CptS 570 MACHINE LEARNING HOMEWORK-3

Sharath Kumar Karnati

10th November 2024

---

1 Analytical Part (6 percent)

Questions

Question 1: Suppose $x = (x_1, x_2, \cdots, x_d)$ and $z = (z_1, z_2, \cdots, z_d)$ 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 represents the standard Euclidean distance:

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

Hint: Use Jensen's inequality – If X is a random variable and f is a convex function, then

$$f(E[X]) \leq E[f(X)].$$

Solution : Approach 1:

We know that the standard Euclidean distance between two points x and z is given by:

$$d(x, z) = \sqrt{(x_1 - z_1)^2 + (x_2 - z_2)^2 + \cdots + (x_d - z_d)^2}$$

Squaring both sides:

$$d^2(x, z) = (x_1 - z_1)^2 + (x_2 - z_2)^2 + \cdots + (x_d - z_d)^2$$

By applying Jensen's inequality, which states that for a convex function f:

$$f(E[X]) \leq E[f(X)]$$

If we consider $f(x) = x^2$, we can write:

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

Thus, we conclude:

$$\sum_{i=1}^{d} f((x_i - z_i)^2) \leq d^2(x, z)$$

This shows that the left-hand side is bounded by the squared Euclidean distance, i.e., the Euclidean distance serves as an upper bound.

Approach 2:

Now, let's define $m_i = (x_i - z_i)$ as the difference between the components of x and z. We can express the average difference as:

$$\frac{1}{\sqrt{d}} \sum_{i=1}^{d} |x_i - z_i|$$

Squaring both sides, we obtain:

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

By applying Jensen's inequality again, we have:

$$f\left(\frac{1}{d} \sum_{i=1}^{d} m_i \right) \leq \frac{1}{d} \sum_{i=1}^{d} f(m_i)$$

Thus, applying this to the sum of squared differences:

$$\frac{1}{d} \left(\sum_{i=1}^{d} m_i \right)^2 \leq \frac{1}{d} \sum_{i=1}^{d} m_i^2$$

This demonstrates that the left-hand side (the normalized sum of absolute differences) is less than or equal to the squared Euclidean distance. When d is large, this approximation becomes tighter, which is helpful for efficiently computing nearest neighbors in high-dimensional spaces.

(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:

From **Approach 1**, we can see that computational efficiency is improved since calculating the squared Euclidean distance avoids the need for the square root operation. The nearest neighbors search becomes faster as we no longer need to compute the exact Euclidean distance, yet the relative distances between points remain the same.

From **Approach 2**, the left-hand side involves computing the difference between each coordinate of points x and z, summing these differences, and then squaring the result. This method is computationally simpler than calculating the full Euclidean distance (on the right-hand side), which involves summing the squares of differences and taking a square root. Thus, using this approach can reduce the computational complexity in nearest neighbor searches, resulting in faster computations.

Question 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 challenge of constructing an efficient data structure for identifying the closest point from a set of n points to a given query point is referred to as the nearest neighbor problem. This problem becomes more complex as the dimensionality of the points increases. A number of algorithms have been devised to address this issue, but they often struggle with the exponential growth in both time and space complexity as the number of dimensions, d, increases.

To mitigate this, various approximation techniques have been introduced. One such technique is the *Randomized c-approximate R-near neighbor* or (c, R)-NN. This approach builds a data structure that, with a certain probability, can efficiently identify a point within a distance cR of a given query point. By constructing multiple instances of this data structure, the likelihood of success increases. Assuming $R = 1$ or normalizing all points by R simplifies the problem, leading to what is known as the *c-approximate near neighbor problem* or *c-NN*.

Locality-Sensitive Hashing (LSH) A popular technique for approximating nearest neighbors in high-dimensional spaces is Locality-Sensitive Hashing (LSH). In this approach, two points p and q are considered to be R-near neighbors if the distance between them is at most R. Let H be a collection of hash functions that map points from R^d into a universe U. The general idea is that points that are close to each other in the original space are hashed to nearby locations in U, while distant points are hashed to farther locations. A family H is called locality-sensitive if it satisfies certain properties that ensure this behavior.

In particular, the LSH method computes distances using the Euclidean metric and employs a hash function to assign probabilities to points. A higher probability indicates that a point is close to the query point, while a lower probability suggests that the point is farther away.

High-Dimensional Nearest Neighbor Search The problem of nearest neighbor search in high-dimensional spaces, particularly for k-NN, involves handling the so-called fragmentation zone sensitivity. The aim is to minimize the distance between two points P and Q, where R defines the maximum allowable distance. Hash functions tend to produce more collisions for points that are far apart, and R is used as a threshold for neighbor proximity. A locality-sensitive hashing region is only used by the algorithm when it satisfies the following conditions:

$\mathrm{P}_H[h(q) = h(p_1)] \geq P_1$,
$P_H[h(q) = h(p_1)] \leq P_2, \quad where \quad P_1 > P_2$.

Using these criteria, the nearest neighbor algorithm is developed based on the family of hash functions H. The differences between P_1 and P_2 are not substantial, allowing for customization in the algorithm's behavior. The customization involves creating a data structure, $g(P)$, where j is a variable ranging from 1 to L. At designated locations, algorithmic queries are facilitated by this

data structure. For each L, points are taken from $g(Q)$, and the point with the shortest distance to Q is returned.

Randomized vs Non-Randomized Approaches The paper discusses both randomized and non-randomized strategies for determining the closest point. In the non-randomized approach, a value of R is chosen that closely matches the neighborhood requirements, with C serving as the proximity factor. In contrast, the randomized method selects R randomly.

Additionally, the paper explores the concept of the Hamming distance, which is vital in the LSH framework, particularly for binary vectors.

Question 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,$\cdots$,rk, where ri corresponds to the ith rule. Is it possible to convert the rule set Rinto an equivalent decision tree? Explain your construction or give a counterexample.

Answer:

The process of converting a rule set R into a decision tree is straightforward because each path in the decision tree corresponds to a combination of rules, which effectively partitions the space into regions filled with data points.

To begin, the rules are combined since we know that a decision tree can be constructed to mirror the rule set R. For each criterion, the numerical values are sorted into a list, and then the list is split in half using a dichotomous approach, where the middle value serves as the reference point. This recursive splitting continues until each list is fully divided. The algorithm first eliminates any attributes that do not appear in any of the rules. For example, given attributes m_1, m_2, and m_3 with values n_1, n_2, and n_3, respectively, consider the following:

- Rule $r_1 : m_1 = n_1$,
- Rule $r_2 : m_1 = n_3 and m_2 = n_2$.

Any missing attributes in the rules are treated as null to ensure consistency. Thus, the rules become:

- $r_1 : m_1 = n_1, m_2 = n_2, m_3 =$ "$notvalid$",
- $r_2 : m_1 = n_3, m_2 = n_2, m_3 =$ "$notvalid$".

At this point, one attribute is chosen as the root node, and the remaining attributes branch out from it. Three lists are generated based on various factors, such as influence, dependency, and the number of valid values for each attribute. The first list prioritizes attributes with the least occurrence of "not valid." The second list ranks attributes with similar "not valid" counts, choosing those with the least dependency on other attributes. Finally, the third list considers attributes with the fewest distinct values. When multiple attributes have similar counts, the most suitable attribute for the given condition is chosen. This process is repeated recursively until no more relevant attributes remain.

Any remaining rules that couldn't initially be assigned are distributed among the branches, and the branches are further divided into child nodes. If all the rules at a node consistently classify the data into the same class, the node becomes a leaf node.

Question 4 :

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: In generative classifiers, the joint probability $p(x, y)$ is considered, where x represents the input and y represents the class label. Using Bayes' rule, predictions are made by determining $p(y|x)$ and selecting the class label with the highest likelihood. Naive Bayes and other generative classifiers tend to show an increasing asymptotic error as the number of examples grows, and they converge to this error faster when the dataset size is smaller.

Discriminative classifiers, on the other hand, either directly model the posterior $p(y|x)$ or map the input x to the class label y. For these classifiers, the sample complexity is linear in the VC (Vapnik–Chervonenkis) dimension, which means the number of parameters used in the VC dimension grows polynomially or linearly. Logistic regression is an example of a discriminative classifier.

The performance of these classifiers is measured by examining if the generative classifier has already reached its asymptotic error or if the discriminative classifier is approaching its lower asymptotic error. Let h_{gen} and h_{dis} represent the generative and discriminative classifiers, respectively, and let $h_{gen,\infty}$ and $h_{dis,\infty}$ represent their asymptotic versions. In general, discriminative classifiers like logistic regression have a lower asymptotic error compared to generative classifiers like Naive Bayes, meaning $\epsilon(h_{gen,\infty}) \geq \epsilon(h_{dis,\infty})$.

To approach their respective asymptotic errors, discriminative classifiers need $O(n)$ training examples, while generative classifiers require $O(\log n)$ examples. Initially, generative classifiers perform better than discriminative classifiers when the number of training examples is low, as they converge faster. However, as more examples become available, the discriminative classifier eventually surpasses the generative classifier in terms of performance, achieving a lower asymptotic error.

Question 5 :

Naive Bayes vs. Logistic Regression 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.

Answer:

Naive Bayes often delivers more accurate results due to its assumption that the features in the training data are independent. This leads to faster and more precise convergence of the algorithm's outputs. According to Dietterich (1998) (accessible at http://sci2s.ugr.es/keel/pdf/algorithm/articulo/dietterich1998.pdf), accuracy tends to decrease when there is a high degree of correlation between variables and they are not independent. Since Naive Bayes depends on the assumption of feature independence, it consistently performs better in such scenarios.

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: It is acknowledged that as the size of the training data approaches infinity, logistic regression tends to outperform and yield more accurate results. While Naive Bayes is generally recognized for its effectiveness, it comes with a higher computational cost. As the dataset grows, Naive Bayes struggles to fit the data due to the rapid increase in feature combinations, leading to potential inefficiencies. In contrast, logistic regression usually excels in such scenarios, as it is better equipped to manage the challenges of high-dimensional feature spaces.

c. Can we compute P(X) from the learned parameters of a Naive Bayes classifier? Please explain your reasoning.

Answer: In the Naive Bayes classifier, it is possible to calculate $P(x)$. It's important to note that Naive Bayes doesn't explicitly generate parameters. Instead, the classifier's parameters need to be derived to compute the full probability distribution. This is done through marginalization, which leverages the set of conditional probabilities. A comparable approach is applied in coding, where $P(x)$ is computed in part by determining the probability for terms such as "wise" and "future."

d. Can we compute P(X) from the learned parameters of a Logistic Regression classifier? Please explain your reasoning.

Answer:

In logistic regression, calculating $P(x)$ directly is not feasible. Instead, logistic regression focuses on using conditional probabilities to estimate $P(x)$. As a discriminative model, logistic regression is concerned with determining the conditional probabilitiess based on the given data.

Question 6: Please read the following paper and briefly summarize the key ideas as you understood: Thomas G. Dietterich: Ensemble Methods in Machine Learning. Multiple Classifier Systems 2000: 1-15 https://web.engr.oregonstate.edu/ tgd/publications/mcs-ensembles.pdf

Answer:

The aim is to generate classifiers for each set of training instances (denoted by S). A classifier represents the actual function of f's hypotheses. These classifiers can be combined into ensembles, which make collective decisions on new data. To perform better than individual classifiers, ensemble classifiers must be both accurate and diverse. For example, if one classifier in an ensemble makes an incorrect prediction, the others may still make the correct decision, ensuring that the overall ensemble provides the right classification. Thus, it is crucial for classifiers in an ensemble to have uncorrelated errors and to keep individual error rates below 0.5.

Here are some key points from the paper:

1. Ensemble methods classify new data using a weighted vote of predictions from multiple classifiers. These algorithms generate a group of classifiers that contribute to the final classification through a weighted voting system.

2. Based on a set of training cases S, a learning algorithm generates a classifier. This classifier predicts the corresponding y-values for new x-values and reflects a hypothesis about the true function f. These classifiers are denoted as $h_1, h_2, \ldots, h_l$. To classify new data, the predictions of the classifiers in the ensemble are aggregated, often through voting. Research in supervised learning has actively explored how to build efficient classifier ensembles, showing that groups of classifiers often outperform individual ones in terms of accuracy.

3. Classifiers are said to be diverse when their error rates on new X values resemble random guessing, meaning their accuracy is similar to that of a classifier making random predictions.

4. The reasons for building classifier ensembles are as follows:

   - **Statistical:** Ensembles help when there is insufficient training data to fully explore the hypothesis space and choose the one that best matches the true function f. By combining classifiers, ensembles allow for more accurate estimation of the true function.
   - **Computational:** In cases where reliable hypothesis generation may get stuck in local optima, ensembles provide a method to initiate searches from various points, improving the chances of finding an optimal solution and avoiding local optima.
   - **Representational:** If the true function f is not adequately captured by the entire hypothesis space, ensembles increase the representational capacity by combining hypotheses, better reflecting the complexities of the underlying function.

5. The methods for building ensembles include:

- **Bayesian Voting:** Uses the conditional probability distribution for each hypothesis h and calculates a weighted sum of all hypotheses for a new training example x.
- **Manipulating Training Instances:** Bootstrap replications of the original training set are used to calculate h on different subsets, a method known as bagging, which works well with data that contains unstable training instances.
- **Manipulating Input Features:** Hypotheses are calculated based on different subsets of features. This approach is effective when the dataset has redundant features.
- **Modifying Output Target:** The K-classes are divided into two subsets labeled as 1 and 0, and repeated iterations create different output groupings. These iterations form the ensemble.
- **Injecting Randomness:** Randomness is introduced into the ensemble-building process by running the algorithm with the same training data but different initial weights.

6. AdaBoost improves the weighted voting process to handle residual errors that prior trees could not correct. However, in noisy environments, AdaBoost can overfit, especially when it focuses too heavily on misclassified examples, which can lead to overfitting and reduced performance.

Question 7: We need to perform statistical tests to compare the performance of two learning algorithms on a given learning task. Please read the following paper and briefly summarize the key ideas as you understood: Thomas G. Dietterich: Approximate Statistical Test For Comparing Supervised Classification Learning Algorithms. Neural Computation 10(7): 1895-1923 (1998) http://sci2s.ugr.es/ keel/pdf/algorithm/articulo/dietterich1998.pdf

Answer: The design, analysis, and application of machine learning algorithms for classification present several challenges, and current statistical methods are often considered insufficient for addressing these issues. The study aims to address the following key question: "Given two algorithms A and B and a small dataset S, which algorithm would produce more accurate classifiers when trained on a dataset of similar size to S?" In response, the study introduces novel statistical strategies to address these challenges. In total, nine potential questions are explored in this research.

Given the small size of the dataset, the study utilizes resampling and holdout techniques to answer these questions. It is assumed that the size of the training set influences the performance of both learning algorithms.

Let f be the objective function that partitions the population's points into K classes (with $K = 2$), given a dataset X that serves as the population. From X, a random sample S is selected using a predefined probability distribution D. Pairs of the form $(x, f(x))$ are generated by applying $f(x)$ to every $x \in S$ to create the training examples.

A random selection is made from X for the test case using a fixed probability distribution. According to the null hypothesis, the error rates for both classification algorithms will be the same. Let f_A and f_B represent the respective results of algorithms A and B. The null hypothesis is formulated as follows:

$$\Pr[f'_A(x) = f_A(x)] = \Pr[f'_B(x) = f_B(x)]$$

The study presents five statistical tests summarized as follows:

1. **McNemar's Test:** A contingency table is constructed after using two algorithms to classify the test set. The total number of testing examples is given by:
$$n = Errormadebyboth f'_A$$
and f'_B (n_{00})+$Errormadeby f'_A$ but not by f'_B (n_{01})+$Errormadeby f'_B$ but not by f'_A (n_{10})+$Errormadebyneither f'_A$ nor f'_B (n_{11})$If the two algorithms, A$ and B, perform differently on a training set, the null hypothesis is rejected.

2. **A Test for the Difference of Two Proportions:** This test measures the error rate differential between A and B. The error rates for A and B are $P_A = \frac{n_{00}+n_{01}}{n}$ and $P_B = \frac{n_{00}+n_{10}}{n}$. If P_A and P_B are independent, $P_A - P_B$ is assumed to have a normal distribution. The data is used to calculate:
$$z = \frac{P_A - P_B}{\sqrt{2P(1-P)/n}}$$
where $P = \frac{P_A+P_B}{2}$. The null hypothesis is rejected if $|z| > 1.96$.

3. **Resampled Pair t-Test:** Two sets are created from a random sample S: a training set R and a testing set T. Misclassified instances are recorded after 30 trials. The null hypothesis states that the statistic follows a t-distribution with $n - 1$ degrees of freedom. If the value exceeds 2.04523, the null hypothesis can be rejected.

4. **k-Fold Cross-validated Paired t-Test:** K disjoint sets, $T_1, T_2, \ldots, T_K$, each of equal size, are created from set S. Each T_i is used as a test set while all other sets T_j (where $j \neq i$) are used for training. This method encounters the training set overlap problem when computing the statistic.

5. **5x2cv Paired t-Test:** This test is an extension of the 5x5 cross-validation paired t-test, where the numerator of the t-statistic is replaced with the observed difference from a single fold of k-fold cross-validation. After splitting the data into two equal-sized sets, S_1 and S_2, five replications of 2-fold cross-validation are performed. The training and testing sets are denoted as S_1 and S_2, respectively. Simulations with both test and real data yield successful results. The 5x2cv test is considered the most effective, although it may be more computationally expensive. McNemar's test also performs well.

Question 8:

(Finite-Horizon MDPs.) Our basic definition of an MDP in class defined the reward function R(s) to be a function of just the state, which we will call a state reward function. It is also common to define a reward function to be a function of the state and action, written as R(s,a), which we will call a state-action reward function. The meaning is that the agent gets a reward of R(s,a) when they take action ain state s. While this may seem to be a significant difference, it does not fundamentally extend our modeling power, nor does it fundamentally change the algorithms that we have developed. a) Describe a real world problem where the corresponding MDP is more naturally modeled using a state-action reward function compared to using a state reward function.

Answer :

Vehicle navigation and robotics are two real-world applications where a state-action reward function may be easier to represent than a state reward function.

Consider a scenario where an autonomous car or robot needs to navigate a dynamic, obstacle-filled environment. In such cases, the reward for a specific state may not fully capture the complex nature of the decision-making process. The optimal choice in a given situation may depend on both the specific action taken and the current state.

For example, a state-action reward function may assign a larger reward to an action that moves the robot away from an obstacle, thereby increasing safety when the robot is in close proximity to the obstruction. However, if the robot is in an open space, actions that effectively guide it towards its goal might be more rewarding.

By using a state-action reward function, the agent can differentiate between various actions in the same state, enabling more sophisticated decision-making. This modeling choice becomes important when the effects of an action depend not only on the current state but also on the specific action taken during that state.

b) Modify the Finite-horizon value iteration algorithm so that it works for state-action reward functions. Do this by writing out the new update equation that is used in each iteration and explaining the modification from the equation given in class for state rewards.

Answer :

The original finite horizon value iteration is as follows:

$$V_0(s) = R(s), \quad \forall s$$

$$V_R(s) = R(s) + \max_a \sum_{s'} T(s,a,s') V_{K-1}(s'), \quad \pi(s,k) = \arg\max_a \sum_{s'} T(s,a,s') V_{K-1}(s')$$

Update Equation:

$$V_0(s) = 0, \quad \forall s$$

Since there is no initial action, the reward is set to zero. Rewards are only assigned upon the execution of an action.

$$V_k(s) = \max_a \left[R(s,a) + \sum_{s'} T(s,a,s') V_{K-1}(s') \right]$$

Here, $R(s, a)$ indicates that rewards are only given when an action is performed because it is a part of a maximizing function. Hence, it is essential to optimize this function.

$$V_k(s, k) = \arg\max_a \sum_{s'} T(s, a, s')V_{K-1}(s')$$

c) Any MDP with a state-action reward function can be transformed into an "equivalent" MDP with just a state reward function. Show how any MDP with a state-action reward function R(s,a) can be transformed into a different MDP with state reward function R(s), such that the optimal policies in the new MDP correspond exactly to the optimal policies in the original MDP. That is an optimal policy in the new MDP can be mapped to an optimal policy in the original MDP. Hint: It will be necessary for the new MDP to introduce new "book keeping" states that are not in the original MDP.

Answer :

An effective method for converting an MDP with a state-action reward function $R(s, a)$ into an equivalent MDP with a state reward function $R(s)$ involves the introduction of "bookkeeping" states. This allows us to track information regarding actions performed within the original MDP while maintaining the dynamics of the process.

Let the original MDP be denoted as

$$M = (S, A, P, R, \gamma),$$

where:

- S represents the set of states,
- A is the set of available actions,
- P is the state transition probability function,
- $R(s, a)$ is the reward function based on state-action pairs,
- γ is the discount factor.

Next, we define the transformed MDP as $M' = (S', A', P', R', \gamma')$ with the following components:

- $S' = S \times A$, which is the product of the original state and action spaces,
- $A' = A$, meaning the action set remains unchanged,
- P' is derived from the original P but applies to the new state space,
- $R'(s')$ is the reward function in the modified MDP, which is calculated based on the original $R(s, a)$,
- $\gamma' = \gamma$, ensuring the discount factor stays consistent.

The transition probability function in the transformed MDP is given by:

$$P'((s,a),s') = P(s' \mid s,a)$$

Similarly, the reward function is:

$$R'(s') = R(s,a)$$

It can be shown that the optimal policies in the original MDP and the transformed MDP align perfectly.

Consider π, the policy in M where $\pi(s) = a$ for all $s \in S$, and assume π' is the optimal policy in M'. Under π, the corresponding state-action pair (s,a) in M' will be visited with probability 1 for every state-action pair (s,a) in M. As a result, the expected cumulative reward obtained by π in M is equal to the expected cumulative reward obtained by π' in M'.

This transformation ensures the preservation of optimal policies while enabling us to work with an MDP that uses a state reward function. Although the transformed MDP has a larger state space due to the addition of bookkeeping states, the fundamental characteristics of the optimal policies remain unchanged.

Question 9:

(k-th Order MDPs.) A standard MDP is described by a set of states S, a set of actions A, a transition function T, and a reward function R. Where T(s,a,s) gives the probability of transitioning to safter taking action ain state s, and R(s) gives the immediate reward of being in state s. A k-order MDP is described in the same way with one exception. The transition function T de- pends on the current state sand also the previous k1 states. That is, T(sk1,···,s1,s,a,s) = Pr(s—a,s,s1,···,sk1) gives the probability of transitioning to state sgiven that action a was taken in state s and the previous k1 states were (sk1,···,s1). Given a k-order MDP M = (S,A,T,R) describe how to construct a standard (First-order) MDP M = (S,A,T,R) that is equivalent to M. Here equivalent means that a solution to Mcan be easily converted into a solution to M. Be sure to describe S , A , T, and R. Give a brief justification for your construction.

Answer :

Let S' represent the state space of the transformed MDP, M', where each state in S' is expressed as $(s, s_1, s_2, \ldots, s_{k-1})$, with each element being an element from the original state space S.

The action space for the transformed MDP M' remains the same as that of the original MDP, so $A' = A$. Consequently, the reward function for M', denoted by $R'(s, s_1, s_2, \ldots, s_{k-1})$, is simply $R(s)$. This means that the reward in the new MDP depends solely on the state s of the original MDP.

The transition function for M' is defined as:

$$T'(s, s_1, s_2, \ldots, s_{k-1}, a, s') = \Pr(s' \mid a, s, s_1, s_2, \ldots, s_{k-1})$$

This indicates that the transition function in M' ensures the accurate tracking of historical states. The k-order transition provides the probability of reaching a new state s' after executing an action a from a sequence of previous states.

The policy for the transformed MDP M' is denoted by $\pi'(s, s_1, s_2, \ldots, s_{k-1})$, which represents the policy for a k-order MDP. Hence, for every k-order MDP M, there exists an equivalent MDP M'.

Question 10 :

Some MDP formulations use a reward function R(s,a) that depends on the action taken in a state or a reward function R(s,a,s) that also depends on the result state s (we get reward R(s,a,s) when we take action a in state s and then transition to s). Write the Bellman optimality equation with discount factor for each of these two formulations.

Answer: Bellman Optimality Equations

1. Reward Function $R(s)$ For a given reward function $R(s)$, the Bellman Optimality Equation is expressed as:

$$V^*(s) = R(s) + \beta \max_{s'} T(s, a, s')V^*(s')$$

Where:

- $V^*(s)$ represents the optimal value function for state s,
- β is the discount factor,
- $T(s, a, s')$ is the state transition probability function.

2. State-Action Reward Function $R(s, a)$ For a reward function dependent on both state and action $R(s, a)$, the equation becomes:

$$V^*(s) = \max_a \left[R(s, a) + \beta \sum_{s'} T(s, a, s')V^*(s') \right]$$

In this case:

- The value of state s is obtained by maximizing the immediate reward $R(s, a)$ over all possible actions a, and the expected future values are discounted by β.

3. State-Action-State Reward Function $R(s, a, s')$ When the reward function depends on state, action, and the next state $R(s, a, s')$, the Bellman Optimality Equation is:

$$V^*(s) = \max_a \left[\sum_{s'} T(s, a, s')R(s, a, s') + \beta \sum_{s'} T(s, a, s')V^*(s') \right]$$

Here:

- The equation involves a summation over the possible next states s', where both the reward and transition probabilities are considered.

Question 11:

Consider a trivially simple MDP with two states S= s0,s1and a single action A= a. The reward function is R(s0) = 0 and R(s1) = 1. The transition function is T(s0,a,s1) = 1 and T(s1,a,s1) = 1. Note that there is only a single policy for this MDP that takes action a in both states. a) Using a discount factor = 1 (i.e. no discounting), write out the linear equations for evaluating the policy and attempt to solve the linear system. What happens and why?

Answer: Let the policy be P, with values $v_0 = v_P(s_0)$ and $v_1 = v_P(s_1)$. The corresponding linear equations are:

$$v_0 = R(s_0) + \beta v_1$$

$$v_1 = R(s_1) + \beta v_1$$

where β is the discount factor and $\beta = 1$. Substituting the values into the equations:

$$v_0 = 0 + (1)v_1$$

$$v_0 = v_1$$

$$v_1 = 1 + (1)v_1$$

$$v_1 = 1 + v_1$$

As a result, the equations have no solution for v_0 and v_1, indicating that the policy is not well-defined.

b) Repeat the previous question using a discount factor of = 0.9.

Answer:

Let the policy be P, with values $v_0 = v_P(s_0)$ and $v_1 = v_P(s_1)$. The corresponding linear equations are:

$$v_0 = R(s_0) + \beta v_1$$

$$v_1 = R(s_1) + \beta v_1$$

where β is the discount factor and $\beta = 0.9$. Substituting the values into the equations:

$$v_0 = 0 + (0.9)v_1$$

$$v_0 = (0.9)v_1$$

$$v_1 = 1 + (0.9)v_1$$

Solving for v_1:

$$v_1 = 10$$

Substituting $v_1 = 10$ into the equation for v_0:

$$v_0 = 9$$

Thus, the values of v_0 and v_1 are 9 and 10, respectively.

2. Programming Part :

Model Performance Metrics

Training Accuracy: 0.9596273291925466
Testing Accuracy: 0.8316831683168316

Class	Precision	Recall	F1-Score	Support
0	0.79	0.73	0.76	37
1	0.85	0.89	0.87	64
Accuracy	0.83			
Macro avg	0.82	0.81	0.82	101
Weighted avg	0.83	0.83	0.83	101

Table 1: Classification Report