Task 1

Import libraries

In [ ]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from collections import Counter
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

Balance classes in dataset

In [ ]:
```
columns = ['fLength', 'fWidth', 'fSize', 'fConc', 'fConc1', 'fAsym',
           'fM3Long', 'fM3Trans', 'fAlpha', 'fDist', 'class']

data = pd.read_csv('/kaggle/input/telescope-data/telescope_data.csv', names=columns

gamma = data[data['class'] == 'g']
hadron = data[data['class'] == 'h']

gamma_balanced = gamma.sample(n=len(hadron), random_state=42) # random_state gives

balanced_data = pd.concat([gamma_balanced, hadron])

balanced_data = balanced_data.sample(frac=1, random_state=42).reset_index(drop=True

X = balanced_data.drop('class', axis=1) # Seperate features from dataset for traini
y = balanced_data['class'].map({'g': 0, 'h': 1}) # Map gamma to 0 and hadron to 1
```

Splitting the dataset

In [ ]:
```
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_sta

X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, rand

print(f"Training set size: {len(X_train)}")
print(f"Validation set size: {len(X_val)}")
print(f"Test set size: {len(X_test)}") # Just to make sure
```

Manual KNN

In [ ]:
```python
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))

class KNN:
    def __init__(self, k):
        self.k = k

    def fit(self, X, y):
        self.X_train = X.to_numpy() # Convert dataset to numpy arrays for vector ma
        self.y_train = y.to_numpy()

    def predict(self, X):
        X = X.to_numpy()
        predictions = [self._predict(x) for x in X]
        return np.array(predictions)

    def _predict(self, x):
        distances = [euclidean_distance(x, x_train) for x_train in self.X_train]

        k_indices = np.argsort(distances)[:self.k] # Indices of nearest k points
        k_nearest_labels = [self.y_train[i] for i in k_indices] # Get their labels

        most_common = Counter(k_nearest_labels).most_common(1) # Get label with mos
        return most_common[0][0] # Return winning label

    def evaluate(self, X, y):
        X = X.to_numpy()
        y = y.to_numpy()
        predictions = self.predict(X)
        accuracy = np.sum(predictions == y) / len(y)
        return accuracy
```

Test manual KNN

In [ ]:
```python
scaler = StandardScaler() # Normaliza features for (hopefully) better results
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

X_train_scaled = pd.DataFrame(X_train_scaled, columns=X.columns) # Return datatype
X_val_scaled = pd.DataFrame(X_val_scaled, columns=X.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X.columns)

k_values = range(1, 20)
manual_accuracies = []

for k in k_values:
    knn = KNN(k=k)
    knn.fit(X_train_scaled, y_train)
    accuracy = knn.evaluate(X_val_scaled, y_val)
    manual_accuracies.append(accuracy)
    print(f"k={k}, Validation Accuracy={accuracy:.4f}")
```

```
best_k_manual = k_values[np.argmax(manual_accuracies)]
print(f"Best k for manual implementation: {best_k_manual}")
```

KNN using scikit-learn

In [ ]:
```
sklearn_accuracies = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    y_pred = knn.predict(X_val_scaled)
    accuracy = accuracy_score(y_val, y_pred)
    sklearn_accuracies.append(accuracy)
    print(f"k={k}, Validation Accuracy={accuracy:.4f}")

best_k_sklearn = k_values[np.argmax(sklearn_accuracies)]
print(f"Best k for sklearn implementation: {best_k_sklearn}")
```

Plot accuracy for each K

In [ ]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.plot(k_values, manual_accuracies, label='Manual Implementation', marker='o')
plt.plot(k_values, sklearn_accuracies, label='Scikit-Learn', marker='x')
plt.xlabel('k Value')
plt.ylabel('Validation Accuracy')
plt.title('Validation Accuracy vs. k Value')
plt.axvline(x=best_k_manual, color='blue', linestyle='--', label=f'Best Manual k={b
plt.axvline(x=best_k_sklearn, color='orange', linestyle='--', label=f'Best Sklearn
plt.legend()
plt.grid()
plt.show()
```

Testing models using best K

In [ ]:
```
# Manual knn test
manual_knn = KNN(k=best_k_manual)
manual_knn.fit(X_train_scaled, y_train)
y_pred_manual = manual_knn.predict(X_test_scaled)

# Sklearn knn test
sklearn_knn = KNeighborsClassifier(n_neighbors=best_k_sklearn)
sklearn_knn.fit(X_train_scaled, y_train)
y_pred_sklearn = sklearn_knn.predict(X_test_scaled)

def evaluate_model(y_true, y_pred, model_name):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
```

```
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    cm = confusion_matrix(y_true, y_pred)

    print(f"\n{model_name} Evaluation:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print("Confusion Matrix:")
    print(cm)

evaluate_model(y_test, y_pred_manual, "Manual KNN")
evaluate_model(y_test, y_pred_sklearn, "Scikit-Learn KNN")
```