Reasoning Qs of ML for Mid I

Identify whether the below statements are True or False.

1. Training error is high when a machine learning model undergoes underfitting.

✅ **True** — underfitting means the model is too simple, it performs poorly even on the training data, hence high training error.

2. The training error or accuracy doesn't provide an (optimistically) biased estimate of the generalization performance in the machine learning model.

❌ **False** — training accuracy is usually **optimistic** (better than test performance), so it **is biased** as an estimate of generalization.

3. KNN algorithm is considered as a parametric model.

❌ **False** — KNN is **non-parametric**, because it does not assume a fixed functional form; complexity grows with training data. k is a hyperparameter not a paramter.

4. Hamming distance is just like the Manhattan distance applied to binary feature vectors.

✅ **True** — for binary vectors, Manhattan distance = sum of absolute differences = number of mismatched bits = Hamming distance.

5. Increasing the value of k in KNN generally results in a model with higher bias and lower variance.

✅ **True** — small k = flexible (low bias, high variance). Large k = smoother (high bias, low variance).

6. In KNN, the value of k represents the number of features in the dataset.

❌ **False** — k = number of **nearest neighbors** considered for classification/regression. Number of features is fixed by dataset.

7. If 50 Patients are actually diagnosed as diabetic, and the model correctly identifies 45 of them as diabetic. 50 patients are not diabetic, but the model incorrectly classifies 10 of them as diabetic. The F1-score of the model would be 85.6%.

Let's compute:

- **TP = 45** (diabetic correctly identified)
- **FN = 5** (missed diabetics)
- **FP = 10** (non-diabetics misclassified)
- **TN = 40**

Precision = TP / (TP+FP) = 45 / (45+10) = 45/55 ≈ **0.818**

Recall = TP / (TP+FN) = 45 / 50 = **0.9**

F1 = 2 × (P×R) / (P+R) = 2×(0.818×0.9)/(0.818+0.9)

= 2×0.736 / 1.718 ≈ **0.857 = 85.7%**

Given statement: 85.6% ≈ matches (rounding).

✅ **True**

Q: Suppose we increased the size of the testing set. Would this likely improve or deteriorate the performance of the model on new data? Why?

Answer: Increasing the size of the testing set does not improve the model itself because training is already complete. However, it makes the **performance evaluation more reliable**, since more test data provides a better estimate of how the model will behave on unseen data. Thus, the reported performance (accuracy, error, etc.) becomes more stable and trustworthy, but the actual model performance does not improve.

In short: it neither improves nor worsens the model but gives a more accurate estimate of its performance.

Q: Machine learning studies the design and development of algorithms that learn from the data and improve their performance through experience. Justify the statement by giving real life example.

Answer: Machine Learning focuses on building algorithms that automatically learn patterns from data and improve over time without being explicitly programmed. For example, in **spam email detection**, the model is trained on a dataset of emails labeled as “spam” or “not spam.” Over time, as it processes more emails, it learns to recognize new spam patterns (like suspicious keywords or sender addresses) and improves its ability to filter out spam messages. This shows how algorithms can learn from data and experience to perform better.

Q: Define the terms ‘Model training’ and ‘Feature vector’?

Answer:

- **Model Training:** Model training is the process of feeding data into a machine learning algorithm so it can learn the underlying patterns and relationships. During training, the model adjusts its internal parameters to minimize error and improve predictions on unseen data.

- **Feature Vector:** A feature vector is a numerical representation of an object or instance that the machine learning model uses as input. Each element of the vector corresponds to a measurable attribute (feature). For example, in predicting house prices, a feature vector may include `[size, number_of_rooms, location_score]`.

Q: Why is it called the bias–variance tradeoff?

Answer:

The bias–variance tradeoff describes the balance between two types of errors in a machine learning model:

- **Bias:** Error due to overly simple assumptions in the model (underfitting). High bias means the model cannot capture important patterns in data.
- **Variance:** Error due to excessive sensitivity to training data (overfitting). High variance means the model performs well on training data but poorly on unseen data.

The "tradeoff" comes from the fact that reducing bias usually increases variance, and reducing variance usually increases bias. The goal is to find the right balance where total error (bias² + variance + noise) is minimized.

Q: Explain the concept of cross-validation in machine learning.

Answer:

Cross-validation is a technique used to **evaluate model performance** more reliably by splitting the dataset into multiple parts (folds). The most common method is **k-fold cross-validation**, where the dataset is divided into *k* equal folds. The model is trained on *k-1* folds and tested on the remaining fold, and this process is repeated *k* times with different test folds each time.

The final performance is the average across all folds. This helps:

- Reduce overfitting,
- Provide a better estimate of generalization,

- Ensure the evaluation is not biased by one particular train-test split.

Q: What is the purpose of splitting the dataset into training, validation, and testing sets?

Answer:

Splitting the dataset helps evaluate and tune machine learning models effectively:

- **Training Set:** Used to train the model by adjusting its parameters.
- **Validation Set:** Used to tune hyperparameters and select the best model configuration without touching the test set.
- **Testing Set:** Used at the very end to estimate the true generalization performance of the final model on unseen data.

This ensures that the model is not just memorizing data but can generalize well.

Q: Differentiate between Parameters and Hyperparameters in machine learning.

Answer:

- **Parameters:** Internal variables of a model learned automatically during training. Example: weights and biases in a neural network, or coefficients in linear regression.
- **Hyperparameters:** External settings chosen **before training** that control the learning process. They are not learned from data but set by the practitioner. Example: learning rate, number of neighbors (*k*) in KNN, number of trees in Random Forest.

👉 Parameters are learned, hyperparameters are chosen.

6. Why does increasing the dataset size often reduce variance?

Increasing the dataset size provides the model with more diverse examples of the underlying data distribution. This reduces the model's sensitivity to noise in the

training data, leading to more stable predictions on new data, and therefore lower variance.

7. What happens when a very small value of K is selected in nearest neighbor?

If K is very small (e.g., K = 1), the model becomes highly sensitive to noise in the training data, which can cause overfitting. It may capture idiosyncrasies of individual data points rather than general patterns.

8. Below are some applications, write from which ML problem they belong to:

a. Identify objects or entities within images → **Computer Vision / Image Classification / Object Detection (Supervised Learning).**

b. Forecast crop yields for agriculture → **Regression (Supervised Learning).**

c. Determine the sentiment of a piece of text → **Natural Language Processing / Text Classification (Supervised Learning).**

9. KNN is known as instance-based learning. Why?

KNN does not learn an explicit model. Instead, it stores all training instances and makes predictions by comparing a new input to these stored instances. Predictions are based directly on the similarity (distance) to the nearest neighbors, which is why it is called **instance-based learning**.

10. How the choice of K affects bias and variance

- **Small K (e.g., K=1):**
  - Low bias → model closely fits training data
  - High variance → sensitive to noise → overfitting
- **Large K:**

- High bias → smooth decision boundary → may miss local patterns → underfitting
- Low variance → less sensitive to individual noisy points

Tip: Choose K via **cross-validation** for best trade-off.

11. Effect of unnormalized features

- KNN uses **distance metrics** (Euclidean, Manhattan).
- Features with larger scale dominate distances → small-scale features are ignored

  Example:

- Feature 1 (Height in meters): 1–2
- Feature 2 (Income in thousands): 10–100
- Income dominates Euclidean distance → model biased toward it

Solution: Normalize/standardize all features (Min-Max or Z-score).

12. Effect of distance metric choice

- **Euclidean:** Straight-line distance → sensitive to large differences
- **Manhattan:** Sum of absolute differences → less sensitive to outliers
- **Minkowski:** General form → parameter `p` adjusts distance type
  - p=1 → Manhattan, p=2 → Euclidean

Impact: Choice changes nearest neighbors → may change predictions.

13. Why KNN is “lazy learning” / instance-based

- **No model is explicitly trained**
- KNN **stores all training instances**

- Predictions are made on-demand using nearest neighbors
- Pros: Simple, flexible
- Cons: Slow for large datasets, requires memory

14. Handling categorical features in KNN

- Convert categories to **one-hot encoding** → avoids false ordinal relationships
- For binary/categorical data, **Hamming distance** can be used instead of Euclidean
- Example: Color = {Red, Blue, Green} → [1,0,0], [0,1,0], [0,0,1].

15. Effect of imbalanced classes

- KNN may be biased toward **majority class**
- Example: 90% class A, 10% class B → nearest neighbors likely class A → minority ignored
- **Solutions:**
  - Weighted KNN (closer neighbors count more)
  - Oversampling minority class or undersampling majority class
  - Adjust distance or voting to account for class imbalance

16. What happens if you train and test on the same dataset

- The model may appear to perform extremely well (**low training error**), but **generalization is poor**.
- High risk of **overfitting** → model memorizes training data instead of learning patterns.
- Test metrics are **misleading**, giving false confidence in the model.

17. How cross-validation reduces overfitting

- **K-Fold Cross-Validation:**
  - Split dataset into K folds
  - Train on K-1 folds, validate on 1 fold → repeat K times
  - Average validation error gives **robust estimate** of performance
- **Effect:**
  - Reduces reliance on a single train/validation split
  - Helps select hyperparameters that **generalize better**

18. Detecting underfitting and overfitting using validation error

Scenario	Training Error	Validation Error	Reasoning
Underfitting	High	High	Model too simple, cannot capture patterns
Overfitting	Low	High	Model too complex, memorizes training data
Good Fit	Low/Medium	Low/Medium	Model captures patterns without memorizing

19. Why do we minimize a loss function during training

- Loss function quantifies **how far predictions are from actual labels**.
- Minimizing loss:
  - Guides the **model to better fit training data**
  - Optimizes parameters (weights) to **reduce prediction error**
- Example: Gradient descent updates weights to reduce loss.

20. How weighting classes helps with imbalanced data

- Problem: Minority class contributes less to loss → model biased toward majority class
- **Weighted loss:** Assign higher weight to minority class → increases its impact on optimization

21. What is Class Imbalance?

- **Class imbalance** occurs when one class (majority) has **much more data** than another class (minority).
- Example: In a medical dataset, 95% of patients are healthy (class A), 5% are sick (class B).

Problem: Standard metrics like accuracy can be **misleading** because predicting the majority class most of the time still gives high accuracy.

22. Why Accuracy Can Be Misleading

Accuracy formula:

$$\text{Accuracy} = TP+TN/(TP+TN+FP+FN)$$

Example:

Actual	Predicted
A	A
A	A
A	A
B	A
B	A

- Total samples = 5
- Correct predictions = 3 (TP for class A)
- Accuracy = 3/5 = 60%

Even though **all minority class B instances are misclassified**, accuracy seems **moderately good**, hiding poor performance on the minority class.

Extreme example:

- 90% class A, 10% class B
- Model predicts **all as A** → Accuracy = 90%
- But **minority class never detected** → model is practically useless for class B

23. Why F1-score is Better

F1-score formula:

$F1 = 2 \cdot Precision \cdot Recall / Precision + Recall$

- **Precision:** Out of all predicted positives, how many are correct
- **Recall:** Out of all actual positives, how many are detected

Example (minority class B):

- Predicted all as majority → TP = 0, FP = 0, FN = 10
- Precision = 0, Recall = 0 → F1 = 0

Interpretation:

- F1-score **penalizes the model** for failing to predict the minority class.
- Unlike accuracy, F1 considers **both false positives and false negatives**, making it more reliable for imbalanced datasets

Metric	Sensitive to imbalance?	Pros	Cons
Accuracy	Yes	Easy to compute	Can be misleading for minority class
F1-score	No	Balances precision and recall, highlights minority class performance	Slightly more complex

24. Label Encoding vs One-Hot Encoding

Encoding Type	Description	Example (Color: Red, Blue, Green)
Label Encoding	Assigns integer values to categories (ordinal mapping)	Red → 1, Blue → 2, Green → 3
One-Hot Encoding	Creates a separate binary vector for each category	Red → [1,0,0], Blue → [0,1,0], Green → [0,0,1]

25. Why One-Hot Encoding Avoids Ordinal Assumptions

- Label encoding implies **order** (e.g., Green > Blue > Red)
- KNN and distance-based models treat numeric values as **continuous** → may introduce false distances
- One-hot encoding ensures **all categories are equidistant**, avoiding bias from artificial order

26. When Label Encoding is Harmful in KNN / Distance-Based Algorithms

- KNN calculates **distance** between feature vectors
- Label encoded categories may suggest **false closeness**:
  - Example: Red=1, Green=3 → distance between Red and Green = 2, but actually categories are **nominal**
- Can mislead predictions → one-hot encoding preferred for categorical features

27. How Encoding Affects Cosine Similarity / Euclidean Distance

- **Cosine Similarity:** Measures angle between vectors

    - Label encoding may give large vector differences → artificially reduces similarity
    - One-hot ensures **correct binary vector comparison**
- **Euclidean Distance:** Directly affected by numerical value differences
    - Label encoding creates **false distances**, while one-hot treats each category equally

28. Model, Hypothesis, Algorithm – Differentiation

Term	Definition
Algorithm	Step-by-step procedure to find a model (e.g., KNN, Gradient Descent)
Model	Learned function that predicts output from input
Hypothesis	Candidate function within the hypothesis space that maps inputs to outputs

29. How Hypothesis Space Affects Underfitting / Overfitting

- **Hypothesis space:** Set of all candidate models a learning algorithm can choose from
- **Too small hypothesis space:** Model cannot capture patterns → **underfitting**
- **Too large hypothesis space:** Model may memorize training data → **overfitting**

30. How Learning Algorithm Selects the Best Hypothesis

- Learning algorithm uses **training data** and **loss function** to evaluate hypotheses
- Selects the hypothesis that **minimizes training loss** and ideally **generalizes well** to validation/test data

- Example: Gradient descent iteratively updates parameters to minimize loss → finds best hypothesis in space

31. Does MSE(Mean Squared Error) always increases when we add a new feature?

- **Training MSE**:

  No, adding a new feature will **never increase training MSE** (in linear regression with least squares).

  Why? Because the model can always **ignore** the new feature (set its coefficient to 0). This means training MSE either **decreases or stays the same**.

- **Test/Validation MSE**:

  Not necessarily. On unseen data, adding irrelevant/noisy features often **increases MSE** because the model overfits the training data. So, test MSE can **increase** with extra features.

👉 Summary:

- Training MSE → **Never increases** with more features.
- Test/Validation MSE → **Can increase** if features cause overfitting.

32. Does choosing the value of k using hyperparameter tuning guarantee no overfitting?

- If by **k** you mean something like:
  - **k in k-NN** → Small k = overfitting, large k = underfitting. Hyperparameter tuning (via cross-validation) helps pick the "best" k, but it doesn't *guarantee* no overfitting — it just balances bias vs variance.
  - **k in regularization (like Ridge with λ or Lasso with λ)** → Hyperparameter tuning also reduces overfitting risk but doesn't eliminate it completely.

👉 Cross-validation helps you **estimate** generalization error and pick k to minimize it, but:

- If data is small/noisy → still possible to overfit.
- If cross-validation is not representative → also possible to overfit.

Q: Consider a Natural Language processing (NLP) application in which a chatbot communicates to the queries in a text form. Assume that there are two documents as shown in table.1 and the chatbot has to find the document class of Document 3 either alpha or beta while executing the KNN algorithm. Find the cosine similarity between the two documents.

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|}$$

Table.1 Words and Document

Words	Document 1	Document 2	Document 3
Machine	1	1	1
Learning	1	0	1
is	1	1	0
an	1	1	0
interesting	1	1	0
not	0	1	1
course	0	1	1
Classification class	**Alpha**	**Beta**	?

Step 1: Represent documents as vectors

- Document 1: [1, 1, 1, 1, 1, 0, 0]
- Document 2: [1, 0, 1, 1, 1, 1, 1]

- Document 3: [1, 1, 0, 0, 0, 1, 1]

Step 2: Cosine similarity formula

cosine_similarity(A,B)=A·B/||A||·||B||

Where:

- A·B=∑iAiBi
- ||A||=∑iAi^2
- ||B|| = ∑iBi^2

Step 3: Compute similarity of Document 3 with Document 1

- Dot product: D3·D1 = 2
- Norms: ||D3||=5 , ||D1|| = sqrt(5)
- Cosine similarity: cos(D3,D1)=2/2·sqrt5≈0.447

Step 4: Compute similarity of Document 3 with Document 2

- Dot product: D3.D2 = 3
- Norms: ||D2|| = sqrt(6)
- Cosine similarity: cos(D3,D2)=3/2·sqrt6≈0.612

Step 5: Classify Document 3 using KNN

- Cosine similarity with Document 1 (Alpha) ≈ 0.447
- Cosine similarity with Document 2 (Beta) ≈ 0.612

Since 0.612 > 0.447, Document 3 is closer to Document 2.

Document 3 class = Beta

b) Use KNN algorithm with k = sqrt(9) to classify the signal. Use Euclidean distance, Manhattan distance and weight feature distances with weight 0.6. Compare the results obtained with each distance.

IoT Devices	Noise level	Signal strength	Class
1	34	50	Lower power device
2	45	46	Lower power device
3	43	34	High power device
4	32	28	Lower power device
5	34	45	??

Step 1: Set K for KNN

k=sqrt{9} = 3

We will classify Device 5 using its neighbors based on distances.

Step 2: Compute distances

Device 5 vector: [34,45]

Other devices:

- Device 1 → [34, 50]
- Device 2 → [45, 46]
- Device 3 → [43, 34]
- Device 4 → [32, 28]

(a) Euclidean Distance

Formula:

d(A,B)=sqrt((x1−x2)^2+(y1−y2)^2)

Calculations:

- **Device 1:**

(34−34)^2+(45−50)^2=0+25=5

- **Device 2:**

(34−45)^2+(45−46)^2=121+1=122≈11.05

- **Device 3:**

(34−43)^2+(45−34)^2=81+121=202≈14.21

- **Device 4:**

(34−32)^2+(45−28)^2=4+289=293≈17.12

Nearest 3 neighbors (Euclidean): Device 1 (5), Device 2 (11.05), Device 3 (14.21)

Classes: Lower, Lower, High

Majority → Lower power device

(b) Manhattan Distance:

d(A,B)= ∣ x1−x2 ∣ + ∣ y1−y2 ∣

1. Device 1: |34-34| + |45-50| = 0 + 5 = 5
2. Device 2: |34-45| + |45-46| = 11 + 1 = 12
3. Device 3: |34-43| + |45-34| = 9 + 11 = 20
4. Device 4: |34-32| + |45-28| = 2 + 17 = 19

Nearest 3 neighbors (Manhattan): Device 1 (5), Device 2 (12), Device 4 (19)

- Classes: Lower, Lower, Lower
- Majority → **Lower power device**

(c) Weighted Euclidean Distance (weight = 0.6 for noise level, 0.4 for signal strength)

dw(A,B)=sqrt(0.6∗(x1−x2)^2+0.4∗(y1−y2)^2)

1. Device 1: ≈3.16
2. Device 2: ≈8.54
3. Device 3: ≈9.85
4. Device 4: ≈10.86

Nearest 3 neighbors (Weighted): Device 1 (3.16), Device 2 (8.54), Device 3 (9.85)

- Classes: Lower, Lower, High
- Majority → **Lower power device**

Interpretation:

- All distance metrics predict the same class for Device 5 → **Lower power device**.
- Weighting features slightly changes the distance values but does not change the majority vote in this case.

Exam Question

You are given a CSV dataset `employee_data.csv` that contains the following columns:

- **EmployeeID** (Unique identifier for each employee)
- **Name** (Categorical, string type)
- **Age** (Numeric)
- **Salary** (Numeric)
- **Department** (Categorical, e.g., "HR", "IT", "Finance")
- **Married** (Boolean type with values `True` / `False`)

The dataset may contain **missing values** in some columns.

Answer the following questions using Python (with **pandas**, **matplotlib**, and **sklearn** where needed).

(a) Importing the Dataset

Write Python code to import the dataset into a DataFrame using pandas. Display the first 5 rows.

(b) Null Values

1. Write the command to find the total number of missing values in each column.
2. For numerical columns (like Age, Salary), replace missing values with the column mean.
3. For any column that has more than **30% missing values**, drop that column completely from the dataset.

(c) Column Renaming

Rename the column **"Salary"** to **"Employee_Salary"** in the DataFrame.

(d) Encoding Categorical and Boolean Columns

1. Apply **Label Encoding** to the column **"Married"** (True → 1, False → 0).
2. Apply **One-Hot Encoding** to the column **"Department"** (e.g., HR, IT, Finance).

(e) Visualization

Draw a **scatter plot** between **Age** and **Employee_Salary** using matplotlib, where the points are colored based on the Department.

```
# --------------------------
# (a) Importing the dataset
# --------------------------
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv("employee_data.csv")

# Display first 5 rows
print("First 5 rows of dataset:")
print(df.head())

# --------------------------
# (b) Handling Missing Values
# --------------------------

# 1. Find total number of missing values in each column
print("\nMissing values per column:")
print(df.isnull().sum())

# 2. Fill missing values in numerical columns with mean
```

```
for col in df.columns:
    if df[col].dtype in ['int64', 'float64']:  # numerical columns
        df[col].fillna(df[col].mean(), inplace=True)

# 3. Drop columns having more than 30% missing values
threshold = 0.3 * len(df)
df = df.dropna(axis=1, thresh=threshold)

print("\nAfter handling missing values:")
print(df.head())

# --------------------------
# (c) Column Renaming
# --------------------------
df.rename(columns={"Salary": "Employee_Salary"}, inplace=True)

print("\nColumns after renaming:")
print(df.columns)

# --------------------------
# (d) Encoding Categorical/Boolean Columns
# --------------------------

# 1. Label Encoding for 'Married' column (True/False → 1/0)
if 'Married' in df.columns:
    le = LabelEncoder()
    df['Married'] = le.fit_transform(df['Married'])

# 2. One-Hot Encoding for 'Department' column
if 'Department' in df.columns:
    df = pd.get_dummies(df, columns=['Department'], drop_first=True)

print("\nData after encoding:")
print(df.head())

# --------------------------
```

```
# (e) Scatter Plot
# --------------------------

# Scatter plot between Age and Employee_Salary
plt.figure(figsize=(8, 6))
if 'Employee_Salary' in df.columns and 'Age' in df.columns:
    plt.scatter(df['Age'], df['Employee_Salary'], c='blue', alpha=0.6)

plt.title("Scatter Plot: Age vs Employee_Salary")
plt.xlabel("Age")
plt.ylabel("Employee_Salary")
plt.grid(True)
plt.show()
```