CS 536: Homework 2

Solutions

1 Decision Lists (10 pts)

Show that the number of decision lists over n Boolean variables is at most $n!8^n$. Let Y be the $0/1$ output of a decision list over n Boolean variables $X_1, X_2, \ldots X_n$. Show that Y can also be written as a linear classifier over $X_1, X_2, \ldots X_n$, i.e., Y can be written as $sgn(b + \sum_i w_i X_i)$.

Solution: Let's count the number of decision lists of a particular length l. There are $\binom{n}{l}$ ways to pick the l variables in the list and $l!$ ways to arrange them in a linear order. For each variable x_i in the list, there are 2 ways to choose the value that will be tested,, i.e. $X_i = 0$ or $X_i = 1$ and 2 ways to choose the output value at that node. Hence the number of lists of length l is $\leq \binom{n}{l} l! 4^l$. The total number of decision lists $\leq \sum_{l=0}^{n} \binom{n}{l} l! 4^l \leq n! 5^n$.

Note: We will give you points if you argued that the number of lists is at most $n!4^n$.

Given a length l decision list with variables in the order $X_1, X_2, \ldots X_l$, assume that the check made at step i is $X_i = a_i$ and the output at step i is y_i. Notice that $a_i \in \{0, 1\}$ and $y_i \in \{-1, 1\}$. Then the function computed by the list can be written as $sgn(\sum_{i=1}^{l} 2^{l-i+1} y_i (X_i)^{a_i} (1 - X_i)^{(1-a_i)} + y_{l+1})$. Since $a_i \in \{0, 1\}$, each term in the summation is a linear function of X's. This can be rearranged and written as $sgn(b + \sum_i w_i X_i)$.

k-decision lists:
The decision lists that we saw in the class test the value of one variable at each step. In a k-decision list is one can test the value of any function of up to k variables in each step. So, the decision lists from the lecture are 1-decision lists. Show that if $Y = f(X_1, X_2, \ldots X_n)$ where $f()$ is a k-decision list, then given training data, one can find a k-decision list of zero training error in time $O(n^k)$. How much training data would one need to guarantee the true error[1] of the list you found will likely be at most ϵ?

Solution: One solution is to map the data into a new high dimensional feature space. For every subset S of up to k variables, say $X_1, X_2, \ldots X_k$ and for every assignment $\vec{a} = a_1, a_2, \ldots a_k$ to these variables, add a corresponding feature $X_{S,\vec{a}}$ in the new space. $X_{S,\vec{a}}$ is a 0/1 valued feature that takes on value 1 if the variables in the set S have the assignment $\vec{a}$. Now the problem reduces to finding a 1-decision list in this new feature space and we can use the algorithm for 1-decision lists. The total number of features is at most $k\binom{n}{k}2^k \leq O(n^k 2^k)$. The runtime of the algorithm will be polynomial in m and the number of features, i.e., $O(n^k 2^k)$. The sample complexity will be $m \geq \frac{1}{\epsilon^2} \log(|H|) = \Omega(\frac{1}{\epsilon^2} n^k 2^k \log(n))$.

2 Perceptron and Margin (20 pts)

Design a training set consisting of $O(n)$ points in $\Re^n$ that is linearly separable but on which the perceptron algorithm will take $\exp(\Omega(n)))$ steps to converge. **Note:** the Euclidean length of every point in your construction should be at most polynomial in n.

[Hint: Consider using a subset of $\{0, 1\}^n$]

[1] As in classification, the error here is 0/1 loss, i.e., the probability that a new example drawn from the same distribution as the training set will be labeled incorrectly.

Solution: The idea is to construct a training instance that consists of labeled points in $\{0,1\}^n$ such that any linear classifier that separates these points has to use exponentially large weights. Since running the perceptron on a subset of $\{0,1\}^n$ can update a weight value by at most 1 at each step, this would imply that the Perceptron algorithm will take an exponential amount of time to converge. Here is one such dataset:

Assume that n is even. A similar construction can be obtained for odd values. The dataset consists of n points $X_1, X_2, \ldots X_n$. For $i = 1, 2, \ldots n/2$, point X_{2i} is labeled positive has $1's$ in the following coordinates: $2i, 2i-1, 2i-3, 2i-5, \ldots 1$. Similarly, point X_{2i-1} is labeled negative has $1's$ in the following coordinates: $2i-1, 2i-2, 2i-4, 2i-6, \ldots 2$. Any weight vector $w = (w_1, w_2, \ldots w_n)$ that classifies these points correctly will have to satisfy $y_i(w \cdot X_i) > 0$ for all the points. This gives the following inequalities

$$|w_{2i}| > |w_{2i+1}| + |w_{2i+3}| + |w_{2i+5}| + |w_1|, \ \forall i = 1, \ldots n/2$$

and

$$|w_{2i-1}| > |w_{2i}| + |w_{2i+2}| + |w_{2i+4}| + |w_2|, \ \forall i = 1, \ldots n/2$$

It is easy to check that any w satisfying the above inequalities will eventually use exponentially large weight values.

3 SVMs with no label (20 pts)

You are given a set of m points of unit length in $\Re^d$ but with no labels. However, you are given the promise that there exists a non-trivial[2] way to label them as $+/-$ such that they become separable with margin $\frac{1}{4}$. Give an efficient algorithm to find such a labeling. Your algorithm should run in time polynomial in m and d.

Solution: Let $X_1, X_2, \ldots, X_m$ be the dataset. We know that the dataset is separable with margin $\gamma = \frac{1}{4}$. Hence, there exists a labeling $L^* = (y_1, y_2, \ldots y_m)$ of the data points such that when we run Perceptron on $(X_1, y_1), (X_2, y_2), \ldots (X_m, y_m)$, it will stop in at most $\frac{1}{\gamma^2} = 16$ steps. The hypothesis output by perceptron is a linear combination of training examples, i.e. it is of the form $w = \sum_i y_i X_i$. Hence, there exists a linear combination of at most 16 training examples, such that the linear classifier given by that combination induces the labeling L^*. This suggests the following algorithm

- Enumerate all sets of up to 16 examples in the training set.
- For each set, say $X_1, X_2, \ldots X_{16}$ enumerate all 2^{16} labelings of these points.
- For each set $X_1, X_2, \ldots X_{16}$ and labeling $y_1, y_2, \ldots y_{16}$, use the classifier $w = \sum_{i=1}^{16} y_i X_i$ to label the entire set of points.
- Run SVM on the entire labeled set to check if the margin is $\frac{1}{4}$. If yes, output the labeled data set.

The runtime of the algorithm is at most $O(m^{16} 2^{16} T(m, d))$ where $T(m, d)$ is the time to run SVM on the data and is also polynomial in m, d.

4 Kernels (10 pts)

Recall that $K : X \times X \mapsto \Re$ is a legal kernel if there exists an implicit function ϕ such that $K(x, y) = \phi(x) \cdot \phi(y)$. (Here, X is our space of examples.) Often the easiest way to prove that some function is a legal kernel is to build it out of other legal kernels. In particular, suppose $K(x, y) = \phi(x) \cdot \phi(y)$ and $K'(x, y) = \phi'(x) \cdot \phi'(y)$ where $\phi : X \mapsto \Re^N$ and $\phi' : X \mapsto \Re^{N'}$ for some N, N' . (Let's not worry about infinite-dimensional implicit feature spaces.)

1. Show that for any constant $c \geq 0$, cK is a legal kernel.
   Solution: Let $\phi : X \mapsto \Re^N$ be the mapping for the kernel K. Then the mapping $\sqrt{c}\phi$ is a valid feature mapping for cK, i.e. $cK(x, x') = \sqrt{c}\phi(x) \cdot \sqrt{c}\phi(x')$.

[2] A trivial labeling is one that assigns all of them as + or -.

2. Show that the sum, $K + K'$, is a legal kernel.
   Solution: Let ϕ and ϕ' be the mappings for K and K', then the mapping $\phi''(x) = (\phi(x), \phi'(x))$ is a valid mapping for $K + K'$.
3. Show that the product, KK', is a legal kernel.
   Solution: Let ϕ, ϕ' be the feature maps for K and K'. Let $\phi(x) = (\phi_1(x), \phi_2(x), \ldots \phi_N(x))$. Consider the map $\phi''(x) = ((\phi_1(x)\phi'(x), \phi_2(x)\phi'(x), \ldots \phi_N(x)\phi'(x)))$. This is a valid map for $K(x, x')K'(x, x')$.

5 Decision trees and Decision lists (10pts)

Decision tree rank: The rank of a decision tree is defined as follows. If the tree is a single leaf then the rank is 0. Otherwise, let r_L and r_R be the ranks of the left and right subtrees of the root, respectively. If $r_L = r_R$ then the rank of the tree is $r_L + 1$. Otherwise, the rank of the tree is the maximum of r_L and r_R.

1. Prove that a decision tree with ℓ leaves has rank at most $\log_2 \ell$.
   Solution: Proof is by induction on ℓ. Base case with $\ell = 1$ is trivial. Now assume that the claim is true for all $\ell <= t - 1$. Consider a tree with t leaves. Let r_L and r_R be the two subtrees of the root. Both of them have strictly less than t leaves and hence have rank at most $\log_2(t - 1)$. Also, one of them, say r_L will have $\leq \frac{t}{2}$ leaves and hence will have $rank(r_L) \leq \log_2(\frac{t}{2})$. We have two cases: i) $r_L = r_R$. In this case $rank(r) = rank(r_L) + 1 \leq \log_2(\frac{t}{2}) + 1 = \log_2(t)$. ii) $r_L \neq r_R$. In this case $rank(r) = max(rank(r_L), rank(r_L)) \leq \log_2(t)$.
2. Show that if one can fit a training data using a decision tree with ℓ leaves, then one can also fit the data using a k-decision list for $k = \log_2 \ell$. You can assume binary features.
   Solution: If a decision tree has rank ℓ, then to decide the label of any example, one has to check at most $\log_2 \ell$ variables. Hence, one can write a $\log_2 \ell$-decision list where each entry in the list checks if the example falls into one of the leaves of the tree.

6 More Kernels (10 pts)

1. Show that if $K : X \times X \mapsto \Re$ is a legal kernel then so is K^d for any $d \in \mathbb{N}$. Here, K^d refers to the Kernel that produces dot products as $K^d(x, x')$.
   Solution: From problem 4 we know that the product of any two legal kernels is a legal kernel. This implies that K^d is a legal kernel.
2. Let $\hat{X}$ denote the set of all finite subsets of X. Prove that if K is a valid kernel on $X \times X$ then
   $$\hat{k}(A, B) = \sum_{x \in A, x' \in B} k(x, x')$$
   is a valid kernel on $\hat{X} \times \hat{X}$.
   Solution: Let $\phi : X \mapsto \Re^N$ be the feature map for K. Define $\phi' : \hat{X} \mapsto \Re^N$ as $\phi'(A) = \sum_{x \in A} \phi(x)$. Here the sum is coordinate wise. Then $\hat{k}(A, B) = \sum_{x \in A, x' \in B} k(x, x') = \sum_{x \in A, x' \in B} \phi(x) \cdot \phi(x') = \sum_{x \in A} \phi(x) \cdot (\sum_{x' \in B} \phi(x')) = \phi'(A) \cdot \phi'(B)$.
3. Prove that if $\sigma : X \mapsto X$ is a function and $K(x, x')$ is a valid kernel, then so is $K(\sigma(x), \sigma(x'))$.
   Solutions: Let ϕ be the mapping for K. Define $\phi'(x) = \phi(\sigma(x))$. Then $K(\sigma(x), \sigma'(x)) = \phi(\sigma(x)) \cdot \phi(\sigma(x')) = \phi'(x) \cdot \phi'(x')$.
4. Given a kernel K with feature map Φ, construct a corresponding normalized kernel K' by normalizing the feature map Φ such that all points have unit length in the new space. Give an expression for computing dot products in the normalized feature space.
   Solution: The new feature mapping will be $\Phi'(x) = \frac{\Phi(x)}{\|\Phi(x)\|} = \frac{\Phi(x)}{\sqrt{\Phi(x) \cdot \Phi(x)}}$. The new kernel will be $K'(x, x') = \frac{K(x,x')}{\sqrt{K(x,x)}\sqrt{K(x',x')}}$.

5. Given an example of a kernel with two valid feature maps Φ_1 and Φ_2, mapping into spaces of different dimensions.

   Solution: Let $X \subset \Re^d$ consist of $v_1, v_2, \ldots v_d$ where the v_i's are orthogonal. Consider the kernel K defined as follows: $K(v_i, v_j) = v_{i,1}v_{j,1}$, if $v_{i,1}$ and $v_{j,1}$ have the same sign, else $K(v_i, v_j) = 0$. One feature map for this kernel is $\phi : X \mapsto \Re^2$, such that $\phi(v_i) = (\frac{v_{i,1}}{\sqrt{2}}, \frac{sign(v_{i,1})v_{i,1}}{\sqrt{2}})$. Another feature map $\phi' : X \mapsto \Re^d$ is $\phi(v_i) = (\frac{v_{i,1}}{\sqrt{2}}, \frac{sign(v_{i,1})v_{i,1}}{\sqrt{2}}, v_i)$.

7 Kernel Perceptron (20pts)

The basic Perceptron algorithm that we saw in the class will loop forever if the data is not linearly separable. One way to fix this is to stop the algorithm after T passes over the data for some fixed value T. Is this a good strategy? Justify your answer with the help of examples.

Solution: This is not a good strategy. Perceptron is very sensitive to the order in which the examples are given. Hence stopping it abruptly at some T can produce very different hypotheses depending on T. This is what you should have observed in your experiments.

Download the web spam dataset from the course webpage. This is a dataset of web pages classified as spam or not spam.

In the downloaded archive you will find:
webspam_wc_normalized_unigram.svm: This file contains info on 350000 web pages. Each webpage is a feature of length 254. The file contains info for each webpage, one per line. Each line has the following format *label feature_index1:feature_val1 feature_index2:feature_val2 ..... .* Label is $+1/-1$. "feature_index:feature_val" format describes that the given feature index has the corresponding value. For instance if the line is *[+1, 10:0.001 112:1.3]*, then it means the document has label +1, feature number 10 has value 0.001, feature number 112 has value 1.3 and all other features are 0.

1. Randomly partition the dataset into 250000 examples for training and 100000 for testing.
2. Implement the Perceptron algorithm mentioned above. Train it with different values of T. Plot T vs accuracy on test data. Which value T^* gives the best tradeoff between training time and accuracy?
3. Fix the value of T^* found above. Again randomly partition the dataset into 250000 examples for training and 100000 for testing. Implement a kernelized version of your algorithm with $T = T^*$. Choose a Gaussian Kernel and experiment with different values of σ. Plot σ vs accuracy on test set. Which value of σ achieves the best accuracy on the test set?
4. Come up with your own modification of the Perceptron algorithm that can outperform your implementation both in terms of train time and test set accuracy. Describe your new algorithm and show results.

   Solution: Perceptron is very sensitive to the stopping point. In order to overcome this, a practical implementation is the *voted* perceptron method. Here, instead of maintaining just one classifer, one maintains a list of classifiers. Every time a mistake is made, the current classifier w_{t-1} is added to the list and we move on to the new classifier $w_t = w_{t-1} + y_i x_i$. At prediction time, a weighted combination of the all the classifiers in the list is used rather than just the current classifier. See here for more details: http://curtis.ml.cmu.edu/w/courses/index.php/Voted_Perceptron.