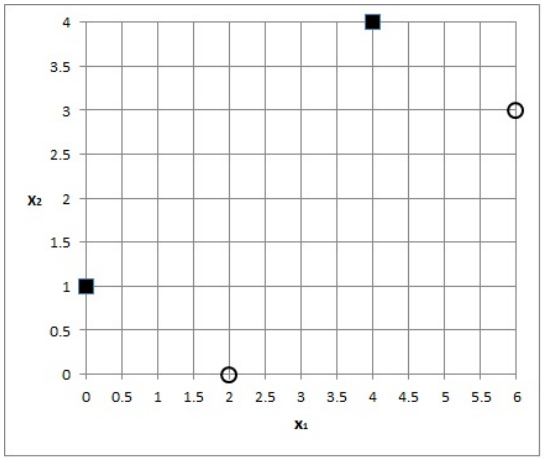

Figure 3: The training set S_1

(e) *Bonus Problem (optional):* If the x_2-coordinate of four example points in S_1 were multiplied by 5, what would happen to the boundary for $k = 1$? Draw another picture. How might this effect cause problems when working with real data?

Problem 3: kNN Implementation [50 points]

In this task, you will implement a k-Nearest Neighbor method for the *Collaborative Filtering* problem. Collaborative Filtering (or CF for short) seeks to understand how to make good recommendations to a user for new movies, products, music, etc. that he or she might like. Such methods solve this problem by seeking patterns in the tastes of similar users, and they often avoid relying on content-based information (such as the audio signal of a song).

In this particular instance, we will work with recommending songs. As a substitute for users, we will be analyzing playlists collected from various radio stations. To this end, we will not be considering the ordering of songs in a playlist here – just which songs occur in a playlist and how many times. In this way we intend to approximate the idea of users' music collections with these radio playlists, so for the rest of the assignment we'll just speak in terms of the users we're simulating instead of playlists. Each of the lines in the *user_train.txt* file represents an individual user in a format similar to a standard format for data used in machine learning applications. Namely, each line is of the format:

$$< \texttt{user ID} > \ - \ < \texttt{feature index} >:< \texttt{feature value} > \\ < \texttt{feature index} >:< \texttt{feature value} > \ ...$$

Any feature index which is not explicitly mentioned is assumed to be equal to zero. This way, sparse data (which is common in many applications) can be efficiently represented. In user_train.txt, feature indices correspond to song IDs, and feature values correspond to the number of times a user has played that song, so that each line gives us a histogram of plays over songs. You could also imagine that each line gives us one user's rating of how much he likes each of the songs. The file *song_mapping.txt* contains the mapping from feature indices (or song IDs) to song titles and artists.

Each of the users' collections in user_train.txt has had 60% of the original songs removed. This is intended to simulate songs that a user eventually bought or liked – songs that we know the user later thought were good. In this simulation, then, we have access to a set of songs that the user likes, and we want to try to recommend songs to the user that he or she will like. The removed songs appear in the file *user_test.txt*, where each line has the format:

$$< \texttt{user ID} > \ - \ < \texttt{removed song} > < \texttt{removed song} > \ ...$$

We will try to guess which songs the user will like by using our kNN Collaborative Filtering technique. In a nutshell, we will use kNN to come up with a ranking of the best songs for a given user.

First, our performance metric will be *precision at 10*, which tells us what proportion of the 10 highest ranked items are relevant (or here, how many of the 10 highest ranked items appear in the set of removed songs, or how many of the top retrieved songs are liked by the user). The metric for a single ranking is:

$$\text{Prec@10} = \frac{|R_{rel}|}{|R|}$$

where R is the set of the top 10 returned items and R_{rel} is the subset of those items which are liked (appear in the removed songs). This metric is for a single user – to report an aggregate performance measure on the entire dataset, you should report the average precision at 10 over all the users.

The nearest neighbor technique you will implement works as follows. First, for a query user u (with a music collection represented by point x_u in the space), find the top k nearest neighbors $N_k(u)$ (given a similarity metric $K(x, y)$ to be defined shortly). Now find the ranking vector r_u so that:

$$r_u = \begin{cases} \frac{1}{k} \sum_{j \in N_k(u)} x_j & \text{(Unweighted case)} \\ \\ \frac{\sum_{j \in N_k(u)} x_j K(x_j, x_u)}{\sum_{j \in N_k(u)} K(x_j, x_u)} & \text{(Weighted case)} \end{cases}$$

The ranking will come from sorting the components of the vector r_u – the songs that correspond to indices with very large weight are considered to be more relevant than those with small weight, since large weight means a song was played frequently by similar users. Note: the songs removed from a user's collection have had all plays removed, so when retrieving songs, you should only consider the ten most relevant songs which *do not already occur in the query user u's collection.* We will consider the following three similarity metrics for $K(x, y)$:

- Inverse of Euclidean distance: $K(x, y) = \frac{1}{||x-y||_2^2}$
- Dot product: $K(x, y) = x \cdot y$
- Cosine distance: $K(x, y) = \frac{x \cdot y}{||x||_2 ||y||_2}$

You will have to implement the k-Nearest Neighbor algorithm described above for this Collaborative Filtering technique. Your code should accept the following options as parameters:

- *k:* the number of neighbors to consider

- *Weighted?:* Boolean flag indicating whether to use the weighted or unweighted versions of the algorithm
- *Similarity Metric:* integer parameter indicating which similarity metric to use: 0 for inverse euclidean distance, 1 for dot product, 2 for cosine distance.
- *User Query:* integer parameter for a single user to query. If this argument is set to a non-default value, then the code should compute the ranking for this user alone, and should display the ten songs occuring most frequently in this user's collection, as well as the top ten highest ranked retrieved songs for this user.
- *Artist Query:* a string parameter for a musical artist to query. If this argument is set to a non-default value, then the code should 1) construct a music collection vector where each song from this artist is played once, display ten of the artist's songs to the user, and then 2) compute the ranking of songs for this user's collection and display the top ten highest ranked retrieved songs for that user (like in the previous problem). Do the artist matching the simple way, by checking the entire string from song_mapping.txt for any occurrence of the artist name (i.e. even if the artist name only occurs in the title, we want that song).

You should complete each of the following experiments and exercises (a) through (h) and **make sure to submit a printout of your code and a README explaining how it was intended to be run when submitting the solutions to the rest of the assignment**:

(a) *Qualitative evaluation:* Before anything else, let's get a feel for how the method actually works for recommending music. Using the cosine metric, $k = 100$, and the weighted version of the algorithm, do a query for the artist *Metallica* and show the results. Now, pick one or two of your own favorite artists, perform queries for them, and show the results (this works best if you pick an artist that has a decent number of songs and plays in the data). What do you think of the method's recommendations? Are they any good? About how many of the songs do you actually like or think you might like?

(b) *Baselines:* Now start your quantitatve analysis by implementing two baseline methods against which you'll compare your kNN method and report their performance:

- *Random:* Take the list of all song IDs and randomly shuffle it. Use the first 10 songs in the shuffled list, in that order, as your ranking.
- *Popularity-based:* Rank the songs in decreasing order according to the total number of plays each one has in any user's music collection in the training data.

(c) *Effect of k:* Using the inverse Euclidean distance metric and the unweighted version of the method, vary the value of k, trying out the values in {1, 3, 5, 10, 25, 50, 100, 250, 500, 1000}, and observe the effect on performance. Plot the performance across values of k on a log scale. Summarize and explain your findings. Note that for this

task, a number from 0.6 to 0.7 would be considered quite good (this would mean that roughly two thirds of the recommendations made are ones the user liked), and for this metric you can probably expect something lower than that.

(d) *Effect of Metric:* Now repeat the experiments (and plots) from part (c) with each of the other metrics. Which metric works the best? Why do you think this is?

(e) *Complexity:* For a given k, n training points, and dimensionality d, what is the complexity of testing a new point? Give your answer using big-O notation.

(f) *Effect of Sparsity:* Now suppose the data is sparse (only a few non-zero features for each example). Say that on average, s of the d features are non-zero for an example. What is the complexity of testing a new point now?

(g) *Weighted kNN:* Using the cosine metric and $k = 10$, report the performance with the weighted kNN approach. Describe how it compares with the performance in the unweighted case.

(h) *Bonus problem (optional):* Can you come up with another way to improve on the method? If so, what's the best precision at 10 that you can achieve?