Assignment Code: DA-AG-013

# SVM & Naive Bayes | Assignment

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

Ans:- A **Support Vector Machine (SVM)** is a powerful **supervised machine learning algorithm** used for **classification** and **regression** tasks, but it's most commonly used for **binary classification**.

---

## 🧠 Basic Idea of SVM

SVM aims to **find the best boundary (hyperplane)** that separates data points of different classes with the **maximum margin** (i.e., the largest distance between the boundary and the nearest points from both classes). These nearest points are called **support vectors**.

---

## 📌 How SVM Works — Step-by-Step

### 1. Linearly Separable Case

If the data can be separated by a straight line (in 2D) or a plane/hyperplane (in higher dimensions):

- SVM finds the **optimal hyperplane** that separates the two classes.
- The hyperplane is chosen such that it has the **maximum margin**.
- The points that lie **closest** to the hyperplane are the **support vectors**, and they define the margin.

**Mathematical objective:**
Maximize margin = 2∣∣w∣∣\frac{2}{||w||}∣∣w∣∣2,
where www is the weight vector of the hyperplane.

### 2. Non-Linearly Separable Case

If the data **is not linearly separable**:

- SVM uses a technique called the **kernel trick** to **map data to a higher-dimensional space** where it **becomes linearly separable**.
- Common **kernel functions**:
    - **Linear**: K(x,y)=x⋅yK(x, y) = x \cdot yK(x,y)=x⋅y
    - **Polynomial**: K(x,y)=(x⋅y+c)dK(x, y) = (x \cdot y + c)^dK(x,y)=(x⋅y+c)d
    - **RBF (Gaussian)**: K(x,y)=exp(−γ∥x−y∥2)K(x, y) = \exp(-\gamma ||x - y||^2)K(x,y)=exp(−γ∣∣x−y∣∣2)

**3. Soft Margin (for noisy data)**

- Real-world data often has **overlapping classes** or noise.
- SVM introduces a **penalty parameter** `C` to allow some misclassifications while still aiming for a wide margin.
    - **High C** → less tolerance for misclassification (can overfit).
    - **Low C** → more tolerance (can underfit).

---

## 📊 Example

Imagine you want to classify emails as "spam" or "not spam". SVM will:

- Convert each email into a vector of features (word counts, etc.).
- Find the optimal boundary that separates spam from not spam emails with the highest margin.
- Use support vectors (critical examples) to define this boundary.

---

## ✅ Advantages of SVM

- Works well in **high-dimensional spaces**.

- Effective when the number of features > number of samples.
- **Robust** to overfitting, especially with the right kernel and regularization.

### ❌ Limitations

- Not suitable for **large datasets** (training time is slow).
- Less effective on **overlapping** classes.
- Choosing the **right kernel** and tuning hyperparameters can be complex.

---

### 🔧 Real-World Applications

- **Text classification** (e.g., spam detection, sentiment analysis)
- **Image classification**
- **Bioinformatics** (e.g., protein classification)
- **Face detection** in images

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

Ans:- The **difference between Hard Margin and Soft Margin SVM** lies in how strictly the algorithm separates the classes and handles **misclassification or overlapping data**.

---

## 🔴 1. Hard Margin SVM

### 💡 Definition:

Hard Margin SVM tries to **find a hyperplane that separates the data with **no misclassifications**.
It assumes the data is **perfectly linearly separable**.

### 📌 Characteristics:

- No data points are allowed **inside the margin** or on the wrong side of the hyperplane.
- Maximizes the **margin** while keeping **zero training error**.
- Works only when the data is **clean and noise-free**.

### ⚠️ Limitations:

- **Very sensitive to outliers** or noise.
- Fails when even a **single point** is misclassified or slightly overlaps.

### 🔎 Use Case:

Rarely used in real-world applications because most real datasets are noisy or not perfectly separable.

---

## 🟡 2. Soft Margin SVM

### 💡 Definition:

Soft Margin SVM **allows some misclassifications** or violations of the margin to achieve better generalization on noisy or overlapping data.

### 📌 Characteristics:

- Introduces **slack variables ($\xi_i$)** to allow some points to lie **inside the margin** or be **misclassified**.
- Controlled by the **regularization parameter `C`**:
    - **High `C`** → tries to classify all training examples correctly (less tolerance to misclassification, risk of overfitting).
    - **Low `C`** → allows more misclassification for better generalization (risk of underfitting).

### ✅ Benefits:

- **Robust to outliers** and noisy data.
- Balances the **trade-off between margin size and classification error**.

### 🔎 Use Case:

Widely used in **real-world problems** like spam detection, image classification, medical diagnosis, etc.

---

## 🆚 Summary Table:

| Feature | Hard Margin SVM | Soft Margin SVM |
|---|---|---|
| Tolerance to error | ❌ No tolerance (strict separation) | ✅ Allows some error/misclassification |
| Suitable for | Perfectly separable data | Real-world noisy or overlapping data |
| Flexibility | Rigid | Flexible |
| Use of slack variables | ❌ No | ✅ Yes ($\xi_i$) |
| Regularization (C) | Not applicable | ✅ Controls trade-off between margin and error |
| Sensitive to outliers | ✅ Highly sensitive | ❌ Less sensitive |

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

Ans:- 🧠 **What is the Kernel Trick in SVM?**

The **Kernel Trick** is a technique used in **Support Vector Machines (SVM)** to enable them to perform **non-linear classification**.

🔍 **Problem:**

- In many real-world datasets, the data **cannot be separated linearly** (i.e., by a straight line or hyperplane).
- Simply using a linear SVM won't work well in such cases.

💡 **Solution: Kernel Trick**

The **Kernel Trick** allows SVM to:

- **Map the original data** into a **higher-dimensional feature space**, where a linear separation **is possible**.
- Do this **implicitly**, without actually computing the transformation — by using a **kernel function**.

> ✅ So, instead of computing φ(x) explicitly, we compute **K(x, x′) = φ(x)·φ(x′)** directly, which is much faster and more efficient.

---

⚙️ **How it Works:**

- Suppose x and x′ are input vectors.
- A kernel function **K(x, x′)** returns the **dot product of the two vectors in a higher-dimensional space**:
  $K(x,x') = \phi(x) \cdot \phi(x')$

---

# 🔵 Common Kernel Example: Radial Basis Function (RBF) Kernel

### Formula:

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

- γ\gammaγ controls how far the influence of a single training example reaches.
- A **higher γ** means only nearby points affect the decision boundary.

### ✅ Use Case:

- Used when the **decision boundary is very complex and curved**.
- Suitable for **non-linear problems** like:
    - Handwritten digit classification (like MNIST)
    - Face recognition
    - Bioinformatics (e.g., classifying proteins, genes)

---

### 🔁 Intuition Behind RBF Kernel:

Imagine you're trying to classify points shaped like two circles:

- Inner circle = Class A
- Outer ring = Class B
  → These are **not linearly separable in 2D**, but they **are** separable in a higher-dimensional space with the RBF kernel.

---

## ✅ Summary of the Kernel Trick

| Aspect | Description |
|---|---|
| Purpose | Enables SVM to handle **non-linear data** |
| How it works | Computes dot products in **high-dimensional space** without actual mapping |
| Key advantage | **Transforms non-linear problems into linear ones** in transformed space |
| Common kernels | Linear, Polynomial, RBF (Gaussian), Sigmoid |
| Example kernel | **RBF kernel** — works well on complex, curved decision boundaries |

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

Ans:- 🧠 **What is a Naïve Bayes Classifier?**

A **Naïve Bayes Classifier** is a **supervised machine learning algorithm** based on **Bayes’ Theorem**, used for **classification tasks**. It’s especially popular for text data (like spam detection or sentiment analysis).

---

## 📌 Bayes’ Theorem Recap:

$P(C|X)=P(X|C)\cdot P(C)P(X)P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}P(C|X)=P(X)P(X|C)\cdot P(C)$

Where:

- $P(C|X)P(C|X)P(C|X)$ = Probability of class **C** given features **X** (posterior)
- $P(X|C)P(X|C)P(X|C)$ = Likelihood of features **X** given class **C**
- $P(C)P(C)P(C)$ = Prior probability of class **C**
- $P(X)P(X)P(X)$ = Probability of features **X**

The classifier predicts the class **C** that **maximizes** this posterior probability.

---

### 🧾 **Example:**

Say you want to classify an email as **spam** or **not spam** based on the words it contains:

- XXX = [‘win’, ‘money’, ‘now’]
- The model calculates the probability of each class given the words, and picks the one with the **highest probability**.

---

## 🧐 Why is it called "Naïve"?

Because it **naively assumes** that:

> All features (words, attributes) are **independent of each other**, given the class.

In reality, this assumption is **almost never true** (e.g., in a sentence, words are related). But this **simplifies the computation** and still works surprisingly well in many applications.

---

### ✅ Example of the "Naïve" Assumption:

In text classification:

- Suppose you're classifying messages.
- You have features like:
    - Word 1: "buy"
    - Word 2: "now"

The **naïve assumption** says:

> The presence of "buy" is independent of the presence of "now", **given** the class label (e.g., spam).

Even though in real life, these words often appear **together** in spam messages!

---

## ✅ Advantages:

- **Simple and fast**
- **Works well with high-dimensional data** (like text)
- **Performs surprisingly well** even with the naive assumption
- Requires **less training data**

## ❌ Limitations:

- Assumes **feature independence** (which is rarely true)
- Not great when features are **highly correlated**
- Poor performance on **very small datasets**

---

## 🔧 Real-World Applications:

- **Spam detection**
- **Sentiment analysis**
- **Document categorization**
- **Medical diagnosis**

---

## 🔚 Summary:

| Feature | Description |
|---|---|
| Based on | **Bayes' Theorem** |
| Assumption | Features are **conditionally independent** given the class |
| Why "naïve"? | Assumes independence between features (often unrealistic) |
| Best use cases | **Text classification**, spam filtering, sentiment analysis |

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

Ans:- ## ✅ 1. Gaussian Naïve Bayes

### 📌 Description:

Assumes that the **features follow a Normal (Gaussian) distribution**.

### 📊 Suitable for:

- **Continuous (numerical) features**
- e.g., age, height, temperature, etc.

### 🔍 Formula:

$P(x_i|C) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i - \mu)^2}{2\sigma^2}\right)$

Each feature's likelihood is computed assuming it's Gaussian distributed.

### 💡 Example Use Case:

- **Medical diagnosis** (e.g., classify if a patient has a disease based on continuous lab results)
- **Iris flower classification** (based on petal/sepal width/length)

---

## ✅ 2. Multinomial Naïve Bayes

### 📌 Description:

Assumes the features represent **discrete counts** (i.e., number of times a feature appears).

### 📊 Suitable for:

- **Count-based data**, especially **text data** where features are word frequencies or term frequency-inverse document frequency (**TF-IDF**).
- Doesn’t work well with negative or continuous features.

### 💡 Example Use Case:

- **Spam detection** (based on word counts in emails)
- **News classification**

- **Sentiment analysis**

### ⚠️ Note:

This is the most common Naïve Bayes variant used in **Natural Language Processing (NLP)**.

---

## ✅ 3. Bernoulli Naïve Bayes

### 📌 Description:

Assumes that each feature is a **binary value** (0 or 1), representing the **presence or absence** of a feature (not frequency).

### 📊 Suitable for:

- **Binary/Boolean features**
- Works well when you're only interested in whether a word occurs or not (not how often)

### 💡 Example Use Case:

- **Text classification** where we only care if a word appears (e.g., spam vs. not spam)
- **Document classification** with binary features (e.g., has the word “money”? yes/no)

---

## 🔚 Summary Table

| Variant | Data Type | Assumption | Example Use Case |
|---|---|---|---|
| **Gaussian** | Continuous | Features follow **Normal distribution** | Medical data, Iris dataset |
| **Multinomial** | Count (integers) | Features are **counts/frequencies** | Spam detection, news classification |

| | | | |
|---|---|---|---|
| **Bernoulli** | Binary (0/1) | Features are **present/absent** | Text classification (presence-only) |

---

## 🤔 Which One to Use?

- **Use Gaussian** → for numerical features (e.g., heights, test scores)
- **Use Multinomial** → for word counts or TF-IDF in NLP tasks
- **Use Bernoulli** → for binary text features or presence/absence scenarios

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.

Ans:- Here's a complete Python program to:

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

```
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 the dataset 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 the SVM Classifier with a linear kernel
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)

# Predict on the test set
```

```
y_pred = svm_model.predict(X_test)

# Calculate and print the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")

# Print the support vectors
print("\nSupport Vectors:")
print(svm_model.support_vectors_)
```

✅ Output Example (will vary slightly each run):

```
Model Accuracy: 1.00

Support Vectors:
[[5.1 3.8 1.5 0.3]
 [4.6 3.2 1.4 0.2]
 ...
]
```

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.

Ans:- Here's a complete Python program to:

- Load the **Breast Cancer** dataset
- Train a **Gaussian Naïve Bayes** model
- Print the **classification report** (precision, recall, F1-score)

✅ Python Code:

```
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 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 the Gaussian Naive Bayes model
gnb = GaussianNB()
gnb.fit(X_train, y_train)

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

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

### 📊 Output Example (may vary slightly):

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

   malignant       0.96      0.94      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.

Ans:- Here's a complete Python program to:

- Load the **Wine** dataset
- Train an **SVM Classifier** using **GridSearchCV**
- Find the best `C` and `gamma` values
- Print the **best hyperparameters** and **accuracy**

---

## ✅ Python Code:

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

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

# Split into train and test 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)

# Define the parameter grid
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [0.001, 0.01, 0.1, 1],
    'kernel': ['rbf']
}

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

# Best parameters and accuracy
best_model = grid.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

# Output results
print("Best Hyperparameters:")
print(grid.best_params_)
print(f"\nTest Accuracy: {accuracy:.2f}")
```

🧪 Output Example (varies by run):

```
Best Hyperparameters:
{'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}

Test Accuracy: 1.00
```

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.

Ans:- Here's a Python program that:

- Loads a synthetic **text dataset** using `fetch_20newsgroups` from `sklearn.datasets`
- Trains a **Naïve Bayes Classifier** (Multinomial Naive Bayes is best suited for text)
- Computes and prints the **ROC-AUC score**

We'll use **binary classification** (e.g., 'sci.space' vs. 'rec.autos') for ROC-AUC to make sense.

---

### ✅ Python Code:

```
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

# Load a binary subset of the 20 newsgroups dataset
categories = ['sci.space', 'rec.autos']
newsgroups = fetch_20newsgroups(subset='all', categories=categories)

# Features and labels
X = newsgroups.data
y = newsgroups.target  # 0 or 1

# Vectorize the text using TF-IDF
vectorizer = TfidfVectorizer()
X_vectorized = vectorizer.fit_transform(X)

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_vectorized, y, test_size=0.2,
random_state=42)

# Train the Naive Bayes model
```

```
model = MultinomialNB()
model.fit(X_train, y_train)

# Predict probabilities
y_probs = model.predict_proba(X_test)[:, 1]

# Compute ROC-AUC score
roc_auc = roc_auc_score(y_test, y_probs)
print(f"ROC-AUC Score: {roc_auc:.2f}")
```

🧪 Example Output:
ROC-AUC Score: 0.98

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.

Ans:- Here's a detailed and practical approach to building a **Spam Email Classifier** given the challenges you mentioned (text data, class imbalance, and missing values):

---

# 🔧 Step-by-Step Approach

## 1. Preprocessing the Data

✅ **Handle Missing Data:**

- **Text Columns:**
    - If emails are missing (`NaN` in text), drop or fill with empty string: `df['email_text'].fillna('')`
- **Other Features (e.g., metadata):**
    - For categorical: fill with `'unknown'`

- For numerical: fill with mean/median or use KNN imputation if necessary

✅ **Text Vectorization:**

Since emails are unstructured text:

- Use **TF-IDF Vectorization**: Converts text to numerical values based on term frequency and importance.

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X = vectorizer.fit_transform(df['email_text'])
```


```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
X = vectorizer.fit_transform(df['email_text'])
```


```
from sklearn.metrics import classification_report, roc_auc_score, confusion_matrix
```

# 💼 Business Impact

## ✅ Benefits of this solution:

- **Increased Productivity**: Employees won't waste time on spam emails.
- **Security**: Blocks phishing or malicious emails that could harm the organization.
- **Customer Trust**: Ensures legitimate customer communications aren't lost or mislabeled.
- **Automation**: Reduces manual effort in monitoring inboxes and spam folders.

---

# 🔁 Summary

| Task | Solution |
| --- | --- |

| | |
|---|---|
| Preprocessing | TF-IDF + Fill missing text with `''` |
| Model | **Multinomial Naïve Bayes** or SVM with class weights |
| Imbalance | SMOTE, class weights, threshold tuning |
| Metrics | Precision, Recall, F1, ROC-AUC |
| Business Value | Improves security, efficiency, and automation |