Name: Shikhar Varshney	***Roll No : 2315002054***
Class Roll No : 65	***Sec : A (2025-26)***

Part 1

Ans 1)

1) **Core Assumption of Naive Bayes** Naive Bayes assumes that all features are conditionally independent given the class label. This means the presence or absence of one feature does not affect the others, simplifying the computation of probabilities.
2) **Difference between GaussianNB, MultinomialNB, and BernoulliNB**

- **GaussianNB**: Used for continuous data; assumes features follow a normal distribution.
- **MultinomialNB**: Suitable for discrete count data like word frequencies in text classification.
- **BernoulliNB**: Works with binary/boolean features, such as word presence (0 or 1) in documents.

3) **Suitability for High-Dimensional Data** Naive Bayes performs well in high-dimensional spaces because it treats each feature independently, reducing the complexity of the model. It also requires fewer parameters, making it efficient and less prone to overfitting.

Ans 2)

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix

df = pd.read_csv("https://raw.githubusercontent.com/justmarkham/pycon-2016-tutorial/master/data/sms.tsv", sep='\t', names=['label', 'message'])
```

```
df['label'] = df['label'].map({'ham': 0, 'spam': 1})
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df['message'])
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = MultinomialNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

Ans 3)

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
gnb = GaussianNB()
gnb.fit(X_train, y_train)
gnb_pred = gnb.predict(X_test)
```

```
gnb_acc = accuracy_score(y_test, gnb_pred)

lr = LogisticRegression(max_iter=200)

lr.fit(X_train, y_train)

lr_pred = lr.predict(X_test)

lr_acc = accuracy_score(y_test, lr_pred)

dt = DecisionTreeClassifier()

dt.fit(X_train, y_train)

dt_pred = dt.predict(X_test)

dt_acc = accuracy_score(y_test, dt_pred)

print("GaussianNB Accuracy:", gnb_acc)

print("Logistic Regression Accuracy:", lr_acc)

print("Decision Tree Accuracy:", dt_acc);
```

Part 2:

Ans 4)

1. **Entropy and Information Gain** Entropy measures the impurity or randomness in a dataset. Information Gain quantifies the reduction in entropy after a dataset is split on a feature—higher gain means better feature for splitting.
2. **Gini Index vs Entropy** Both are metrics for measuring node impurity. Gini Index is faster to compute and favors binary splits, while Entropy is based on logarithmic probabilities and can be more sensitive to class imbalance.
3. **Overfitting in Decision Trees** A decision tree overfits when it becomes too complex and captures noise in the training data. This can be avoided by limiting tree depth, setting minimum samples per split, or using pruning techniques.

Ans 5)

```
import pandas as pd

from sklearn.model_selection import train_test_split
```

```
from sklearn.tree import DecisionTreeClassifier, plot_tree

from sklearn.metrics import accuracy_score, confusion_matrix

import matplotlib.pyplot as plt

df =
pd.read_csv("https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv")

df['Age'].fillna(df['Age'].median(), inplace=True)

df['Embarked'].fillna('S', inplace=True)

df.drop(['Cabin', 'Name', 'Ticket'], axis=1, inplace=True)

df = pd.get_dummies(df, columns=['Sex', 'Embarked'], drop_first=True)

X = df.drop('Survived', axis=1)

y = df['Survived']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

tree = DecisionTreeClassifier(random_state=42)

tree.fit(X_train, y_train)

plt.figure(figsize=(12, 6))

plot_tree(tree, feature_names=X.columns, class_names=['Not Survived', 'Survived'],
filled=True)

plt.show()

y_pred = tree.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

Ans 6)

```
import numpy as np

depths = range(1, 21)

train_acc = []
```

```
test_acc = []
for d in depths:
    model = DecisionTreeClassifier(max_depth=d, min_samples_split=10, random_state=42)
    model.fit(X_train, y_train)
    train_acc.append(model.score(X_train, y_train))
    test_acc.append(model.score(X_test, y_test))
plt.plot(depths, train_acc, label='Train Accuracy', marker='o')
plt.plot(depths, test_acc, label='Test Accuracy', marker='x')
plt.xlabel('Max Depth')
plt.ylabel('Accuracy')
plt.title('Overfitting Visualization')
plt.legend()
plt.grid(True)
plt.show()
```

Part 3

Ans 7)

1. Difference Between Bagging and Boosting

- **Bagging (Bootstrap Aggregating)**:
    - Trains multiple models independently on random subsets of the data (with replacement).
    - Aims to reduce **variance**.
    - Example: Random Forest.
- **Boosting**:

- Trains models sequentially, where each model tries to correct the errors of the previous one.
- Aims to reduce **bias**.
- Example: AdaBoost, Gradient Boosting.

2. How Random Forest Reduces Variance

- Combines predictions from multiple decision trees trained on bootstrapped samples.
- Introduces randomness in feature selection at each split.
- Aggregating results (majority vote or average) stabilizes predictions and reduces overfitting.

3. Weakness of Boosting-Based Methods

- Sensitive to **noisy data** and **outliers**.
- Can **overfit** if the number of iterations is too high.
- Computationally more expensive than bagging.

Ans 8)

```
import matplotlib.pyplot as plt
import seaborn as sns

importances = rf_model.feature_importances_
features = X.columns
sns.barplot(x=importances, y=features)
plt.title("Feature Importances - Random Forest")
plt.show()
```

Ans 9)

Model Training

- Train either an AdaBoostClassifier or a GradientBoostingClassifier using a suitable classification dataset (e.g., Titanic, Iris, or Wine).
- Ensure proper preprocessing: handle missing values, encode categorical features, and split into training/testing sets.

Performance Comparison

Compare the chosen boosting model with:

- DecisionTreeClassifier (from Task 5)
- RandomForestClassifier (from Task 8)

Metric	Description
Accuracy	Measures overall correctness of predictions
F1-score	Harmonic mean of precision and recall; useful for imbalanced datasets
Training Time *(optional)*	Time taken to train each model;

Expected Output

- A table or summary comparing all three models on the same dataset.
- Example format:

plaintext

Model Accuracy F1-score Training Time (s)

Decision Tree 0.78 0.76 0.05

Random Forest 0.83 0.81 0.20

Gradient Boosting 0.85 0.83 0.35

Optional Enhancements

- Visualize feature importances.
- Plot confusion matrices for deeper insight.