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, you 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.

1. (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. (2 points) Is it possible to produce some set of correct training examples that will get the algorihtm 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.

3. (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.

4. (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

Solution 1

1. Predict PLAYTENNIS from TEMPERATURE,HUMIDITY,WIND,OUTLOOK.

   The initial entropy of the training sample:

$$E(S) = -(\frac{5}{14}log_2\frac{5}{14} + \frac{9}{14}log_2\frac{9}{14}) = 0.9403$$

   We will choose the variable to split on such that the corresponding information gain is maximal.

$$InfoGain = E(S) - \sum_{vals} \frac{\mid S_v \mid}{\mid S \mid} E(S_v)$$

$$InfoGain(S,T) = 0.9403 - \frac{4}{14} - \frac{6}{14}(-(\frac{2}{6}log_2\frac{2}{6} + \frac{4}{6}log_2\frac{4}{6})) - \frac{4}{14}(-(\frac{1}{4}log_2\frac{1}{4} + \frac{3}{4}log_2\frac{3}{4})) = 0.0292$$

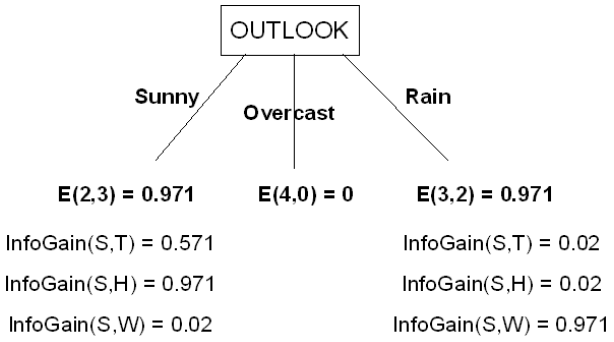
$$InfoGain(S,H) = 0.9403 - \frac{7}{14}(-(\frac{4}{7}log_2\frac{4}{7} + \frac{3}{7}log_2\frac{3}{7})) - \frac{7}{14}(-(\frac{6}{7}log_2\frac{6}{7} + \frac{1}{7}log_2\frac{1}{7})) = 0.1518$$

$$InfoGain(S,W) = 0.9403 - \frac{8}{14}(-(\frac{2}{8}log_2\frac{2}{8} + \frac{6}{8}log_2\frac{6}{8})) - \frac{6}{14} = 0.0481$$

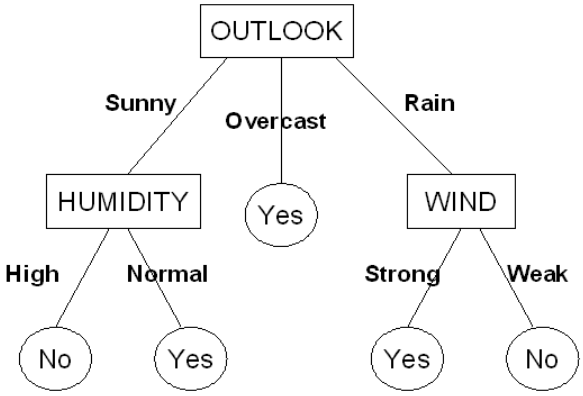
$$InfoGain(S,W) = 0.9403 - \frac{5}{14}(-(\frac{3}{5}log_2\frac{3}{5} + \frac{2}{5}log_2\frac{2}{5})) - \frac{5}{14}(-(\frac{3}{5}log_2\frac{3}{5} + \frac{2}{5}log_2\frac{2}{5}) - \frac{4}{14}0) = 0.2468$$

The first arrtribute to split on is therefore: OUTLOOK.

Next, we choose an attribute to split on in every leaf of the tree:



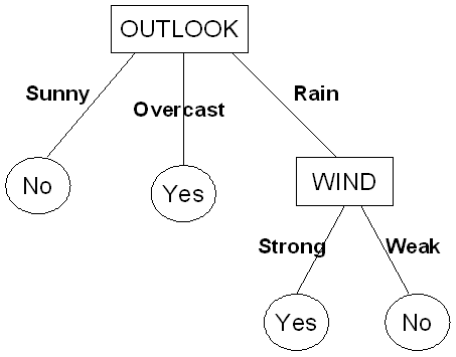
The fully developed tree is:



This tree assigns the correct class to all of the training examples.

2. Yes, it is possible. For example, consider the subset of examples includes {D1,D2,D4,D10,D11,D12}. For this subset, the it Temperature attribute has entropy of 0 (that is, it perfectly predicts *PlayTennis*). The corresponding tree would include *Temperature* only, as a single node.

3. The corresponding tree is:



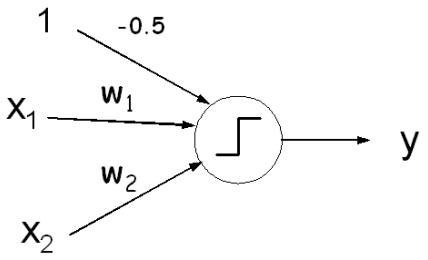
The test set accuracy is $\frac{5}{7}$. In this case, the tree built was too general. In particular, the training set did not include examples where the sunny attribute was positive. Therefore, the model failed to classify such instances in the test set.

In practice, however, the common problem is over-fitting, where there are many training examples and attributes such that fitting the training data perfectly (or as nearly perfectly as possible) leads to an overly specific model, and lower performance on unseen examples.

4. Possible strategies are to add contraints on the minimal number of training examples in a terminal/leaf node (since a leaf that includes only 1 example is likely to be over specific). Or, contrain the tree to include only up to k pre-defined number of levels, since a simple tree is likely to be more general. etc.

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 if 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).

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

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:

- Calculate the net's output Y_i.
- Multiply the error $(Z_i - Y_i)$ by the learning rate η. 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

1. (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.

2. (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).

Solution 2

1. The final weights are $W_1 = 0.7, W_2 = 0.7$:

2. The cost function to be minimized is

   $$Cost = \frac{1}{2} \sum_i E_i^2$$

   $$Cost = \frac{1}{2} \sum_i (Z_i - (-0.5 + W_1 X_{i1} + W_2 X_{i2}))^2$$

   The gradient of the cost function with repspect to the parameters W1,W2:

   $$\frac{dE}{dW_1} = (Z_i - (-0.5 + W_1 X_{i1} + W_2 X_{i2}))(-X_{i1}) = -E_i X_{i1}$$

   $$\frac{dE}{dW_2} = (Z_i - (-0.5 + W_1 X_{i1} + W_2 X_{i2}))(-X_{i2}) = -E_i X_{i2}$$

   Plugging this into the gradient update rule, we get:

   $$W_1 = W_1 + \eta E_i X_{i1}$$

   (and same for W_2)

X_1	X_2	W_1	W_2	Z	Y	Error	W_1	W_2
0	0	0.1	0.3	0	(-0.5) 0	0	0.1	0.3
0	1	0.1	0.3	1	(-0.2) 0	1	0.1	0.5
1	0	0.1	0.5	1	(-0.4) 0	1	0.3	0.5
1	1	0.3	0.5	1	(0.3) 1	0	0.3	0.5
0	0	0.3	0.5	0	(-0.5) 0	0	0.3	0.5
0	1	0.3	0.5	1	(0) 0	1	0.3	0.7
1	0	0.3	0.7	1	(-0.2) 0	1	0.5	0.7
1	1	0.5	0.7	1	(0.7) 1	0	0.5	0.7
0	0	0.5	0.7	0	(-0.5) 0	0	0.5	0.7
0	1	0.5	0.7	1	(0.2) 1	0	0.5	0.7
1	0	0.5	0.7	1	(0) 0	1	0.7	0.7
1	1	0.7	0.7	1	(0.9) 1	0	0.7	0.7
0	0	0.7	0.7	0	(-0.5) 0	0	0.7	0.7
0	1	0.7	0.7	1	(0.2) 1	0	0.7	0.7
1	0	0.7	0.7	0	(0.2) 0	0	0.7	0.7
1	1	0.7	0.7	1	(0.9) 1	0	0.7	0.7

3. In this case, where the examples are not separable, the weights will oscillate rather than converge.

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.

1. 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:

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

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

   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, $w_1, w_2..., w_n$ (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:

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

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

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, $Count(\text{“}cheap''\text{)} = 200$,$Count(\text{“}now''\text{)} = 50$, etc.; and $Count(allwords) = 10000$. Then, $Pr(\text{“}cheap''|SPAM) = \frac{200}{10000} = 0.02$.

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 potrions 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:**

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

   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. Androutsopoulos 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.

Solution 3

1. NB implementation:

   - Training and test accuracy, using all of the training examples: 99.50% (training), and 98.51% (test). The training and test accuracies using random subsets of the training data gives varying results, depending on the data split. However, on average, performance on both the train and test sets improves using more data for training. The training accuracy is usually higher than the test set accuracy, due to some over fitting.

     The given dataset yields very good results. The main reason for that is that the target classes are very well separated. In particular, the EMAIL documents are all formal email messages from the domain of linguistics. Therefore, even a small number of training documents is sufficient to train an effective classifier.

     Note:

     Some students provided accuracy results that count the training and test sets, together. This is a misconception of the experimental design. Since the model is trained using the training data only, the test set should be kept separate, providing an unbiased evaluation of the model performance on unseen data. The training accuracy gives a statistics of how well the model fits the training data. When the two numbers are compared, it can give an indication of over fitting. Evaluating both the train and test accuracies, mixed, is simply not informative.

   - Cross validation:

     The following table describes the results of 10-fold cross validation.

Segment	test size	correct predictions	example accuracies
1	N_1	C_1	92.86
2	N_2	C_2	98.25
3	N_3	C_3	100.00
4	N_4	C_4	100.00
5	N_5	C_5	96.72
6	N_6	C_6	96.72
7	N_7	C_7	98.36
8	N_8	C_8	96.72
9	N_9	C_9	98.36
10	N_10	C_10	96.72

Note that the training data should be split such that the union of the 10 segements gives exactly the training set – that is, every instance of the training set should be in included in exactly one CV segment. Then, the overall CV accuracy is calculated as:

$$Acc^{CV} = \frac{\sum C_i}{\sum N_i}$$

The individual accuracies per each segment may vary due to randomness in instance distribution in every segment. However, the overall CV accuracy should give a realtively good approaximation of model performance. The extent to which this evaluation indeed predicts performance on unseen data depends on how similar the test data distribution is to the train data distribution (if they are similar,

then CV peroformance would be a good measure. Otherwise, performance on the test data may be lower).

Note that 10-fold CV is perferable to 3-fold CV, since it allows to use 90% of the training data in building a model (rather than only 67%). Since 100% of the data is available to us, and model quailty is dependant on the amount of the data used for training, then we are interested to use as much training data as possible to get a realistic evaluation. (This was not asked in the homework, however some of you suggested that, since they were worried about the credibility of accuracy for the relatively small test segments; as explained above, the individual accuracies per segments are less interesting. The variability cancels out when the instances are summed up).

2. KNN implementation:

   K=1: 89.05%

   K=3: 85.57%

   K=5: 85.07%

   K=19: 76.62%

   Some students got better accuracies (at the range of 95% accuracy, where performance usually picked at K=3). These high numbers were due to some pre-processing of the text. For example, elimination of punctunation tokens. You were not asked to perform such filtering. Either way, both types of implementations were acceptable.

3. The top words, as defined, are those words that discriminate best between the two classes of EMAIL and SPAM. The complete definition of Information Gain for feature (word) selection is as follows:

$$IG = -\sum_{C_i} Pr(C_i)log(Pr(C_i)) + Pr(w)\sum_{C_i} Pr(C_i|w)log(Pr(C_i|w)) + Pr(\overline{w})\sum_{C_i} Pr(C_i|\overline{w})log(Pr(C_i|\overline{w}))$$

   where $C_i \in \{EMAIL, SPAM\}$, and w is a candidate word.

   We did not require that you use this formula. Other variations were acceptable. However, whatever measure you came up with, it is clear that the most discriminative words are those that appear only in one of the classes (either EMAIL or SPAM). Such words have entropy zero.

   Following is the list of words that have entropy 0 (that is, appear only in SPAM or only in EMAIL), which appear in the training corpus more than 50 times, as required. Therefore, the top 10 selected words should have been any subset of this list:

   5599, 3d, abstract, abstracts, acquisition, advertise, aol, approaches, bills, bonus, capitalfm, chair, chomsky, click, cognitive, committee, comparative, computational, context, customers, deadline, discourse, evidence, ffa, floodgate, grammar, grammatical, hundreds, income institute, investment, lexical, linguist, linguistic, linguistics, linguists, marketing, millions, multi-level, offshore, papers, parallel, phonology, product, programme, remove, researchers, sales, secrets, sell, semantic, semantics, sessions, structure, submissions, syntax, texts, theoretical, theory, translation, win, workshop

   Performance of NB and KNN, using the top T words:

   There was hardly a consensus on the actual results. This is due to several reasons:

   - For KNN, once the top words have been selected, you can either include non-top-words in the document length' normalization term ($\sqrt{(x_{1A}^2 + x_{2A}^2 + ...)}$, for document A), or not. It appears that it is much better to consider only top words and completely discard the irrelevant words, also for length normalization.
   - the variation in the word selection criteria.

   A trend that was observed in applying word selection was an improvement of the KNN algorithm results. 20 or 50 words are a too restrictive set of feature for effective classification. However, considering the top 200 discriminative words, for example, forms a more effective space for the cosine similarity measure. Considering all words gave lower performance for KNN, showing that this algorithrm (or, more precisely, the cosine similarity measure that was applied) is sensitive to noise. Naive Bayes, on the other hand, proved to be relatively robust to noise.

4. Extensions

   Among the features that were suggested: document length, count of words that appear in some external dictionary, which includes "obvious" spam words. You also suggested to consider only very short or only long words, by the intuition that these words are most discriminative. This is in fact similar to the model applied, with a heuristic feature selection. This actually improved preformance (mostly for KNN). Some of you pointed out that the subject line may be more informative comparing with other email content. This can be implemented, for example, by calculating a separate score for the email content, for the email subject line, and then weighting the two into one final score (weights can be learned from data as well). You also suggested that non-spam messages are more grammatical. A simple related feature could be bigram counts (two word sequecnes) in addition to the single words (of course, this would result in a much larger feature space – where feature selection should be useful). Interestingly, a group of students replaced the majority vote mechanism of KNN with a locally trained NB model (that is, using the K nearest neighbors for a local model training). This led to performance improvements.

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 insrall 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 Truble.

Solution 4

Applying the algorithm for 2 clusters split CA into a north and south.

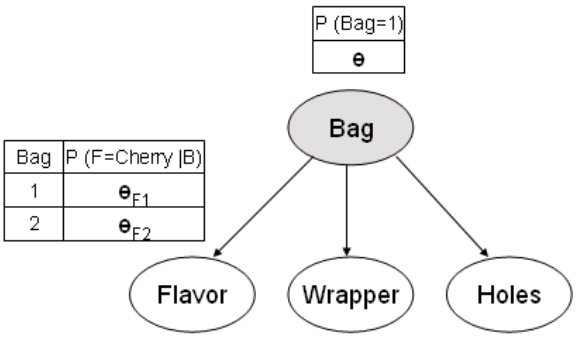
Applying the algorithm for 10 clusters clearly identified the silicon valley area as one cluster (where houses are relatively new and expensive, median income is high), the area of San Francisco as another etc. While SF and Silicon Valley may have a somewhat similar profile (at least, compared with mid CA), they were separated mainly due to the latitude/longtitue properties, that involve a dimension of geographical cohesiveness.

This question was a qualitative one, and was intended to let you experiment with a real world clustering task. Hopefully it gave you some intuition as for how the preset number of clusters affects the clustering

results. Too few clusters may not reveal the information that is in the data. Too many clusters may hurt data analysis as well, as it may not generalize (and in this sense, it is equivalent to over fitting).

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 θ is the prior probability that a candy comes from bag 1; θ_{F1} and θ_{F2} are the probabilities that the flavor is cherry, given that the candy comes from Bag 1 abd Bag 2 respectively; θ_{W1} and θ_{W2} give the probabilities that the wrapper is red; and θ_{H1} and θ_{H2} give the probabilities that the candy has a hole. In the figure, the bag is a hidden variable because, one 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:

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

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:

	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

- 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 will 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).

Solution 5

- Explain why the EM would not work if there were just two attributes in the model rather than three.

  With three attributes, there are seven parameters in the model and the empirical data give frequencies for $2^3 = 8$ classes, which supply 7 independent numbers since the 8 frequencies have to sum to the total sample size. Thus, the problem is neither under- nor over-constrained. With two attributes, there are five parameters in the model and the empirical data give frequencies for $2^2 = 4$ classes, which supply 3 independent numbers. Thus, the problem is severely underconstrained. There will be a two-dimensional surface of equally good ML solutions and the original parameters cannot be recovered.

- Show the calculations for the first iteration of EM starting from:
  $\theta^{(0)} = 0.6, \theta^{(0)}_{F1} = \theta^{(0)}_{W1} = \theta^{(0)}_{H1} = 0.6, \theta^{(0)}_{F2} = \theta^{(0)}_{W2} = \theta^{(0)}_{H2} = 0.4$

$$\theta^{(1)} = \frac{\Sigma_{j=1}^{1000} P(Bag = 1 | f_j, w_j, h_j)}{1000}$$

$$\theta^{(1)} = \frac{1}{1000} \Sigma_{j=1}^{1000} \frac{P(f_j|Bag = 1)P(w_j|Bag = 1)P(h_j|Bag = 1)P(Bag = 1)}{\Sigma_{i\in\{1,2\}} P(f_j|Bag = i)P(w_j|Bag = i)P(h_j|Bag = i)P(Bag = i)}$$

$$\theta^{(1)} = \frac{273}{1000} \frac{\theta^{(0)}_{F1}\theta^{(0)}_{W1}\theta^{(0)}_{H1}\theta^{(0)}}{\theta^{(0)}_{F1}\theta^{(0)}_{W1}\theta^{(0)}_{H1}\theta^{(0)} + \theta^{(0)}_{F2}\theta^{(0)}_{W2}\theta^{(0)}_{H2}(1-\theta^{(0)})} + \dots$$

$$\frac{93}{1000} \frac{\theta^{(0)}_{F1}\theta^{(0)}_{W1}(1-\theta^{(0)}_{H1})\theta^{(0)}}{\theta^{(0)}_{F1}\theta^{(0)}_{W1}(1-\theta^{(0)}_{H1})\theta^{(0)} + \theta^{(0)}_{F2}\theta^{(0)}_{W2}(1-\theta^{(0)}_{H2})(1-\theta^{(0)})} + \dots$$

$$\frac{104}{1000} \frac{\theta^{(0)}_{F1}(1-\theta^{(0)}_{W1})\theta^{(0)}_{H1}\theta^{(0)}}{\theta^{(0)}_{F1}(1-\theta^{(0)}_{W1})\theta^{(0)}_{H1}\theta^{(0)} + \theta^{(0)}_{F2}(1-\theta^{(0)}_{W2})\theta^{(0)}_{H2}(1-\theta^{(0)})} + \dots$$

$$\frac{90}{1000} \frac{\theta^{(0)}_{F1}(1-\theta^{(0)}_{W1})(1-\theta^{(0)}_{H1})\theta^{(0)}}{\theta^{(0)}_{F1}(1-\theta^{(0)}_{W1})(1-\theta^{(0)}_{H1})\theta^{(0)} + \theta^{(0)}_{F2}(1-\theta^{(0)}_{W2})(1-\theta^{(0)}_{H2})(1-\theta^{(0)})} + \dots$$

$$\frac{79}{1000} \frac{(1-\theta^{(0)}_{F1})\theta^{(0)}_{W1}\theta^{(0)}_{H1}\theta^{(0)}}{(1-\theta^{(0)}_{F1})\theta^{(0)}_{W1}\theta^{(0)}_{H1}\theta^{(0)} + (1-\theta^{(0)}_{F2})\theta^{(0)}_{W2}\theta^{(0)}_{H2}(1-\theta^{(0)})} + \dots$$

$$\frac{100}{1000} \frac{(1-\theta^{(0)}_{F1})\theta^{(0)}_{W1}(1-\theta^{(0)}_{H1})\theta^{(0)}}{(1-\theta^{(0)}_{F1})\theta^{(0)}_{W1}(1-\theta^{(0)}_{H1})\theta^{(0)} + (1-\theta^{(0)}_{F2})\theta^{(0)}_{W2}(1-\theta^{(0)}_{H2})(1-\theta^{(0)})} + \dots$$

$$\frac{94}{1000} \frac{(1-\theta^{(0)}_{F1})(1-\theta^{(0)}_{W1})\theta^{(0)}_{H1}\theta^{(0)}}{(1-\theta^{(0)}_{F1})(1-\theta^{(0)}_{W1})\theta^{(0)}_{H1}\theta^{(0)} + (1-\theta^{(0)}_{F2})(1-\theta^{(0)}_{W2})\theta^{(0)}_{H2}(1-\theta^{(0)})} + \dots$$

$$\frac{167}{1000} \frac{(1-\theta^{(0)}_{F1})(1-\theta^{(0)}_{W1})(1-\theta^{(0)}_{H1})\theta^{(0)}}{(1-\theta^{(0)}_{F1})(1-\theta^{(0)}_{W1})(1-\theta^{(0)}_{H1})\theta^{(0)} + (1-\theta^{(0)}_{F2})(1-\theta^{(0)}_{W2})(1-\theta^{(0)}_{H2})(1-\theta^{(0)})} + \dots$$

$$\theta^{(1)} = 0.22797 + 0.06438 + 0.072 + 0.045 + 0.05469 + 0.05 + 0.047 + 0.05138$$

$$\theta^{(1)} = 0.6124$$

The computation for the other parameters proceeds similarly according to:

$$\theta_{F1}^{(1)} = \frac{\Sigma_{j:f_j=cherry} P(Bag = 1 | f_j = cherry, w_j, h_j)}{\Sigma_j P(Bag = 1 | f_j, w_j, h_j)}$$

The solutions are given in the text.

- What happens if we start will all the parameters set to the same value p? (Hint: you may find it helpful to investigate this empirically before deriving the general result.)

  Assume $\theta = 0.5$. With every parameter identical, the new parameter values for bag 1 will be the same as those for bag 2, by symmetry. Intuitively, if we assume initially that the bags are identical, then it is as if we had just one bag. Likelihood is maximized under this constraint by setting the proportions of candy types within each bag to the observed proportions.

- 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).

$$L = 273 \log(\theta\theta_{F1}\theta_{W1}\theta_{H1} + (1-\theta)\theta_{F2}\theta_{W2}\theta_{H2})$$
$$+93 \log(\theta\theta_{F1}\theta_{W1}(1-\theta_{H1}) + (1-\theta)\theta_{F2}\theta_{W2}(1-\theta_{H2})) + \ldots$$

$$\frac{\delta L}{\delta \theta_{H1}} = 273 \frac{\theta\theta_{F1}\theta_{W1}}{\theta\theta_{F1}\theta_{W1}\theta_{H1} + (1-\theta)\theta_{F2}\theta_{W2}\theta_{H2}}$$
$$+93 \frac{\theta\theta_{F1}\theta_{W1}}{\theta\theta_{F1}\theta_{W1}(1-\theta_{H1}) + (1-\theta)\theta_{F2}\theta_{W2}(1-\theta_{H2})}$$

Now if $\theta_{F1} = \theta_{F2}, \theta_{W1} = \theta_{W2}, and \theta_{H1} = \theta_{H2}$, the denominators simplify, everything cancels, and we have

$$\frac{\delta L}{\delta \theta_{H1}} = \theta[\frac{273}{\theta_{H1}} - \frac{93}{(1-\theta_{H1})} + \ldots]$$
$$= \theta[\frac{550}{\theta_{H1}} - \frac{450}{(1-\theta_{H1})}]$$

First, note that $\frac{\delta L}{\delta \theta_{H2}}$ will have the same value except that θ and $(1-\theta)$ are reversed, so the parameters for bags 1 and 2 will move in lock step if $\theta = 0.5$. Second, note that $\frac{\delta L}{\delta \theta_{H1}} = 0$ when $\theta_{H1} = 550/(450 + 550)$, i.e., exactly the observed proportion of candies with holes. Finally, we can calculate second derivatives and evaluate them at the fixed point. For example, we obtain

$$\frac{\delta^2 L}{\delta \theta_{H1}^2} = N\theta^2\theta_{H1}(1-\theta_{H1})(2\theta_{H1} - 1)$$

which is negative (indicating the fixed point is a maximum) only when $\theta_{H1} < 0.5$. Thus, in general the fixed point is a saddle point as some of the second derivatives may be positive and some negative. Nonetheless, EM can reach it by moving along the ridge leading to it, as long as the symmetry is unbroken.