CBCS SCHEME

USN | | | | | | | | | | |

21AD62

Sixth Semester B.E. Degree Examination, June/July 2024
Data Science & its Applications

Time: 3 hrs. Max. Marks: 100

Note: Answer any FIVE full questions, choosing ONE full question from each module.

Module-1

1 a. Describe dispersion and variance and write the python code to compute the variance. **(07 Marks)**

b. Discuss random variables with an example in detail. **(07 Marks)**

c. Explain standard deviation and interquartile range and write python code to compute standard deviation and interquartile range. **(06 Marks)**

OR

2 a. Explain Bar Chart, Line Chart and Histogram with help of diagram. **(07 Marks)**

b. Discuss Conditional probability with an example in detail. **(07 Marks)**

c. Explain Correlation and describe the impact of outlier on correlation. **(06 Marks)**

Module-2

3 a. Explain P-Values with an example. **(07 Marks)**

b. Write Python program to plot Line chart by assuming your own data and explain the various attributes of line chart. **(06 Marks)**

c. Describe A/B test with an example. **(07 Marks)**

OR

4 a. A certain disease affects 1% of the population. A test for the disease has a 99% sensitivity (true positive rate) and a 99% specificity (true negative rate). If a person tests positive, what is the probability that they actually have the disease? **(07 Marks)**

b. Describe how data can be manipulated by considering an example. **(06 Marks)**

c. Explain cleaning and munging of data with an example. **(07 Marks)**

Module-3

5 a. Explain support vector machines in detail. **(07 Marks)**

b. Discuss digression in detail. **(06 Marks)**

c. Discuss the need for fitting the model in multiple regressions. **(07 Marks)**

OR

6 a. Discuss Goodness of Fit in detail. **(06 Marks)**

b. Write Python snippet for Accuracy, Precision, Recall and F_1 score. **(07 Marks)**

c. Explain Feature Extraction and Feature selection. **(07 Marks)**

Module-4

7 a. Discuss perceptron neural network in detail. **(10 Marks)**

b. Explain layer abstraction in deep learning. **(10 Marks)**

OR

8	a.	Write python program to compute loss and optimization in deep learning.	**(10 Marks)**
	b.	Explain feed forward neural network in detail with a neat diagram.	**(10 Marks)**

Module-5

9	a.	Describe n-Gram language models in detail.	**(10 Marks)**
	b.	Explain Eigen Vector centrality in detail.	**(10 Marks)**

OR

10	a.	Explain item based collaborative filtering.	**(10 Marks)**
	b.	Discuss matrix factorization in detail.	**(10 Marks)**

* * * * *

21AD62-DATA SCIENCE AND APPLICATIONS

VTU ANSWER KEY

1a) **1. Dispersion**

In general statistical terms, **dispersion** refers to the extent to which data points in a dataset differ from a central value (such as the mean or median). It is a broad term that encompasses any measure that quantifies how spread out the data is.

Some common measures of dispersion include:

- **Range**: The difference between the maximum and minimum values in a dataset.
- **Variance**: A measure of how far each data point is from the mean, on average (squared differences).
- **Standard Deviation**: The square root of the variance, providing a measure of spread in the original units of the data.
- **Interquartile Range (IQR)**: The range between the 25th and 75th percentiles of the data (middle 50% of the data).

So, **dispersion** is a general term for any method that measures spread or variability, and variance is one specific measure of dispersion.

2. Variance

Variance is a **specific statistical measure of dispersion** that quantifies the average squared deviation of each data point from the mean. In simple terms, it gives you an idea of how spread out the data is around the mean value.

Formula for Variance:

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$$

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

```python
def calculate_variance(data):
    # Calculate mean
    mean = sum(data) / len(data)

    # Calculate variance
    variance = sum((x - mean) ** 2 for x in data) / len(data)
    return variance

def calculate_dispersion(data):
    # Dispersion can be calculated as the range (max - min) of the dataset
    return max(data) - min(data)
```

1b) A **random variable** is a numerical outcome of a random process or experiment. It is a function that assigns a real number to each outcome of a random experiment. Random variables are used to quantify the outcomes of stochastic (random) phenomena and are categorized into two main types:

1. **Discrete Random Variables**: These can take on a finite or countably infinite set of values. Examples include the number of heads when flipping a coin multiple times or the number of cars passing through a toll booth in a given time period.
2. **Continuous Random Variables**: These can take on any value within a continuous range (which could be an interval of real numbers). Examples include the height of a person or the time taken for a chemical reaction to complete.

1c) **Standard Deviation (SD)**

The **standard deviation** (SD) is a measure of the spread or dispersion of a set of data points around the **mean** (average). It quantifies how much individual data points deviate from the mean of the dataset. The greater the standard deviation, the more spread out the data is. A smaller standard deviation indicates that the data points are closer to the mean.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

Interquartile Range (IQR)

The **Interquartile Range** (IQR) is a measure of statistical dispersion, specifically the range within which the middle 50% of the data lies. It is the difference between the **third quartile** (Q3) and the **first quartile** (Q1).

Formula for IQR

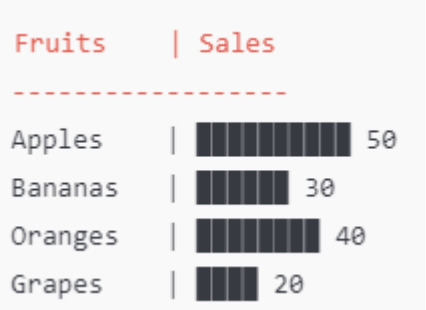
$$IQR = Q3 - Q1$$

Where:

- $Q3$ = Third quartile (75th percentile)
- $Q1$ = First quartile (25th percentile)

2a) **1. Bar Chart**

A **bar chart** (or **bar graph**) is a graphical representation of data in which individual bars represent categories or discrete data points. The length of each bar is proportional to the value or frequency of the category it represents.



2. Line Chart

A **line chart** (or **line graph**) is a type of chart used to represent data points over time or ordered categories, with a continuous line connecting the points.

Use Case

- **Time series data**: Line charts are best for visualizing data that changes over time, such as stock prices, temperature variations, or sales trends.
- It shows the **trend** or **pattern** of data points, making it easier to see increases, decreases, or cyclical changes.

3. Histogram

A **histogram** is a graphical representation of the **distribution of a dataset**, where data is divided into **bins** or **intervals**, and the frequency of data points within each bin is displayed using bars.

Use Case

- **Continuous data**: Histograms are used when you want to see how data is distributed across continuous intervals. It's especially useful for visualizing the **frequency distribution** of numerical data.

2b) **Conditional probability** is the probability of an event occurring given that another event has already occurred. In mathematical terms, the conditional probability of event A given event B is denoted as:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Where:

- $P(A|B)$ is the probability of event A occurring given that event B has already occurred.
- $P(A \cap B)$ is the probability of both events A and B occurring.
- $P(B)$ is the probability of event B occurring.

2c) **Correlation** is a statistical measure that expresses the extent to which two variables are related. It provides insights into the relationship between variables, whether they increase or decrease together (positive correlation), or whether one increases while the other decreases (negative correlation). However, **outliers** can have a significant impact on correlation, sometimes distorting the results and leading to misleading conclusions.

3a) The **p-value** is the probability that the observed results (or more extreme results) would occur if the **null hypothesis** were true. In simpler terms, it measures how compatible your data is with the null hypothesis. The null hypothesis generally represents the idea that there is **no effect** or **no relationship** between the variables.

- A **small p-value** (typically less than or equal to 0.05) suggests that the observed data is **unlikely** under the null hypothesis, leading to the rejection of the null hypothesis.
- A **large p-value** (greater than 0.05) suggests that the observed data is **likely** under the null hypothesis, and thus there is **no significant evidence** to reject the null hypothesis. +EXAMPLE

3b) import matplotlib.pyplot as plt

hours_spent_studying = [10, 9, 2, 15, 10, 16, 11, 16]

scores_in_final_exam = [95, 80, 10, 50, 45, 98, 38, 93]

Plotting the data

plt.plot(hours_spent_studying, scores_in_final_exam, marker='*', color='red', linestyle='-')

Adding labels and title

plt.xlabel('Number of Hours Spent Studying')

plt.ylabel('Score in Final Exam')

plt.title('Effect of Study Hours on Exam Performance')

Displaying the plot

plt.grid(True)

plt.show()

3c) **A/B Testing** is a controlled experiment that compares two versions (A and B) of a single variable to determine which one performs better. It's widely used in marketing, web development, and product testing to optimize user experiences, design changes, or business decisions. +EXAMPLE

4a)

The formula for Bayes' Theorem is:

$$P(\text{Disease}|\text{Positive Test}) = \frac{P(\text{Positive Test}|\text{Disease}) \cdot P(\text{Disease})}{P(\text{Positive Test})}$$

1. Calculate $P(\text{Positive Test})$:
   - $P(\text{Positive Test}|\text{Disease}) = 0.99$ (sensitivity)
   - $P(\text{Disease}) = 0.01$
   - $P(\text{Positive Test}|\text{No Disease}) = 0.01$ (false positive rate)
   - $P(\text{No Disease}) = 0.99$

$$P(\text{Positive Test}) = (0.99 \times 0.01) + (0.01 \times 0.99)$$

$$P(\text{Positive Test}) = 0.0099 + 0.0099 = 0.0198$$

2. Calculate the posterior probability using Bayes' Theorem:

$$P(\text{Disease}|\text{Positive Test}) = \frac{(0.99 \times 0.01)}{0.0198}$$

$$P(\text{Disease}|\text{Positive Test}) = \frac{0.0099}{0.0198} \approx 0.5$$

4b) **1. Data Cleaning**

Data cleaning (or data scrubbing) is the process of identifying and correcting errors or inconsistencies in the data. Common tasks in this step include:

- **Handling missing data**: Fill missing values using methods like mean/median imputation, forward/backward fill, or dropping missing rows/columns.
- **Outlier detection**: Identify and treat outliers, which may skew the results of analysis or modeling.
- **Removing duplicates**: Detect and remove duplicate records to ensure that the dataset doesn't have redundant information.
- **Standardization of data types**: Ensure that all columns are in the appropriate data type (e.g., numeric, categorical, datetime).
- **Fixing data entry errors**: Correct typos, inconsistent formatting, or incorrect values.

2. Data Transformation

Data transformation refers to modifying the format, structure, or values of data to make it suitable for analysis:

- **Scaling and normalization**: Rescale data to a standard range, such as transforming numerical features to have zero mean and unit variance, or rescaling values to a specific range (e.g., between 0 and 1).

- **Feature engineering**: Creating new features based on existing data (e.g., extracting day of the week from a date column or creating interaction terms between features).
- **One-hot encoding**: Convert categorical variables into binary vectors, where each category in a column becomes a separate column.
- **Log transformation**: Apply logarithmic transformations to skewed data to make it more normally distributed.
- **Binning**: Grouping continuous values into discrete categories, such as grouping ages into ranges (e.g., 18-25, 26-35).

4c) **Data Munging (also known as Data Wrangling) is the process of transforming raw, unstructured, or messy data into a clean, structured format that is suitable for analysis and modeling. It involves a series of techniques that are typically required to reshape, clean, and format data so that it can be easily used in statistical analysis, machine learning models, or visualization.**

Data Cleaning

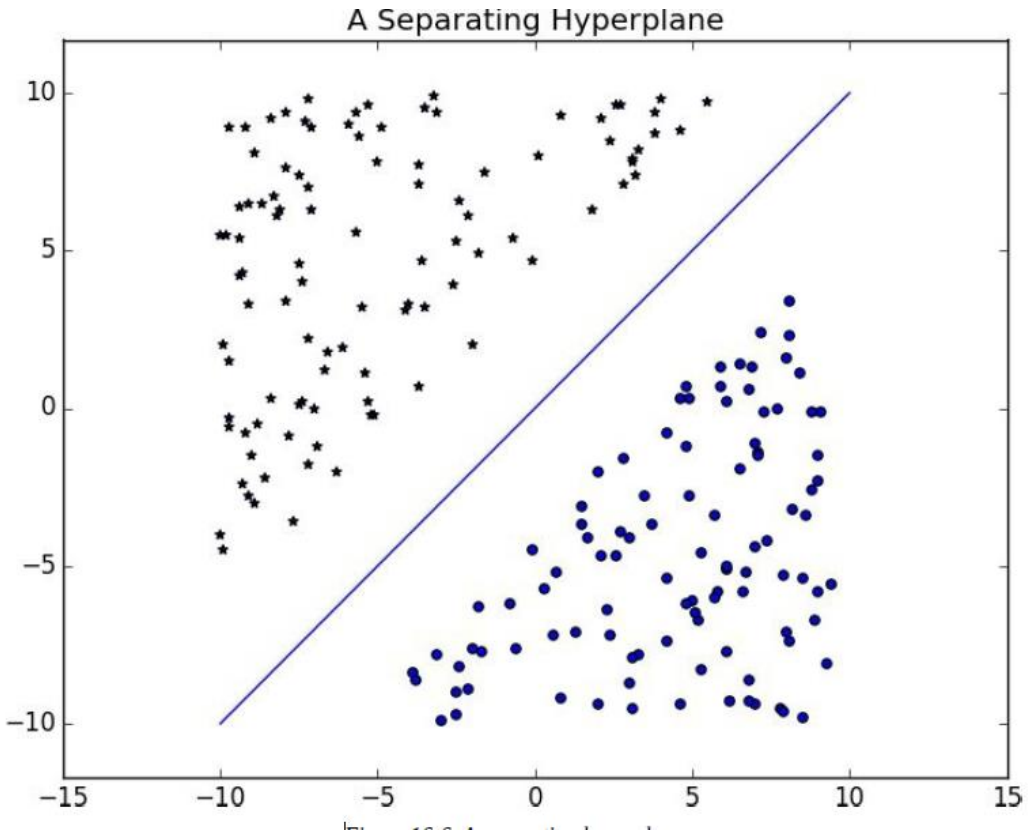
Data cleaning focuses on fixing errors and inconsistencies in the dataset to ensure the data is correct and usable.

Common Data Cleaning Tasks:

- **Handling Missing Data**
    - **Detect missing data**: Missing values can appear as NaN, None, or an empty cell in datasets.
    - **Ways to handle missing data**:
        - **Imputation**: Fill missing values with a default value, the mean, median, or mode of the column.
        - **Forward fill / Backward fill**: Use the preceding or succeeding value in a time-series or ordered dataset.
        - **Dropping missing values**: Remove rows or columns with missing values if they are insignificant or sparse.
        - **Custom imputation**: Use domain-specific knowledge to fill in missing data (e.g., replacing missing birth dates with the median or mode).

5a) An alternative approach to classification is to just look for the hyperplane that “best” separates the classes in the training data. This is the idea behind the support vector

machine, which finds the hyperplane that maximizes the distance to the nearest point in each class.



5b) Rather than training each tree on all the inputs in the training set, we train each tree on the result of bootstrap_sample(inputs). Since each tree is built using different data, each tree will be different from every other tree. (A side benefit is that it's totally fair to use the nonsampled data to test each tree, which means you can get away with using all of your data as the training set if you are clever in how you measure performance.) This technique is known as bootstrap aggregating or bagging

5c)

Now imagine that each input x_i is not a single number but rather a vector of k numbers $x_{i1}, \ldots, x_{ik}$. The multiple regression model assumes that:

$$y_i = \alpha + \beta_1 x_{i1} + \ldots + \beta_k x_{ik} + \varepsilon_i$$

In multiple regression the vector of parameters is usually called β. We'll want this to include the constant term as well, which we can achieve by adding a column of ones to our data:

```
beta = [alpha, beta_1, ..., beta_k]
```

and:

```
x_i = [1, x_i1, ..., x_ik]
```

6a)Goodness of fit-Mean,R-squared

6b) **def** accuracy(tp, fp, fn, tn):

correct = tp + tn

total = tp + fp + fn + tn

return correct / total

def precision(tp, fp, fn, tn):

return tp / (tp + fp)

def recall(tp, fp, fn, tn):

return tp / (tp + fn)

def f1_score(tp, fp, fn, tn):

p = precision(tp, fp, fn, tn)

r = recall(tp, fp, fn, tn)

6c) *Features* are whatever inputs we provide to our model.

In the simplest case, features are simply given to you. If you want to predict someone'salary based on her years of experience, then years of experience is the only feature you

have. That's where a combination of *experience* and *domain expertise* comes into play. +Example

7a) *perceptron*, which approximates a single

neuron with *n* binary inputs. It computes a weighted sum of its inputs and "fires" if that

weighted sum is zero or greater:

def step_function(x):

return 1 **if** x >= 0 **else** 0

def perceptron_output(weights, bias, x):

"""returns 1 if the perceptron 'fires', 0 if not"""

calculation = dot(weights, x) + bias

return step_function(calculation)

7b)Write about neural networks

8a) import numpy as np

import matplotlib.pyplot as plt

```
# Generate some random data for the example
np.random.seed(0)
X = 2 * np.random.rand(100, 1)  # 100 random data points in one feature
y = 4 + 3 * X + np.random.randn(100, 1)  # Linear relation with some noise

# Initialize model parameters (weights) randomly
theta = np.random.randn(2, 1)  # Random initialization of weights and bias

# Add x0 = 1 for the bias term (intercept)
X_b = np.c_[np.ones((100, 1)), X]  # Add bias term (column of ones)

# Mean Squared Error (MSE) loss function
def compute_mse_loss(X, y, theta):
    m = len(y)
    predictions = X.dot(theta)
    loss = (1 / (2 * m)) * np.sum((predictions - y) ** 2)
    return loss

# Gradient of MSE loss function (for linear regression)
def compute_gradient(X, y, theta):
    m = len(y)
    predictions = X.dot(theta)
    gradients = (1 / m) * X.T.dot(predictions - y)
    return gradients

# Gradient Descent for Optimization
def gradient_descent(X, y, theta, learning_rate=0.1, n_iterations=1000):
    loss_history = []
```

```
    for iteration in range(n_iterations):
        gradients = compute_gradient(X, y, theta)
        theta -= learning_rate * gradients  # Update the parameters
        loss = compute_mse_loss(X, y, theta)  # Compute loss
        loss_history.append(loss)

    return theta, loss_history

# Apply Gradient Descent to optimize the parameters
optimized_theta, loss_history = gradient_descent(X_b, y, theta, learning_rate=0.1, n_iterations=1000)

# Print optimized parameters (weights and bias)
print("Optimized parameters (theta):", optimized_theta)

# Plot the loss history over iterations
plt.plot(loss_history)
plt.title('Loss over Iterations')
plt.xlabel('Iterations')
plt.ylabel('MSE Loss')
plt.show()

# Plot the regression line
plt.scatter(X, y, color='blue', label='Data points')
plt.plot(X, X_b.dot(optimized_theta), color='red', label='Fitted line')
plt.xlabel('Feature X')
plt.ylabel('Target y')
plt.legend()
plt.show()
```

8b) *feed-forward* neural network that consists of discrete *layers* of neurons,

each connected to the next. This typically entails an input layer (which receives inputs and feeds them forward unchanged), one or more “hidden layers” (each of which consists of neurons that take the outputs of the previous layer, performs some calculation, and passes the result to the next layer), and an output layer (which produces the final outputs).

ust like the perceptron, each (noninput) neuron has a weight corresponding to each of its inputs and a bias. To make our representation simpler, we’ll add the bias to the end of our weights vector and give each neuron a *bias input* that always equals 1.

def neuron_output(weights, inputs):

return sigmoid(dot(weights, inputs))

9a) An **N-gram** is a contiguous sequence of N items from a given text or speech. These items can be words, characters, or other linguistic units.

- **Unigram (1-gram)**: A sequence of one word.
    - Example: "I", "like", "pizza"
- **Bigram (2-gram)**: A sequence of two consecutive words.
    - Example: "I like", "like pizza", "pizza is"
- **Trigram (3-gram)**: A sequence of three consecutive words.
    - Example: "I like pizza", "like pizza is"

And so on for higher-order N-grams (4-gram, 5-gram, etc.).

2. Basic Idea

In an N-gram model, you predict the probability of a word occurring based on the previous N-1 words. For example, in a bigram model, you would predict the probability of the next word given the previous word.

9b) **Eigenvector Centrality** is a centrality measure used in **graph theory** and **network analysis** that identifies the most important nodes in a network based on the connections they have to other important nodes. It's a measure of **relative influence** where nodes that are connected to other high-influence nodes are considered more central or important.

Formally, the **Eigenvector Centrality** of a node vvv is the iii-th component of the **eigenvector** corresponding to the largest eigenvalue of the adjacency matrix AAA of the network.

The centrality x of node v is given by the equation:

$$A \cdot x = \lambda \cdot x$$

Where:

- A is the adjacency matrix of the network (i.e., a matrix where element A_{ij} represents the weight or presence of an edge between node i and node j).
- x is the vector of centralities (the eigenvector).
- λ is the largest eigenvalue (also called the **principal eigenvalue**) of the matrix A.

10 a) **Item-Based Collaborative Filtering**

- **Similarity between items**: The system computes the similarity between items based on users' interactions (ratings, views, etc.). This similarity is typically measured using similarity metrics like **cosine similarity**, **Pearson correlation**, or **adjusted cosine similarity**.
- **Recommendation**: To recommend items to a user, the system looks at the items that the user has interacted with (e.g., items they have rated or liked) and recommends other items that are similar to those items based on the calculated similarities.

Steps for Item-Based Collaborative Filtering

1. **Collect the Data**: You need data that shows how users interact with items. This could be ratings, clicks, or purchases.
2. **Construct the Item-Item Similarity Matrix**: Compute the similarity between all pairs of items. For example, for each pair of items, measure how often they are rated similarly by users.
3. **Generate Recommendations**: Once the similarity matrix is built, for each user, recommend items that are most similar to the items they have already interacted with (i.e., items that have high similarity scores).

10 b) **Matrix Factorization Concept**

Let's assume you have a **user-item interaction matrix** RRR, where rows represent users and columns represent items, and the values represent interactions (such as ratings).

For a matrix R of size $m \times n$ (where m is the number of users and n is the number of items), matrix factorization tries to approximate R by the product of two smaller matrices:

- P: An $m \times k$ matrix (user matrix).
- Q: A $k \times n$ matrix (item matrix).

Here k is the **latent dimensionality** or the number of hidden factors. The idea is that each user and each item is represented by a vector in a **latent factor space** of dimensionality k. The objective of matrix factorization is to find these latent vectors that, when multiplied, approximate the original user-item interaction matrix R.

$$R \approx P \times Q^T$$

Where:

- P contains the **user feature vectors** (late ctors for users).
- Q contains the **item feature vectors** (latent factors for items).