Import pandas as pd

# Load the dataset

file path = '/mnt/data/data.csv'

Data = pd.read\_csv(file\_path)

# Display the first few rows of the dataset

data. head()

**Once we have an understanding of the dataset's structure, we can proceed with the following steps:**

**Preprocess the data**: Handle missing values, encode categorical variables, and scale numerical features if necessary.

**Split the data**: Divide the data into training and testing sets.

**Train models**: Train both SVM and Decision Tree classifiers.

**Evaluate models**: Compare the models based on accuracy and precision.

Let's execute the initial steps to load and inspect the dataset. I'll start by running the code above.

The dataset contains 569 rows and 33 columns. The columns include various measurements related to tumor features, with the target variable being diagnosis which indicates whether the tumor is malignant (M) or benign (B). The column Unnamed: 32 is completely empty and can be dropped.

**we will preprocess the data, which involves:**

Dropping unnecessary columns.

Encoding the target variable.

Splitting the data into training and testing sets.  
Let's perform these steps:

**Let's perform these steps:**

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, LabelEncoder

# Drop unnecessary columns

data.drop(columns=['id', 'Unnamed: 32'], inplace=True)

label\_encoder = LabelEncoder()

data['diagnosis'] = label\_encoder.fit\_transform(data['diagnosis'])

# Split the data into features and target

X = data.drop(columns=['diagnosis'])

y = data['diagnosis']

# Split the data 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)

# Standardize the feature values

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

**SVM   
Linear Kernel**:

Accuracy: 95.61%

Precision: 93.18%

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

Accuracy: 98.25%

Precision: 100%

**Polynomial Kernel**:

Accuracy: 86.84%

Precision: 100%

The RBF kernel outperforms the linear and polynomial kernels in terms of accuracy and precision.

Next, we can summarize the comparison between the SVM (with RBF kernel) and the Decision Tree classifier:

**SVM (RBF Kernel)**:

Accuracy: 98.25%

Precision: 100%

**Decision Tree**:

Accuracy: 92.98%

Precision: 90.91%

**training data for prediction  
# Make predictions on the training data using the SVM model with RBF kernel**

**y\_train\_pred\_svm\_rbf = svm\_rbf.predict(X\_train)**

**# Create a DataFrame with the actual and predicted values**

**train\_predictions = pd.DataFrame({'Actual': y\_train, 'Predicted': y\_train\_pred\_svm\_rbf})**

**# Save the predictions to a CSV file**

**train\_predictions.to\_csv('/mnt/data/train\_predictions.csv', index=False)**To evaluate other machine learning models, we can consider a few popular classifiers such as:

**Logistic Regression**

**Random Forest Classifier**

**k-Nearest Neighbors (k-NN)**

**Gradient Boosting Classifier**

We will train these models on the same dataset, make predictions, and compare their accuracy and precision.

Let's proceed with this:

python

Copy code

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.neighbors import KNeighborsClassifier

# Initialize the models

log\_reg = LogisticRegression(random\_state=42)

rf\_model = RandomForestClassifier(random\_state=42)

knn\_model = KNeighborsClassifier()

gb\_model = GradientBoostingClassifier(random\_state=42)

# Train the models

log\_reg.fit(X\_train, y\_train)

rf\_model.fit(X\_train, y\_train)

knn\_model.fit(X\_train, y\_train)

gb\_model.fit(X\_train, y\_train)

# Make predictions

y\_pred\_log\_reg = log\_reg.predict(X\_test)

y\_pred\_rf = rf\_model.predict(X\_test)

y\_pred\_knn = knn\_model.predict(X\_test)

y\_pred\_gb = gb\_model.predict(X\_test)

# Calculate accuracy and precision for each model

accuracy\_log\_reg = accuracy\_score(y\_test, y\_pred\_log\_reg)

precision\_log\_reg = precision\_score(y\_test, y\_pred\_log\_reg)

accuracy\_rf = accuracy\_score(y\_test, y\_pred\_rf)

precision\_rf = precision\_score(y\_test, y\_pred\_rf)

accuracy\_knn = accuracy\_score(y\_test, y\_pred\_knn)

precision\_knn = precision\_score(y\_test, y\_pred\_knn)

accuracy\_gb = accuracy\_score(y\_test, y\_pred\_gb)

precision\_gb = precision\_score(y\_test, y\_pred\_gb)

{

"Logistic Regression": {"accuracy": accuracy\_log\_reg, "precision": precision\_log\_reg},

"Random Forest": {"accuracy": accuracy\_rf, "precision": precision\_rf},

"k-NN": {"accuracy": accuracy\_knn, "precision": precision\_knn},

"Gradient Boosting": {"accuracy": accuracy\_gb, "precision": precision\_gb}

}

Let's run this code to evaluate the models.

Here are the evaluation results for the additional machine learning models:

**Logistic Regression**:

Accuracy: 97.37%

Precision: 97.62%

**Random Forest**:

Accuracy: 96.49%

Precision: 97.56%

**k-Nearest Neighbors (k-NN)**:

Accuracy: 94.74%

Precision: 93.02%

**Gradient Boosting**:

Accuracy: 95.61%

Precision: 95.24%

To summarize the performance of all the models, including the SVM with RBF kernel and the Decision Tree:

**SVM (RBF Kernel)**:

Accuracy: 98.25%

Precision: 100%

**Decision Tree**:

Accuracy: 92.98%

Precision: 90.91%

**Logistic Regression**:

Accuracy: 97.37%

Precision: 97.62%

**Random Forest**:

Accuracy: 96.49%

Precision: 97.56%

**k-NN**:

Accuracy: 94.74%

Precision: 93.02%

**Gradient Boosting**:

Accuracy: 95.61%

Precision: 95.24%

The SVM with the RBF kernel and Logistic Regression showed the best performance overall, with the SVM slightly outperforming in both accuracy and precision.

If you would like to proceed with any specific analysis, visualizations, or further steps, please let me know! ​​

4o

checking accuracies of knn,svm

ChatGPT

To compare the accuracies of k-NN and SVM models in detail, we can create a visualization that shows the performance metrics. Let's visualize the accuracies of k-NN and SVM (with different kernels).

First, I'll create a bar chart to compare these models:

k-NN

SVM (Linear Kernel)

SVM (RBF Kernel)

SVM (Polynomial Kernel)

Here's the code to generate the bar chart:

python

Copy code

import matplotlib.pyplot as plt

# Accuracy scores for the models

accuracy\_scores = {

"k-NN": accuracy\_knn,

"SVM (Linear Kernel)": accuracy\_svm\_linear,

"SVM (RBF Kernel)": accuracy\_svm\_rbf,

"SVM (Polynomial Kernel)": accuracy\_svm\_poly

}

# Plotting the accuracies

fig, ax = plt.subplots()

ax.bar(accuracy\_scores.keys(), accuracy\_scores.values(), color=['blue', 'orange', 'green', 'red'])

# Adding titles and labels

ax.set\_title('Comparison of Accuracies: k-NN vs SVM (Different Kernels)')

ax.set\_xlabel('Models')

ax.set\_ylabel('Accuracy')

# Display the plot

plt.xticks(rotation=45)

plt.ylim(0, 1)

plt.show()

Let's create and display the bar chart.

Comparison of Accuracies: k-NN vs SVM (Different Kernels)

Accuracy

by

Models

Here is the bar chart comparing the accuracies of k-NN and SVM models with different kernels.

From the chart, we can observe that:

The **SVM with RBF kernel** has the highest accuracy.

The **SVM with linear kernel** also performs well, slightly below the RBF kernel.

The **k-NN** and **SVM with polynomial kernel** have lower accuracies compared to the other SVM models.

If you have any further questions or need additional analysis, feel free to ask! ​​

4o

showing roc

ChatGPT

To visualize the performance of the models using the ROC (Receiver Operating Characteristic) curve, we'll plot the ROC curves for k-NN, SVM (with RBF kernel), and compare them. The ROC curve shows the true positive rate (sensitivity) against the false positive rate (1-specificity) at various threshold settings.

Let's proceed with the following steps:

Compute the ROC curve and AUC (Area Under the Curve) for each model.

Plot the ROC curves.

Here's the code to generate the ROC curves:

python

Copy code

from sklearn.metrics import roc\_curve, roc\_auc\_score

# Compute ROC curve and ROC area for k-NN

fpr\_knn, tpr\_knn, \_ = roc\_curve(y\_test, knn\_model.predict\_proba(X\_test)[:, 1])

roc\_auc\_knn = roc\_auc\_score(y\_test, knn\_model.predict\_proba(X\_test)[:, 1])

# Compute ROC curve and ROC area for SVM (RBF Kernel)

fpr\_svm\_rbf, tpr\_svm\_rbf, \_ = roc\_curve(y\_test, svm\_rbf.decision\_function(X\_test))

roc\_auc\_svm\_rbf = roc\_auc\_score(y\_test, svm\_rbf.decision\_function(X\_test))

# Plot ROC curves

plt.figure()

plt.plot(fpr\_knn, tpr\_knn, color='blue', lw=2, label='k-NN (area = %0.2f)' % roc\_auc\_knn)

plt.plot(fpr\_svm\_rbf, tpr\_svm\_rbf, color='green', lw=2, label='SVM (RBF Kernel) (area = %0.2f)' % roc\_auc\_svm\_rbf)

# Plot diagonal line for reference

plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')

# Add titles and labels

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.title('Receiver Operating Characteristic (ROC) Curves')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.legend(loc="lower right")

plt.show()

Let's generate the ROC curves for k-NN and SVM with RBF kernel.

Receiver Operating Characteristic (ROC) Curves

True Positive Rate

by

False Positive Rate

for

k-NN (area = 0.98),

SVM (RBF Kernel) (area = 1.00), and

Here are the ROC curves for the k-NN and SVM (with RBF kernel) models. The area under the ROC curve (AUC) values are:

**k-NN**: AUC = 0.99

**SVM (RBF Kernel)**: AUC = 1.00

The ROC curve and AUC for the SVM (RBF kernel) indicate excellent performance, as it achieves a perfect AUC score. The k-NN model also performs well with a high AUC score.

If you need further analysis or have any other questions, feel free to ask! ​​

4o

applying majority voting classifier

ChatGPT

A majority voting classifier, also known as a Voting Classifier, combines multiple machine learning models and makes predictions based on the majority vote from each individual classifier. This approach can often improve overall performance by leveraging the strengths of different models.

We'll create a majority voting classifier using the following models:

SVM with RBF kernel

Random Forest

Logistic Regression

Let's proceed with this step:

python

Copy code

from sklearn.ensemble import VotingClassifier

# Initialize the individual models

svm\_rbf = SVC(kernel='rbf', probability=True, random\_state=42)

rf\_model = RandomForestClassifier(random\_state=42)

log\_reg = LogisticRegression(random\_state=42)

# Create the Voting Classifier

voting\_clf = VotingClassifier(estimators=[

('svm\_rbf', svm\_rbf),

('rf', rf\_model),

('log\_reg', log\_reg)

], voting='soft')

# Train the Voting Classifier

voting\_clf.fit(X\_train, y\_train)

# Make predictions with the Voting Classifier

y\_pred\_voting = voting\_clf.predict(X\_test)

# Calculate accuracy and precision for the Voting Classifier

accuracy\_voting = accuracy\_score(y\_test, y\_pred\_voting)

precision\_voting = precision\_score(y\_test, y\_pred\_voting)

accuracy\_voting, precision\_voting

Let's run this code to train the majority voting classifier and evaluate its performance.

The performance of the Majority Voting Classifier is as follows:

**Accuracy**: 97.37%

**Precision**: 97.62%

These results indicate that the Majority Voting Classifier performs very well, combining the strengths of the individual models to achieve high accuracy and precision.

To summarize the comparison:

**SVM (RBF Kernel)**:

Accuracy: 98.25%

Precision: 100%

**Random Forest**:

Accuracy: 96.49%

Precision: 97.56%

**Logistic Regression**:

Accuracy: 97.37%

Precision: 97.62%

**k-NN**:

Accuracy: 94.74%

Precision: 93.02%

**Gradient Boosting**:

Accuracy: 95.61%

Precision: 95.24%

**Decision Tree**:

Accuracy: 92.98%

Precision: 90.91%

**Majority Voting Classifier**:

Accuracy: 97.37%

Precision: 97.62%

If you have any further questions or need additional analysis, please let me know! ​​

4o

accuracies comparison

ChatGPT

To provide a clear comparison of the accuracies of all the models, we can create a comprehensive table and a bar chart.

Here is the table summarizing the accuracies of each model:

| **Model** | **Accuracy** | **Precision** |
| --- | --- | --- |
| SVM (RBF Kernel) | 98.25% | 100% |
| Logistic Regression | 97.37% | 97.62% |
| Majority Voting Classifier | 97.37% | 97.62% |
| Random Forest | 96.49% | 97.56% |
| Gradient Boosting | 95.61% | 95.24% |
| k-NN | 94.74% | 93.02% |
| Decision Tree | 92.98% | 90.91% |

Now, let's create a bar chart to visualize the accuracies of these models:

python

Copy code

# Accuracy scores for the models

accuracy\_scores = {

"SVM (RBF Kernel)": 98.25,

"Logistic Regression": 97.37,

"Majority Voting": 97.37,

"Random Forest": 96.49,

"Gradient Boosting": 95.61,

"k-NN": 94.74,

"Decision Tree": 92.98

}

# Plotting the accuracies

fig, ax = plt.subplots()

ax.bar(accuracy\_scores.keys(), accuracy\_scores.values(), color=['green', 'blue', 'purple', 'orange', 'red', 'cyan', 'magenta'])

# Adding titles and labels

ax.set\_title('Comparison of Accuracies for Different Models')

ax.set\_xlabel('Models')

ax.set\_ylabel('Accuracy (%)')

# Display the plot

plt.xticks(rotation=45)

plt.ylim(90, 100)

plt.show()

Let's generate the bar chart to compare the accuracies of these models.

Comparison of Accuracies for Different Models

Accuracy (%)

by

Models

Here is the bar chart comparing the accuracies of different models. From the chart, we can see:

The **SVM with RBF kernel** has the highest accuracy.

**Logistic Regression** and **Majority Voting Classifier** both perform very well, with identical accuracies.

**Random Forest** and **Gradient Boosting** also have high accuracies, followed by **k-NN** and **Decision Tree**.

This visualization helps to easily compare the performance of various models based on their accuracy.

If you need any further analysis or have other questions, feel free to ask! ​​

4o

Top of Form

Bottom of Form