Problem Set 4: Machine Learning

6.034 Fall 2003

`http://www.ai.mit.edu/courses/6.034f/`

Due: Wednesday, November 12, 2003

(Recommended: Problems 1-3 by Nov. 3rd)

For this problem set, you will need to copy the following files to your ps4 directory: `id-trees.scm`, `learn-utils.scm`, `loader.scm`, `movies-code.scm`, `nearest-neighbors.scm`, `ps4-publictest.scm`, `ps4.scm`, `tester.scm`*, and all of the* `*.data` *files. The instructions for running the id-tree and nearest-neighbors code is included in the relevant .scm file.*

After a brief period of employment at the defense contracting startup company, Ben was fired because of a bug in one of his pieces of code that began posting large amounts of confidential data to the Internet. He was about ready to give up on ever writing software again when his old friend Alyssa P. Hacker called him with good news.

"Ben, I heard about the incident at the startup and although I'm sure you're upset about the whole situation, I have some good news. I'm starting up a consulting company called McLearning & Co. and I'm looking for some new analysts. We're focusing on creating learning systems and with your 6.034 background you'd be a perfect candidate."

Ben instantly took Alyssa up on her offer, deciding that life as a consultant would suit him just fine. After a few weeks of training, Alyssa told him she had a project for him. Ben and another McLearning consultant, Reggie Larity, would be moving to Hollywood and helping the film studio New Lambda Pictures learn about what kinds of people will enjoy different genres of movies.

Problem 1: Nearest Neighbors

Upon arriving at the client site, Ben and Reggie were introduced to Ouda Verdept, the CTO of New Lambda Pictures. After introductions and pleasantries, she gave a quick briefing to Ben and Reggie.

"In recent years our company has lost a sense of the kinds of customers who watch our movies, and it's hurting us in business. So the CEO decided to fire the Market Research division and use machine learning systems to figure out consumer preferences instead."

"Every year we produce movies in the following genres:"

- Adventures
- Art-flick
- Sci-fi
- Comedy
- Suspense/Horror
- Drama

"In the past, we've had customers rate movies on a scale of 0 (terrible) - 5 (fantastic). Additionally, when we hired our last set of consultants from Hollywood Consulting Group (HCG), they told us that from the studies they performed, most customers like different movies within the same genre equally. For example, if a customer really liked **Gladiator**, they will most likely enjoy **Romeo Must Die** as well."

"HCG also told us that we could classify our customers into the following types of people:"

- Average-Joe
- Arty
- Non-Violent
- Kids
- Curmudgeon
- Optimist

"The only problem is, they never did anything with all of this information. We heard one of their consultants mutter something about nearest neighbors classification but they never got actually delivered a

working system. We'd like you to finish the implementation effort so we can actually start classifying and recommending movies. We'll give you the data sets we already have and the code that HCG has written."

Ben and Reggie decided to split the problem up. Ben would get the nearest neighbors system up and working, so they would have something to demo quickly, while Reggie would investigate other learning systems that might work faster or produce better models.

Being a bit overwhelmed at first, Ben started by looking through the existing nearest-neighbors code (*nearest-neighbors.scm*). He also found that HCG developed two sets of training data: *sim-movie-loose-train.data* and *sim-movie-tight-train.data*. The loose training data has a much larger range of preferences per customer type while the tight training data has a lower range that preferences can take per customer type.

After groveling through the code, Ben realized that most of the code he needed was already written. All he needed was a way of evaluating how "near" two people's ratings were, and he would be able to build a nearest neighbors function. Opting for the simplest solution, Ben chose Euclidean distance as a measure for closeness.

(Part A) Comparing Data

Write `compare` *which will take two customer preference vectors and return the square of the Euclidean distance.*[1] *Assume that the preferences which are given as input to the compare function are lists of equal length.*

Ben thought that he was already done with the entire project until he realized that some of the customer data might be incomplete. For example, not every customer will have seen every movie and for every movie they haven't seen, their preference will be `#f`. These customers would be the ones that would be targeted by the movie company so Ben decides to deal with missing preferences by ignoring the values that have `#f`.

(Part B) Comparing Partial Data

Write `compare-missing` *which will take two customer preference vectors and not consider preference entries which are* `#f`*. As a special case, if all pairs of entries have at least one* `#f`*, then the the system should return a large number, make it 1000.*

With a way of comparing samples, Ben was finally ready to implement nearest neighbors. To do so, he first sorted the database, then chose the best results.

(Part C) Nearest Neighbors

Write `sort-database-by-nearness`*, a function to sort the database given a particular value of the* `pref` *variable. It should place the better matches earlier in the list. After that, write* `k-nearest neighbor`*, an algorithm using Euclidean distance.*

Optional: Interpolating Preferences

Once he had the nearest neighbors, Ben was able to predict new values simply by filling in blanks with the average preference of the nearest neighbors.

Write the function Ben used to generate expected preferences.

Problem 2: ID-Trees

Ben showed his working preference-guesser to Ouda, who was very pleased. He was quite happy as well, until she threw him a hard question: "Which movies are the best indicators about what somebody will like?"

Ben stumbled a bit in answering. "Well, it's not very easy to answer questions like that with a nearest neighbors system. We could run some sort of statistics on it maybe... but really the best answer is that HCG was inferior to McLearning as consultants and it's all their fault. We'll be able to answer all those questions really soon, though, with the identification trees system that Reggie has been working on."

[1] Euclidean distance is straight line distance: if the preference has two dimensions and the distance in one dimension is 4 while the other is 3 then `compare` should return $3^2 + 4^2 = 25$.

"Good," said Ouda, "I want to see it in a couple days."

Ben rushed off to find Reggie, and get back to work on the ID tree system that Reggie had been working on as an alternative to nearest neighbors.

In general, Ben liked the identification tree system better than the nearest neighbors system anyway. Since an ID tree groups data points together with a series of tests, then once the space is divided up into regions of homogeneous data points, the classification is much faster, since it no longer requires computing distances between individual data points. The hard part, though, is constructing the tree, which can take a long time as it consider different possible tests to apply.

Only two functions remained to code: generating possible test thresholds and calculating the disorder of a pair of sets. Ben and Reggie finished them up quickly, and tried out their tree function.

(Part A) Decision Thresholds *Write a function that computes the possible thresholds for decisions. Your function,* `compute-thresholds`*, should take a list of numbers and return all intermediate values.*

(Part B) Computing Disorder *Write the function that actually computes the disorders. Your function,* `disorder-of-threshold` *should take as input two lists of data points (see* `learn-utils.scm` *for exact data structure format) and compute the average disorder measure needed for the ID tree.*

Finally, with the ID tree function built, Ben was able to answer Ouda's question. Building an ID tree based on the data set `sim-movie-tight-train.data`, he discovered that nine of the movies were good indicators of preference.

(Part C) Analyzing an ID Tree *Build an ID tree using* `sim-movie-tight-train.data` *and analyze it to determine which nine movies are used in the tree. Note that the movie names are listed in* `movies-code.scm`*. You may find the variable* `*movie-names*` *from* `movies-code.scm` *useful for turning numbers into names.*

Optional Problem:

When Ben tried the same routine on `sim-movie-loose-train.scm`*, he found it produced a much bigger tree, one that contained 24 tests and had a lot of very small leaves. This tree is badly overfitted, and performs badly when tested. Figure out a way to prevent overfitting while constructing an ID tree, code it up, and show that it works.*

Problem 3: Understanding Threshold Computations (ID-Trees)

Ouda then asked Ben a new question, a question about predicting whether people would watch a given movie, based on other movies they had just watched. "For example," said Ouda, "Here is a data set for people who watched the Matrix Reloaded and X-Men United, two of the most popular movies of the summer. The axes are the amount they liked each movie. A point is a "+" if the person plans to watch The Matrix: Revolutions and a "-" if they don't." (See Figure 1)

"We want to know how liking two action movies corresponds with watching a third. Can your system build predictions?" Ben nodded, and quickly produced an ID tree predicting whether a person will watch The Matrix: Revolutions based on how much they liked X-Men United and Matrix Reloaded.

(Part A) Action Movie Identification Tree *Calculate the ID tree which Ben constructed from the data in Figure 1, using the greedy algorithm and standard disorder calculations. Break ties by choosing Matrix before X-Men.*

You should express the tree as a nested scheme list, where each branch is a list of `test left-branch right-branch` *and each leaf is the symbol* `no` *or* `yes`*. Please format your tests and tree such that for each branch the lower value is on the left. (i.e. all tests are ">" with false going to the left branch)*

For example, the tree in Figure 2 is linearized as: `((> L 1.5) No ((> R 0.9) ((> L 5) No Yes) Yes))`

(Part B) Action Movie Disorders *For each non-leaf node in the tree you produced, enter the average disorder (accurate to two decimal places). Order your entries as a left-right traversal of the tree. Each entry*

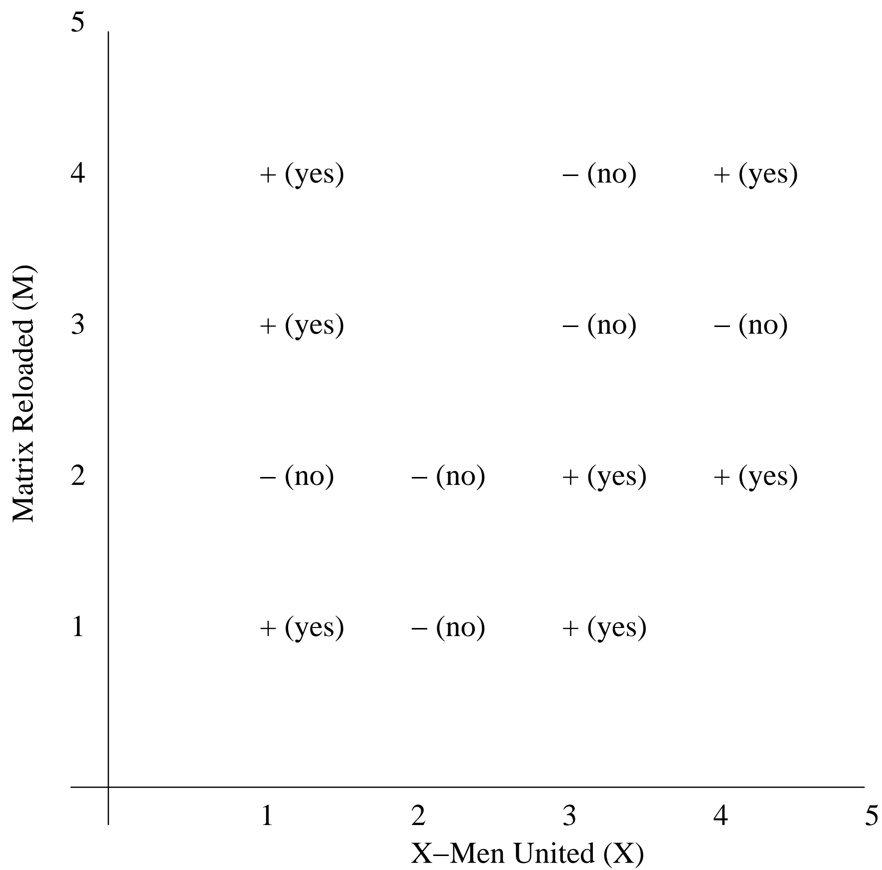

Figure 1: X-Men vs. Matrix Reloaded predicting Matrix: Revolutions

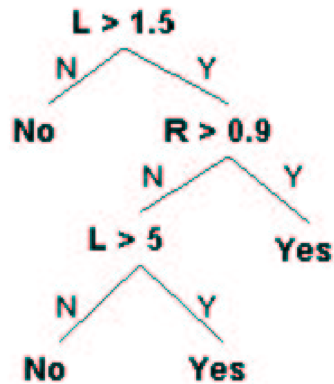

Figure 2:

should be described by the list of tests from root to most recent test, followed by the disorder. For example, your list of tests followed by the disorder may look like `((> L 1.5) (> R 0.9) 0.15)` for the node labeled `(> R 0.9)` on the diagram.

(Part C) Down-With-Love Identification Tree

Ben realizes that the numeric ratings for movies may not make sense when taken out of context. So, he wants to find out what circumstances affect people's ratings of a movie. He picks the recent romantic comedy "Down With Love" and does a survey among people who have watched it. He finds the following statistics:

- 8 women and 14 men liked the movie and would watch it again.
- 6 women and 13 men passionately hated the movie and under no circumstances would they tolerate watching it again.
- Of the 22 people who liked the movie, 4 women and 9 men watched it while on a date and their date went well.
- Of the 19 that did not like the movie, 2 women and 3 men watched it while on a date and their date went well.
- Of the 19 that did not like the movie, 3 women and 1 men watched it while on a date and their date did not go well.
- Of the people that went on a date, none of the people who did not enjoy their dates liked the movie.
- Of the people who did not watch the movie on while on a date 4 women and 5 men liked the movie while 1 woman and 9 men did not.

From these statistics, Ben was able to construct an ID tree showing the relative importance of dating and gender in determining whether a person liked "Down With Love".

Using the predicates `female?` *and* `date-status?`*, build an ID tree representing Ben's findings: note that the predicate* `date-status?` *returns three answers: "bad", "good", and "none", which you should branch on in that (alphabetical) order. Use the same tree representation as before.*

(Part D) Down-With-Love Disorders *For each non-leaf node in the tree you produced, enter the average disorder (accurate to two decimal places). Order your entries as a left-right traversal of the tree. Each entry should be a list of tests from root to most recent test, followed by the disorder. For example, your lists of tests followed by the disorder may look like:* `(date-status? 0.15)`.

Looking at the disorder values he gets, Ben decides that the features he has considered are actually not predictive of whether people like the movie. He gives up on identification trees and his social experiment related to moviegoer reviews and starts something completely different. Predicting the class of viewers using neural nets.

Problem 4: Neural-Nets

(Part A) To experiment with neural nets, Ben starts with the 2-dimensional feature spaces, S1-S4, shown at the top row of Figure 3. His goal is to use one of the perceptron nets, N1-N4, shown at the bottom row of Figure 3 to separate Artsy (label A) viewers from Average-Joes (label B) and Curmudgeons (label C). His network should return 1.0 if an unknown person is an Artsy viewer and 0.0 if he is in one of the other two viewer groups.

Note that the given networks are perceptron networks. Remember that perceptron units apply a threshold function to the weighted sum of their inputs, and return either 0.0 or 1.0.

Match each of the feature spaces in Figure 3 with the simplest perceptron network that can produce the shown decision boundaries.

(Part B) You are given the simple neural net sigmoid unit (See Figure 4) and the initial conditions shown below:

Initial values of inputs (x) and weights (w) and the desired output (y*):

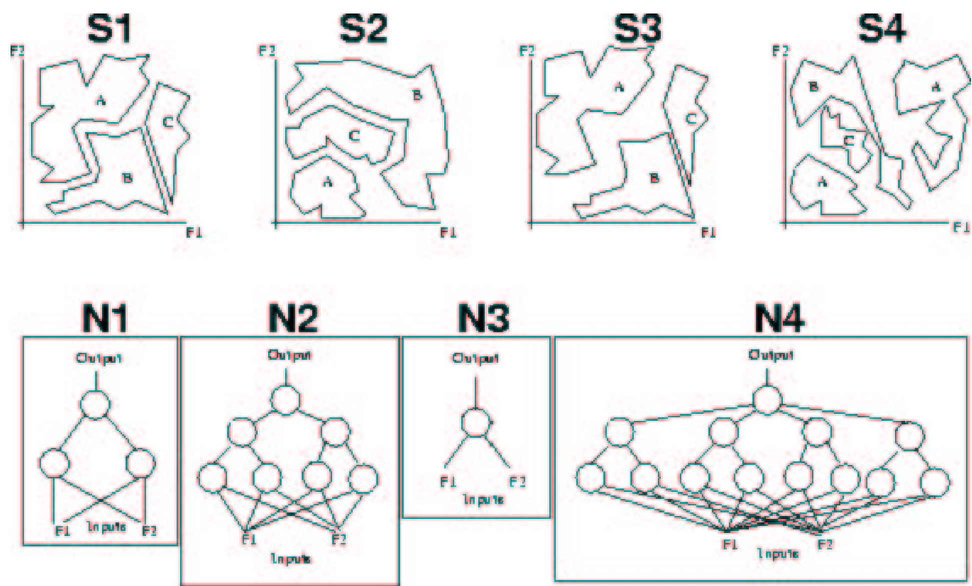

Figure 3:

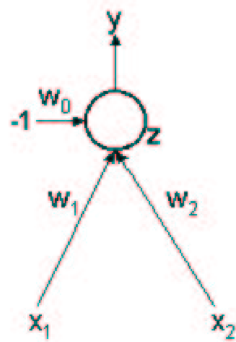

Figure 4:

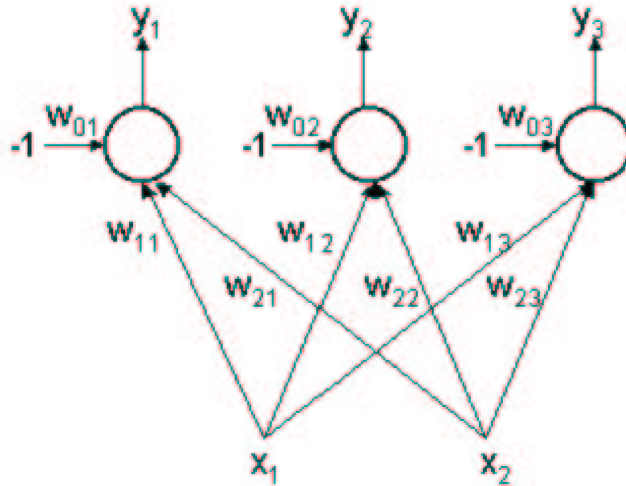

Figure 5:

- $x_1 = 0.1$,
- $x_2 = 1$,
- $w_0 = 0.05$,
- $w_1 = 0.05$,
- $w_2 = 0.05$,
- $y* = 1$.

Calculate the initial actual output, the new set of weights after the first iteration of standard backpropagation algorithm with a learning rate (r) of 8, and the actual output with the updated weights. Enter your answer into the .scm file as a list of the updated weights and the initial and final actual output, in that order. i.e. (w_0 w_1 w_2 *initial-y final-y*)

(Part C) Here you'll use a neural network to solve a three-class classification problem. To do this we introduce three output neurons and predict the label based on which output is the largest. For example, the output of (1,0,0), i.e. $y_1 = 1$, $y_2 = 0$ and $y_3 = 0$, corresponds to a label of 1, (0,1,0) corresponds to a label of 2 and (0,0,1) gives a label of 3. The neural net that we'll use for this purpose is given in Figure 5.

You can assume that the activation functions (also known as threshold functions, also known as transfer functions) for the three neurons are the typical sigmoid (squashed S) function. The output of neuron i is computed by first evaluating the total input to the neuron, $z_i = w_{1i}x_1 + w_{2i}x_2 - w_{0i}$, and passing this through the sigmoid function to derive the output: $y_i = sigmoid(z_i)$.

Recall that the derivative of the sigmoid function with respect to z is $y(1 - y)$.

Assume that all the weights are initially zero and that the error function is the squared error :

$E = 1/2(y_1 - y_1^*)^2 + 1/2(y_2 - y_2^*)^2 + 1/2(y_3 - y_3^*)^2$

The network is presented with the following example:

- inputs = $(x_1, x_2) = (1, -1)$
- target label = 3

Using a learning rate of 8, calculate the values of the weights after updating using the backpropagation rule with this single training example.

Now, using the new weights, identify the labels that the network would predict for the following input sets:

- inputs = $(x_1, x_2) = (1, -1)$,
- inputs = $(x_1, x_2) = (1, 0)$,
- inputs = $(x_1, x_2) = (1, 1)$.

Wrap-up

When you have completed this problem set, you should be able to `ssh` *into* `athena.dialup.mit.edu` *and type:*

```
cd ~/6.034-files/ps4/
add scheme
scheme
(load "tester.scm")
(test-file "ps4" "ps4-publictest")
```

This will evaluate your problem set based on the public test cases; you should pass them all! Good luck!