Gaussian naive bayes

```
from sklearn.naive_bayes import GaussianNB
import numpy as np
X = np.array([[0,0],
              [0,1],
              [1,0],
              [2,1],
              [2,2],
              [1,1],
              [2,0]])
y = np.array([0, 0, 0, 1, 1, 0, 1])
model = GaussianNB()
model.fit(X, y)
new_fruit = np.array([[1, 0]])
prediction = model.predict(new_fruit)
print("Predicted class:", "Apple" if prediction == 0 else
"Orange")
```

Multinomial naive bayes

```
from sklearn.naive_bayes import MultinomialNB

from sklearn.feature_extraction.text import CountVectorizer

documents = ["I like programming", "python is great", "I hate bugs", "like solving problems", "I hate error"]

labels = [1, 1, 0, 1, 0]

vectorizer = CountVectorizer()

X = vectorizer.fit_transform(documents)

y = labels

model = MultinomialNB()

model.fit(X, y)

new_docs = ["i hate python", "i like reading"]

X_new = vectorizer.transform(new_docs)

predictions = model.predict(X_new)

print(predictions)
```

linear svm

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification, make_circles
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
X, y = make_classification(n_samples=100, n_features=2, n_informative=2,
n_redundant=0, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
linear_svm = SVC(kernel='linear')
linear_svm.fit(X_train, y_train)
y_pred_linear = linear_svm.predict(X_test)
print("Linear SVM:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_linear):.2f}")
```

```
print(classification_report(y_test, y_pred_linear))

plt.figure(figsize=(8, 6))

xx, yy = np.meshgrid(np.linspace(X[:, 0].min(), X[:, 0].max(),
100),

np.linspace(X[:, 1].min(), X[:, 1].max(), 100))

Z = linear_svm.decision_function(np.c_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, levels=[Z.min(), 0, Z.max()], alpha=0.3,
colors=['blue', 'red'])

plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k',
cmap=plt.cm.Paired)

plt.title("Linear SVM Decision Boundary")

plt.show()
```

Nonlinear svm

```
from sklearn.datasets import make_circles
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
import numpy as np


X, y = make_circles(n_samples=100, noise=0.1, factor=0.3, random_state=42)
# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Train an SVM with RBF kernel
nonlinear_svm = SVC(kernel='rbf', C=1, gamma='auto')
nonlinear_svm.fit(X_train, y_train)
# Make predictions
```

```
y_pred_nonlinear = nonlinear_svm.predict(X_test)

# Evaluate performance

print("Non-linear SVM:")

print(f"Accuracy: {accuracy_score(y_test,
y_pred_nonlinear):.2f}")

print(classification_report(y_test, y_pred_nonlinear))

# Plot decision boundary

plt.figure(figsize=(8, 6))

xx, yy = np.meshgrid(np.linspace(X[:, 0].min(), X[:, 0].max(),
100),

np.linspace(X[:, 1].min(), X[:, 1].max(), 100))

Z = nonlinear_svm.decision_function(np.c_[xx.ravel(),
yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, levels=[Z.min(), 0, Z.max()], alpha=0.3,
colors=['blue', 'red'])

plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k',
cmap=plt.cm.Paired)

plt.title("Non-linear SVM Decision Boundary")

plt.show()
```