

# 15-381 Spring 2007

## Assignment 6: Learning

Questions to Einat (einat@cs.cmu.edu)

Spring 2007  
Out: April 17  
Due: May 1, 1:30pm Tuesday

The written portion of this assignment must be turned in at the beginning of class at 1:30pm on May 1st. Type or write neatly; illegible submissions will not receive credit. Write your name and andrew id clearly at the top of your assignment. If you do not write your andrew id on your assignment, you will lose 5 points.

The code portion of this assignment must be submitted electronically by 1:30pm on May 1st. To submit your code, please copy all of the necessary files to the following directory:

`/afs/andrew.cmu.edu/course/15/381/hw6_submit_directory/yourandrewid`

replacing yourandrewid with your Andrew ID.

**Late Policy.** Both your written work and code are due at 1:30pm on 4/3. Submitting your work late will affect its score as follows:

- If you submit it after 1:30pm on 5/1 but before 1:30pm on 5/2, it will receive 90% of its score.
- If you submit it after 1:30pm on 5/2 but before 1:30pm on 5/3, it will receive 50% of its score.
- If you submit it after 1:30pm on 5/3, it will receive no score.

### **Collaboration Policy.**

You are to complete this assignment individually. However, you are encouraged to discuss the general algorithms and ideas in the class in order to help each other answer homework questions. You are also welcome to give each other examples that are not on the assignment in order to demonstrate how to solve problems. But we require you to:

- not explicitly tell each other the answers
- not to copy answers
- not to allow your answers to be copied

In those cases where you work with one or more other people on the general discussion of the assignment and surrounding topics, we ask that you specifically record on the assignment the names of the people you were in discussion with (or “none” if you did not talk with anyone else). This is worth five points: for each problem, your solution should either contain the names of people you talked to about it, or “none.” If you do not give references for each problem, you will lose five points. This will help resolve the situation where a mistake in general discussion led to a replicated weird error among multiple solutions. This policy has been established in order to be fair to everyone in the class. We have a grading policy of watching for cheating and we will follow up if it is detected.

## Problem 1 - Decision Trees (20 points)

Use the ID3 algorithm for the following question.

- (10 points) Build a decision tree from the given tennis dataset. You should build a tree to predict PlayTennis, based on the other attributes (but, do not use the Day attribute in your tree.). Show all of your work, calculations, and decisions as you build the tree.  
What is the classification accuracy?
- (2 points) Is it possible to produce some set of correct training examples that will get the algorithm to include the attribute Temperature in the learned tree, even though the true target concept is independent of Temperature? if no, explain. If yes, give such a set.
- (5 points) Now, build a tree using only examples D1–D7. What is the classification accuracy for the training set? what is the accuracy for the test set (examples D8–D14)? explain why you think these are the results.
- (3 points) In this case, and others, there are only a few labelled examples available for training (that is, no additional data is available for testing or validation). Suggest a concrete pruning strategy, that can be readily embedded in the algorithm, to avoid over fitting. Explain why you think this strategy should work.

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Table 1: The play tennis dataset

## Problem 2 - Neural Networks (10 points)

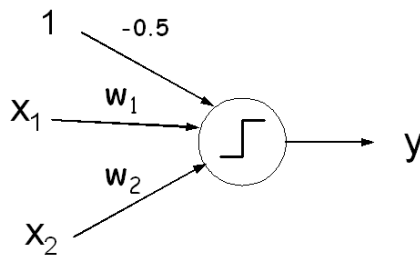
Given is the following single neuron perceptron. In this one-layer perceptron model, the neuron calculates a weighted sum of inputs. Then, it applies a threshold to the result: if the sum is larger than zero, the output is 1. Otherwise, the output is zero.

Consider the following examples, where  $Z$  is the *desired* output (indeed, this is the OR function).

In this question, you will apply the Perceptron update algorithm to *automatically learn the network's weights*, so that it classifies correctly all the training examples. The algorithm is simple:

Iterate through the training examples, one by one (if the last example was used, and the algorithm hasn't converged yet, start again from the first example, and so forth.).

For every example  $i$ :



$X_1$	$X_2$	$Z$
0	0	0
0	1	1
1	0	1
1	1	1

- Calculate the net's output  $Y_i$ .
- Multiply the error  $(Z_i - Y_i)$  by the learning rate  $\eta$ . Add this correction to any weight for which the input in the example was non-zero. That is, if for the current example  $i$   $X_1 = 1$ , then update  $W'_1 \rightarrow W_1 + \eta(Z_i - Y_i)$ , etc.
- If the network outputs the correct result for all of the training set examples, conclude.

Our questions:

$X_1$	$X_2$	$W_1$	$W_2$	$Z$	$Y$	Error	$W_1$	$W_2$
0	0	0.1	0.3					
0	1							
1	0							
1	1							
0	0							
0	1							
1	0							
1	1							
...								

Table 2: Results format

- (3 points) Apply the algorithm for the given training examples. Use learning rate  $\eta = 0.2$ . Assign the weights the initial values  $W_1 = 0.1, W_2 = 0.3$ . Give your results as specified in Table 1. You should expect to getting the final weights within only a few passes over the training examples.
- (5 points) The perceptron training algorithm is in fact a simple gradient descent update. In this question, you will derive this algorithm. The approach for training a perceptron here is to minimize a squared error function.
  - Give the definition of a *squared error* function,  $E$ , in terms of  $W_1, W_2, X_{i1}, X_{i2}$  and  $Z_i$ .
  - Each weight should now be updated by taking a small step in the opposite direction of its gradient (so as to minimize the error):

$$W' = W - \eta \nabla E(W)$$

Show how this translates into the algorithm that you applied in the previous question.

3. (2 points) In practice, the training example may be noisy. Suppose that there are *contradicting* examples in the training set: for example, an additional example, where  $X_1 = 1, X_2 = 1, Z = 0$ . How do you think this will affect the algorithm's behavior? (you are welcome to go ahead and try).

## Problem 3 - Naive Bayes and KNN (60+20 bonus points)

In this assignment you will build classifiers that should distinguish between valid email messages and spam.

You are given a labelled dataset, posted on the class website. The dataset is split into two zipped files: *train* and *test*. Each file will unpack into a directory that includes two kinds of email messages (one per file): SPAM (file names starting with sp) and MAIL (file names starting with numbers).

You will build classifiers using the *train data only*. The classifier performance should be evaluated using the *test data only*.

We will represent every message as a bag of words, as detailed below.

- Following is some relevant facts that you may want to use in building a Naive Bayes (NB) classifier. As a Bayesian classifier, it computes the following:

$$Pr(SPAM|message) = \frac{Pr(SPAM)Pr(message|SPAM)}{Pr(message)} \sim Pr(SPAM)Pr(message | SPAM)$$

$$Pr(MAIL|message) = \frac{Pr(MAIL)Pr(message|MAIL)}{Pr(message)} \sim Pr(MAIL)Pr(message | MAIL)$$

Your classifier will write out "SPAM" if  $Pr(SPAM|message) > Pr(MAIL|message)$  and "MAIL" otherwise.

How to compute those quantities?  $Pr(SPAM)$  and  $Pr(MAIL)$  should be easy given the *training data*. The other two terms,  $Pr(message|SPAM)$  and  $Pr(message|MAIL)$ , are where the "naive" part of Naive-Bayes comes in.

Let us represent a message as a sequence of  $n$  words,  $w_1, w_2, \dots, w_n$  (where  $w_i$  may be equal to  $w_j$ ). That is, we consider as features the occurrences of the message words.

The Naive-Bayes model assumes independence between the features. Thus, it computes the probability of a message conditioning on SPAM and MAIL as a product of the feature values probabilities, given SPAM and MAIL. So we have:

$$Pr(message|SPAM) = Pr(w_1(message)|SPAM) \times Pr(w_2(message)|SPAM) \dots \times Pr(w_n(message)|SPAM)$$

$$Pr(message|MAIL) = Pr(w_1(message)|MAIL) \times Pr(w_2(message)|MAIL) \dots \times Pr(w_n(message)|MAIL)$$

What are the probabilities  $Pr(w_i(message)|SPAM), Pr(w_i(message)|MAIL)$ ?

These can be readily obtained from the training data as well. For that, you should count the occurrences of words in the training data for each class. For example, suppose that in the SPAM training messages,  $Count("cheap") = 200, Count("now") = 50$ , etc.; and  $Count(allwords) = 10000$ . Then,  $Pr("cheap"|SPAM) = \frac{200}{10000} = 0.02$ .

However, there is a small fix to this simple ratio. While you obtain word counts from the data available for training, the messages in the test set may contain some words that did not occur in the training set (these are Out-Of-Vocabulary words, or OOV). In these cases  $Pr(w_k(message)|SPAM)$  would be

zero, meaning that  $Pr(message|SPAM)$  be zero as well. To avoid that, you may need to *smooth* your probability estimates.

The most simple smoothing mechanism is Laplace's law or add-one. Here, you simply add 1 to the count of each word (including the OOV token); you have to adjust the total count, as well, in order to produce a probability distribution:

$$P(w|CLASS) = \frac{Count(w|CLASS) + 1}{N(CLASS) + V}$$

The vocabulary size is the count of all the *unique* words that were observed in both the training and the test datasets. Per the previous example, suppose  $V = 1000$ . Then,

$$P(\text{"cheap"}|SPAM) = \frac{Count(\text{"cheap"}|SPAM) + \lambda}{N(SPAM) + V} = \frac{200 + 1}{10000 + 1000}$$

What is the probability of an unseen word in this case? (it's the same, only that  $Count(OOV)=0$ )

Finally, multiplying many small probabilities, you may run into underflow issues. To avoid that, the product  $\prod(w_i|CLASS)$  can be exchanged with  $\sum \log(w_i|CLASS)$ .

- (15 points) Write code that trains a NB classifier, as described. Your code should:
  - a. train a model based on the training data
  - b. output the accuracy rates per the training and test sets, where

$$Accuracy = \frac{\# - of - correctly - classified - messages}{all - messages}$$

- (5 points) Train a NB using *random* subsets of 20%, 40%, 60%, 80%, and 100% of the training data. Report the corresponding accuracies per the train (that is, the net portion of training data that was used) and test data. **Since results vary depending on the particular random set used, report average results over 5 runs.** Give your conclusions.
  - (5 points) Implement cross validation as follows. Split the training set randomly into 10 portions, of the same size. Train a model 10 times - where in every iteration one of the training data portions serves as the test set, and the model is trained on the rest of the data. Report the accuracy per individual test portion, and the overall cross-validation accuracy. Do you find the cross validation results reliable, comparing to evaluating accuracy using a separate test set?
2. Write code that implements a KNN classifier, for the same problem. For that, you will represent every message as a vector of word counts. The vector length is the vocabulary size (per the training data). Naturally, most of its entries would equal zero. For example, suppose that  $V=1000$ , and that  $i = 52$  is the index of the word "cheap". If message  $m$  has "cheap" appearing 7 times in it, then its representing vector will have 7 in the 52nd index.

For every message in the test set, calculate the similarity to each of the messages in the training set **using a cosine similarity measure**. Consider the top  $k$  most similar messages, to determine the test message class. **The cosine similarity measure is calculated as follows:**

$$Sim(A, B) = \frac{AB}{|A||B|} = \frac{x_{1A}x_{1B} + x_{2A}x_{2B} + \dots}{\sqrt{(x_{1A}^2 + x_{2A}^2 + \dots)}\sqrt{(x_{1B}^2 + x_{2B}^2 + \dots)}}$$

where  $x_{iA}$  is the count of the word of index  $i$  in document  $A$ .

- (10 points) Write the code for KNN classification.
- (5 points) Train the classifier using  $K = 1, K = 3, K = 5, K = 19$ . What are the corresponding accuracies on the test data?

3. (20 points) Now, as a pre-processing step, allow to choose the top  $T$  features (words). **We recommend that you use information gain as your criteria. Consider only those words that appear more than 50 times in the training data.** Note that you need *not* select the features in a hierarchical manner.

What are the top 10 most predictive words found by your measure?

What is the test accuracy, using NB and KNN models (**assign  $K=3$** ) that are structured using all of the training data, for  $T = 20, T = 50, T = 100, T = 200, T = 500$ ? Explain your results. What are the effects of feature selection on each of these algorithms?

4. (20 points, bonus!) Suggest your own model, including new features, which you think may be useful for this problem. One example of a feature is the length (in bytes) of a message. Another feature is the number of words. Or the number of occurrences of the letter Q. You can come up with features very easily. Use the KNN or the NB model.

Did you succeed to improve performance on the test set (comparing to previous results)? provide your features, type of model (NB or KNN) and results. If you managed to improve performance you will get 20 points for this question. Otherwise, you will get some of these points, depending on how we like your model... (we will try to be generous). Also, we may publish your model, if it does substantially well. (But, hey, remember that you are not allowed to look at the test set!)

The dataset we are using is due to:

I. Androustopoulos et al., "An Evaluation of Naive Bayesian Anti-Spam Filtering". In Proceedings of the Workshop on Machine Learning in the New Information Age, 11th European Conference on Machine Learning, Barcelona, Spain, pp. 9-17, 2000.

Thanks to Prof. Noah Smith for making it available.

#### Command line instructions:

Name your interface class as 'EMAILSPAM'. The command line arguments for your code should be as follows:

-dir: the root directory (that includes train and test)

-alg: NB/KNN

-mode: CV/TEST (cross validation of 10-way-split, versus train-and-test)

-ratio: % of training data used to build the model

-K: specifies  $K$  for the KNN algorithm (meaningless otherwise)

-T: the number of top selected features used (if '0', then use all features)

For example: EMAILSPAM -dir DIR -alg NB -mode TEST -ratio 80 -K 0 -T 50 – this will train a NB model using 80% of the training data, and the top 50 features. It will output accuracy rate per the training (the 80% used) and test sets.

## Problem 4 - K-Means (10 points)

Download the California Housing dataset from the course web page.

This dataset includes statistics about the median income, median age, average number of rooms per person, average number of persons in a household etc. The statistics were gathered per geographical block.

In this question, you will run the k-means algorithm to cluster this data. We should first install the Weka package (<http://www.cs.waikato.ac.nz/ml/weka/>).

Start Weka ('explorer' mode) and load in the data. Note, you may need to change the default memory usage for this exercise. Select the Cluster tab (select the 'use training data' mode).

Choose SimpleKMeans as the clustering algorithm. Use the default setting, which is for 2 clusters. This implementation of KMeans performs data normalization automatically.

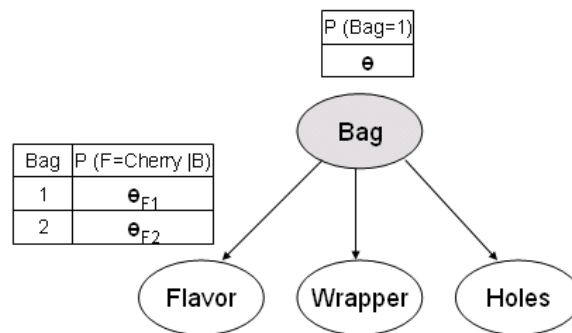
Cluster the data. Write down the SSE (sum squared error), cluster means, within cluster standard deviations for each attribute, etc. Visualize the clustering by right clicking on the results. Plot the latitude along the X axis and longitude along the Y axis. You'll get a plot that looks roughly in the shape of California. Using the cluster means and std. deviations as representative values in the cluster, what can you say about the clusters that were found? How do they compare with what you know about cities and areas in California? What can you attribute the gaps to? It will definitely be helpful to have a map of California handy.

Change K from 2 to 10 (click the 'simpleKMeans' box to reach this option) and repeat the clustering process above. Tabulate cluster means and standard deviations. When you change K, do any additional patterns emerge? If so, what are they?

This question is due to Prof. Craig Trumble.

## Problem 5 - EM (10 points - bonus!)

(Question 20.10 from Russel & Norvig.)



Consider the application of EM to learn the parameters for the network in the figure. The figure represents a situation in which there are two bags of candies that have been mixed together. Candies are described by three features: in addition to the *Flavor* and the *Wrapper*, some candies have a *Hole* in the middle and some do not. The distribution of candies in each bag is described by a naive bayes model: the features are independent, given the bag, but the conditional probability distribution for each feature depends on the bag. The parameter  $\theta$  is the prior probability that a candy comes from bag 1;  $\theta_{F1}$  and  $\theta_{F2}$  are the probabilities that the flavor is cherry, given that the candy comes from Bag 1 and Bag 2 respectively;  $\theta_{W1}$  and  $\theta_{W2}$  give the probabilities that the wrapper is red; and  $\theta_{H1}$  and  $\theta_{H2}$  give the probabilities that the candy has a hole. In the figure, the bag is a hidden variable because, once the candies have been mixed together, we no longer know which bag each candy came from. The problem is, to recover the descriptions of the two bags by observing candies from the mixture.

The true net parameters are as follows:

$$\theta = 0.5, \theta_{F1} = \theta_{W1} = \theta_{H1} = 0.8, \theta_{F2} = \theta_{W2} = \theta_{H2} = 0.3$$

	W=Red	W=Red	W=Green	W=Green
	H=1	H=0	H=1	H=0
F=Cherry	273	93	104	90
F=lime	79	100	94	167

That is, the candies are equally likely to come from either bag; the first is mostly cherries with red wrappers and holes; the second is mostly limes with green wrappers and no holes.

1000 samples were generated from this model. The counts for the eight possible kinds of candy are as follows:

- Explain why the EM would not work if there were just two attributes in the model rather than three.
- Show the calculations for the first iteration of EM starting from:  
 $\theta^{(0)} = 0.6, \theta_{F1}^{(0)} = \theta_{W1}^{(0)} = \theta_{H1}^{(0)} = 0.6, \theta_{F2}^{(0)} = \theta_{W2}^{(0)} = \theta_{H2}^{(0)} = 0.4$
- What happens if we start with all the parameters set to the same value  $p$ ? (Hint: you may find it helpful to investigate this empirically before deriving the general result.)
- Write out an expression for the log likelihood of the sampled candy data (in the table) in terms of the parameters, calculate the partial derivatives with respect to each parameter, and investigate the nature of the fixed point reached in the previous question (previous bullet).