Homework 2

Reet Barik, WSU ID: 11630142
CptS 570 Machine Learning

October 29, 2018

Exercise 1. Suppose x = (x1, x2, ... , xd) and z = (z1, z2, ... , zd) be any two points in a high-dimensional space (i.e., d is very large)
a) Try to prove the following, where the right-hand side quantity represent the standard Euclidean distance. $(\frac{1}{\sqrt{d}}\sum_{i=1}^{d} x_i - \frac{1}{\sqrt{d}}\sum_{i=1}^{d} z_i)^2 \leq \sum_{i=1}^{d}(x_i - z_i)^2$
b) We know that the computation of nearest neighbors is very expensive in the high-dimensional space. Discuss how we can make use of the above property to make the nearest neighbors computation efficient?

Answer:

a) To prove:

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

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

Let $x_i - z_i = y_i$. Then substituting in (1), we get

$$\frac{1}{d}(\sum_{i=1}^{d} y_i)^2 \leq \sum_{i=1}^{d}(y_i)^2 \text{.....(2)}$$

Now, we know from Jensen's Inequality, f(E[y]) $\leq$ E[f(y)]. If f(y) = y^2,

$$f((\frac{\sum_{i=1}^{d}(y_i)}{d})^2) \leq \frac{\sum_{i=1}^{d}(y_i)^2}{d}$$

$$f(\frac{(\sum_{i=1}^{d}(y_i))^2}{d^2}) \leq \frac{\sum_{i=1}^{d}(y_i)^2}{d}$$

$$\frac{(\sum_{i=1}^{d}(y_i))^2}{d} \leq \sum_{i=1}^{d}(y_i)^2$$

Substituting y_i back in, we get:

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

b) From the Right Hand side of the given result, we can see that we are computing the Euclidean distance between the two points. The computation in d dimensions will result in a complexity of O(nd).
From the Left Hand side of the given result, it can be observed that we are computing the average of each summations. This reduction in dimension results in the computational complexity of O(n + d). Clearly, this way is much more efficient than the other.

Exercise 2. We briefly discussed in the class about Locality Sensitive Hashing (LSH) algorithm to make the nearest neighbor classifier efficient. Please read the following paper and briefly summarize the key ideas as you understood:
Alexandr Andoni, Piotr Indyk: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Communications of ACM 51(1): 117-122 (2008)
http://people.csail.mit.edu/indyk/p117-andoni.pdf

Answer:

Introduction:
The definition of the nearest neighbor problem states that given a query point, we need to come up with a data structure that returns the point closest to the query point. The points in question exist in a d-dimensional Euclidean space. For low value of d, there already exist efficient algorithms like kd-trees. But these tend to suffer from "the curse of dimensionality" because for a very large d, either the space or query time is exponential in d. This bottleneck may be overcome by approximation where the idea is that an approximate nearest neighbor is as good as the exact nearest one. The article's point of focus is on such algorithms which perform approximate search in high dimensions based on the concept of Locality-Sensitive Hashing. Here, several hash functions are used in such a way that the probability of collision between objects or points that are closer to each other is very high and vice versa. Nearest neighbors can then be chosen from the bucket the query points belongs to. This has found success in areas like web clustering, computational biology, computer vision etc. Another objective of this article is the description of "a recently developed LSH family for the Euclidean distance, which achievies a near-optimal separation between the collision probabilities of close and far points".

Problem Definition:
If we are given a set of points P in a d-dimensional space $\mathbb{R}^d$ and a query point q, then we need to come up with a data structure that will return an R-near neighbor of q in P with a non-zero probability. This can be extended to return each R-near neighbor of q in P with a non-zero probability. [Note: If point p1 is an R-near neighbor of p2, it means that that the distance between p1 and p2 can be at most R].

Locality-Sensitive Hashing:
The algorithm can be broken down into two parts.
Pre-processing: L functions are chosen $g_j, j = 1, 2, ..., L$ by setting $g_j = (h_{1,j}, h_{2,j}, .., h_{k,j})$ for all $h_{i,j}$ are chosen at random from the LSH family. L-many hash tables are constructed where, for each j, the j^{th} hash table contains the dataset points using the function g_j.
Query for q in P: For j in 1,2,...L, from the j^{th} hash table we retrieve all the points from the bucket corresponding to the bucket $g_j(q)$. For each point retrieved, we compute their distances from q and return that/those point/points which satisfy the consition described in the problem definition.

Exisiting LSH Libraries:
The following LSH families that have been discovered have been surveyed in this article:
1. Hamming distance: Hamming distance between two binary vectors is the minimum number of substituions required to transform one to the other. For M-ary vecotrs with small M, there exists an efficient reduction to result in the l_1 distance between those two vectors.
2. l_1 distance: This distance is based on the number of steps required to transform one vector to another using taxicab geometry.
3. l_s distance: A vector r is picked from the so-called s-stable distribution or the Gaussian distribution and its projection onto x is used to define the similarity metric.
4. Jaccard: This is mainly used to compute the similarity between sample sets using the Jaccard coefficient which is the size of the intersection divided by the size of the union of the sample sets.
5. Arccos: This is a similarity metric between two vectors based on the angle between them.
6. l_2 distance: This algorithm projects points into a unit hypersphere in the Euclidean space in order to compute the distance between two points.

Near-Optimal LSH Function for Euclidean Distance:
A new family of LSH is presented in this section with query time of the order of $1/c^2$ for large n. This is done by 'ball-partitioning' of the projection space instead of the 'grid-partitioning' approch used in previous attempts. Locating a cell containing a given point wouldn't require unbounded amount of time if each ball is replaced by a grid of balls. This will result in a finite amount of balls to be enough to cover all points in the projected space.

Exercise 3. We know that we can convert any decision tree into a set of if-then rules, where there is one rule per leaf node. Suppose you are given a set of rules R = r1, r2, , rk, where ri corresponds to the ith rule. Is it possible to convert the rule set R into an equivalent decision tree? Explain your construction or give a counterexample.

Answer:

To answer this question we need to take a look at the set of rules that we are given.

Case 1: For any given set of rules:
We can prove by a counterexample that it's not possible to form a decision tree for any set of rules provided:
Example 1:
Rule 1: if (x > 0) then y = 1
Rule 2: if (x > 0) then y = 0
Here, we have two contradicting rules on the same feature. And it is obvious that a decision tree cannot be constructed from such rules.
Example 2:
Rule 1: if (Outlook == Sunny AND Humidity == Normal)
Rule 2: if (Outlook == Sunny AND Temperature == Low)
Here, we can see that to construct a decision tree, the branch corresponding to Outlook == Sunny has 2 options as far as the next attribute to split on is considered. Due to this ambiguity, it is not possible to construct a decision tree.

Case 2: For a set of rules that is obtained from a decision tree:
If we are given a set of rules that have been obtained from a decision tree, it is possible to reconstruct the decision tree for that given set of rules as follows:
Rearrange all rules so that the first attribute checked is same for all rules. That attribute corresponds to the split attribute at the root node of the decision tree. The subset of rules where the first check is of the form Attribute1 == Value1, correspond to one branch of the root node. This way we can partition the set of rules. For each subset of rules corresponding to one branch coming out of the root node, we follow the process recursively to find the next node and split condition for that branch. We continue this process till all the rules are exhausted.

Exercise 4. You are provided with a training set of examples (see Figure 1). Which feature will you pick first to split the data as per the ID3 decision tree learning algorithm? Show all your work: compute the information gain for all the four attributes and pick the best one.

Answer:

In ID3, we choose the best feature to split on based on the Information Gain of each feature. Gain of a set S for a feature A is given by G(S,A) = Entropy(S) - Entropy(S | A).
Here, Entropy(S) = - (5/14) log (5/14) - (9/14) log (9/14) = 0.94
Now,
Gain(S, Outlook) = Entropy(S) - Entropy(S | Outlook)
= 0.94 - 5/14 (- (3/5) log (3/5) - (2/5) log (2/5)) - 4/14 (- (4/4) log (4/4) -(0/4) log (0/4))
- 5/14 (- (3/5) log (3/5) - (2/5) log (2/5))
= 0.246
Gain(S, Temperature) = Entropy(S) - Entropy(S | Temperature)

= 0.94 - 4/14 (- (2/4) log (2/4) - (2/4) log (2/4)) - 6/14 (- (4/6) log (4/6) -(2/4) log (2/4)) - 4/14 (- (3/4) log (3/4) - (1/4) log (1/4))
= 0.029
Gain(S, Humidity) = Entropy(S) - Entropy(S | Humidity)
= 0.94 - 7/14 (- (3/7) log (3/7) - (4/7) log (4/7)) - 7/14 (- (6/7) log (6/7) -(1/7) log (1/7))
= 0.151
Gain(S, Wind) = Entropy(S) - Entropy(S | Wind)
= 0.94 - 8/14 (- (6/8) log (6/8) - (2/8) log 2/8)) - 6/14 (- (3/6) log (3/6) -(3/6) log (3/6))
= 0.048

We can see that the Informaton Gain is maximum for the attribute 'Outlook'. Therefore, 'Outlook' will be picked first to split the data as per ID3 decision tree learning algorithm.

Exercise 5. Please read the following paper and briefly summarize the key ideas as you understood (You can skip the proofs, but it is important to understand the main results):
Andrew Y. Ng, Michael I. Jordan: On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes. NIPS 2001: 841-848 http://ai.stanford.edu/∼ang/papers/nips01-discriminativegenerative.pdf

Answer:

Introduction:
Discriminative classifiers learn P(Y | X) directly from the training data while the Generative classifiers learn a joint distribution P(X, Y) and use that to compute P(Y | X). For the purpose of the paper, Logistic Regression has been taken as a representative classifier for Discriminative and Naive Bayes Classifier has been taken as a representative classifier for Generative. This pair of classifiers is then used to show that the asymptotic error for Generative models is higher than that of Discriminative models. And also, Generative models approach asymptotic error faster than Discriminative models.

Preliminaries:
In case of Generative classifier, If $X = \{0, 1\}^n$ be some discrete data, in the n-dimensional space, $Y = \{T, F\}$ be the output labels, and a joint distribution D over the training set S. Then a Naive Bayes model will predict T, only if the following condition is positive:

$$l_{Gen}(x) = \sum_{i=1}^{n} log \frac{\hat{p}(x_i|y = T)}{\hat{p}(x_i|y = F)} + log \frac{\hat{p}(y = T)}{\hat{p}(y = F)}$$

if X is continuous such that $X = [0, 1]^n$, then the distribution is considered to be a Gaussian one.
In case of Discriminative classifier, it predicts T only if the following linear discrimininent fuction is positive:

$$l_{Dis}(x) = \sum_{i=1}^{n} \beta_i x_i + \theta$$

Analysis of Algorithms:
The following inferences can be drawn with respect to Naive Bayes Classifier and Logistic Regression classifier:
1. $Error(h_{Dis,\infty}) \leq Error(h_{Gen,\infty})$, where h_{Gen} and h_{Dis} are a generative-discriminative pair.
2. The number of examples needed to reach asymptotic error for Logistic Regression model or a Discriminative Classifier is of the order of n.
3. Lemma 3 says that number of examples required to train a Logistic Regression model is of the order of $O(n)$ while that for a Naive Bayes model is $O(log(n))$. This can be generalised to all pairs of Discriminative-Generative model pairs.
4. Theorem 4 says that given an $Error(h_{Gen})$ of a Generative classifier will reach the $Error(h_{Gen,\infty})$ with an upper bound of $G(O(\sqrt{\frac{1}{m}logn})))$ and not take unbounded time.
5. This proposition states that even if we choose $\omega(1)$ fraction of features, the expected $l_{Dis,\infty}(x)$ will be of the order $\omega(n)$, which is very large. So the chance of choosing the right random variable from that is near zero.
6. Suppose $G(.) \leq E_0/2 + F(.)$ and some $E_0 > 0$ while the assumptions of theorem 4 hold, then, the bound in theorem 4 to hold with high probability, it suffices to pick m = $\Omega(logn)$

Conclusions:
The inference from the whole exercise has been succintly put in the paper as follows: Even though Discriminative learning has a lower aymptotic error, Generative learning may catch up faster to it's higher asymptotic error. Hence, with increasing training example, Naive Bayes initially does better but Logistic Regression eventually catches up and ends up performing better.

Exercise 6. a. Let us assume that the training data satisfies the Naive Bayes assumption (i.e., features are independent given the class label). As the training data approaches infinity, which classifier will produce better results, Naive Bayes or Logistic Regression? Please explain your reasoning.
b. Let us assume that the training data does NOT satisfy the Naive Bayes assumption. As the training data approaches infinity, which classifier will produce better results, Naive Bayes or Logistic Regression? Please explain your reasoning.

Answer:

a) The formulation used to compute P(Y | X) in case of Naive Bayes can be used to derive the formulation of P(Y | X) for Logistic Regression. This transformation from one form to other is shown below:

$$
\begin{aligned}
P(Y=1 \mid X) &= \frac{P(Y=1)P(X \mid Y=1)}{P(Y=1)P(X \mid Y=1) + P(Y=0)P(X \mid Y=0)} \\
&= \frac{1}{1 + \frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)}} \\
&= \frac{1}{1 + exp(ln(\frac{P(Y=0)}{P(Y=1)})) + exp(ln(\frac{P(X|Y=0)}{P(X|Y=1)}))} \\
&= \frac{1}{1 + exp(w_0 + \sum_{i=1}^{n} w_i X_i)}
\end{aligned}
$$

Despite the apparent interchangeability of form between the Logistic Regression Classifier and the Naive Bayes Classifier, we know from the observations obtained from the previous question that with increasing training example, Naive Bayes initially does better but Logistic Regression eventually catches up and ends up performing better.

b) If the training data does NOT satisfy the Naive Bayes Assumption, then the following result will hold true:

$$
P(X \mid Y) \neq \prod_{i=1}^{d} P(X_i \mid Y)
$$

Hence, the asymptotic error of Naive Bayes classifier will be higher than that of Logistic Regression classifer which doesn't need to depend on the independance of the features given a class label. Hence, we can conclude that the Logistic Regression Classifier will perform better than Naive Bayes Classifier as training data approaches infinity.

Exercise 7. a. Can we compute P(X) from the learned parameters of a Naive Bayes classifier? Please explain your reasoning.
b. Can we compute P(X) from the learned parameters of a Logistic Regression classifier? Please explain your reasoning.

Answer:

a) In Naive Bayes Classifier, we compute P(Y | X) using Bayes Rule after learning P(Y) and P(X | Y) as shown below:

$$
P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} => \frac{P(X|Y)P(Y)}{\sum_Y P(X,Y)}
$$

Given that our features are binary, we can see that,

$$
P(X) = \sum_Y P(X,Y) = P(X|Y)P(Y) + P(X|\neg Y)P(\neg Y)
$$

Since P(Y) is known, P(¬Y) can be computed. The value of P(X |¬Y) can be computed from the data. Hence, using these two P(X) can be easily computed from learned parameters of Naive Bayes Classifier.

b) In Logistic Regression, we compute P(Y|X) without learning P(Y) or P(X|Y). Hence, we cannot compute P(X) from the learned parameters of a Logistic Regression Classifier.

Exercise 8. In the class, we looked at the log-likelihood derivation and the corresponding gradient ascent algorithm to find the parameters of a binary logistic regression classifier (see slide 12 and slide 13). We want to extend the log-likelihood derivation and parameter learning algorithm to the multi-class case. Suppose we have K different classes, and the posterior probability can be represented using the so-called soft-max function (see slide 18):
$P(y = k|x) = \frac{exp(w_k.x)}{\sum_{i=1}^{K} exp(w_i.x)}$
a. Derive the log-likelihood and the corresponding gradient ascent algorithm to find the parameters.
b. Add a regularization term to the log-likelihood objective (see slide 16), and derive the gradient ascent update rule with the additional change.

Answer:

a) The log-likelihhod Derivation and paramter learning for multiclass case is done using one hot encoding of output label

$$L = \sum_{j=1}^{k} y_j \log \widehat{y}_j$$

$$y_j = \frac{exp(w_j x)}{\sum_{i=1}^{k} exp(w_i x)}$$

Taking partial derivative on L

$$\frac{\partial L}{\partial w} = \frac{\partial}{\partial w} \sum_{j=1}^{k} y_j \log \widehat{y}_j$$

Solving it further will give us

$$\frac{\partial L}{\partial w} = \sum y_j \frac{\partial}{\partial w} \log \widehat{y}_j$$

$$\frac{\partial L}{\partial w} = \sum y_j \frac{1}{y_j} \frac{\partial \widehat{y}_j}{\partial w} \quad .....(1)$$

Substituting for y_j to find the partial derivative $\frac{\partial \widehat{y}_j}{\partial w}$

$$\frac{\partial \widehat{y}_j}{\partial w} = \frac{\partial}{\partial w} \left(\frac{exp(w_j x)}{\sum exp(w_k x)} \right)$$

$$\frac{\partial \widehat{y}_j}{\partial w} = \frac{x \quad exp(w_j x) \quad \sum(w_k x) - x \quad exp(w_j x) exp(w_j x)}{(\sum exp(w_k x))^2}$$

$$\frac{\partial \widehat{y_j}}{\partial w} = x\left[\frac{exp(w_j x)}{\sum exp(w_k x)} - \left(\frac{exp(w_j x)}{\sum exp(w_k x)}\right)^2\right]$$
$$= x[\widehat{y_j} - \widehat{y_j}^2]$$

Substituting this on (1), we get

$$\frac{\partial L}{\partial w_j} = \sum y_j \quad x(\widehat{y_j} - \widehat{y_j}^2) \quad .....(2)$$

The above holds good for the correct class of j, For incorrect class of j will give us

$$\frac{\partial \widehat{y_j}}{\partial w_k} = \frac{\partial}{\partial w_k}\left(\frac{exp(w_j x)}{\sum exp(w_k x)}\right)$$
$$= -\widehat{y_j}\widehat{y_k}x$$

Substituting in (1) will give us

$$\frac{\partial L}{\partial w_k} = -\widehat{y_j}\widehat{y_k}x.....(3)$$

The general form from (2) and (3) can be written as follows

$$\frac{\partial y_j}{\partial w} = (y_j - \widehat{y_j})x^i \quad .....(4)$$

b) The log-likelihood objective is given by

$$L = arg \quad \max_w \left\{ \sum_i \log P(y^i|x^i, w) - \lambda \parallel w \parallel^2 \right\}$$

In the above equation the regularization term is $\lambda \parallel w \parallel^2$. Now, we can say that:

$$\frac{\partial L}{\partial w} = \frac{\partial}{\partial w}\left(\sum \log P(y^i|x^i, w)\right) - \frac{\partial}{\partial w}(\lambda \parallel w \parallel^2)$$

From (4) we try to find derivative for the above equation, which will be

$$\frac{\partial L}{\partial w} = (y_j - \widehat{y_j})x^i - \lambda\frac{\partial}{\partial w}(\parallel w \parallel^2)$$

$$\frac{\partial L}{\partial w} = (y_j - \widehat{y_j})x^i - \lambda(\parallel w \parallel)$$

which is the gradient ascent with regularization term.

Exercise 9. a. Implement the Naive Bayes Classifier (with Laplace Smoothing) and run it on the training data. Compute the training and testing accuracy.
b. Run the off-the-shelf Logistic Regression classifier from Weka (weka.classifiers.functions.Logistic) or scikit-learn on the training data. Compute the training and testing accuracy.

Answer:

a) Naive Bayes' Classifier:
Training accuracy: 0.9316770186335404
Testing accuracy: 0.7821782178217822

b) Scikit-learn Logistic Regression Classifier:
Training accuracy: 0.9440993788819876
Testing accuracy: 0.7821782178217822

Exercise 10. a) Implement the ID3 decision tree learning algorithm that we discussed in the class. The key step in the decision tree learning is choosing the next feature to split on. Implement the information gain heuristic for selecting the next feature. Please see lecture notes or https://en.wikipedia.org/wiki/ID3_algorithm for more details. I explained how to select candidate thresholds for continuous features: Sort all candidate values for feature f from training data. Suppose f1, f2, ... , fn is the sorted list. The candidate thresholds are chosen as fi + (fi+1 fi)/2 for i=1 to n.
b) Run the decision tree construction algorithm on the training examples. Compute the accuracy on validation examples and testing examples.
c) Implement the decision tree pruning algorithm discussed in the class (via validation data).
d) Run the pruning algorithm on the decision tree constructed using training examples. Compute the accuracy on validation examples and testing examples. List your observations by comparing the performance of decision tree with and without pruning.

Answer:

a) Code submitted for review in the designated dropbox folder.

b) The following values were observed:
Training accuracy: 1.0
Testing accuracy: 0.6559440559440559
Dev accuracy: 0.6578249336870027

c)

d)