Problem Set 1

Jingwei Li -- jli779

1. For almost every case we have discussed where we are doing supervised learning, we have assumed a deterministic function. Imagine instead a world where we are trying to capture a non-deterministic function. In this case, we might see training pairs where the x value appears several times, but with different y values. For example, we might be mapping attributes of humans to whether they are likely to have had chicken pox. In that case, we might see the same kind of person many times but sometimes they will have had chicken pox, sometimes not.
We would like to build a learning algorithm that will compute the probability that a particular kind of person has chicken pox. So, given a set of training data where each x is mapped to 1 for true or 0 for false:
1) Derive the proper error function to use for finding the ML hypothesis using Bayes Rule. You should go through a similar process as the one used to derive least squared error in the lessons.
Answer:

Assume the training data D is of the form D={<x1,d1>,<x2,d2>,...,<xm,dm>} and each training example is drawn independently=>

$$P(D|h) = \prod_{i=1}^{m} P(x_i, d_i|h)$$

since x is independent of h, so

$$P(D|h) = \prod_{i=1}^{m} P(x_i, d_i|h) = \prod_{i=1}^{m} P(d_i|h, x_i)P(x_i) \qquad (*)$$

$$P(d_i|h, x_i) = \begin{cases} h(x_i) & \text{if } d_i = 1 \\ (1 - h(x_i)) & \text{if } d_i = 0 \end{cases}$$ which can be re-expresed as:

$$P(d_i|h, x_i) = h(x_i)^{d_i}(1 - h(x_i))^{1-d_i}$$ substitute this into equation (*)

$$P(D|h) = \prod_{i=1}^{m} h(x_i)^{d_i}(1 - h(x_i))^{1-d_i} P(x_i)$$

Now we write an expression for the maximum likelihood hypothesis

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} h(x_i)^{d_i}(1 - h(x_i))^{1-d_i} P(x_i)$$

The last term is a constant independent of h, so it can be dropped

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} h(x_i)^{d_i}(1 - h(x_i))^{1-d_i}$$

==>

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \sum_{i=1}^{m} d_i \ln h(x_i) + (1 - d_i) \ln(1 - h(x_i)) \qquad (**)$$

2) Compare and contrast your result to the rule we derived for a deterministic function perturbed by

zero-mean gaussian noise. What would a normal neural network using sum of squared errors do with these data? What if the data consisted of x,y pairs where y was an estimate of the probability instead of 0s and 1s?

Answer:

----For a deterministic function perturbed by zero-mean Gaussian noise, minimizing the sum of the squared errors produces the maximum likelihood hypothesis. By contrast, for this problem where the assumption is that the observed Boolean value is a probabilistic function of the input instance, the rule that maximize the equation (**)(i.e., minimize the cross entropy)seeks the maximum likelihood hypothesis.

----For a normal neural network that seeks to minimize the sum of squared errors, it can perform a gradient ascent rather than gradient descent search and the weight vector is adjusted in the direction of the gradient, using the following update rule:

$$w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$$

where

$$\Delta w_{jk} = \eta \sum_{i=1}^{m} (d_i - h(x_i))\ x_{ijk}$$

Δw_{jk} is derived as following: denote the quantity in (**) as G(h, D)

$$\begin{aligned}
\frac{\partial G(h, D)}{\partial w_{jk}} &= \sum_{i=1}^{m} \frac{\partial G(h, D)}{\partial h(x_i)} \frac{\partial h(x_i)}{\partial w_{jk}} \\
&= \sum_{i=1}^{m} \frac{\partial (d_i \ln h(x_i) + (1 - d_i) \ln(1 - h(x_i)))}{\partial h(x_i)} \frac{\partial h(x_i)}{\partial w_{jk}} \\
&= \sum_{i=1}^{m} \frac{d_i - h(x_i)}{h(x_i)(1 - h(x_i))} \frac{\partial h(x_i)}{\partial w_{jk}}
\end{aligned}$$

To keep our analysis simple, suppose our neural network is constructed from a single layer of sigmoid units. In this case we have

$$\frac{\partial h(x_i)}{\partial w_{jk}} = \sigma'(x_i) x_{ijk} = h(x_i)(1 - h(x_i)) x_{ijk}$$

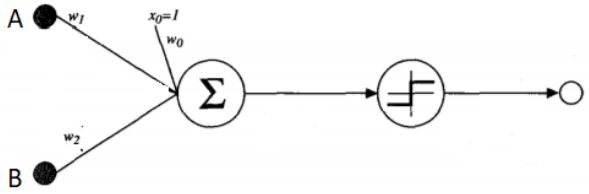
===> $$\frac{\partial G(h, D)}{\partial w_{jk}} = \sum_{i=1}^{m} (d_i - h(x_i))\ x_{ijk}$$

---- If the data consisted of x,y pairs where y was an estimate of the probability, it will be more like the deterministic function discussed in the lecture, assuming Gaussian noise. The most likely hypothesis will be the one that minimize the sum of squared errors between the observed training values yi (instead of di which can be any real numbers, yi is a probability and is between 0 and 1)and the hypothesis prediction h(xi).

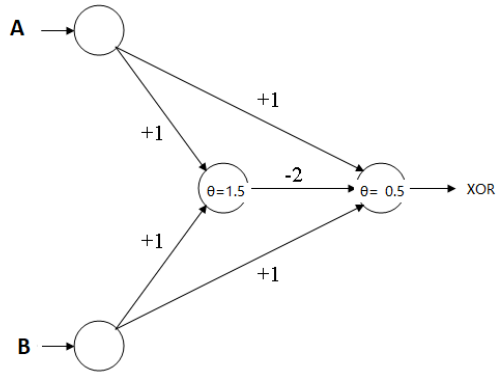
$$h_{ML} = \underset{h \in H}{\operatorname{argmin}} \sum_{i=1}^{m} (y_i - h(x_i))^2$$

2. Design a two-input perceptron that implements the boolean function $A \wedge \neg B$.
Answer:

w_0=-0.3, w_1=0.5, w_2= -0.5



Design a two-layer network of perceptrons that implements $A \oplus B$ ($\oplus$ is XOR).
Answer:



3. Derive the perceptron training rule and gradient descent training rule for a single unit with output o, where $o = w_0 + w_1x_1 + w_1x^2_1 + \ldots + w_nx_n + w_nx^2_n$. What are the advantages of using gradient descent training rule for training neural networks over the perceptron training rule?

Answer:

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d\in D}2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$

$$= \sum_{d\in D}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \text{W0-W1(X1d+X1d\^2)-...-Wi(Xid+Xid\^2)-...-Wn(Xnd+Xnd\^2))}$$

$$\frac{\partial E}{\partial w_i} = \sum_{d\in D}(t_d - o_d)(-x_{id} - x_{id}{}^2)$$

$$\Delta w_i = \eta\sum_{d\in D}(t_d - o_d)\ (\ x_{id} + x_{id}{}^2\ )$$

The perceptron rule converges to a weight vector that correctly classifies al training examples, provided the training examples are linear separable. Gradient descent rule does not have that requirement. When it is applied to not linearly separable examples, it will converges toward a best-fit approximation to target concept (of course, gradient descent can also be used to train linear separable examples)

4. Explain how you can use Decision Trees to perform regression? Show that when the error function is squared error, then the expected value at any leaf is the mean. Take the Boston Housing dataset (https://archive.ics.uci.edu/ml/datasets/Housing) and use Decision Trees to perform regression.

Answer:

We can construct a decision tree for regression only replacing the information gain with reduction of error.

In more details: the most common method for building a regression tree based on a sample of an unknown regression surface consists of trying to obtain the model parameters that minimize the least squares error criterion. webuilding a regression tree tries to minimize the least squares error criterion.So constructing a decision tree is all about finding splitting attribute that maximizes the decrease in the error of the tree resulting from this spliting. The dataset then is divided based on the values of the selected attributes and a branch will stop splitting if some termination criteria is met (e.g., zero standard deviation). The process is run recursively on the non-leaf branches until all data is processed. When the number of instances is>1 in one leaf node, the mean is calculated as the final value for the target.

Here is result from running the boston housing dataset using regression tree:

=== Run information ===

Scheme: weka.classifiers.trees.REPTree -M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0
Relation: housing
Instances: 506
Attributes: 14
CRIM
ZN
INDUS
CHAS
NOX
RM
AGE
DIS
RAD
TAX
PTRATIO
B
LSTAT
MEDV
Test mode: 10-fold cross-validation

=== Classifier model (full training set) ===

REPTree
==========

RM < 6.8
| LSTAT < 14.75
| | CRIM < 8.05
| | | RM < 6.54
| | | | TAX < 222.5 : 27.2 (4/13.26) [2/83.08]
| | | | TAX >= 222.5

```
|  |  |  |  |  LSTAT < 7.62 : 24.42 (28/1.49) [16/48.47]
|  |  |  |  |  LSTAT >= 7.62
|  |  |  |  |  |  RM < 5.84 : 19.19 (21/4.34) [6/3.17]
|  |  |  |  |  |  RM >= 5.84
|  |  |  |  |  |  |  PTRATIO < 18.65
|  |  |  |  |  |  |  |  TAX < 412.5 : 22.21 (34/3.75) [19/4.69]
|  |  |  |  |  |  |  |  TAX >= 412.5 : 19.08 (4/2.82) [2/0.55]
|  |  |  |  |  |  |  PTRATIO >= 18.65
|  |  |  |  |  |  |  |  RAD < 4.5 : 18.86 (11/3.28) [7/12.79]
|  |  |  |  |  |  |  |  RAD >= 4.5 : 20.65 (33/2.58) [14/8.19]
|  |  |  RM >= 6.54
|  |  |  |  LSTAT < 6.93 : 29.03 (11/5.62) [12/50.91]
|  |  |  |  LSTAT >= 6.93
|  |  |  |  |  PTRATIO < 20.95 : 25.62 (13/5.49) [4/6.17]
|  |  |  |  |  PTRATIO >= 20.95 : 21.7 (2/0.49) [0/0]
|  |  CRIM >= 8.05 : 50 (2/0) [0/0]
|  LSTAT >= 14.75
|  |  LSTAT < 19.05
|  |  |  CRIM < 0.61
|  |  |  |  AGE < 73.3 : 20.7 (8/1.99) [2/3.9]
|  |  |  |  AGE >= 73.3 : 18.42 (16/1.96) [7/7.33]
|  |  |  CRIM >= 0.61 : 15.43 (27/6.17) [14/3.72]
|  |  LSTAT >= 19.05
|  |  |  NOX < 0.6 : 17.24 (11/7.42) [8/27.03]
|  |  |  NOX >= 0.6 : 12.45 (20/9.68) [6/1.21]
RM >= 6.8
|  RM < 7.42 : 32.13 (35/19.45) [23/32.28]
|  RM >= 7.42 : 45.1 (21/44.72) [9/17.82]

Size of the tree : 35

Time taken to build model: 0.03 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.866
Mean absolute error                      2.9954
Root mean squared error                  4.4076
Relative absolute error                 46.638  %
Root relative squared error             49.9055 %
Total Number of Instances              452
Ignored Class Unknown Instances                54
```

5. Suggest a lazy version of the eager decision tree learning algorithm ID3. What are the advantages and disadvantages of your lazy algorithm compared to the original eager algorithm?

Answer:

Store instances during training phase and start building decision tree using ID3 at classification phase. You will still use info gain to decide what the best attribute is, but you only need to build the tree on the branches that has the value of the attribute in the test sample. So that you can ignore all other irrelevant paths in the tree, but only construct a path that help classify your test sample.

The advantage is that you can save computation time this way by only building part of the tree that will help classify your test sample. The disadvantage is that, however, when the test sample size increases, you might still end up building the whole tree to classify all test samples.

6. Imagine you had a learning problem with an instance space of points on the plane and a target function that you knew took the form of a line on the plane where all points on one side of the line are positive and all those on the other are negative. If you were constrained to only use decision tree or nearest-neighbor learning, which would you use? Why?
Answer:
The problem's instance space are points on a plane and let us assume that the linear separator's function takes the form of $f(x)=w_0+w_1x$. The goal is to find the approximation of w_0 and w_1.

With decision trees, regression tree in this case: theoretically, we need to go through infinite number of w_1 and w_0 to calculate the standard deviation reduction and decide on the best splitting. Or in other words, if there are N possible values for w_1, for example, we will have to go through N-1possible splitting. Also, with regression tree, we will have to take all training examples for function approximation. The computation is expensive. However, regression trees will not get hugely affected by outliners/noise.

With nearest neighbor (locally weighted regression), we only need to focus on local training samples of your query at testing time instead constructing a function at training time for all instances. The computation is much less expensive. However, it constructs an explicit approximation to f over a local region. So if you end up querying many samples, the computation can also be expensive. The other thing is local approximation is very susceptible to the choice of k (number of nearest neighbors) – A small value of k means that noise will have a higher influence on the result. A large value make it computationally expensive and kinda defeats the basic philosophy behind KNN.

In general: Consider the following dataset: N samples, each has k attributes.
1. naive KNN: O(1) [training Time] + O(NK) [query Time] = O (NK)
*2. naive decision tree: O(N^2 * K * log(N)) [training Time] + O(log(N)) [query Time] = O(N^2 * K)*
So in most cases if we properly choose the number of neighbors (to control the influence of noise) and do not perform too many queries, I will choose to use nearest neighbor learning for this problem.

7. Give the VC dimension of the following hypothesis spaces. Briefly explain your answers.
   1. An origin-centered circle (2D)
      Answer:
      VC-dim = 2. It is easy to see how an origin-centered circle can shatter 2 points. However, no set of three points is shatter-able by origin-centered circles. E.g., any set of three points at some radii $r1 \leq r2 \leq r3$ from the origin, and no function will be able to label the points at r1 and r3 with + while labeling the point at r2 with −.

   2. An origin-centered sphere (3D)
      Answer:
      VC-dim = 2. The same reasoning as for the 2D case applies.