

# HW3

John Ye

November 3, 2024

## 1

### 1a

Let  $m = \frac{1}{d} \sum_{i=1}^d (x_i - z_i)$  to be the mean of the difference between  $x_i$  and  $z_i$ , then,  $m^2 = (\frac{1}{d} \sum_{i=1}^d (x_i - z_i))^2$ . Consider  $f(x) = x_i - z_i$ , by using the Jensen's inequality, we have

$$\begin{aligned} f\left(\frac{1}{d} \sum_{i=1}^d (x_i - z_i)\right) &\leq \frac{1}{d} \sum_{i=1}^d f(x_i - z_i) \\ \left(\frac{1}{d} \sum_{i=1}^d (x_i - z_i)\right)^2 &\leq \frac{1}{d} \sum_{i=1}^d (x_i - z_i)^2 \end{aligned} \tag{1}$$

then, multiply both side by  $d$ , and get

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

and that complete the proof.

### 1b

By using the previous property, we can calculate the approximation of the Euclidean distance between two points using an average-based representation rather than computing it directly, which makes the computation simpler. The equation also provides a lower bound for the sum of the squared differences, so we can calculate the lower bound first to check whether the two points are

close, if the result is within a certain range, we then compute the Euclidean distance between the points, otherwise, we consider the points are not close. This could reduce the computational cost

## 2

This paper mainly discusses algorithms for performing approximate search in high dimension based on the concept of locality-sensitive hashing (LSH). The main idea is to hash the points using several hash functions to ensure that for each function the probability of collision is much higher for objects that are close to each other than for those that are far apart. Then, near neighbors can be determined by hashing the query point and retrieving elements stored in buckets containing that point.

Sometimes when the gap between the probabilities are really small, the LSH family  $H$  cannot be used, an amplification process is needed in order to achieve the desired probabilities of collision. The amplification is used to increase the gap between two probabilities involves constructing new hash functions that make similar points more likely and dissimilar points less likely to collide.

The authors also discuss a new LSH family for Euclidean distance which significantly improves upon some other LSH approaches. This family uses multi-dimensional projection instead of one-dimensional projection and the query time can be improved from  $\rho(c) = \frac{1}{c^2} + O(\log \log(\frac{n}{\log^{\frac{1}{3}} n}))$  to  $\rho(c) = \frac{1}{c^2}$  when  $n$  is large enough.

## 3

Yes, it is possible to convert the rule set  $R$  into an equivalent decision tree. We can collect all the features and conditions for all rules, and then let the each feature to be a split point for the tree, and each condition to be a value of a particular feature that determines how the tree should be proceeded. Then we use the conditions to create the branches for the tree. Each path from the root to a leaf is a rule from the rule set, the leaf node is the outcome defined by the rule.

## 4

This paper mainly discusses the comparison between generative and discriminative classifiers as typified by logistic regression and naive Bayes. The author shows that the generative model has a higher asymptotic error as the number of training examples becomes large than the discriminative model. However, the generative model may also approach its asymptotic error much faster than the discriminative model. The authors proves that when the number of training examples  $m$  is increased, the naive Bayes classifier would initially perform better, but eventually the logistic regression would catch up and overtake the performance of the naive Bayes.

## 5

### 5a

Naive Bayes would produce better result. Based on the result from “On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes”, the generative naive Bayes classifier may converge more quickly to its asymptotic error, which means when the number of training examples  $m$  is increased, the generative naive Bayes would initially do better, but the discriminative logistic regression would eventually catch up and may overtake the performance of naive Bayes. Although eventually, these two classifiers may have similar performance, but in this case, naive Bayes would converge faster and have more efficiency than logistic regression

### 5b

If the data does not satisfy the Naive Bayes assumption, then logistic regression will produce better results. Naive Bayes makes predictions by assuming the features are conditionally independent given the class label, but logistic regression learns a direct map from the inputs to the class labels. Therefore, in this case, the logistic regression would produce better results than naive Bayes.

### 5c

Yes, we can compute  $P(X)$  from the learned parameters of a Naive Bayes classifier. Since Naive Bayes assumes that all features are conditionally independent given the class label  $Y$ , then  $P(X|Y) = \prod_{i=1}^d P(X_i|Y)$ , where  $X_i$  is

each feature in the data. In order to compute the marginal probability  $P(X)$ , we have  $P(X) = \sum_Y P(Y)P(X|Y)$ . Since  $P(Y)$  and  $P(X_i|Y)$  are learned in Naive Bayes classifier, then  $P(X)$  can be computed using these parameters.

## 5d

No, we cannot compute  $P(X)$  from the learned parameters of a Logistic Regression classifier. In order to compute the marginal probability  $P(X)$ , we need the  $P(X)$  or  $P(X|Y)$ . However, since logistic regression is a discriminative model, it does not model the distribution of the feature  $X$ . Therefore, it is impossible to compute  $P(X)$  from learned parameters of a Logistic Regression.

## 6

This paper discusses the ensemble methods and explains why ensembles can often perform better than any single classifier. There are three fundamental reasons for this. The first reason is statistical. The learning algorithm may perform badly when there are no sufficient data, it may find many different hypotheses in a space that all give the same accuracy on the training data. By using an ensemble method, the algorithm can average the votes and reduce the risk of overfitting. The second reason is computational. Many learning algorithms may perform some local search that may get stuck in local optima. The ensemble method can run these algorithms multiple times from many different starting points, which improves the chances of finding a better hypothesis. The third reason is representational. In most applications of machine learning, the true function cannot be represented by a single hypothesis. By using the ensemble methods, multiple classifiers can be combined to produce a better representation.

The author also discusses the methods for constructing ensembles. The first method mentioned is Bayesian Voting, which primarily addresses the statistical component of ensembles by using a probabilistic approach to weight different hypotheses by their posterior probabilities. The second method is constructing ensembles manipulates the training examples to generate multiple hypotheses using Bagging or Boosting. This works especially well for unstable learning algorithms where the output classifier undergoes major changes in response to small changes in the training data. The third technique is to manipulate the set of input features available to the learning algorithm, which only works when the input features are highly redundant.

The fourth technique is to manipulate the output value that are given to the learning algorithm. The last is to inject randomness into the learning algorithm

The paper also compares different ensemble methods. Boosting may outperform Bagging when there is little noise in the data. However, when the noise is increased, Boosting may cause overfitting. When the dataset is really large, Randomization can outperform Bagging since randomization creates diversity under all conditions. However, the quality of the decision tree may be decreased.

## 7

This paper mainly discusses five approximate statistical tests for determining whether one learning algorithm outperforms another on a particular learning task. To design and evaluate statistical test, four sources of variation need to be identified. First, for small test datasets, there is the random variation in the selection of the test data that is used to evaluate the learning algorithms. The second source of random variation results from the selection of the training data. The third source of variance can be internal randomness in the learning algorithm, and the last source of variation need to be handled by statistical tests in random classification error.

Next, the author explains five statistical tests. The first test is McNemar's test. In this test, the datasets are splitted into a training set and a test set, both algorithms will be trained on the training set and tested on the test set. This test is based on the goodness-of-fit test that compares the distribution of counts expected under the null hypothesis to the observed counts, where the null hypothesis is both algorithms have the same error rate. The McNemar's test should only be applied if the sources of variability are small. Also, since the original training datasets is been splitted into training and test set, the size of the training set is decreased, therefore, we also need to assume that the relative difference observed on the splitted training set will still hold for the original training set size. The second test is a test for the difference of two proportions, which measures the difference between the error rate of two algorithms. However, since the probability of the error rate for both algorithms are measured on the same test set, they are not independent, and similar to the McNemar's test, the splitted training set has smaller size than the original training set. The third test is the resampled paired  $t$  test, which tests the algorithms and checks the difference in accuracy

between them by resampling the training set multiple times. The drawback for this test is that the error rates are not independent, and the test sets overlaps. The fourth test is the  $k$ -fold cross-validated paired  $t$  test, which is similar to the previous  $t$  test except that instead of randomly dividing the original datasets,  $k$ -fold cross-validation will be used on the datasets. By using this test, each test set will be independent of the others, however, the training set overlap may still occurs. The last test is the 5 replications of 2-fold cross-validation (5x2cv) paired  $t$  test, runs five iterations of the 2-fold cross-validation, which ensuring each data split is tested multiple times. Five replications is used because exploratory studies showed that using fewer or more than five replications increased the risk of Type I error. A possible explanation is with fewer replications, the noise in the measurement of the variances may become troublesome, and with more replications, the lack of independence among variances becomes troublesome.

The author next discusses the simulation experiment design for those five tests, by measuring the probability of the Type I error of the algorithms, where the Type I error occurs when the null hypothesis is true and the learning algorithm rejects the null hypothesis. By setting up the simulation for the tests, the author shows a graph contains the probability of Type I error for all five tests. The result shows that the resampled  $t$  test has the highest error rate, and the 5x2cv test performed best among all five tests. After excluded the resampled  $t$  test from further experiments, the author continues to experiment the tests on realistic data with real learning algorithm. The author measures the power of the tests, which is the ability to reject the null hypothesis when it is false. The result for the power analysis shows that the cross-validated  $t$  test has more power compare to other tests, however, this test has the highest Type I error rate. Then, for the rest of the three tests, 5x2cv  $t$  test performs the best among all the tests.

In conclusion, the 5x2cv  $t$  test is the most powerful tests among tests with acceptable Type I error. The McNemar's test performs well in most experiments although it is not as powerful as the 5x2cv  $t$  test. The difference of two proportions test has limitations especially when the algorithms have different regions of poor performance. The cross-validated  $t$  test has the highest power among all tests, and it could be used where Type II error is more important.

## 8

### 8a

One real world problem for choosing the state-action reward function over state reward function is smart car. The state reward function  $R(s)$  for a smart car may represent if the car sees a green light, the agent will choose to proceed and receive the same amount of reward everytime. However, in real world, it is not always the case for driver to proceed on a green light. By using the state-action reward function, if the car sees a green light, and there is a pedestrian crossing the road, if the agent chooses to proceed, it will receive a smaller or negative reward because it should not be a preferred action, and if the agent decides to stop and wait for the pedestrian, it will receive a larger reward. By using the state-action reward function, the agent's decision-making ability will be improved.

### 8b

The original finite-horizon value iteration in the slide is

$$V^k(s) = R(s) + \max_a \sum_{s'} T(s, a, s') \cdot V^{k-1}(s')$$

where  $V^k(s)$  is the value of state  $s$  at time  $k$ ,  $R(s)$  is the reward at state  $s$ ,  $T(s, a, s')$  is the transition probability from state  $s$  to the next state  $s'$  under action  $a$ , and  $V^{k-1}(s')$  is the value function for next states  $s'$  at time  $k - 1$ . In order to make this function works for state-action reward functions, we need to replace  $R(s)$  with  $R(s, a)$ , which is the reward at state  $s$  when taking action  $a$ . Therefore, the new value iteration would be:

$$V^k(s) = \max_a \sum_{s'} R(s, a) + T(s, a, s') \cdot V^{k-1}(s')$$

In the original value iteration function, the reward only depends on the state  $s$ , while in the modified function, the reward will depend on both state  $s$  and action  $a$ . In the modified function, the policy will consider to maximize both immediate reward and the expected reward for future states.

### 8c

Let  $M_1$  to denote the original MDP with state  $S$  and actions  $A$ , the state-action reward function  $R(s, a)$  and the transition probabilities  $T(s, a, s')$ , and let  $M_2$  to denote the new MDP with state  $S'$  which includes the states  $S$  from

original MDP, and new book keeping states  $S(s, a)$ , which used to track of which action was taken in the original states. Then, the new reward function  $R(s')$  can be defined as

$$R(s') = R(s_{(s,a)}) = R(s, a)$$

which means the book keeping state has the equivalent reward to the original state-action reward.

The new transition function is

$$T(s_{(s,a)}, s') = T(s, a, s')$$

After the reward function and the transition function are defined, the maximum reward for  $M_2$  will be ensured to be equivalent to the original MDP  $M_1$ , therefore, the optimal policy in the new MDP  $M_2$  can be mapped to an optimal policy in the original MDP  $M_1$ .

## 9

In order to construct a standard MDP  $M'$  that is equivalent to  $M$ , we need to first define the  $S', A', T', R'$  in  $M'$ .

$S'$  in  $M'$  represents a sequence of previous states from  $(s_{k-1}, \dots, s)$  in  $M$ .

$A'$  in  $M'$  represents actions that are the same from  $A$  in  $M$ .

$T'$  in  $M'$  represents the transition function from state  $(s_{k-1}, \dots, s_1, s)$  to the new state  $(s, s_{k-1}, \dots, s_2)$ , which makes  $T'((s_{k-1}, \dots, s_1, s), a, (s, s_{k-1}, \dots, s_2))$  is equivalent to  $T(s_{k-1}, \dots, s_1, s, a, s')$  in  $M$ .

$R'$  in  $M'$  represents the reward function that makes  $R'((s_{k-1}, \dots, s_1, s))$  is equivalent to  $R(s)$  in  $M$ .

Once all the constructions are completed,  $M'$  can be directly maps to an optimal solution for  $M$ .

## 10

The Bellman optimality equation for  $R(s, a)$  is

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

where  $R(s, a)$  is the reward for taking action  $a$  in current state  $s$ , and the discount factor  $\beta$  only affects the value on all future states after taking the



action  $a$ .

The Bellman optimality equation for  $R(s, a, s')$  is

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

where  $R(s, a, s')$  is the reward for taking action  $a$  after transitioning from current state  $s$  to the next state  $s'$ . In this case, the reward  $R$  and the future discounted value  $\beta V^*$  are weighted by the transition probabilities  $T$ .

## 11

### 11a

When  $\beta = 1$ , the Bellman equation for state  $s_0$  is

$$\begin{aligned} V(s_0) &= R(s_0) + \beta \cdot T(s_0, a, s_1) \cdot V(s_1) \\ &= 0 + 1 \cdot V(s_1) \\ &= V(s_1) \end{aligned} \tag{2}$$

the Bellman equation for state  $s_1$  is

$$\begin{aligned} V(s_1) &= R(s_1) + \beta \cdot T(s_1, a, s_1) \cdot V(s_1) \\ &= 1 + 1 \cdot V(s_1) \\ &= 1 + V(s_1) \end{aligned} \tag{3}$$

The equation has no finite solution since  $\beta = 1$ , future rewards are fully valued without any discounting, and the reward in  $s_1$  accumulates indefinitely.

### 11b

When  $\beta = 0.9$ , the Bellman equation for state  $s_0$  is

$$\begin{aligned} V(s_0) &= R(s_0) + \beta \cdot T(s_0, a, s_1) \cdot V(s_1) \\ &= 0 + 0.9 \cdot V(s_1) \\ &= 0.9V(s_1) \end{aligned} \tag{4}$$

the Bellman equation for state  $s_1$  is

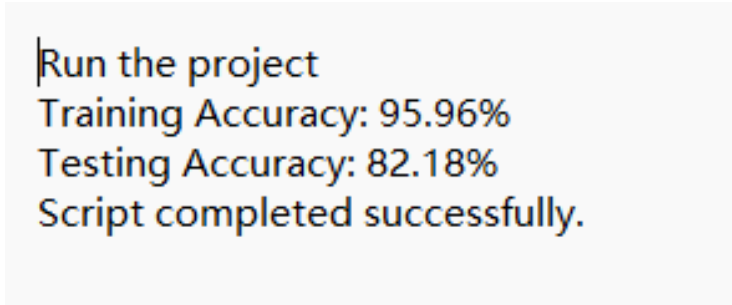
$$\begin{aligned} V(s_1) &= R(s_1) + \beta \cdot T(s_1, a, s_1) \cdot V(s_1) \\ &= 1 + 0.9 \cdot V(s_1) \\ &= 1 + 0.9V(s_1) \\ V(s_1) - 0.9V(s_1) &= 1 \\ 0.1V(s_1) &= 1 \\ V(s_1) &= 10 \end{aligned} \tag{5}$$

Then,  $V(s_0) = 0.9V(s_1) = 9$

Therefore, the solution for  $\beta = 0.9$  is  $V(s_0) = 9$  and  $V(s_1) = 10$

## 12 Programming Output

The output.txt after executing the run\_code.bat:

A screenshot of a terminal window with a light gray background. The text is displayed in a monospaced font. The first line is a prompt character followed by the command 'Run the project'. The subsequent lines show the results: 'Training Accuracy: 95.96%', 'Testing Accuracy: 82.18%', and 'Script completed successfully.'.

```
|Run the project  
Training Accuracy: 95.96%  
Testing Accuracy: 82.18%  
Script completed successfully.
```

Figure 1: output.txt