CS412: Introduction to Data Mining, Fall 2023, Homework 4

Name: Teng Hou (tenghou2)

Q1

a. *i. To calculate the answer for* $P(F=1,T=0,S=1,A=1)$*, we could calculate* $P(F=1)$,$P(T=0)$,$P(S=1)$,$P(A=1)$*. And then we multiply them together which will be the answer.*
From the figure and table, we could know that $P(F=1)=0.1$*,* $P(T=0)=1-P(T=1)=1-0.01=0.99$ *And when* $F=1$ *is given, then we could find* $P(S=1)=0.9$*. Then as* $F=1$ *and* $T=0$ *are given,* $P(A=1)=0.95$*. So we can find the final answer:*

$$P(F=1,T=0,S=1,A=1)=0.1*0.99*0.9*0.95=0.08465$$

ii. To calculate the answer for $P(F=1,T=1,S=0,A=1)$*, we could calculate* $P(F=1)$,$P(T=1)$,$P(S=0)$,$P(A=1)$*. And then we multiply them together which will be the answer.*
From the figure and table, we could know that $P(F=1)=0.1$*,* $P(T=1)=0.01$ *And when* $F=1$ *is given, then we could find* $P(S=0)=0.1$*. Then as* $F=1$ *and* $T=1$ *are given,* $P(A=1)=0.8$*. So we can find the final answer:*

$$P(F=1,T=1,S=0,A=1)=0.1*0.01*0.1*0.8=0.00008$$

b. *i. Since that we could get* $P(F=1,A=1)$ *by calculating* $P(F=1,T=0,S=1,A=1)+P(F=1,T=1,S=1,A=1)+P(F=1,T=0,S=0,A=1)+P(F=1,T=1,S=0,A=1)$
From a), we know that $P(F=1,T=0,S=1,A=1)=0.08465$ *and* $P(F=1,T=1,S=0,A=1)=0.00008$.
And $P(F=1,T=0,S=0,A=1)=0.1*0.99*0.1*0.95=0.00941$
And $P(F=1,T=1,S=1,A=1)=0.1*0.01*0.9*0.8=0.00071$
So

$$P(F=1,A=1)=0.08465+0.00008+0.00941+0.00071=0.09485$$

ii. To calculate $P(A=1)$*, we could calculate the sum of*
$P(F=0,T=0,S=0,A=1)=0.9*0.99*0.9*0.0001=0.00008$
$P(F=0,T=0,S=1,A=1)=0.9*0.99*0.1*0.0001=0.00001$
$P(F=0,T=1,S=0,A=1)=0.9*0.01*0.9*0.8=0.00648$
$P(F=0,T=1,S=1,A=1)=0.9*0.01*0.1*0.8=0.00071$
And from a) and b) i.
$P(F=1,T=0,S=0,A=1)=0.00941$

$P(F=1,T=0,S=1,A=1)=0.08465$
$P(F=1,T=1,S=0,A=1)=0.00008$
$P(F=1,T=1,S=1,A=1)=0.00071$
So we could get

$P(A=1)=0.00008+0.00001+0.00648+0.00072+0.00941+0.08465+0.00008+0.00072$

$$P(A=1)=0.10214$$

c. *According to Bayes rule, we could calculate*

$$P(F=1|A=1)=\frac{P(A=1|F=1)}{P(A=1)}=\frac{P(F=1,A=1)}{P(A=1)}$$

So the answer is

$$P(F=1|A=1)=\frac{0.09486}{0.10215}=0.92863$$

Q2

a. *Yes, we could train such a soft margin linear SVM. From the given figure of the dataset, it is obvious that it's impossible to completely divide the dataset into two 2-class. But if we allow some mistakes, i.e we use slack variables, we could train a soft margin linear SVM. Because by using slack variables, we could have some data located on the wrong side.*
No it is impossible. Just as I mentioned before, the distribution of the data points is concentric circles, so we can not classify this dataset linearly. And this means that there must be some slack variables which is not zero. So we can train such a classifer but it is impossible that all slack variables will be zero.

b. *Yes, I agree with Professor Kernel's claim.*
In the original two-dimensional space, the dataset is distributed in the form of concentric circles, which means that any linear classifier cannot effectively distinguish between the two categories. However, when we apply a polynomial kernel mapping to project the data points into a high-dimensional space, the structure that was originally not linearly separable may become linearly separable. This is because the additional dimensions in the high-dimensional space provide more degrees of freedom to capture the nonlinear relationships in the data.
And according to the kernel function in the question, we could find that there includes $(x_1^i)^2$ and $(x_2^i)^2$, which could expresses the distance from the point to original point in the original two-dimensional space. And in high dimensional space, this could be come the method to linearly divide the dataset, which could train a hard-margin SVM.

c. *i.*
No. The kmeans algorithm is based on the distance between the clusters and each data. So the cluster will tend to put the cluster on the overlap area of two circle, which will not accurately distinguish two circles. Or we can say that the kmeans algorithm is not suitable to discover clusters with non-convex shapes and it can only detect clusters that are linearly separable, and the dataset given is a non-convex shape.
ii.
Yes. Under the influence of the Radial Basis Function (RBF) kernel, each data point is mapped into an infinite-dimensional space, where the new features of the data points are defined based on their relative proximity to all other points. This mapping helps to reveal potentially hidden structures in the data, making data that was not linearly separable in low-dimensional space become linearly separable in the new feature space. For data with concentric circular shapes, traditional k-means clustering cannot correctly differentiate ring-like structures because it only considers Euclidean distance. However, the RBF kernel, through its nonlinear mapping, "unfolds" the concentric circles from the original two-dimensional space into a high-dimensional space. In this space, the inner and outer circles can be seen as two groups of points distributed along a certain dimension, and they can be separated by a linear dividing line.

Q3

a. *d:*
This is the paramter that specifies the maximum depth of the trees. Tree depth is the length of the longest path from one tree's root to its leaves. As the d gets bigger, the tree can capture more information about the data.
m:
When we hope to decide on which attribute to split at each node in the tree, the Random Forests will not consider all attribute but only a random subset of them. Then the parameter m will define the number of attributes to be considered for each split. We need to pay attention on the choice of m, if it is too large, then there will be high variance and overfitting. If m is too small, then the forset may not capture all relevant attribute, leading to bias.
T:
This parameter defines the number of trees that should be grown in the forest. It's important to find a balance of this parameter. As although the forest will reduces the risk of overfitting as the T gets larger, there will be a certain point where adding more trees has a diminishing return on performance and increases the cost of computing.

b. *Training:*
During the training phase of Random Forests with Random Input selection, each tree is constructed by taking a bootstrap sample from the training dataset. The trees are built using the CART methodology, where at each node within a tree, a randomly selected subset of 'm' features is considered for splitting, rather than all 'd' available features. This introduces randomness and ensures that the trees in the forest are diverse. Each tree is grown to its maximum depth 'd' as specified, without pruning, to form a strong learner with low bias. This process is repeated until a forest of 'T' trees has been created.
Prediction:
For prediction on a test point, each of the 'T' trees in the forest provides a vote for a class, and the class that gets the majority of votes is chosen as the final prediction. This majority voting mechanism integrates the predictions from all the individual trees to decide on the most probable class for the test point.

c. *The probability that any specific sample is not picked during the bootstrap sampling is* $(1-\frac{1}{n})^n$. *So the expected number of unique samples from the original set of n samples in the bootstrapped sample is*

$$(1-(1-\frac{1}{n})^n)\times n$$

When n is large, the equation $(1-\frac{1}{n})^n$ *will get closed to* $\frac{1}{e}$, *so the expected number of unique samples is*

$$n\times(1-\frac{1}{e})\approx 0.63212n$$

d. *I do not agree with Professor Forest. As the m stands for the number of attributes to be considered for each split, when we make m smaller, it could reduce the variance. But when the m get too small, then the forest may not capture all relevant attribute, leading to bias. This is due to that there will be many weak trees as m is set to 1. If the trees are* *too weak, the forest's accuracy can actually decrease, because the trees might not capture the underlying patterns in the data effectively. And this will cause the error from the bias, and then leading to higher variance.*

Q4

a. *For basic gradient descent, the update equation is:*

$$\theta_{t+1,i} \leftarrow \theta_{t,i} - \eta_t g_{t,i}$$

where $g_{t,i} = \nabla L(\theta_t)$, which is the gradient at old θ of lost function. And i means the i-th component.
For Adagrad, the update equation is

$$\theta_{t+1,i} \leftarrow \theta_{t,i} - \eta_{t,i} g_{t,i}$$

where $g_{t,i} = \nabla L(\theta_t)$ and $\eta_{t,i} = \frac{\eta}{\sqrt{\Sigma_{\tau=1}^{t} g_{\tau,i}^2}}$
Similar aspects:
i. Both two methods use gradient information to update the parameter θ.
ii. These two methods are both iterative algorithms and for each step, they both need to calculate the gradient.
iii. In each iteration, the parameters are updated in the direction opposite to the gradient to reduce the value of the loss function.
Different aspects:
For basic gradient descent, the learning rate stay the same, but for Adagrad, the learning rate changes. Or we can say that, Adagrad has an adaptive learning rate. This learning rate depends on the cumulative square of all past gradients for that parameter. This means that for dimensions where the parameter is frequently updated, Adagrad will decrease its learning rate, while for dimensions with infrequent updates, the learning rate remains relatively larger.
This is because that in practice, in high dimensional feature (parameter) spaces, only a few features are very informative and many features are irrelevant. By the way of adaptive learning rate, the gradient descent does feature specific learning rates.

b. *When the loss function $L(\theta)$ does not depend on the parameter $\theta_i \in \mathbb{R}$, then the gradient of the parameter will tends to close to zero. Then in Adagrad algorithm, this parameter's cumulative gradient will be small. And then as the learning rate of Adagrad algorithm depends on the inverse of the sum of squares, so the learning rate will be large. So the learning rate of the parameter will be relative large. In conclusion, smaller gradients leads to large step size, larger gradients leads to small step size.*
Then as the $L(\theta)$ does not depend on θ_i, then the parameter will not do some contributions to the gradient, then leading to a large step size.

c. *The main limitation of Adagrad for deep learning is that it remembers $g_{\tau,i}$ for all τ in the history, which makes all historical τ get the same weight. And as the times of iteration get larger, the cumulative gradient will get larger and larger which makes the learning rate get really small in the end. And this may lead to the model terminate to learn when the model does not converge.*
Adam tries to solve this by calculating the first moment estimate and second moment estimate.

For first moment estimate, we could calculate by:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

For second moment estimate, we could calculate by:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

And Compute bias corrected versions of first and second moment by:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Then the final update equation is:

$$\theta_{t+1} \leftarrow \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

With this method, The Adam optimization algorithm overcomes the limitations of Adagrad by combining momentum and adaptive learning rate strategies. It adjusts the update step for each parameter using the square root of the second-order moment estimate and the first-order moment estimate, which leverages the benefits of momentum for accelerating gradient descent while preventing the learning rate from diminishing too rapidly. Adam also introduces a bias correction mechanism to adjust the moment estimates during the initial phase of iterations, ensuring the algorithm starts off correctly. These features provide Adam with adaptive learning rates for each parameter that do not reduce drastically with iterations like in Adagrad, leading to better performance in deep learning, especially in handling non-convex optimization problems.

Q5

a. *We can consider a similarity measure $w(x, x_i)$ that assigns a weight of 1 to the nearest neighbor of the test point x and 0 to all other points. so the prediction $\hat{y}(x)$ is just the y_i value of the nearest neighbor to x. This is because the neighbor has the highest similarity and all other points have zero weight in the prediction sum. So we can say that nearest neighbor regression is a regression model of the form (2).*

b. *If x_i is one of the k-nearest neighbor of x, then it will get a non-zero weight of $\frac{1}{k}$. And if x_i is not one of the k-nearest neighbor, it will only get a zero weight. So this method makes sure that only k nearest neighbors will effect the final prediction $\hat{y}(x)$*
Or we can use equation to show the answer:

$$w(x, x_i) = \frac{1}{d(x, x_i)^2} = \frac{1}{||x - x_i||^2}$$

c. *No.*
As the dataset is given by $(x_i, y_i), i \in [n]$, so we need check all n points and calculate the distance between themselves and the test point x. And the process to scan all the points is computed in $O(n)$. And we need to sort the k minimum distance, so the final time is must larger than $O(n)$, which means we can not use only $O(k)$ time. And if we use a list that contain current k minimum distance point. Then the whole process is around $O(nlogk)$, which is larger than $O(k)$. So the prediction can not be computed in $O(k)$ time.

d. *The attention weights $w(x, x_i)$ are computed based on a parameterized function of the test point x and each of the data point x_i. And this function involves a set of parameters that can be learned during training.*
After we compute the weights, the regression prediction $\hat{y}(x)$ is obtained by taking the weighted sum of the target values y_i, and the weights here is given by the attention mechanism.
For the whole process, we can first compute $\hat{v}$, it could be computed as

$$\hat{v} = \Sigma_{i=1}^{n} w(q, k_i) v_i$$

and

$$w(q, k_i) = softmax(\frac{< q, k_i >}{\sqrt{d}}) = \frac{exp\frac{<q,k_i>}{\sqrt{d}}}{\Sigma_{i=1}^{n}\frac{<q,k_i>}{\sqrt{d}}}$$

And this process is similar with form (2). The final prediction is the sum of values weighted by a similarity measure. And the similarity measure is parameterized and learned from the training.