CPT_S 570: Homework #3

Instructor: Jana Doppa

Sheng Guan

Problem 1

Answer:

a)

Let $X = x - z = (x_1 - z_1, x_2 - z_2, ..., x_d - z_d)$, let $f(x) = x^2$,obviously $f(x)$ is a convex function.

$$\mathbb{E}(X) = \frac{\sum_{i=1}^{d}(X_i)}{d} = \frac{\sum_{i=1}^{d}(x_i - z_i)}{d} \tag{1}$$

$$\mathbb{E}[f(X)] = \frac{\sum_{i=1}^{d}(X_i^2)}{d} = \frac{\sum_{i=1}^{d}(x_i - z_i)^2}{d} \tag{2}$$

Based on Jensen's inequality, we can get:

$$f(\mathbb{E}(X)) = (\frac{\sum_{i=1}^{d}(x_i - z_i)}{d})^2 \leq \mathbb{E}[f(X)] = \frac{\sum_{i=1}^{d}(x_i - z_i)^2}{d} \tag{3}$$

then

$$(\frac{1}{\sqrt{d}}\sum_{i=1}^{d} x_i - \frac{1}{\sqrt{d}}\sum_{i=1}^{d} z_i)^2 \leq \sum_{i=1}^{d}(x_i - z_i)^2 \tag{4}$$

b)

In a high-dimensional space, computing distance of two points is very expensive, especially we need to compute all pair-wise distances. We can pre-calculate $\sum_{i=1}^{d} x_i$ for each nodes, then use $(\frac{1}{\sqrt{d}}\sum_{i=1}^{d} x_i - \frac{1}{\sqrt{d}}\sum_{i=1}^{d} z_i)^2$ as the distance value. Since the distance function is one-dimensional, the computational complexity of NN computation is reduced.

Problem 2

Answer:

The problem in this paper want to solve *nearest neighbor* problem. There has several efficient algorithms can solve *nearest neighbor* problem efficiently but only when the dimension d is low. When it comes to high dimension, these algorithms cannot perform well since the distance computation in high dimensional space is very expensive.

The authors try to get a tradeoff between computation time and accuracy. In the related work, there exists several algorithm designed for this purpose. They can return an approximate nearest neighbor in a short time, and this neighbor is almost as good as the exact one.

This paper introduces an *locality-sensitive hashing* (LSH) based approximate algorithm. It uses several hash functions to hash "close" points into same buckets. Then giving a query point, it uses the same function to hash it to some buckets and find near neighbors in buckets containing that point. It is somehow like the coarse-fine search, which first locate a section and then find the match ones within that section. This paper also introduces some LSH families for various distance measures and a LSH function which can be used in this near neighbor algorithm.

The key insight behind the solution is that in high dimension space, calculate the exact distance between query point and all other points is quite expensive. Also, given all points, they have some kind of space distribution. For a query point, actually we do not need to calculate the distance between it and all other points. What will contribute to the final result is those close to the query point. Thus use LSH we can avoid calculating the distance between query points and their "far" points. By considering only the "near"

neighbors, we still can output the result with some confidence. This is reasonable tradeoff between time complexity and accuracy.

The most important thing here is how to make sure we choose a certain *locality-sensitive hash functions* which can give some result confidence. Given a set P of points and any query point q, we use a probability δ and the each neighbor of q in P is with probability $1 - \delta$. This is important as it makes this work meaningful, which means we can spend less time but get almost the same result. The authors also propose two strategies to scan through the buckets and retrieve the points stored in them, that yields a solution to the randomized c-approximate R-near neighbor problem and randomized R-near neighbor reporting problem correspondingly. In the end, they also propose a near-optimal LSH functions for Euclidean Distance. In this scenario, grid partitioning does not result in any improvement. Instead, they use the method of ball partitioning

The strengths of this method is that it provides a good way to get a "good enough" result instead of spending a long time computing the exact value. It is very efficient and can be used in different scenarios to accelerate the computation.

Problem 3

Answer:
Yes.
The $AQDT-2$ Algorithm can convert a set of rules into an equivalent decision tree.
To facilitate such a process, the system creates a special data structure for each concept description (ruleset). This structure has fields such as the number of rules, the number of conditions in each rule, and the number of attributes in the rules. The system also creates an array of attribute descriptions.

Input: A set of rules, a decision making situation, and a set of testing examples.
Output: A decision structure that suits the given decision making situation.
Step 1: Evaluate each attribute occurring in the ruleset context using the LEF attribute ranking measure. Select the highest ranked attribute. Suppose it is attribute A.
Step 2: Create a node of the tree (initially, the root, afterwards. a node attached to a branch). and assign to it the attribute A. In the standard mode. create as many branches from the node. as there are legal values of the attribute A , and assign these values to the branches. In the compact mode, create as many branches as there are disjoint value sets of this attribute in the decision rules. and assign these sets to the branches.
Step 3: For each branch, associate with it a group of rules from the ruleset context that contain a condition satisfied by the value(s) assigned to this branch. For example, if a branch is assigned values i of attribute A, then associate with it all rules containing condition $[A = i \vee ...]$. If a branch is assigned values $i \vee j$, then associate with it all rules containing condition $[A = i \vee j \vee ...]$. Remove from the rules these conditions. If there are rules in the ruleset context that do not contain attribute A, add these rules to all rule groups associated with the branches stemming from the node assigned attribute A. (This step is justified by the consensus law: $[x = l] \equiv \{[x = l]\&[y = a] \vee [x = l]\&[y = b]\}$, assuming that a and b are the only legal values of y.) All rules associated with the given branch constitute a ruleset context for this branch.
Step 4: If all the rules in a ruleset context for some branch belong to the same class, create a leaf node and assign to it that class. If all branches of the trees have leaf nodes. stop. Otherwise, repeat steps 1 to 4 for each branch that has no leaf.

Problem 4

Answer:

$$H(x) = -\frac{9}{14}log\frac{9}{14} - \frac{5}{14}log\frac{5}{14} = 0.94 \tag{5}$$

Consider outlook first, it has three values, Sunny(2 yes and 3 no), Overcast(4 yes) and Rain(3 yes and 2no)

$$H(x|Outlook) = \frac{5}{14}(-\frac{2}{5}log\frac{2}{5} - \frac{3}{5}log\frac{3}{5}) + \frac{4}{14} * 0 + \frac{5}{14}(-\frac{2}{5}log\frac{2}{5} - \frac{3}{5}log\frac{3}{5}) = 0.694 \tag{6}$$

Also we can get:

$$H(x|Temperature) = \frac{4}{14}(-\frac{2}{4}log\frac{2}{4} - \frac{2}{4}log\frac{2}{4}) + \frac{6}{14}(-\frac{4}{6}log\frac{4}{6} - \frac{2}{6}log\frac{2}{6}) + \frac{4}{14}(-\frac{3}{4}log\frac{3}{4} - \frac{1}{4}log\frac{1}{4}) = 0.911 \tag{7}$$

$$H(x|Humidity) = \frac{7}{14}(-\frac{3}{7}log\frac{3}{7} - \frac{4}{7}log\frac{4}{7}) + \frac{7}{14}(-\frac{6}{7}log\frac{6}{7} - \frac{1}{7}log\frac{1}{7}) = 0.789 \tag{8}$$

$$H(x|wind) = \frac{8}{14}(-\frac{6}{8}log\frac{6}{8} - \frac{2}{8}log\frac{2}{8}) + \frac{6}{14}(-\frac{3}{6}log\frac{3}{6} - \frac{3}{6}log\frac{3}{6}) = 0.892 \tag{9}$$

So we choose Outlook as the first feature. Now if outlook is overcast, we can get a pure node. If outlook is sunny, let's recompute the gain:

$$H(x|Temperature) = \frac{2}{5} * 0 + \frac{2}{5}(-\frac{1}{2}log\frac{1}{2} - \frac{1}{2}log\frac{1}{2}) + \frac{1}{5} * 0 = 1 \tag{10}$$

$$H(x|Humidity) = \frac{3}{5} * 0 + \frac{2}{5} * 0 = 0 \tag{11}$$

$$H(x|wind) = \frac{3}{5}(-\frac{2}{3}log\frac{2}{3} - \frac{1}{3}log\frac{1}{3}) + \frac{2}{5}(-\frac{1}{2}log\frac{1}{2} - \frac{1}{2}log\frac{1}{2}) = 0.951 \tag{12}$$

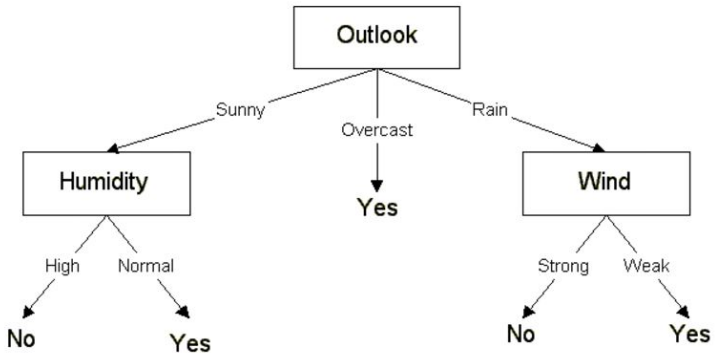
Thus we choose Humidity as the second feature, no matter humidity choose high or normal, we all get pure nodes.
Now consider if outlook is Rain, recompute the gain:

$$H(x|Temperature) = \frac{3}{5}(-\frac{2}{3}log\frac{2}{3} - \frac{1}{3}log\frac{1}{3}) + \frac{2}{5}(-\frac{1}{2}log\frac{1}{2} - \frac{1}{2}log\frac{1}{2}) = 0.951 \tag{13}$$

$$H(x|Humidity) = \frac{3}{5}(-\frac{2}{3}log\frac{2}{3} - \frac{1}{3}log\frac{1}{3}) + \frac{2}{5}(-\frac{1}{2}log\frac{1}{2} - \frac{1}{2}log\frac{1}{2}) = 0.951 \tag{14}$$

$$H(x|Wind) = \frac{3}{5} * 0 + \frac{2}{5} * 0 = 0 \tag{15}$$

Thus we choose wind as the feature, no matter wind choose strong or weak, we all get pure nodes.
ID3 algorithm stop here, we can output the decision tree:



Problem 5

Answer: Generative classifiers learn a model of the joint probability, p(x,y), of the inputs x and the label y, and make their predictions by using Bayes rules to calculate $p(y|x)$, and then picking the most likely label y. Descriminative classifiers model the posterior $p(y|x)$ directly, or learn a direct map from inputs x to the class labels.
In this paper, the authors try to compare the generative and discriminative learning by considering the naive Bayes model and its discriminative analog, logistic regression/ linear classification, and show: a)The generative model does indeed have a high asymtotic error than the discriminative model which complies with people's thinking that "one should solve the problem directly and never solve a more general problem as an intermediate step". However, the generative model may also approach its asymptotic error much faster than the discriminative model.
The results imply that even though the discriminative logistic regression algorithm has a lower asymptotic error, the generative naive Bayes classifier may also converge more quickly to its higher asymptotic error. Thus, as the number of training examples m is increased, one would expect generative naive Bayes to initially do better, but for discriminative logistic regression to eventually catch up to, and quite likely overtake, the performance of naive Bayes.

Problem 6

Answer:
a)When the training data approaches infinity and under conditional independence assumptions, the Logistic regression is discriminative counterpart of Naive Bayes. Therefore, if the data satisfies conditional independence assumptions, Naive Bayes and Logistic Regression will produce equivalent results.

b)Logistic Regression will produce better results, since it doesn't assume that data satisfies conditional independence.

Problem 7

Answer:
a)Yes
Naive Bayes is a generative classifier. We can obtain P(X) by marginalizing $P(X|Y)$ over the class variable. For instance:

$$P(X) = \sum_y P(X|Y = y)P(Y = y) \tag{16}$$

b)No
Logistic regression is a discriminative classifier, that estimates $P(Y|X)$, not $P(X|Y)$. In order to calculate $P(X)$, we need to know $P(X|Y)$.

Problem 8

Answer:
a) For each training example (x_i, y_i),

$$logP(Y; w) = log \prod_i P(x_i, y_i; w) = \sum_i logP(x_i, y_i; w) \tag{17}$$

Joint distribution

$$P(y, x) = [\arg\max_w log(Y; w)] = [\arg\max_w \sum_i logP(x_i, y_i; w)] = [\arg\max_w \sum logP(y_i|x_i; w)P(x_i; w)] \tag{18}$$

Calculate the weights, W is the parameter.

$$\mathrm{W}^* = \underset{W}{\operatorname{argmax}} \prod_{i=1}^{n} P(y_i|x_i, W)$$

Calculate the log-likelihood:

$$\mathrm{l(W)} = \log \prod_{j=1}^{n} \prod_{k=1}^{K} \mu_{jk}^{y_{jk}} log\mu_{jk} = \sum_{j=1}^{n} (\sum_{k=1}^{K} y_{jk} w_k x_j - log \sum_{i=1}^{K} e^{w_i x_j})$$

$$\mu_{jk} = \mathrm{P}(y_j = k|x_j, W), y_{jk} = 1_{\{y_j = k\}}, \text{and } 1_{\{\cdot\}}$$

Do gradient ascent update rules:

$$g_k(W) = \nabla_{w_k} l(W) = \frac{\partial}{\partial w_k} \sum_{j=1}^{n} \left(\sum_{i=1}^{K} y_{ji} w_i x_j - log \sum_{i'=1}^{K} e^{w_{i'} x_j} \right)$$

$$= \sum_{j=1}^{n} \left(\frac{\partial}{\partial w_k} \sum_{i=1}^{K} y_{ji} w_i x_j - \frac{\partial}{\partial w_k} log \sum_{i'=1}^{K} e^{w_{i'} x_j} \right)$$

$$= \sum_{j=1}^{n} \left(y_{jk} x_j - \frac{\frac{\partial}{\partial w_k} \sum_{i'=1}^{K} e^{w_{i'} x_j}}{\sum_{i'=1}^{K} e^{w_{i'} x_j}} \right)$$

$$= \sum_{j=1}^{n} \left(y_{jk} x_j - \frac{e^{w_{i'} x_j} x_j}{\sum_{i'=1}^{K} e^{w_{i'} x_j}} \right)$$

$$= \sum_{j=1}^{n} (y_{jk} - \mu_{jk}) x_j$$

b) we add a regularization term:
Adding a regularization term to the likelihood objective:

$$\underset{W}{\operatorname{argmax}}\left\{\sum_i logP\left(y^i\middle|x^i,w\right)-\lambda\left||w|\right|^2\right\}$$

$$\mathrm{w}=\mathrm{w}+\eta\left(\sum_i(y^i-\hat{u}^i)x^i-\lambda w\right)$$

$$\nabla_{w_k}l(W)=\sum_{i=1}^{n}\left(y_{ik}-\hat{u}_{ik}\right)x_i$$

Problem 9

Answer:

a)
Implementation please see the code.
b)
validation accuracy=58.8%, and testing accuracy=67.4%
Please note that the validation set is randomly shuffled, the accuracy results in each run is different.
c)
Implementation please see the code.
d)
validation accuracy =67.187% and testing accuracy=73.267%.
The pruning process successfully reduces the tree size and improves the accuracy performance a little bit with the tradeoff of more computation power required.
e)
Training accuracy=93.167%, and testing accuracy=77.227%
f)
Training accuracy=100%, and testing accuracy=71.2871%