6.867 Machine Learning

Problem Set 4

Due date: Wednesday November 3

Please address all questions and comments about this problem set to `6867-staff@csail.mit.edu`. You will need to use MATLAB for some of the problems but essentially all the code is provided. If you are not familiar with MATLAB, please consult

`http://www.ai.mit.edu/courses/6.867/matlab.html`

and the links therein.

Problem 1: Feature Selection

The task and the data

In this problem you will study feature selection on the task of predicting vowels (a,e,i,o,u) in English from the preceding letters. To predict if the current letter is or is not a vowel, we look at up to eight letters before the current position. For each of the previous letters we only look at whether they themselves are vowels or not.

Because letters belonging to different words are likely to be much less correlated than letters within a word, we discard any letters before the beginning of the current word. We still keep 8 features for every letter, but we give a special "stop" value to the features before the beginning of the word. Thus each feature can take three values, with the following meaning:

- $x_i = 0$ if the letter at position $-i$ belongs to the same word and is not a vowel
- $x_i = 1$ if the letter at position $-i$ belongs to the same word and is a vowel
- $x_i = 2$ if position $-i$ is past the beginning of the word

For instance, the letter 's' in the word 'English' would be represented by the feature vector $(1, 0, 0, 0, 1, 2, 2, 2)$. The class label is 1 if the current letter is a vowel and 0 otherwise.

We have preprocessed the text of *Alice in Wonderland* by Lewis Carroll into this feature representation, and split it into the training set `p1_train.dat`, and test set `p1_test.dat`.

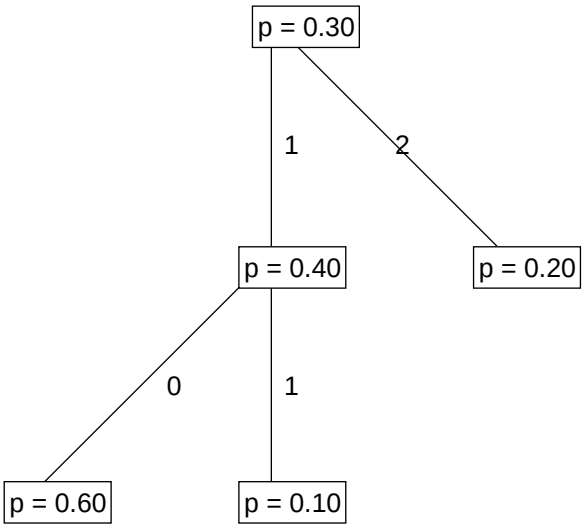

Figure 1: A probabilistic-suffix-tree classifier

There are 10789 letters for training, and 96926 for testing. Each row in each data file is of the form $y\ x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7\ x_8\ n$, where n represents the number of repeated samples with the same label and feature representation.

The classifier

A *probabilistic suffix tree* (PST) is a rooted tree whose edges match feature values such that consecutive edges on the path from the root match consecutive features. Each node in the tree is associated with a predictive distribution $P(y|x_1, x_2, \ldots, x_i)$, where $x_1, x_2, \ldots, x_i$ are the feature values on the unique path from the root of the tree to that node.

In order to classify a sample $(x_1, x_2, \ldots, x_8)$, we start from the root of the tree and match feature values (traverse edges) as far down as possible. We predict with the distribution associated with the last matched node. For example, in Figure 1

$$P(y = 1|0\ldots) = 0.3 \quad \text{(the root because no edge from the root is labeled 0)}$$
$$P(y = 1|2\ldots) = 0.2$$
$$P(y = 1|12\ldots) = 0.4$$
$$P(y = 1|10\ldots) = 0.6$$
$$P(y = 1|11\ldots) = 0.1$$

In order to train a PST we start with the root and try to extend it (select features). The predictive distribution associated with each new node (including the root), that is p, is estimated based on all the training examples that match the path from root to the node. In other words, p for the root node is simply the overall frequency of training samples labeled 1. Similarly, the leftmost child of the root in the figure considers only samples

(letters) that have a preceding vowel, and so on. The predictive distributions associated with the nodes already in the tree do not change as we add new nodes.

The feature selection question consists of deciding which labeled edge to add to which of the possible available nodes.

Feature selection on PST's

The principle for feature selection demonstrated here is to choose the feature representation that has the most impact on the uncertainty in the class label. The classical measure of uncertainty in a random variable is its *entropy* (shown here for the binary class label):

$$H(Y) = \sum_y P(y) \log \frac{1}{P(y)} = -P(y=1) \log P(y=1) - P(y=0) \log P(y=0) \tag{1}$$

Suppose now that we need to predict the class label y given the feature representation $\mathbf{x}$. The knowledge of $\mathbf{x}$ should reduce the uncertainty in the class label. To measure the uncertainty after observing $\mathbf{x}$, we compute the average entropy of Y given $\mathbf{X} = \mathbf{x}$ for all possible values of $\mathbf{x}$:

$$H(Y|\mathbf{X}) = \sum_{\mathbf{x}} P(\mathbf{x}) H(Y|\mathbf{X} = \mathbf{x}) = -\sum_{\mathbf{x},y} P(\mathbf{x}, y) \log P(y|\mathbf{x}) \tag{2}$$

In terms of a PST with K nodes, let $P_i(Y)$ be the label distribution associated with node i, and N_i the number of training samples that fall under that node (and not under any of its children). Let N be the total number of training samples. Then the estimate of the conditional entropy of Y is:

$$H(Y|\mathbf{X}) = \sum_{i=1}^{K} \frac{N_i}{N} H_i(Y) \tag{3}$$

In other words, since each $\mathbf{x}$ is associated with the node we would use to predict the label for it, the overall conditional entropy is simply the sum of the entropies of these predictive distributions weighted by the fraction of training samples that use the nodes.

If we add a leaf $K+1$ to node j, some of the training samples that were classified by node j will now be classified instead by leaf $K+1$. Let N_{K+1} be their number and $P_{K+1}(Y)$ the label distribution associated with the new node based on these N_{K+1} samples. Then the conditional entropy after adding node $K+1$ is given by:

$$H^{\text{new}}(Y|\mathbf{X}) = \left(\sum_{i=1, i \neq j}^{K} \frac{N_i}{N} H_i(Y) \right) + \frac{N_j - N_{K+1}}{N} H_j(Y) + \frac{N_{K+1}}{N} H_{K+1}(Y) \tag{4}$$

Thus the effect of adding leaf $K+1$ is the reduction of uncertainty in Y by the following amount:

$$H(Y|\mathbf{X}) - H^{\text{new}}(Y|\mathbf{X}) = \frac{N_{K+1}}{N} (H_j(Y) - H_{K+1}(Y)) \tag{5}$$

To grow the tree, we consider all possible leafs and greedily select the leaf that maximizes the reduction of conditional entropy (*information gain*).

Matlab files

We have provided the following Matlab files

`new_feature.m` Can be used to create a PST with only the root. The syntax is

```
pst = {new_feature(p1_train, [ ])}
```

`add_feature.m` Extend a PST with a new leaf chosen by the information criterion

```
pst = add_feature(p1_train, pst)
```

`eval_pst_probs.m` Evaluate the probability of class 1 on a test set. The function returns a probability for each point.

`plot_pst.m` Displays a PST.

Questions

1. *(10 points)* Suppose that in a training set we get the following training samples:

y	x	count
0	0	0
0	1	10
0	2	5
1	0	1
1	1	2
1	2	5

   Compute $H(Y)$, $H(Y|x=0)$, $H(Y|x=1)$, $H(Y|x=2)$, and $H(Y|X)$. Also, compute the reduction in conditional entropy if we add the edge 0, 1, or 2 to the root of an empy PST trained on this set. What is the first edge that should be added to the PST?

2. *(10 points)* Train a PST of 10 nodes (including the root) on `p1_train` and hand in the plot of the PST. Mark on the plot beside each node the order in which it was chosen, and the error rate on the test set `p1_test`. In computing the error rate note that you must weight the errors by their count `p1_test(:,end)`.

3. *(10 points)* Out of the constructed PST tree select two paths that assign the lowest and the highest probability to $y = 1$. Express the decision rules represented by the two paths in plain English (in terms of vowels, consonants, and beginning of words). You only need to turn in the two sentences, and an explanation of whether they make sense.

4. *(5 points)* Can the feature selection method extend a node that tests for "feature = 2"? Explain why or why not.

Problem 2: boosting

We consider here a simple AdaBoost algorithm and its properties. Suppose we have already added $m-1$ weak classifiers (run $m-1$ boosting iterations) so that the combined discriminant function is given by

$$h_{m-1}(\mathbf{x}) = \hat{\alpha}_1 h(\mathbf{x}; \hat{\theta}_1) + \hat{\alpha}_{m-1} h(\mathbf{x}; \hat{\theta}_{m-1}) \tag{6}$$

We assume here that the weak component classifiers (e.g., decision stumps) $h(\mathbf{x}; \theta)$ return ± 1 labels and that the votes are non-negative. The next weak classifier to be added is trained to optimize (minimize) the weighted classification error

$$\epsilon_m = \frac{1}{2} - \frac{1}{2} \sum_{i=1}^{n} \tilde{W}_i^{(m-1)} y_i h(\mathbf{x}; \theta_m) \tag{7}$$

where $\tilde{W}_i^{(m-1)}$ is the weight on the i^{th} training example after $m-1$ boosting iterations. The weights are normalized to sum to one.

Let $\hat{\theta}_m$ denote the parameters of the new weak classifier, $\hat{\epsilon}_m$ the weighted error that it achieves, and

$$\tilde{W}_+^{(m-1)} = \sum_{i:\, y_i = h(\mathbf{x}_i; \hat{\theta}_m)} \tilde{W}_i^{(m-1)} \tag{8}$$

$$\tilde{W}_-^{(m-1)} = \sum_{i:\, y_i \neq h(\mathbf{x}_i; \hat{\theta}_m)} \tilde{W}_i^{(m-1)} \tag{9}$$

the total weight placed on examples that $h(\mathbf{x}; \hat{\theta}_m)$ classifies correctly ("+") and incorrectly ("-"), respectively.

The votes α_m associated with the new weak classifier are obtained by minimizing

$$Z_m(\alpha_m) = \sum_{i=1}^{n} \tilde{W}_i^{(m-1)} \exp\big(- y_i \alpha_m h(\mathbf{x}_i; \hat{\theta}_m)\big) \tag{10}$$

$$= \tilde{W}_+^{(m-1)} \exp(-\alpha_m) + \tilde{W}_-^{(m-1)} \exp(\alpha_m) \tag{11}$$

with respect to α_m. We have used here the fact that $y_i h(\mathbf{x}_i; \hat{\theta}_m) = 1$ when the example is correctly classified and -1 otherwise. Let $\hat{\alpha}_m$ be the resulting minimizing value.

The weights on the training examples are finally updated according to

$$\tilde{W}_i^{(m)} = \frac{1}{Z_m(\hat{\alpha}_m)} \tilde{W}_i^{(m-1)} \exp\big(- y_i \hat{\alpha}_m h(\mathbf{x}_i; \hat{\theta}_m)\big) \tag{12}$$

where $\tilde{W}_i^{(0)} = 1/n$. Note that $Z_m(\hat{\alpha}_m)$ obtained earlier is the normalization constant for the weight update.

Each boosting iteration performed in this manner is guaranteed to decrease the average exponential loss. In other words,

$$L(h_m) = \frac{1}{n}\sum_{i=1}^{n} \exp\big(-y_i h_m(\mathbf{x}_i)\big) \tag{13}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \exp\big(-y_i h_{m-1}(\mathbf{x}_i)\big) \exp\big(-y_i \hat{\alpha}_m h(\mathbf{x}_i;\hat{\theta}_m)\big) \tag{14}$$

should be lower than

$$L(h_{m-1}) = \frac{1}{n}\sum_{i=1}^{n} \exp\big(-y_i h_{m-1}(\mathbf{x}_i)\big) \tag{15}$$

We can see this, for example, as follows

$$L(h_m) = L(h_{m-1}) \cdot \overbrace{\frac{1}{L(h_{m-1})}\frac{1}{n}\sum_{i=1}^{n} \exp\big(-y_i h_{m-1}(\mathbf{x}_i)\big)}^{\tilde{W}_i^{(m-1)}} \exp\big(-y_i \hat{\alpha}_m h(\mathbf{x}_i;\hat{\theta}_m)\big) \tag{16}$$

$$= L(h_{m-1}) \sum_{i=1}^{n} \tilde{W}_i^{(m-1)} \exp\big(-y_i \hat{\alpha}_m h(\mathbf{x}_i;\hat{\theta}_m)\big) \tag{17}$$

$$= L(h_{m-1}) Z_m(\hat{\alpha}_m) \tag{18}$$

where $Z_m(\hat{\alpha}_m) < 1$ since $Z_m(0) = 1$ and $\hat{\alpha}_m$ is chosen to minimize $Z_m(\alpha_m)$.

1. *(5 points)* Express the weighted error $\hat{\epsilon}_m$ as a function of $\tilde{W}_+^{(m-1)}$ and $\tilde{W}_-^{(m-1)}$.

2. *(5 points)* Find the minimizing value $Z_m(\hat{\alpha}_m)$ as a function of $\tilde{W}_+^{(m-1)}$ and $\tilde{W}_-^{(m-1)}$. Also express the result in terms of $\hat{\epsilon}_m$.

3. *(5 points)* Show that the classification error of $h_m(\mathbf{x})$ on the training set, denoted here as $\hat{\text{err}}(h_m)$, is bounded by

$$\hat{\text{err}}(h_m) \leq \prod_{k=1}^{m} 2\sqrt{\hat{\epsilon}_k(1-\hat{\epsilon}_k)} \tag{19}$$

   You can assume here that $L(h_m)$, as discussed in the lecture, serves as an upper bound on $\hat{\text{err}}(h_m)$.

4. Adaboost is particularly easy to implement. We have, however, provided you with almost all the necessary MATLAB code.

(a) *(5 points)* Please complete our implementation of Adaboost by substituting in how the votes α are set for each new decision stump. You only need to modify `boost_ada.m`.

(b) *(10 points)* `cancer.mat` contains training and test data for a simple leukemia classification problem. The two dimensional inputs correspond to normalized gene expression levels for two different genes measured across different tissues. Use `call_boosting` to train the boosting classifier and plot the number of test errors as a function of the number of boosting iterations. Modify the routine to also plot the minimum *voting margin*, minimum taken over the training examples, as a function of the boosting iterations. Note that `eval_boost.m` can also return the total number of votes. What can you say about the plots?

(c) *(5 points)* Since the data set is two dimensional, it is easy to visualise the decision regions of the resulting boosting classifier. Use `plot_decision.m` to plot the decision regions of the trained AdaBoost classifier after 50 decision stumps.

5. *(5 points)* Suppose we have collected a finite number of component classifiers from successive boosting iterations, and use the binary ± 1 outputs of these component classifiers as features. Thus, with m component classifiers, we have a new $m-$dimensional feature space. We define a linear classifier without a bias term as follows:

$$f(\mathbf{x}; \mathbf{w}) = w_1 h(\mathbf{x}; \hat{\theta}_1) + \ldots + w_m h(\mathbf{x}; \hat{\theta}_m) \tag{20}$$

where $\hat{\theta}_1, \ldots, \hat{\theta}_m$ are fixed by boosting. We train this linear classifier by also minimizing the average exponential loss on the training examples. Is the resulting linear classifier equivalent to the one returned by AdaBoost? Justify your answer.

Problem 3: VC-dimension

In this problem, we will investigate the VC-dimension of sets of classifiers. A *classifier* here is a function from some input space to the binary class labels $+1, -1$. A classifier can also be described as a subset of the input space which gets the label $+1$. For example, a linear classifier in the plane $\Re^2$, can be described by a half-plane. For this reason, we can discuss the family of linear classifiers as the set of all half-planes (and possibly also the plane itself and the empty set).

We say that a class (i.e. set) $\mathcal{H}$ of classifiers *shatters* a set of points $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ if we can classify the points in X in all possible ways. More precisely, *for all* 2^n possible labeling vectors $y_1, y_2, \ldots, y_n \in \{-1, 1\}^n$, there exists a classifier $h \in \mathcal{H}$ such that $h(\mathbf{x}_i) = y_i$ for all i. For any possible labelings of the points, there has to be a classifier in our set that reproduces those labels. Using the set notation for classifiers, this means that for any subset of examples $X' \subseteq X$ (indicating the subset of points labeled $+1$), there exist a classifier $h \in \mathcal{H}$ such that $X \cap h = X'$ (the set of points for which h assigns label $+1$ includes X' but

not the rest of X). It is important to understand that shattering is a property of a set of classifiers– not of a single classifier. A single classifier cannot shatter even a single point.

The *VC-dimension* of a set $\mathcal{H}$ of classifiers is the size of the largest set of points X that can be shattered by $\mathcal{H}$.

1. *(10 points)* Let C be the set of all possible circles in $\Re^2$. Consider a set of classifiers $\mathcal{H}$ obtained by mapping a circle $c \in C$ to a classifier $h_c \in \mathcal{H}$ such that $h_c(\mathbf{x})$ returns $+1$ when $\mathbf{x}$ lies in the interior of c, and -1 otherwise. What is the VC dimension of $\mathcal{H}$? Justify your answer.

2. Decision stumps, by themselves, are not very powerful classifiers. For instance, a single vertical (or horizontal) decision stump can only shatter 2 points in $\Re^2$. However, combining multiple decision stumps can give rise to more complex classifiers. In this part, we calculate the VC dimensions of some combinations of decision stumps.

   For points in $\Re^2$, calculate the VC-dimension of the following two sets of classifiers:

   (a) *(10 points)* Convex combinations of two vertical decision stumps.

   (b) *(10 points)* Convex combinations of one vertical and one horizontal decision stump.

   As always, explain how you arrived at your answer.