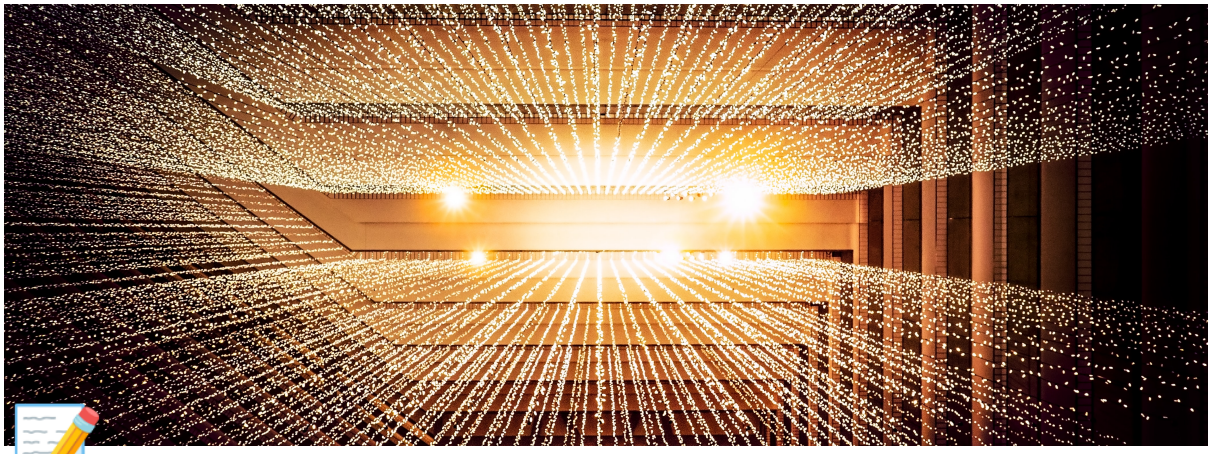

Mid-Term Summary

1. Understand the concepts of rank, invertible, determinant, and singular regarding the matrix.

1. **Rank:** The rank of a matrix is the maximum number of linearly independent rows or columns in the matrix. It is denoted by the symbol "r". A matrix with rank "r" means that there are "r" linearly independent rows or columns in the matrix.

   eg:

   ```
   A = [1 2 3,
       4 5 6,
       7 8 9]
   ```

   To find the rank of matrix A, we can perform row operations to get the matrix in row echelon form. After performing row operations, we get:

   ```
   A = [1 2 3,
       0 -3 -6,
       0 0 0]
   ```

   The first and second rows of the matrix are linearly independent, but the third row is a linear combination of the first two rows. Therefore, the rank of A is 2.

2. **Invertible:** A matrix is invertible if it has a unique solution for its inverse. An invertible matrix is also called a nonsingular matrix because its determinant is nonzero. An invertible matrix has a unique inverse that can be found by using various methods, such as Gaussian elimination or the adjugate method.

   eg: Since the rank of A is less than its order (3), A is not invertible or nonsingular.

3. **Determinant:** The determinant is a scalar value associated with a square matrix. It is used to determine whether the matrix is invertible or not. The determinant of a matrix is denoted by the symbol `|A|` or `det(A)`. It is a product of the diagonal elements of the matrix that are multiplied by their respective cofactors. If the determinant of a matrix is nonzero, the matrix is invertible, and if it is zero, the matrix is singular or non-invertible.

4. **Singular:** A matrix is singular if it is not invertible. It means that the matrix does not have a unique solution for its inverse. A singular matrix has a determinant of zero, which means that its columns or rows are linearly dependent. Singular matrices are also called degenerate matrices.

   eg: The determinant of A is calculated as follows:

   ```
   det(A) = 1 * (-3) * 0 + 2 * 6 * 0 + 3 * 4 * (-3) - 7 * (-3) * 0 - 8 * 2 * (-3) - 9 * 4 * 0
          = 0
   ```

   Since the determinant of A is zero, the matrix A is singular.

   Therefore, the matrix A has rank 2, is not invertible, and is singular.

> 💡 In summary, the rank of a matrix is the maximum number of linearly independent rows or columns, an invertible matrix is a nonsingular matrix that has a unique solution for its inverse, the determinant is a scalar value associated with a square matrix that determines if the matrix is invertible, and a singular matrix is a non-invertible matrix that has a determinant of zero.

2. Understand the concept of inner product (dot product) and outer product.

The inner product, also known as the dot product, is an operation that takes two vectors and produces a scalar. It is denoted by a dot (·) between the two vectors. If we have two vectors a and b of the same dimension, their dot product is defined as:

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

where a1, a2, ..., an and b1, b2, ..., bn are the components of vectors a and b, respectively. The dot product has several important properties, such as:

- The dot product of two vectors is commutative, i.e., `a · b = b · a`.
- The dot product is distributive over addition, i.e., `a · (b + c) = a · b + a · c`.
- The dot product of a vector with itself is equal to the squared magnitude of the vector, i.e., `a · a = ||a||^2`, where `||a||` is the magnitude of vector a.

The outer product, also known as the tensor product, is an operation that takes two vectors and produces a matrix. It is denoted by a cross (×) between the two vectors. If we have two vectors a and b of dimensions n and m, respectively, their outer product is defined as:

```
a × b = [a1b1 a1b2 ... a1bm
         a2b1 a2b2 ... a2bm
         ... ...
         anb1 anb2 ... anbm]
```

The outer product has several important properties, such as:

- The outer product of two vectors is not commutative, i.e., a × b ≠ b × a in general.
- The outer product is distributive over addition, i.e., (a + c) × b = a × b + c × b.
- The rank of the outer product matrix is at most 1, i.e., the outer product of two non-zero vectors always results in a matrix of rank 1.

The inner product and outer product are both important operations in linear algebra and have various applications in areas such as physics, engineering, and computer science.

3. Know the relation between trace of *ATA* and singular values of *A*, and can prove this through SVD (Assignment 1)

(b) Another commonly used norm is the Frobenius norm:

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}, \quad \text{where } A \text{ is a } m \times n \text{ matrix}$$

If we square it to eliminate the square root, the same operation can be calculated as

$$\|A\|_F^2 = \text{trace}(A^T A)$$

From class, remember that the trace of a matrix is the sum of its diagonal elements. If desired, you can verify that the above expressions match by working through yourself how the diagonal elements of $A^T A$ are calculated. Your task is to prove that:

$$\text{trace}(A^T A) = \sigma_1^2 + \sigma_2^2 + \cdots + \sigma_r^2,$$

the sum of the squares of the singular values of A. Hints:

- Use some of the SVD properties that you used in (a).
- One property of the trace is that it is "basis independent," which means, if we have an invertible matrix B, $\text{trace}(B^{-1}AB) = \text{trace}(A)$.
- Using the above properties, you should be able to reduce the expression to a matrix whose diagonal elements are $\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_r^2$.

$$A^T A = (U \Sigma V^T)^T (U \Sigma V^T)$$

$$A^T A = U^T \Sigma^T U^T U \Sigma V^T$$

Here $U U^T = I$ and $V V^T = I$

$$A^T A = \Sigma^T \Sigma$$

Applying Trance

$$\text{Trance}(\Sigma^T \Sigma)$$

$$\text{trance}(A^T A) = \sigma_1^2 + \sigma_2^2 + \sigma_3^2 \ldots\ldots + \sigma_n^2$$

4. Understand Categorical and numeric attributes and their difference.

Categorical attributes are qualitative variables that represent discrete categories or groups. They do not have any intrinsic order or hierarchy. Examples of categorical attributes include gender (male, female), color (red, green, blue), and type of vehicle (sedan, SUV, truck).

Numeric attributes, on the other hand, are quantitative variables that represent a measurable quantity. They can take on continuous or discrete values, and they have an intrinsic order or hierarchy. Examples of numeric attributes include age (in years), height (in centimeters), and income (in dollars).

The main difference between categorical and numeric attributes is that the former represent qualitative information in terms of categories or groups, while the latter represent quantitative information in terms of numbers or values. This difference has important implications for data analysis and machine learning, as the methods used to analyze categorical and numeric attributes are often different.

For example, when analyzing categorical attributes, one might use frequency tables, contingency tables, or chi-square tests to identify patterns or relationships between different categories. When analyzing numeric attributes, one might use descriptive statistics (e.g., mean, standard deviation) to summarize the data, or correlation and regression analysis to identify relationships between different numeric attributes.

5. Understand the concept of sampling with/without replacement, e.g., probability of picking two persons one by one, and both being women (See slides).

(A) Sampling with replacement:

The probability of selecting a woman on the first draw is p(W) = 0.51. Since we are sampling with replacement, the probability of selecting a woman on the second and third draws is also p(W) = 0.51.

Therefore, the probability of selecting three women with replacement is:

```
p(W, W, W) = p(W) * p(W) * p(W)
= 0.51 * 0.51 * 0.51
= 0.132651
```

So the probability of selecting three women with replacement is approximately 0.133 or 13.3%.

(B) Sampling without replacement:

The probability of selecting a woman on the first draw is p(W) = 0.51. However, since we are sampling without replacement, the probability of selecting a woman on the second draw depends on whether the first draw was a woman or a man. If the first draw was a woman, then there are 50 women and 49 men remaining, so the probability of selecting a woman on the second draw is 50/99. If the first draw was a man, then there are 51 women and 48 men remaining, so the probability of selecting a woman on the second draw is 51/99.

Similarly, the probability of selecting a woman on the third draw depends on whether the first two draws were women or men.

Therefore, the probability of selecting three women without replacement is:

```
p(W, W, W) = p(W) * p(W|W or M) * p(W|W and W or M)
= 0.51 * (50/99) * (49/98)
= 0.127686
```

So the probability of selecting three women without replacement is approximately 0.128 or 12.8%.

6. Understand the reservoir sampling algorithms and know how to use this strategy. Understand the conclusion based on the algorithm.

Reservoir sampling is a technique used in statistics and computer science to randomly sample a fixed number of items from a larger data set that cannot be held entirely in memory. It is often used in situations where the entire data set is not available at once, such as when processing large data streams or when dealing with distributed data.

The basic idea behind reservoir sampling is to select a fixed-size sample from a larger population without knowing the size of the population in advance. The algorithm maintains a reservoir, which is initially empty, and sequentially processes each item in the data stream. At each step, the algorithm either adds the item to the reservoir with a certain probability or discards it.

The probability of adding an item to the reservoir is proportional to the current size of the reservoir divided by the total number of items processed so far. This ensures that each item has an equal chance of being included in the final sample.

`eg:` Suppose we have a large data stream consisting of 1 million items, and we want to randomly select a sample of 100 items from the stream using reservoir sampling. We can apply the following formula to calculate the probability of selecting each item in the stream:

$$P(\text{selecting item } i) = \begin{cases} \frac{k}{i} & \text{if } i \leq k \\ \frac{k}{n} & \text{if } i > k \end{cases}$$

where k is the size of the sample we want to select, i is the index of the item in the stream, and n is the total number of items in the stream.

Initially, we create an empty reservoir of size 100, and start processing the items in the stream one by one. For the first 100 items in the stream, we simply add them to the reservoir since we don't have any items to compare them to.

Once we have processed the first 100 items, we begin selecting items with a certain probability. For each subsequent item in the stream, we generate a random number between 0 and 1. If the random number is less than k/i, we replace a randomly selected item from the reservoir with the current item. Otherwise, we discard the current item and move on to the next item in the stream.

This process continues until we have processed all 1 million items in the stream. At this point, we have a reservoir containing 100 randomly selected items from the stream, and each item in the stream had an equal probability of being selected.

Reservoir sampling can be used in a wide range of applications, such as selecting a random sample of web pages to crawl in a search engine, or selecting a random sample of customers for a marketing campaign.

Python code:

```
import random

def reservoir_sampling(stream, k):
    reservoir = []
    for i, item in enumerate(stream):
        if i < k:
            reservoir.append(item)
        else:
            j = random.randint(0, i)
            if j < k:
                reservoir[j] = item
    return reservoir
```

In this implementation, `stream` is an iterable that represents the data stream, and `k` is the size of the sample to be selected. The algorithm initializes an empty `reservoir` and processes each item in the stream sequentially.

If the number of items processed so far is less than `k`, the item is added to the reservoir. Otherwise, a random index `j` between 0 and `i` is selected, where `i` is the current index of the item in the stream. If `j` is less than `k`, the item replaces the item at index `j` in the reservoir.

Finally, the algorithm returns the selected `reservoir`, which contains a random sample of `k` items from the data stream.

Reservoir sampling is a powerful and widely used technique in many areas of computer science and statistics, and it can be adapted and extended in many ways to suit specific applications.

7. Understand “Rhine Paradox” and know how to compute the probability of “at least one occurred among thousands of cases”

Rhine Paradox: The Rhine Paradox is a statistical phenomenon that refers to the apparent contradiction between two types of statistical evidence about the same data set. Specifically, it refers to the paradoxical situation where a strong correlation between two variables is observed at the group level, but little or no correlation is observed at the individual level.

For example, consider a study of the relationship between income and education in two regions of a country, A and B. When the data from the two regions are analyzed separately, a strong positive correlation between income and education is observed in each region. However, when the data from both regions are

combined and analyzed together, the correlation between income and education is much weaker or even disappears.

This paradox occurs because of a statistical concept known as Simpson's paradox, which refers to the reversal of a trend when data is grouped. In the case of the Rhine paradox, the underlying cause is often differences in the composition of the groups being compared.

In order to avoid the Rhine paradox, it is important to carefully consider the level of analysis and the variables being compared, and to use appropriate statistical methods that account for any potential confounding factors.

`eg:` **If you toss a fair coin ten times, what is the probability of at least one head? Please also provide the complete deduction as well as probability.**

First, let's calculate the probability of getting no heads in ten coin tosses. The probability of getting tails on one toss is 1/2, so the probability of getting tails on ten tosses in a row is `(1/2)^10 = 1/1024`.

Therefore, the probability of getting at least one head in ten coin tosses is:

```
1 - probability of getting no heads
= 1 - (1/2)^10
= 1 - 1/1024
= 1023/1024
≈ 0.999
```

So the probability of getting at least one head in ten coin tosses is approximately 0.999, or 99.9%.

To understand the above calculation, we can think of it as follows: The probability of getting no heads in ten coin tosses is very low (1 in 1024), which means that the probability of getting at least one head is very high (1023 in 1024 or 99.9%). This makes sense, since in ten coin tosses, it is very likely that at least one of the tosses will result in a head.

8. Understand the meaning of TF and IDF in TF-IDF features, respectively, and why it can help remove stop words.

TF stands for Term Frequency, and IDF stands for Inverse Document Frequency.

Term Frequency (TF) refers to the number of times a particular term appears in a document. The idea behind TF is that words that occur more frequently within a document are more important and relevant to the meaning of the document.

Inverse Document Frequency (IDF) is a measure of how important a term is across multiple documents in a corpus. The idea behind IDF is that words that occur frequently across multiple documents may not be as informative as words that occur less frequently.

TF-IDF (Term Frequency-Inverse Document Frequency) is a technique used to weigh the importance of each term in a document based on how often it appears in the document and how often it appears across a corpus of documents. The TF-IDF score is the product of the term's TF and IDF values.

By using TF-IDF to compute the importance of words in a document, words that are common across all documents in a corpus (such as "the," "and," "a," etc.) will be weighted less heavily than words that are more specific to the particular document being analyzed. As a result, TF-IDF can help remove stop words, which are common words that do not provide much meaning or value to the text.

9. Understand the fruit example in Bayes theorem and can compute the posterior (Assignment 2).

Assuming that "P(F|B)" refers to the probability of selecting an orange given that the fruit comes from a particular box B, and "P(F)" refers to the probability of selecting an orange without any information about which box it came from, we can prove that P(F|B) = P(F) under the given conditions.

Below we have proved:

```
P(A)= P(O) = 1/2
P(F/B) = (P(B/F) * P(F)) / P(B)
P(B/F) = 1/3
P(F) = 1/2

P(B) = P(F) * P(B/F) + P(F) * P(B/F)
     = (1/2 * 1/3) + (1/2 * 1/3)
     = 1/3

P(F/B) = (P(B/F) * P(F)) / P(B)
       = (1/3 * 1/2) / 1/3
       = 1/2
       = P(F)
```

Therefore, `P(F|B) = P(F)` ,under the given conditions.

10. Know the relation between prior, likelihood and posterior.

To find the probability of `p(B=b|F=a)`, we can use Bayes' theorem, which states that:

$$p(B=b|F=a) = \frac{p(F=a|B=b) \cdot p(B=b)}{p(F=a)}$$

where `p(F=a|B=b)` is the likelihood term, `p(B=b)` is the prior term, and `p(F=a)` is the marginal likelihood or evidence term.

Let's calculate each term one by one:

Prior term:
The probability of selecting box b, denoted as `p(B=b)`, is independent of any knowledge regarding the fruit. As there are only two boxes, the prior probability of selecting either box is 1/2.

```
p(B=b) = 1/2
```

Likelihood term:
`p(F=a|B=b)` is the probability of selecting an apple from box b. From the problem statement, we know that box blue has 3 apples and 1 orange, and box red has 2 apples and 6 oranges. Therefore, the probability of selecting an apple from box blue is 3/4, and the probability of selecting an apple from box red is 2/8 (which can be simplified to 1/4). So, the likelihood term is:

```
p(F=a|B=b) = 3/4, if b=blue
p(F=a|B=b) = 1/4, if b=red
```

Evidence term:
`p(F=a)` is the probability of selecting an apple, regardless of which box it came from. To calculate this, we need to consider all the possible ways of selecting an apple:

```
p(F=a) = p(F=a|B=blue) * p(B=blue) + p(F=a|B=red) * p(B=red)
p(F=a) = (3/4 * 1/2) + (1/4 * 1/2)
```

```
p(F=a) = 1/2
```

Now we can put everything together to get the posterior probability:

```
p(B=b|F=a) = p(F=a|B=b) * p(B=b) / p(F=a)
p(B=b|F=a) = (3/4 * 1/2) / (1/2)
p(B=b|F=a) = 3/4
```

Given that the fruit is an apple, the probability of choosing box blue is 3/4, and the probability of choosing box red is 1/4. Hence, it can be inferred that the most probable box that the apple came from is box blue.

11. Whether the likelihood estimation of Gaussian is biased or not, and why (See discussions on μML, σML2 in slides).

The likelihood estimation of Gaussian can be biased or unbiased, depending on the specific scenario. In general, the maximum likelihood estimation (MLE) of the mean and variance of a Gaussian distribution can be biased when the sample size is small or when the data is not normally distributed. The bias occurs because the MLE tends to overestimate the variance and underestimate the mean in such cases.
However, under certain conditions, such as when the sample size is large and the data is normally distributed, the MLE of the mean and variance can be unbiased. For example, the MLE of the sample mean and sample variance is unbiased when the sample is drawn from a normal distribution.
Therefore, whether the likelihood estimation of Gaussian is biased or not depends on the specific situation and the properties of the data. In general, it is important to consider the sample size and the underlying distribution when assessing the bias of the MLE in Gaussian estimation.
You can find further information about bias in likelihood estimation and examples of unbiased estimation in the provided search results, particularly in slides 44/84 and in the book "Gaussian Processes for Machine Learning".

12. Understand the overfitting issue, and reasons for overfitting in decision tree.

Reasons for overfitting:

1. **Too many branches:** Decision trees can easily become too complex if they are allowed to grow without any constraints. This can result in a tree that is tailored to the training data but not generalizable to new data.
2. **Small sample size:** If the training data is small, the decision tree may not have enough information to make accurate predictions, leading to overfitting.
3. **Irrelevant features:** If the decision tree includes features that are not relevant to the problem being solved, it may be overfitting to noise in the data rather than the underlying pattern.
4. **Bias in the data:** If the training data is biased towards certain outcomes, the decision tree may learn this bias and overfit to it.

To prevent overfitting in decision trees, several techniques can be used:

1. **Pruning:** This involves removing branches that do not improve the performance of the tree on the validation set. This can help to simplify the tree and reduce overfitting.
2. **Limiting the depth of the tree:** This can help to prevent the tree from becoming too complex and overfitting to the training data.

3. **Ensembling:** This involves combining multiple decision trees to reduce overfitting and improve generalization performance.
4. **Cross-validation:** This can help to estimate the generalization performance of the decision tree and prevent overfitting.

13. Know the relation between the number of training data, bias and variable in evaluation.

The number of training data, bias, and variables in evaluation are all related to the performance and generalization of a machine learning model.

In general, having more training data can help reduce the bias of a model and improve its generalization performance. This is because more data provides more information for the model to learn from, which can help it capture the underlying patterns in the data more accurately.

However, adding more variables (also known as features or predictors) to the model can increase its complexity and potentially introduce more noise, which can lead to overfitting and reduce generalization performance. This is because the model may start to fit the noise in the training data rather than the underlying patterns.

The bias of a model refers to the degree to which it is simplified and does not capture the complexity of the true underlying pattern. A model with high bias may underfit the data and not perform well on either the training or test data. Increasing the complexity of the model, by adding more variables or increasing the depth of a decision tree, for example, can reduce the bias of the model but may increase its variance and lead to overfitting.

Therefore, it's important to strike a balance between the number of training data, variables, and bias in order to achieve good generalization performance. Adding more training data and carefully selecting relevant variables can help reduce bias and improve performance, while regularization techniques, such as pruning or limiting the depth of a decision tree, can help reduce overfitting and improve generalization performance.

14. Know the definition of N-fold cross validation and leave-one-out.

N-fold cross-validation and leave-one-out (LOO) are two commonly used techniques for evaluating the performance of a machine learning model.

N-fold cross-validation involves partitioning the available data into N subsets (or "folds"). The model is trained on N-1 of these folds and evaluated on the remaining fold. This process is repeated N times, with each fold used exactly once as the validation data. The results from each fold are then averaged to produce an overall estimate of model performance.

Leave-one-out (LOO) is a special case of N-fold cross-validation where N is equal to the number of data samples. In LOO, the model is trained on all but one of the available data samples and evaluated on the left-out sample. This process is repeated for each data sample, so that each sample is left out once. The results from each iteration are then averaged to produce an overall estimate of model performance.

The advantage of N-fold cross-validation and LOO is that they provide a more accurate estimate of a model's generalization performance than simply evaluating the model on the training data. By using a separate validation set, these techniques can help to detect and prevent overfitting and provide a more realistic estimate of the model's performance on new, unseen data.

The choice of which technique to use often depends on the size of the available data and the computational resources available. N-fold cross-validation can be more computationally efficient than LOO, but may

require a larger amount of data to obtain accurate estimates of model performance. LOO can be more accurate for small datasets, but can be computationally expensive for larger datasets.

15. Know the definition of entropy and understand that uniform distribution offers the highest entropy for discrete random variables.

Entropy: In the context of information theory and machine learning, entropy is a measure of the amount of uncertainty or randomness in a system or data. It is commonly used as a measure of the impurity or disorder in a set of data for the purpose of building decision trees or other classification models.

The entropy of a set S is defined mathematically as:

$$H(S) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

where `p(i)` is the proportion of examples in S that belong to class i. The `logarithm base 2` is used in this formula, which means that the units of entropy are bits. The entropy is zero when all examples in S belong to the same class, indicating that there is no uncertainty about the class of new examples. Conversely, the entropy is maximal when the proportions of examples in S are evenly distributed across all classes, indicating a high degree of uncertainty.

Uniform distribution offers the highest entropy for discrete random variables: A uniform distribution offers the highest entropy for discrete random variables because it maximizes the degree of uncertainty or unpredictability in the distribution.

To understand why a uniform distribution has maximum entropy, consider a discrete random variable X that takes on values `x1, x2, ..., xn` with probabilities `p1, p2, ..., pn`, respectively. The entropy of X can be calculated using the formula:

$$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

where the summation is taken over all possible values of X.

If X is a uniform distribution, then the probability of each value xi is equal to 1/n, i.e., pi = 1/n for all i. Substituting this into the formula for entropy gives:

```
H(X) = -Σ (1/n) log2 (1/n)
= -(n/n) log2 (1/n)
= -log2 (1/n)
= log2 n
```

This shows that the entropy of a uniform distribution is proportional to the logarithm of the number of possible values, n. Since the logarithm function grows very slowly, the entropy of a uniform distribution is relatively high when the number of possible values is large.

In contrast, a distribution that is highly concentrated on a few values has low entropy because there is little uncertainty or unpredictability in the distribution. Thus, a uniform distribution offers the highest entropy for discrete random variables because it maximizes the degree of uncertainty or unpredictability in the distribution.

16. Understand that decision tree is not unique for a particular dataset (See tax return samples from the slides).

17. Know the concept of impurity, Gini, classification error, and their relations and trends given different probabilities.

In the context of decision trees and random forests, impurity measures such as Gini impurity and classification error are used to determine how well a split separates the data into classes. The impurity of a node in a decision tree is a measure of how mixed the classes are at that node.

Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. Gini impurity can be expressed mathematically as follows:

$$\text{Gini impurity} = 1 - \sum_{i=1}^{n} (p_i)^2$$

where `p_i` is the probability of the i'th class label in the subset.

Classification error is another measure of impurity that represents the proportion of misclassified instances at a node. It can be expressed mathematically as follows:

$$\text{Classification error} = 1 - \max_{i=1}^{n}(p_i)$$

where `p_i` is the probability of the i'th class label in the subset.

As the probabilities of the different class labels change, the values of the impurity measures also change. When all probabilities are equal (i.e., `p_i = 1/k` for all i, where k is the number of classes), both measures are at their maximum value of 0.5, indicating maximum impurity. When the probability of one class is 1 and the probabilities of all other classes are 0, both measures are at their minimum value of 0, indicating minimum impurity.

In general, the Gini impurity measure is more sensitive to changes in the probabilities of the class labels than the classification error measure. As the probabilities of the different class labels become more skewed (i.e., one class has a much higher probability than the others), the Gini impurity measure decreases faster than the classification error measure. This is because Gini impurity is more sensitive to the probability of the most frequent class label, while classification error is more sensitive to the probability of the least frequent class label.

Overall, the choice of impurity measure depends on the specific problem at hand and the desired behavior of the decision tree or random forest.

18. Can compute entropy, Gini, classification error, and Information Gain in decision tree (Assignment 2).

19. Can compute the pessimistic error in decision tree. (See samples from slides)

The pessimistic error is a measure of the expected error rate of a decision tree classifier that is based on the worst-case scenario. It assumes that the true probability of error is higher than what is observed in the training data, and it is used to avoid overfitting by penalizing complex trees.

An example of a decision tree with pessimistic error can be illustrated using the classic Iris dataset. Suppose we want to predict the species of an Iris flower based on its petal length and width. We can use a decision tree classifier to create a model that can classify new observations.

We can start by splitting the data based on the petal length. If the petal length is less than or equal to 2.45 cm, then we can predict the species as setosa. Otherwise, we can split the data further based on the petal width. If the petal width is less than or equal to 1.75 cm, then we can predict the species as versicolor. Otherwise, we can predict the species as virginica.

This simple decision tree has an error rate of 5% on the training data, which is very low. However, the pessimistic error takes into account the worst-case scenario, which is that the true probability of error is higher than 5%. To calculate the pessimistic error, we can add a penalty term that accounts for the complexity of the tree. The more complex the tree, the higher the penalty.

Suppose we set the penalty term to be equal to the square root of the number of leaves in the tree. In this case, the penalty term is equal to 2, since the tree has two leaves. The pessimistic error of the tree is therefore 5% + 2% = 7%.

By adding the penalty term, we are essentially assuming that the true probability of error is at least 7%. This pessimistic approach helps to prevent overfitting and ensures that the decision tree generalizes well to new data.

The formula to calculate the pessimistic error of a decision tree classifier is:

$$\text{Pessimistic Error} = \text{Training Error} + \text{Penalty Term}$$

The training error is the observed error rate of the decision tree on the training data, while the penalty term is a function of the complexity of the tree. The penalty term is added to the training error to account for the worst-case scenario, where the true error rate is higher than the observed error rate.

One common form of the penalty term is the **square root of** the number of leaves in the tree. This penalty term is a simple way to account for the complexity of the tree, and it is often used in practice. Other penalty terms can also be used, depending on the specific problem and the desired behavior of the decision tree.

The pessimistic error can be used as a measure of the expected error rate of the decision tree on new, unseen data. By penalizing complex trees, the pessimistic error helps to prevent overfitting and ensures that the decision tree generalizes well to new data.

20. Understand general decision boundary in machine learning and what that is in decision tree.

21. Understand concepts of true positive, false positive, precision, recall and their relations (true positive = recall).

True positive (TP) and false positive (FP) are concepts used in binary classification problems to describe the accuracy of a model's predictions. Precision and recall are performance metrics that quantify the model's accuracy using true positive, false positive, true negative, and false negative values.

True positive (TP) refers to the number of instances that are actually positive and are correctly predicted as positive by the model. False positive (FP) refers to the number of instances that are actually negative but are predicted as positive by the model.

Precision is a metric that measures the proportion of true positive predictions out of all the positive predictions made by the model. It is defined as:

$$\text{——}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Recall is a metric that measures the proportion of true positive predictions out of all the actual positive instances in the data. It is defined as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

where FN (false negative) is the number of instances that are actually positive but are incorrectly predicted as negative by the model.

Precision and recall are related in that both metrics are influenced by the number of true positive predictions made by the model. However, they differ in their focus. Precision focuses on the proportion of positive predictions that are correct, while recall focuses on the proportion of actual positive instances that are correctly predicted as positive.

It is not true that true positive is equal to recall. However, recall is directly influenced by the number of true positive predictions, as seen in the formula above. In other words, as the number of true positive predictions increases, the recall value also increases.

22. Can compute the classification cost based on confusion and cost matrix.

The classification cost based on a confusion matrix and cost matrix can be calculated by summing the products of the number of instances in each cell of the confusion matrix and their corresponding costs from the cost matrix.

Assuming a binary classification problem with classes A and B, the confusion matrix is a 2x2 matrix with the number of instances that are classified as true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN). The cost matrix is also a 2x2 matrix with the costs associated with each possible classification outcome.

The general formula for computing the classification cost based on the confusion matrix and cost matrix is:

$$\text{Classification Cost} = Cost(TP) * TP + Cost(FP) * FP + Cost(FN) * FN + Cost(TN) * TN$$

where Cost(TP), Cost(FP), Cost(FN), and Cost(TN) are the costs associated with true positives, false positives, false negatives, and true negatives, respectively. TP, FP, FN, and TN are the corresponding counts from the confusion matrix.

For example, let's assume the following confusion matrix and cost matrix:

Confusion Matrix:

```
| 25 5 |
| 3 17 |
```

Cost Matrix:

```
| 0 10 |
| 5 0 |
```

The classification cost can be computed as follows:

$$\text{Classification Cost} = Cost(TP) * TP + Cost(FP) * FP + Cost(FN) * FN + Cost(TN) * TN$$

```
= (0 * 25) + (10 * 5) + (5 * 3) + (0 * 17) = 85
```

Therefore, the classification cost based on the given confusion matrix and cost matrix is 85.

23. Understand the precision-recall curve and ROC curve and can read the graph and interpret the meaning.

24. Know basic concepts of similarity and distance, and their basic properties, e.g., identity, symmetry.

Distance is a measure of the dissimilarity between two objects, and it is a non-negative quantity that satisfies the following properties:

1. **Identity:** The distance between an object and itself is zero. That is, `d(x, x) = 0`.
2. **Symmetry:** The distance between two objects is the same regardless of the order in which they are considered. That is, `d(x, y) = d(y, x)`.
3. **Triangle Inequality:** The distance between two objects is always less than or equal to the sum of their distances to a third object. That is, `d(x, y) ≤ d(x, z) + d(z, y)`.

Some common distance measures include Euclidean distance, Manhattan distance, and cosine similarity.

Similarity is a measure of the degree of resemblance or likeness between two objects or data points, and it is a non-negative quantity that satisfies the following properties:

1. **Identity:** The similarity between an object and itself is maximum. That is, `sim(x, x) = 1`.
2. **Symmetry:** The similarity between two objects is the same regardless of the order in which they are considered. That is, `sim(x, y) = sim(y, x)`.
3. **Range:** The similarity between two objects lies between 0 and 1, where 0 indicates no similarity and 1 indicates maximum similarity.

Some common similarity measures include cosine similarity, Jaccard similarity, and Pearson correlation coefficient.

> 💡 In summary, distance and similarity are related but opposite concepts. Distance measures quantify the difference or dissimilarity between objects, while similarity measures quantify the degree of resemblance or likeness between objects. Both distance and similarity measures have important properties, such as identity, symmetry, and range, that make them useful in many applications of data analysis and machine learning.

25. Understand L1, L2 distance and their meanings, Hamming distance, Manhattan distance, etc.

L1 distance, also known as **Manhattan distance** or taxicab distance, is a measure of distance between two points in a two-dimensional or n-dimensional space. It is defined as the sum of the absolute differences of their coordinates. Mathematically, the L1 distance between two points p and q in n-dimensional space is given by:

$$\text{L1 distance}(p, q) = \sum_{i=1}^{n} |p_i - q_i|$$

where `p_i` and `q_i` are the i^th coordinates of points p and q, respectively.

L2 distance, also known as **Euclidean distance**, is another measure of distance between two points in a two-dimensional or n-dimensional space. It is defined as the square root of the sum of the squared differences of their coordinates. Mathematically, the L2 distance between two points p and q in n-dimensional space is given by:

$$\text{L2 distance}(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

The L2 distance is a commonly used distance measure in many applications, such as clustering and classification.

Hamming distance is a measure of distance between two binary strings of equal length. It is defined as the number of positions at which the corresponding symbols are different. Mathematically, the Hamming distance between two binary strings a and b of length n is given by:

$$\text{Hamming distance}(a, b) = \sum_{i=1}^{n} (a_i \neq b_i)$$

where a_i and b_i are the ith symbols of strings a and b, respectively.

The Hamming distance is used in applications such as error-correcting codes and DNA sequence analysis.

Manhattan distance is another name for L1 distance. It is called Manhattan distance because it is similar to the distance a taxicab would travel on a rectangular street grid to reach a destination. The term "Manhattan" is used because the streets of Manhattan form a rectangular grid.

In summary, L1 distance measures the sum of the absolute differences between coordinates, L2 distance measures the square root of the sum of the squared differences between coordinates, and Hamming distance measures the number of differing symbols between binary strings. These distance measures have different applications in various fields such as computer vision, data mining, and bioinformatics.

26. Understand the concept of editing distance.

The editing distance, also known as Levenshtein distance, is a measure of similarity between two strings of characters. It is defined as the minimum number of edit operations (insertions, deletions, or substitutions) required to transform one string into another.

For example, consider the strings "kitten" and "sitting". The editing distance between these two strings can be computed as follows:

1. Substitute "s" for "k" to obtain "sitten" (1 edit operation).
2. Substitute "i" for "e" to obtain "sittin" (1 edit operation).
3. Substitute "g" for "t" to obtain "sitting" (1 edit operation).

Therefore, the editing distance between "kitten" and "sitting" is 3.

The editing distance can be computed recursively using dynamic programming. Let D(i, j) be the editing distance between the first i characters of string A and the first j characters of string B. Then, the editing distance can be computed as follows:

```
D(i, j) = 0, if i = 0 and j = 0 (base case)
D(i, j) = i, if j = 0 (delete all i characters in A)
D(i, j) = j, if i = 0 (insert all j characters in B)
D(i, j) = D(i-1, j-1), if A[i] = B[j] (no edit required)
D(i, j) = 1 + min(D(i-1, j), D(i, j-1), D(i-1, j-1)), otherwise (one edit required)
```

The final result is given by D(n, m), where n is the length of string A and m is the length of string B.