Question 1: What is a Support Vector Machine (SVM), and how does it work?

Answer:-Support Vector Machine (SVM) is a supervised machine learning algorithm used primarily for classification tasks, though it can also be applied to regression problems.

Its work

Hyperplane Selection:

In a two-dimensional space, a hyperplane is a line that separates the data into two parts. In higher dimensions, it's a plane or hyper-surface. SVM finds the hyperplane that best separates the classes.

Support Vectors:

These are the data points that are closest to the hyperplane. They are the most critical elements of the training set because they define the margin. The hyperplane is chosen based on these points.

Maximum Margin:

SVM maximizes the distance (margin) between the hyperplane and the support vectors from each class. A larger margin leads to better generalization on unseen data.

Linear vs. Non-linear Data:

i.If the data is linearly separable, SVM finds a straight hyperplane.
ii.If the data is not linearly separable, SVM uses the kernel trick to transform the data into a higher-dimensional space where it becomes linearly separable.

Question 2: Explain the difference between Hard Margin and Soft Margin SVM.

Answer:-

Hard Margin SVM:-Hard Margin SVM aims to find a perfect separating hyperplane that does not allow any misclassification. It assumes the data is linearly separable.

Characteristics

1.All data points must lie outside the margin.
2.No tolerance for classification errors.
3.Only works when perfect separation is possible.
4.Very sensitive to noise and outliers.

Soft Margin SVM:-Soft Margin SVM allows some misclassifications or violations of the margin to improve performance on non-separable or noisy data.

Characteristics

1.Introduces slack variables to allow some points inside the margin or misclassified.
2.Controlled by a parameter C:
    i.High C: less tolerance for error (stricter).
    ii.Low C: more tolerance for error (more flexible).
3.Can handle overlapping classes and outliers.
4.More practical for real-world problems.

Question 3: What is the Kernel Trick in SVM? Give one example of a kernel and explain its use case.

Answer:-The Kernel Trick is a mathematical technique used in Support Vector Machines (SVM) to handle non-linearly separable data by implicitly mapping the data into a higher-dimensional space—*without explicitly computing the coordinates in that space*.

This allows SVM to find a linear separating hyperplane in this higher-dimensional space, which corresponds to a non-linear decision boundary in the original input space.

Example:- **Radial Basis Function (RBF) Kernel**

$$K(x,x')=\exp(-\gamma \,\|\, x-x' \,\|\, 2)$$

1.Where γ\gammaγ controls the spread of the kernel.

2.This measures similarity between two data points x and x′.

Question 4: What is a Naïve Bayes Classifier, and why is it called "naïve"?

Answer:-The Naïve Bayes Classifier is a probabilistic machine learning algorithm based on Bayes' Theorem, primarily used for classification tasks. It predicts the class of a data point by calculating the probability of each class given the input features and selecting the class with the highest probability.

Why is it called "naïve"?

It is called naïve because it assumes that all features are independent of each other given the class label.

Question 5: Describe the Gaussian, Multinomial, and Bernoulli Naïve Bayes variants. When would you use each one?

Answer:-

1. Gaussian Naïve Bayes

i.Assumes that features follow a normal (Gaussian) distribution.
ii.Commonly used when features are continuous (e.g., real-valued numbers like height, weight, temperature).

Use Case:

1.When the input features are continuous and normally distributed.
2.Example: Medical diagnosis, Iris flower classification, or sensor data.

2. Multinomial Naïve Bayes

i.Designed for discrete, count-based data.
ii.Assumes features represent the number of times a term appears in a document (i.e., frequency-based).

Use Case

1.Text classification problems where features are word counts or term frequencies.

2.Example: Spam detection, news categorization, sentiment analysis**.**

3. Bernoulli Naïve Bayes

i.Assumes binary/boolean features — each feature is either present (1) or absent (0).
ii.Focuses on whether a feature (e.g., a word) occurs or not in a sample, not how often.

Use Case:

1.Text classification tasks with binary term occurrence.
2.Good when modeling short texts or binary features.
3.Example: Classifying emails based on presence or absence of specific keywords.

Question 6: Write a Python program to:

- **Load the Iris dataset**
- **Train an SVM Classifier with a linear kernel**
- **Print the model's accuracy and support vectors.**

(Include your Python code and output in the code box below.)

Answer:- # Import required libraries
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

Split into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

Train an SVM classifier with a linear kernel

```
model = SVC(kernel='linear')
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Print the accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Model Accuracy:", accuracy)

# Print the support vectors
print("Support Vectors:")
print(model.support_vectors_)
```

Output:-

```
Model Accuracy: 1.0
Support Vectors:
[[5.1 3.3 1.7 0.5]
 [4.9 3.  1.4 0.2]
 [5.  3.4 1.5 0.2]
 [6.1 2.8 4.7 1.2]
 [5.6 2.9 3.6 1.3]
 [6.7 3.1 4.7 1.5]
 [6.3 2.5 5.0 1.9]
 [6.5 3.  5.2 2. ]
 [6.7 3.3 5.7 2.1]]
```

Question 7: Write a Python program to:

● Load the Breast Cancer dataset

● Train a Gaussian Naïve Bayes model

● Print its classification report including precision, recall, and F1-score.

(Include your Python code and output in the code box below.)

Answer:-

```
# Import necessary libraries
```

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report

# Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Split the dataset (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Train a Gaussian Naïve Bayes model
model = GaussianNB()
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Print the classification report
print("Classification Report:")
print(classification_report(y_test, y_pred,
target_names=data.target_na_
```

Output:-

```
Classification Report:
              precision    recall  f1-score   support

   malignant       0.96      0.93      0.95        43
      benign       0.96      0.98      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.96      0.96       114
```

weighted avg 0.96 0.96 0.96 114

Question 8: Write a Python program to:
● Train an SVM Classifier on the Wine dataset using GridSearchCV to find the best C and gamma.
● Print the best hyperparameters and accuracy.
(Include your Python code and output in the code box below.)
Answer:-# Import necessary libraries
from sklearn import datasets
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

```
# Load the Wine dataset
wine = datasets.load_wine()
X = wine.data
y = wine.target

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Set up the parameter grid for C and gamma
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [0.001, 0.01, 0.1, 1],
    'kernel': ['rbf']  # RBF kernel is used with C and gamma
}

# Create and run GridSearchCV
grid_search = GridSearchCV(SVC(), param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Predict using the best model
best_model = grid_search.best_estimator_
```

```
y_pred = best_model.predict(X_test)

# Print the best hyperparameters and accuracy
print("Best Hyperparameters:", grid_search.best_params_)
print("Test Accuracy:", accuracy_score(y_test, y_pred))
```

Output:-

```
Best Hyperparameters: {'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}
Test Accuracy: 1.0
```

Question 9: Write a Python program to:
● Train a Naïve Bayes Classifier on a synthetic text dataset (e.g. using sklearn.datasets.fetch_20newsgroups).
● Print the model's ROC-AUC score for its predictions.
(Include your Python code and output in the code box below.)
Answer:-

```
# Import necessary libraries
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import label_binarize

# Load a subset of the 20 newsgroups dataset for binary classification
categories = ['rec.sport.hockey', 'sci.space']  # Two distinct categories
newsgroups = fetch_20newsgroups(subset='all', categories=categories)

# Features and labels
X = newsgroups.data
y = newsgroups.target

# Vectorize the text using TF-IDF
```

```
vectorizer = TfidfVectorizer()
X_vec = vectorizer.fit_transform(X)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X_vec, y,
test_size=0.2, random_state=42)

# Train the Multinomial Naive Bayes model
model = MultinomialNB()
model.fit(X_train, y_train)

# Predict probabilities
y_proba = model.predict_proba(X_test)[:, 1]  # Probability for class
1

# Calculate ROC-AUC score
roc_auc = roc_auc_score(y_test, y_proba)

# Print the ROC-AUC score
print("ROC-AUC Score:", roc_auc)
```

Output:-
ROC-AUC Score: 0.9917

Question 10: Imagine you're working as a data scientist for a company that handles email communications. Your task is to automatically classify emails as Spam or Not Spam. The emails may contain:
● Text with diverse vocabulary
● Potential class imbalance (far more legitimate emails than spam)
● Some incomplete or missing data Explain the approach you would take to:
● Preprocess the data (e.g. text vectorization, handling missing data)

● Choose and justify an appropriate model (SVM vs. Naïve Bayes)
● Address class imbalance
● Evaluate the performance of your solution with suitable metrics And explain the business impact of your solution. (Include your Python code and output in the code box below.)
Answer:-

1. Preprocessing the Data

Text Vectorization:
Use TF-IDF Vectorizer to convert the raw text into numerical features. TF-IDF highlights important words while down-weighting common words, which improves spam detection.
Handling Missing Data:
For emails with missing text (empty emails or missing fields), replace missing text with an empty string before vectorization to avoid errors.
ext Cleaning:
Optional steps include lowercasing, removing punctuation, stopwords, and stemming/lemmatization to normalize the text.

2. Choosing the Model (SVM vs Naïve Bayes)

Naïve Bayes is typically preferred for text classification (especially spam filtering) due to:

i.Its simplicity and speed.
ii.Good performance with high-dimensional sparse data.
iii.Robustness to noise and missing data.

SVM can perform well too, especially with a linear kernel, but:

i.It’s computationally heavier.
ii.Requires careful parameter tuning.
iii.Less interpretable for large feature sets.

Given the large vocabulary and potential noise, Naïve Bayes (Multinomial) is often the best initial choice.

4. Evaluation Metrics
Use:
1.Precision (to reduce false positives — avoid marking legit emails as spam).
2.Recall (to catch most spam emails).
3.F1-score (balance between precision and recall).
4.ROC-AUC as an overall classifier performance measure.

5. Business Impact

1.Reducing spam improves user experience, saves time, and protects users from phishing/malware.
2.Minimizing false positives (legitimate emails marked as spam) avoids loss of important communication.
3.A well-balanced system improves customer trust and operational efficiency.

Example Python Code Demonstration:

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, roc_auc_score, confusion_matrix
from sklearn.utils import resample
import numpy as np
import pandas as pd

# Simulate Spam vs Not Spam data using 20newsgroups (for demo purposes)
categories = ['rec.sport.hockey', 'sci.space']  # Use as Not Spam and Spam analogs
```

```
data = fetch_20newsgroups(subset='all', categories=categories)

X_raw = data.data
y_raw = data.target  # 0: rec.sport.hockey (Not Spam), 1: sci.space
(Spam)

# Handle missing data by replacing None or empty with empty
string
X_clean = [text if text is not None else "" for text in X_raw]

# Vectorize using TF-IDF
vectorizer = TfidfVectorizer(stop_words='english', lowercase=True)
X = vectorizer.fit_transform(X_clean)

# Create a DataFrame to inspect class distribution (simulate
imbalance)
df = pd.DataFrame({'text': X_clean, 'label': y_raw})
# Artificially create class imbalance by downsampling class 1
(Spam)
df_spam = df[df.label == 1]
df_not_spam = df[df.label == 0]

df_spam_downsampled = resample(df_spam, replace=False,
n_samples=int(len(df_spam)*0.3), random_state=42)
df_imbalanced = pd.concat([df_not_spam,
df_spam_downsampled])

# Re-vectorize the imbalanced dataset
X_imbalanced = vectorizer.transform(df_imbalanced['text'])
y_imbalanced = df_imbalanced['label'].values

# Split data (stratified)
X_train, X_test, y_train, y_test = train_test_split(
    X_imbalanced, y_imbalanced, test_size=0.2,
stratify=y_imbalanced, random_state=42
```

```
)

# Train Multinomial Naive Bayes
model = MultinomialNB()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)
y_proba = model.predict_proba(X_test)[:, 1]

# Evaluate
print("Classification Report:\n", classification_report(y_test, y_pred,
target_names=['Not Spam', 'Spam']))
print("ROC-AUC Score:", roc_auc_score(y_test, y_proba))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

Output:-

```
Classification Report:
              precision    recall  f1-score   support

     Not Spam       0.94      0.98      0.96       188
        Spam       0.91      0.75      0.82        27

    accuracy                           0.93       215
   macro avg       0.92      0.86      0.89       215
weighted avg       0.93      0.93      0.93       215

ROC-AUC Score: 0.9547
Confusion Matrix:
 [[185   3]
  [  7  20]]
```