CS 7641 Machine Learning: Problem Set 1

Jilong Cui
jcui@gatech.edu

1 PROBLEM 1

Where we are doing supervised learning, we have mostly 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 use attributes of humans to the probability that they have had chicken pox. In that case, we might see the same kind of person many times but only sometimes they may have had chicken pox.

We would like to build a learning algorithm that will compute the probability that a person has chicken pox. So, given a set of training data where each instance 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.

Solution:

D = training dataset

h(x) = hypothesis that gives the probability of a certain instance x being true (having chicken pox in this example)

y_i = true label for instance x_i

n = number of training samples

Find the hypothesis h that maximizes the likelihood of observing the training data D (using maximum likelihood estimation i.e. MLE)

For the ith example in the dataset:

The probability of observing $y_i = 1$ given x_i is $h(x_i)$

The probability of observing $y_i = 0$ given x_i is $1 - h(x_i)$

Likelihood of the data given the hypothesis h:

$$L(D|h) = \Pi_{i=1}^{n} h(x_i)^{y_i} \times (1 - h(x_i))^{(1-y_i)}$$

To maximize L(D|h), equivalently maximize its logarithm. Take logarithm on both side:

$$l(D|h) = \sum_{i=1}^{m} y_i \log(h(x_i)) + (1 - y_i) \log(1 - h(x_i))$$

To find the best h, minimize the negative log likelihood, which can teat as error function:

$$E(h) = -\sum_{i=1}^{m} y_i \log(h(x_i)) + (1 - y_i) \log(1 - h(x_i))$$

So, $h(x_i)$ is the estimated probability that person i has had chicken pox, and y_i is 1 if person I has had chicken pox, and 0 otherwise.

This is the cross-entropy loss, commonly used for probabilistic classification tasks.

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?

Solution:

Deterministic function with Gaussian Noise:

Underlying assumption – a true deterministic function f(x), and the observed outputs y deviate from f(x) due to additive Gaussian noise. In this scenario, use the mean squatted error (MSE) as the error function which is derived from the maximum likelihood estimation under the Gaussian noise assumption.

$$E(h) = \frac{1}{m} \sum_{i=1}^{m} (h(x_i) - y_i)^2$$

For the probabilistic function (non-deterministic), the derived error function is the cross-entropy loss, which calculates the dissimilarity between the predicted probabilities and actual labels (0s and 1s).

For the deterministic function with the Gaussian noise, the error function is the mean squatted error (MSE), measuring the difference between predicted values and true values.

With MSE, a neural network would try to approximate the conditional mean of the target given the input. For the chickenpox example, if the network is trained on data with 0s and 1s as outputs,

it would predict the expected value of the output given an input. In essence, the network would output probabilities.

If the outputs were estimates of probability rather than 0s and 1s: The neural network using MSE would still try to approximate the conditional mean of the target given the input. However, since the output is already a probability, the network would effectively be modeling the expected value of this probability given the input.

In sum,

- For the probabilistic function (non-deterministic), we are directly modeling the uncertainty and variability in the data.
- For the deterministic function with Gaussian noise, we assume that there is an underlying deterministic function and that deviations from this function are due to noise.
- MSE is more suited for regression tasks with continuous outputs, while cross-entropy is more apt for classification tasks where outputs are probabilities.
- A neural network using MSE would output probabilities when trained on 0s and 1s. If trained on probability estimates, it would model the expected value of these probabilities given the input.

2 PROBLEM 2

Design a two-input perceptron that implements the boolean function $A \wedge \neg B$. Design a two-layer network of perceptrons that implements $A \oplus B$ (where $\oplus$ is XOR).

Solution:

For the Boolean function $A \wedge \neg B$, the truth table is:

A	B	$A \wedge \neg B$
0	0	0
0	1	0
1	0	1
1	1	0

To design a perceptron for $A \wedge \neg B$, use the weights and bias as follow:

w1 = 1 (for input A)

w2 = -1 (for input B)

Bias b = -0.5

So, if plug in the values from the truth table into the perceptron equations y = Aw1 + Bw2 + b

For A = 1, B = 0 → y = 1(1) + 0(-1) – 0.5 = 0.5 which is positive and thus, the perceptron fires (i.e., outputs 1).

For all other combinations, the perceptron will output 0.

Implementing $A \oplus B$ using a two-layer network of perceptrons:

The XOR function, represented by ⊕, is given by:

A	**B**	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Break it down into basic logical functions:

A ⊕ B = (A ∧¬ B) ∨ (¬A ∧ B)

So, we need perceptrons for A ∧¬ B, ¬A ∧ B, and a final OR perceptron.

Layer 1:

Perceptron 1: A ∧¬ B

- Weights: w1 = 1, w2 = -1
- Bias: b = -0.5

Perceptron 2: ¬A ∧ B

- Weights: w1 = -1, w2 = 1
- Bias: b = -0.5

Layer 2:

Perceptron for OR:

- Inputs: outputs from Perceptron 1 and Perceptron 2
- Weights: w1 = 1, w2 = 1
- Bias: b = -0.5

3 PROBLEM 3

Derive the perceptron training rule and gradient descent training rule for a single unit with output o, where $o = w0 + w1x1 + w1x1^2 + \ldots + wnxn + wnxn^2$. What are the advantages of using gradient descent training rule for training neural networks over the perceptron training rule?

Solution:

1. Perceptron Training Rule

The perceptron training rule updates the weights based on the error of the actual and predicted outputs. Given a target value t and a learning rate α, the weight update for w_i is given by:

$$\triangle w_i = \alpha(t - 0)x_i$$

$$\triangle w_i^2 = \alpha(t - 0)x_i^2$$

where Δwi and Δwi2 are the weight updates for the linear and quadratic terms respectively.

2. Gradient Descent Training Rule

For gradient descent, we want to minimize the squared error:

$$E = 0.5(t - 0)^2$$

To update each weight, we find the gradient of E with respect to that weight and then update the weight in the direction that decreases the error.

For w_i, the weight for the linear term:

$$\frac{\partial E}{\partial w_i} = (0 - t)\frac{\partial 0}{\partial w_i}$$
$$\frac{\partial E}{\partial w_i} = (0 - t)x_i$$

Weight update:

$$\Delta w_i = -\alpha \frac{\partial E}{\partial w_i} = \alpha(t - 0)x_i$$

For w_i^2, the weight for the quadratic term:

$$\frac{\partial E}{\partial w_i^2} = (0 - t)\frac{\partial 0}{\partial w_i^2}$$
$$\frac{\partial E}{\partial w_i^2} = (0 - t)x_i^2$$

Weight update:

$$\Delta w_i^2 = -\alpha \frac{\partial E}{\partial w_i^2} = \alpha(t - 0)x_i^2$$

4 PROBLEM 4

Explain how one can use Decision Trees to perform regression? Show that when the error function is squared error that the expected value at any leaf is the mean. Take the Boston Housing dataset (http://lib.stat.cmu.edu/datasets/boston) and use Decision Trees to perform regression.

Solution:

Using Decision Trees for Regression (Regression Tree):

1. Node Splitting: Just as in classification, we split nodes based on some criterion. However, instead of minimizing impurity (e.g., Gini or entropy), we try to reduce the variance (or mean squared error) of the target variable within the nodes.

2. Predictions: Instead of predicting a class label in leaves, a regression tree predicts a continuous value. This prediction is typically the mean of the target values of training samples associated with that leaf.

Prove the expected values at any leaf is the Mean with Squared Error:

Given a set of target values y_i in a leaf, the prediction $\hat{y}$ should minimize the total squatted error.

$$\sum_i^n (y_i - \hat{y})^2$$

To find the value of y^ that minimizes the error, take the derivative with respect to y^ and set it to zero:

$$\frac{d}{d\hat{y}} \sum_i^n (y_i - \hat{y})^2 = -2 \sum_i^n (y_i - \hat{y}) = 0$$

From above,

$$\sum_i^n y_i = n\hat{y}$$
$$\hat{y} = \frac{1}{n} \sum_i^n y_i$$

Boston Housing Dataset with Regression Trees:

Split the data into training and testing sets, then train can use DecisionTreeRegressor from Sklearn to perform the regression trees with the mean_squared_error as the evaluation function.

5 PROBLEM 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?

Solution:

The eager decision tree learning algorithm, like ID3, constructs a decision tree from the training data before receiving any test instance. In contrast, a lazy learning algorithm delays the construction of the model until a query is made to the system.

Lazy version of ID3 (LazyD3):

1. Training Phase:

- Store all training data in memory without processing it.

2. Prediction Phase:

- When a test instance is received, begin constructing a decision tree, but only for the parts that will be used to classify the given instance.

- Use ID3 to create a decision tree rooted at the attribute that provides the highest information gain for the given test instance. Split the training data based on this attribute.

- Traverse down the tree using the test instance's attributes. At each node, continue to split the training data and expand the tree based on the attribute with the highest information gain for the subset of training data at that node.

- Once a leaf node is reached or a certain depth is met, classify the test instance based on the majority class of the training data at that node.

Advantages of LazyD3 over Eager ID3:

1. Memory Efficiency: Since the decision tree is constructed on-the-fly, we do not store any model in memory during the training phase.

2. Flexibility: Can better adapt to local characteristics of the test data, potentially providing improved accuracy for certain types of data distributions.

3. Handles Concept Drift: Since a new tree is created for every test instance, it can adapt to changes in data distribution, which is particularly useful in streaming data scenarios where the underlying data distribution might change over time.

Disadvantages of LazyD3 compared to Eager ID3:

1. Computationally Expensive: Constructing a decision tree for every test instance can be time-consuming, especially with large training datasets.

2. Less Global Insight: Eager ID3, by using all training data to construct a tree, may capture more global patterns in the data.

3. Scalability: Lazy learning can be problematic for large datasets as it requires keeping the entire training dataset in memory.

4. Redundant Computations: Repeatedly constructing trees for each test instance may result in a lot of redundant calculations.

In summary, while the lazy version of ID3 can be more adaptive and flexible for specific test instances, it comes at the cost of increased computational complexity and potentially less global insight into data patterns. The choice between the lazy and eager versions would be context-dependent, hinging on the specific needs of the application, the nature of the data, and computational resources available.

6 PROBLEM 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?

Solution:

Decision Tree Learning:

1. Model Form: Decision trees construct rectangular decision boundaries.

2. Linear Boundary: In the two-dimensional plane described, a decision tree can approximate a linear boundary by a series of vertical and horizontal splits. However, this approximation might require a deep tree if the boundary isn't perfectly horizontal or vertical.

3. Generalization: Decision trees will generalize over the instance space based on the training data it has seen. If the training data is comprehensive and well-distributed, it can build a reasonably accurate approximation of the linear boundary.

Nearest-Neighbor Learning:

1. Model Form: Nearest-neighbor doesn't explicitly form a decision boundary; instead, it classifies based on the closest training instances.

2. Linear Boundary: Given that the target function is a line, nearest-neighbor can work very effectively, especially if there's a dense and well-distributed set of training points across the boundary. A test point will be classified based on its proximity to training instances, and given the clear separation by a line, this method can be very accurate.

3. Generalization: Nearest-neighbor does not generalize over the instance space but rather memorizes the training dataset.

Decision:

In this scenario, if we have a dense and well-distributed set of training points spanning the linear boundary, nearest-neighbor learning would likely be the preferable choice. The reason is that it can directly leverage the clear linear separation in the instance space without needing to approximate this boundary through a series of rectangular splits, as a decision tree would.

However, if computational efficiency during the testing phase is a concern (since nearest-neighbor requires comparing a test instance to all training instances), and if we have a decision tree algorithm that can effectively approximate the linear boundary with a reasonably shallow tree, then the decision tree might be considered.

In essence, the choice hinges on the nature of the training data (density and distribution) and specific problem constraints (e.g., computational resources, prediction speed requirements). But purely from a representational perspective given the problem, nearest-neighbor learning seems more naturally suited.

7 PROBLEM 7

Give the VC dimension of these hypothesis spaces, briefly explaining your answers:

1. An origin-centered circle (2D)

2. An origin-centered sphere (3D)

Solution:

The Vapnik-Chervonenkis (VC) dimension provides a measure of the capacity of a hypothesis class in terms of its ability to shatter a set of points. "Shattering" means that for any possible labeling of a set of points, there exists some hypothesis within the class that can separate or realize that labeling without error.

1. An origin-centered circle (2D):

Consider the scenario where we place 3 points on the 2D plane. If these 3 points are placed in such a way that they form the vertices of an equilateral triangle, then using the origin-centered circle hypothesis, we can adjust the radius of the circle to classify any possible combination of labelings for these 3 points. Thus, the origin-centered circle can shatter 3 points.

However, when we place 4 points in such a way that they form the vertices of a square, there's no way an origin-centered circle can shatter all possible labelings of these 4 points (e.g., when one diagonal is labeled positive while the other diagonal is labeled negative).

Thus, the VC dimension for an origin-centered circle in 2D is 3.

2. An origin-centered sphere (3D):

Let's extend the previous logic to 3D. With 4 non-coplanar points, we can adjust the radius of the origin-centered sphere to classify any possible combination of labelings for these 4 points. Therefore, the origin-centered sphere can shatter 4 points.

However, when we consider 5 points such that they form the vertices of a tetrahedron with one point inside, there is no way an origin-centered sphere can shatter all possible labelings of these 5 points (similar to the diagonal reasoning for the circle).

Thus, the VC dimension for an origin-centered sphere in 3D is 4.

In summary, the VC dimension for an origin-centered circle in 2D is 3, while for an origin-centered sphere in 3D, it's 4.

REFERECE

OpenAI. (2023). *ChatGPT* (September 25 Version) [Large language model]. https://chat.openai.com