CSE 517a: Homework #2

Due on Tuesday, February 24th, 2015

Kilian Q. Weinberger 2:30pm

Contents

Tree Structures for Nearest Neighbor

1 Ball Trees

Remember the ball-trees data structure from class. Assume you have a data set $S = \{\vec{x}_1, \ldots, \vec{x}_n\}$ and a ball B with center $\vec{c}$ and radius r, such that:

$$B = \left\{\vec{x}_i \in S : \|\vec{x}_i - \vec{c}\|^2 \leq r^2\right\} \tag{1}$$

(a) Prove that for any test point $\vec{x}_t$ the distance to any point $\vec{x}_i \in B$ is always greater than or equal to the distance from the Ball. (Hint: Make a drawing and use the triangular inequality.)

In some settings of kNN classification it can be the case that one class is much more common than the other. Let's call them negative (very common) and positive (very rare). – Real world examples are fraud detection on credit cards, abnormal behavior detection on security cameras, fault detection in assembly lines etc.
(b) For a test point $\vec{x}_t$, if you knew the distance to the k closest *positive* examples, how could you speed up the kNN classification with ball-trees even further?

(c) How would you adapt your ball-tree data structure to such a setting. Explain why your setup can be much faster for kNN classification in the most common case. (Hint: Construct two different ball-trees and change the pruning rule.)

CART (Classification and Regression Trees)

2 Dynamic Programming for Regression Trees

You are building a regression tree, and your recursive function is called with data $S = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)\}$ (where we have continuous labels $y_i \in \mathcal{R}$). For a leaf let the prediction be p. Your loss-function is the averaged squared-loss:

$$\mathcal{L}(S) = \frac{1}{|S|} \sum_{(\vec{x}_j, y_j) \in S} (y_j - p)^2. \tag{2}$$

(a) Show that the *average* predictor $p \leftarrow \bar{y} = \frac{1}{n} \sum_{(\vec{x}_i, y_i) \in S} y_i$ minimizes $\mathcal{L}$.

If the termination conditions are not met (*i.e.* you do not create a leaf), you need to split. For this, you need to find the best split for any feature f. Assume we have sorted our data set according to some feature f, such that $f_1 \leq f_2 \leq \cdots \leq f_n$, where f_i is the value of feature f in example $\vec{x}_i$. Let all relevant splits be $c_1 \leq c_2 \leq \cdots \leq c_{n-1}$, for example $c_i = \frac{f_i + f_{i+1}}{2}$.
(b) Define the predictors $\bar{y}_L^i, \bar{y}_R^i$ of the left and right sub-trees, after cutting at c_i (left $\leq c_i$) in terms of $y_1, \ldots, y_n$.

(c) Write down the expressions for the loss $\mathcal{L}_L^i$ and $\mathcal{L}_R^i$ on the left and right sub-trees, after cutting at c_i. What is the time complexity of computing both terms for any cut point c_i?

(d) Express $\bar{y}_L^{i+1}, \bar{y}_R^{i+1}$ in terms of $\bar{y}_L^i, \bar{y}_R^i$ and y_{i+1}.

(e) Let $s^i = \sum_{j=1}^{i} y_j^2$ and $r^i = \sum_{j=i+1}^{n} y_j^2$, express $\mathcal{L}_L^{i+1}, \mathcal{L}_R^{i+1}$ in terms of $y_{i+1}, \bar{y}_L^i, \bar{y}_R^i, s^i, r^i$.

(f) Write a full update rule for iteration $i+1$, assuming you have $\bar{y}_L^i, \bar{y}_R^i, s_i, r_i, \mathcal{L}^i, \mathcal{R}^i$.

(g) What is your time complexity for the best-cut search with this method?

(h) What is the time complexity for a best-cut search with Entropy- and Gini impurity measures?

3 Decision Trees

In many applications (e.g. medical sciences) it is not always possible to obtain all features (e.g. medical examinations) on all data examples $\vec{x}_i$ (e.g. patients). This results in some missing features in the train / test data. In those cases we have an additional bit that indicates that a particular feature value does not exist.
(a) How would you adapt decision trees to this sort of data? (Just describe the necessary changes to the algorithm.)

(b) Can you also create a *binary* decision tree that incorporates missing data?

Model Averaging

4 Bagging

Bagging is a method to reduce the variance of a classifier by averaging over several classifiers trained on subsets of the original training data. The subsets are obtained by *uniform* subsampling *with replacement*. i.e. if your data is $S = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)\}$, at each iteration you create a new data set S' with n random picks, picking each example pair with probability $\frac{1}{n}$ *each time*. As a result you could end up with multiple identical pairs, or some not present at all.

Let $p_n(m, k)$ be the probability that you have drawn m unique pairs after k picks with $|S| = n$. So clearly $p_n(m, k) = 0$ whenever $m > k$ (because you cannot end up with more unique elements m than you have drawn), and also $p_n(m, k) = 0$ whenever $m > n$.

(a) What are the base-case values of $p_n(1, 1), p_n(m, 1), p_n(1, k)$?

(b) Assume you are have already picked $k - 1$ elements. What is the probability that the k^{th} pick will *not* increase your number of unique elements m? What is the probability that it will?

(b) Can you express $p_n(m, k)$ in terms of $p_n(m, k - 1)$ and $p_n(m - 1, k - 1)$? (Hint: Use the result of the previous question.).

(c) Write out the formula for $E_{k=n}(\frac{m}{n})$, the expected ratio of unique elements after n picks with $|S| = n$.

(d) Write a little recursive function (in the programming language of your choice) that evaluates $E_{k=n}(\frac{m}{n})$. Plot its value as n increases. What value does it converge to?

(e) If you average over M classifiers, trained on sub-sets $S'_1, \ldots, S'_M$, what is the probability that one input pair is never picked in any of the training data sets S'_i? Plot this function as M increases. (Assuming that n is large enough for the convergence as observed in (c).)

Preparation for Linear classification

5 The sigmoid function

The sigmoid function is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

(a) Plot $\sigma(x)$. What are the extreme values as x becomes very large and very small?

(b) Let $\sigma'(x) = \frac{\partial \sigma}{\partial x}$. Verify that $\sigma'(x) = \sigma(x)\sigma(-x)$.

(c) Is the sigmoid function symmetric about a point? Prove your claim.

(d) An alternative to the sigmoid is the function $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. Is $tanh(x)$ symmetric about a point?

Programming

6 Implementation of Decision Tree

In this assignment you will implement a decision tree. Please first update your svn directory by calling *svn update* in the *svntop* directory.

As Matlab does not support pointers and is really only good with matrices, we will represent a tree through a matrix T. A tree T with q nodes must be of dimensions $6 \times q$, where each column represents a different node in the tree. The i^{th} row represents the following information:

1. **prediction at this node**
2. **index of feature to cut**
3. **cutoff value** c **(**$\leq c$ **: left, and** $> c$ **: right)**
4. **index of left subtree** $(0 = leaf)$
5. **index of right subtree** $(0 = leaf)$
6. **parent** $(0 = root)$

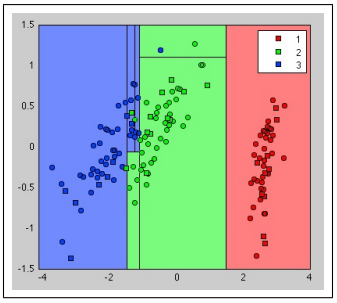

Figure 1: Visualization of a decision tree on the 2d iris data.

Make sure to add at least one test for each function you implement in *hw2tests.m*!
(a) If each parent splits (roughly) binary, what do you expect the value of q to be?

We have implemented a function *entropysplit.m* which searches through all features and returns the best *feature, cut* combination based on information gain (in this case entropy) impurity. You don't need to implement this, it is already there. [For some architectures (Mac, Windows, Linux - all 64bit), we also shipped some compiled binaries to speed things up.] :-)

(b) Implement the function *id3tree.m* which returns a decision tree based on the minimum entropy splitting rule. You can visualize your tree with the command *visualhw2*. (Hint: To speed up your code make sure you initialize the matrix T with the command $T = zeros(6, q)$ with some estimate of q.)

(c) Implement the function *evaltree.m*, which evaluates a decision tree on a given test data set. You can test the accuracy of your implementation with *hw2tictoc*.

(d) Implement the function *prunetree.m*, which returns a decision tree pruned for a validation data set.

(e) Implement the function *forest.m*, which builds a forest of id3 decision trees.

(f) Implement the function *evalforest.m*, which evaluates a forest on a test data set.

(g) Implement the function *boosttree.m*, which applies adaboost on your *id3tree* functions.

(h) Implement the function *evalboost.m*, which learns and evaluates a boosted classifier.

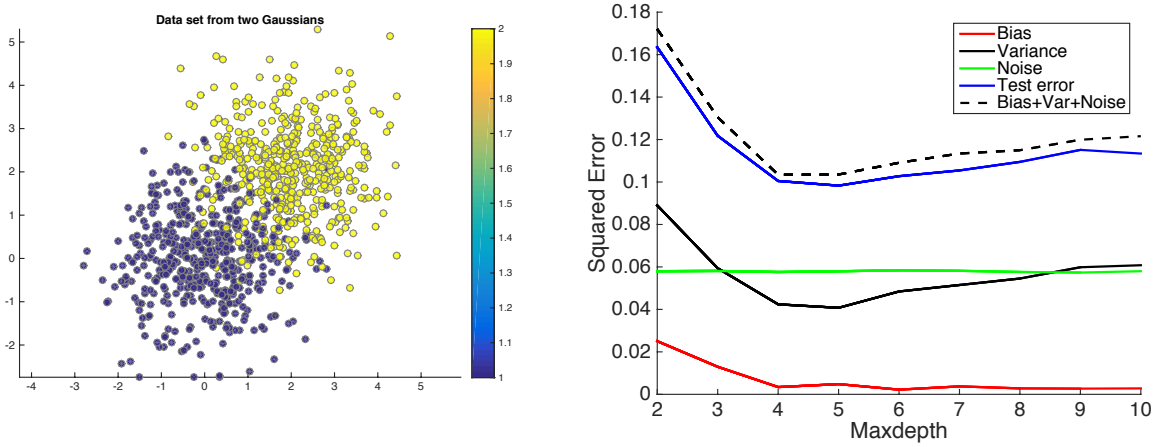

Figure 2: *Left:* a plot of the binary data set with means $\vec{0}$ and $[2,2]^\top$. (Right:) The decomposition of Test error into bias, variance and noise.

7 Bias Variance Trade-off

Remember from class the decomposition of squared error into *bias*, *variance* and *noise*.

$$\underbrace{\mathbb{E}[(h_D(x)-y)^2]}_{\text{Error}} = \underbrace{\mathbb{E}[(h_D(x)-\bar{h}(x))^2]}_{\text{Variance}} + \underbrace{\mathbb{E}[(\bar{h}(x)-\bar{y}(x))^2]}_{\text{Bias}} + \underbrace{\mathbb{E}[(\bar{y}(x)-y(x))^2]}_{\text{Noise}} \tag{4}$$

We will now create a data set for which we can approximately compute this decomposition. The function *toydata.m* generates a binary data set with classes 1 and 2. Both are distributed from Gaussian distributions:

$$p(\vec{x}|y=1) \sim \mathcal{N}(0,I) \text{ and } p(\vec{x}|y=1) \sim \mathcal{N}(\mu_2,I), \tag{5}$$

where $\mu_2 = [2;2]^\top$ (the global variable OFFSET=2 regulates these values).

You will need to edit three functions: *computeybar.m*, *computehbar.m* and *computevariance.m*. Take a look at the function *biasvariancedemo.m* and make sure you understand where they are called and how they contribute to the Bias/Variance/Noise decomposition.

(a) **Noise:** First we focus on the noise. For this, you need to compute $\bar{y}(\vec{x})$ in *computeybar.m*. With the equations in (5) you can compute the probability $p(\vec{x}|y)$. Then use Bayes rule to compute $p(y|\vec{x})$. [You may want to use the function *normpdf*, which is defined for you in *computeybar.m*.]

(b) **Bias:** For the bias you will need $\bar{h}$. We cannot compute the expected value $\bar{h} = \mathbb{E}[h]$, however we can approximate it by training many h_D and averaging their predictions. Edit the file *computehbar.m*. Average over NMODELS different h_D, each trained on a different data set of NSMALL inputs drawn from the same distribution. Feel free to call *toydata.m* to obtain more data sets.

(c) **Variance:** Finally, to compute the variance, we need to compute the term $\mathbb{E}[(h_D - \bar{h})^2]$. Once again, we can approximate this term by averaging over many (NMODELS) models. Edit the file *computevariance.m*.

If you did everything correctly and you call *biasvariancedemo.m*, you should see how the error decomposes (roughly) into bias, variance and noise. If you want the approximation to be more accurate, increase NMODELS and/or NBIG (both these variables simulate infinitely many, so more is better). You can also play around with the variable NSMALL, which regulates how big your actual training is supposed to be.