1. Candidate Elimination

```
import numpy as np
import pandas as pd

# Load data
data = pd.read_csv('2.csv')
X = np.array(data.iloc[:, :-1])  # features
y = np.array(data.iloc[:, -1])   # target

def candidate_elimination(X, y):
    specific = X[0].copy()
    general = [['?' for _ in range(len(specific))] for _ in range(len(specific))]

    print(f"Initial specific: {specific}")
    print(f"Initial general: {general}\n")

    for i, instance in enumerate(X):
        print(f"Step {i+1}: Instance = {instance}, Target = {y[i]}")

        if y[i] == 'yes':  # Positive instance
            for j in range(len(specific)):
                if instance[j] != specific[j]:
                    specific[j] = '?'
                    general[j][j] = '?'
        else:  # Negative instance
            for j in range(len(specific)):
                if instance[j] != specific[j]:
                    general[j][j] = specific[j]
                else:
                    general[j][j] = '?'

        print(f"Specific: {specific}")
        print(f"General: {general}\n")

    general = [h for h in general if h != ['?'] * len(specific)]
    return specific, general

# Run algorithm
final_specific, final_general = candidate_elimination(X, y)
print("FINAL RESULTS:")
print(f"Specific Hypothesis: {final_specific}")
print(f"General Hypotheses: {final_general}")
```

---

2. ID3 (Decision Tree)

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.preprocessing import LabelEncoder

# Load and encode data
data = pd.read_csv("tennis.csv")
X, y = data.iloc[:, :-1], data.iloc[:, -1]

for col in X.columns:
    X[col] = LabelEncoder().fit_transform(X[col])
y = LabelEncoder().fit_transform(y)

# Train and print tree
tree = DecisionTreeClassifier(criterion='entropy', random_state=42)
tree.fit(X, y)
print(export_text(tree, feature_names=list(data.columns[:-1])))
```

3. Backpropagation (BP)

```
import numpy as np

# Data
X = np.array([[2, 9], [1, 5], [3, 6]])
y = np.array([[92], [86], [89]])

# Normalize
X = X / np.amax(X, axis=0)
y = y / 100

# Activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_deriv(x):
    return x * (1 - x)

# Initialize weights
input_size, hidden_size, output_size = 2, 3, 1
w1 = np.random.rand(input_size, hidden_size)
w2 = np.random.rand(hidden_size, output_size)

# Training loop
for i in range(10000):
    # Forward
    hidden = sigmoid(np.dot(X, w1))
    output = sigmoid(np.dot(hidden, w2))
    # Backward
    error = y - output
    d_output = error * sigmoid_deriv(output)
    d_hidden = d_output.dot(w2.T) * sigmoid_deriv(hidden)
    # Update weights
    w2 += hidden.T.dot(d_output)
    w1 += X.T.dot(d_hidden)

# Results
print("Input:\n", X)
print("Predicted Output:\n", output)
print("Loss:\n", np.mean(np.square(y - output)))
```

4. Naive Bayes (NB)

```python
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import pandas as pd
from sklearn.metrics import confusion_matrix, accuracy_score

# Load data
dataset = pd.read_csv('tennis.csv')
X = dataset.iloc[:, :-1].copy()
y = dataset.iloc[:, -1]

print("Data Before Encoding\n", dataset.head())
print("Features Before Encoding\n", X.head())
print("Target Before Encoding\n", y.head())

# Encode features
for column in X.columns:
    X[column] = LabelEncoder().fit_transform(X[column])

print("Features After Encoding\n", X.head())

# Encode target
y = LabelEncoder().fit_transform(y)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)

# Train model
model = GaussianNB()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Results
print("Actual Label: ", y_test)
print("Predicted Label: ", y_pred)
print("Confusion Matrix: \n", confusion_matrix(y_test, y_pred))
print("Accuracy Score: ", accuracy_score(y_test, y_pred) * 100)
```

---

5. EM (Expectation Maximization)

```
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
import sklearn.metrics as sm

# Load data
dataset = load_iris()
X = pd.DataFrame(dataset.data, columns=['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(dataset.target, columns=['Targets'])

print(X)

colormap = np.array(['red', 'lime', 'black'])
plt.figure(figsize=(14, 7))

# Plot 1: Real clusters
plt.subplot(1, 3, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title("Real")

# Plot 2: K-Means clustering
plt.subplot(1, 3, 2)
model = KMeans(n_clusters=3)
model.fit(X)
y_pred = np.choose(model.labels_, [0, 1, 2]).astype(np.int64)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_pred], s=40)
plt.title("KMeans")

# Plot 3: GMM
plt.subplot(1, 3, 3)
scaler = StandardScaler()
xs = scaler.fit_transform(X)
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
y_cluster_gmm = gmm.predict(xs)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_cluster_gmm], s=40)
plt.title("GMM")

# Performance metrics
ari_kMeans = sm.adjusted_rand_score(y.Targets, y_pred)
ari_gmm = sm.adjusted_rand_score(y.Targets, y_cluster_gmm)
print(f"Adjusted Rand Index for K-Means: {ari_kMeans:.4f}")
print(f"Adjusted Rand Index for GMM: {ari_gmm:.4f}")

accuracy_kMeans = np.mean(y.Targets.values.ravel() == y_pred)
accuracy_gmm = np.mean(y.Targets.values.ravel() == y_cluster_gmm)
plt.show()
```

---

6. KNN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load data and split
dataset = load_iris()
X_train, X_test, y_train, y_test = train_test_split(
    dataset.data, dataset.target, test_size=0.2, random_state=30
)

print("Training Labels:", y_train)

# Train KNN model
model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, y_train)

# Predict all test samples at once
y_pred = model.predict(X_test)

# Print predictions
for i in range(len(X_test)):
    actual = dataset.target_names[y_test[i]]
    predicted = dataset.target_names[y_pred[i]]
    print(f"Target: {actual}, Predicted: {predicted}")

# Calculate accuracy
accuracy = model.score(X_test, y_test)
print("Accuracy Score:", accuracy)
```

---

7. Locally Weighted Regression (LWR) using sklearn

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Training data
X = np.array([1, 2, 3, 4, 5])
y = np.array([1, 2, 1.3, 3.75, 2.25])

def lwr_sklearn(x_query, X, y, tau):
    weights = np.exp(-((X - x_query) ** 2) / (2 * tau ** 2))
    model = LinearRegression()
    model.fit(X.reshape(-1, 1), y, sample_weight=weights)
    prediction = model.predict([[x_query]])[0]
    return prediction, model.intercept_, model.coef_[0]

x_query, tau = 3, 1.0
y_pred, intercept, slope = lwr_sklearn(x_query, X, y, tau)

print("Observed value at x=3:", y[X == x_query][0])
print(f"Predicted value at x=3: {y_pred:.3f}")
print(f"Coefficients: Intercepts={intercept:.3f}, Slope={slope:.3f}")

x_vals = np.linspace(1, 5, 100)
y_vals = [lwr_sklearn(x, X, y, tau)[0] for x in x_vals]

plt.scatter(X, y, color='red', label='Data Points')
plt.plot(x_vals, y_vals, color='blue', label='LWR Prediction')
plt.scatter(x_query, y_pred, color='green', label='Predicted Value at x=3')
plt.scatter(x_query, y[X == x_query][0], color='orange', label='Observed value at x=3')
plt.legend()
plt.show()
```

8. SVM

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
import numpy as np
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

# Load data and use only first 2 features for 2D visualization
dataset = load_iris()
X = dataset.data[:, :2]
y = dataset.target

# Keep only classes 0 and 1 (remove class 2)
mask = y != 2
X = X[mask]
y = y[mask]

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train SVM model
model = SVC(kernel='linear')
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)

# Print results
print(f"No of Support Vectors: ", len(model.support_vectors_))
print(f"Support Vectors:\n ", model.support_vectors_)
print(f"Accuracy Score: ", accuracy_score(y_test, y_pred))

# Create decision boundary plot
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min, y_max, 0.02))
grid_points = np.c_[xx.ravel(), yy.ravel()]
grid_points_scaled = scaler.transform(grid_points)
Z = model.predict(grid_points_scaled).reshape(xx.shape)

plt.figure(figsize=(10, 8))
plt.contourf(xx, yy, Z, alpha=0.8)
plt.scatter(X[y==0, 0], X[y==0, 1], color='red', label='Class 0')
plt.scatter(X[y==1, 0], X[y==1, 1], color='green', label='Class 1')
sv_original = scaler.inverse_transform(model.support_vectors_)
plt.scatter(sv_original[:, 0], sv_original[:, 1], s=100, facecolors='none', edgecolors='blue', label='Support Vectors')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('SVM Classification with Linear Kernel')
plt.legend()
plt.show()
```

9. Random Forest

```python
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

# Load data
dataset = load_iris()

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    dataset.data, dataset.target, test_size=0.25, random_state=42
)

# Train model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Results
print("Accuracy Score: ", accuracy_score(y_test, y_pred))
print("Classification Report: ", classification_report(y_test, y_pred))
print("Confusion Matrix: ", confusion_matrix(y_test, y_pred))
```