Implementations of various Machine Leaning methods in Python

By Saumya Gupta

Reg. No. 2022CS18

M.Tech (C.S.E.)

Implementation of Artificial Neural Networks (ANN)

localhost:8890/notebooks/Artificial%20Neural%20Networks%20(ANN).ipynb

jupyter Artificial Neural Networks (ANN) Last Checkpoint: 7 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [7]:
```
import numpy as np
```

In [8]:
```
# Define the sigmoid function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

In [9]:
```
# Set up the training data
X = np.array([[0, 0, 1], [0, 1, 1], [1, 0, 1], [1, 1, 1]])
y = np.array([[0], [1], [1], [0]])
```

In [10]:
```
# Set up the weights for the first layer (input layer -> hidden layer)
w1 = 2 * np.random.random((3, 4)) - 1
```

In [11]:
```
# Set up the weights for the second layer (hidden layer -> output layer)
w2 = 2 * np.random.random((4, 1)) - 1
```

In [12]:
```
# Train the model
for i in range(10000):
    # Forward propagation
    layer1 = sigmoid(np.dot(X, w1))
    layer2 = sigmoid(np.dot(layer1, w2))
```

localhost:8890/notebooks/Artificial%20Neural%20Networks%20(ANN).ipynb

jupyter Artificial Neural Networks (ANN) Last Checkpoint: 8 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

```
    # Backpropagation
    layer2_error = y - layer2
    layer2_delta = layer2_error * (layer2 * (1 - layer2))
    layer1_error = np.dot(layer2_delta, w2.T)
    layer1_delta = layer1_error * (layer1 * (1 - layer1))

    # Update the weights
    w2 += np.dot(layer1.T, layer2_delta)
    w1 += np.dot(X.T, layer1_delta)
```

In [13]:

```
# Test the model
test_input = np.array([1, 0, 1])
test_output = sigmoid(np.dot(sigmoid(np.dot(test_input, w1)), w2))
print(f'Test input: {test_input}, Test output: {test_output}')
```

```
Test input: [1 0 1], Test output: [0.98816852]
```

Implementation of K- Nearest Neighbour Algorithm (KNN)

localhost:8890/notebooks/K-%20Nearest%20Neighbour%20Algorithm.ipynb

jupyter K- Nearest Neighbour Algorithm Last Checkpoint: a few seconds ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [1]:
```
import numpy as np
from collections import Counter
```

In [2]:
```
class KNearestNeighbors:
    def __init__(self, k):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

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

    def _predict(self, x):
        distances = [self._euclidean_distance(x, x_train) for x_train in self.X_train]
        k_nearest_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in k_nearest_indices]
        most_common_label = Counter(k_nearest_labels).most_common(1)[0][0]
        return most_common_label

    def _euclidean_distance(self, x1, x2):
        return np.sqrt(np.sum((x1 - x2) ** 2))
```

localhost:8890/notebooks/K-%20Nearest%20Neighbour%20Algorithm.ipynb

jupyter K- Nearest Neighbour Algorithm Last Checkpoint: a few seconds ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3 (ipykernel)

Run Code

```
        most_common_label = Counter(k_nearest_labels).most_common(1)[0][0]
        return most_common_label

    def _euclidean_distance(self, x1, x2):
        return np.sqrt(np.sum((x1 - x2) ** 2))
```

In [3]:

```
X_train = np.array([[1, 2], [1.5, 1.8], [5, 8], [8, 8], [1, 0.6], [9, 11]])
y_train = np.array([0, 0, 1, 1, 0, 1])
X_test = np.array([[2, 3], [3, 4], [9, 9]])

knn = KNearestNeighbors(k=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print(y_pred) # Output: [0 0 1]
```

```
[0 0 1]
```

Implementation of Linear Discriminant Analysis in Python

Home Page - Select or create a n | LDA (Linear Discriminant Analysis | PCA (Principle Component Analy

localhost:8890/notebooks/LDA%20(Linear%20Discriminant%20Analysis).ipynb

jupyter LDA (Linear Discriminant Analysis) Last Checkpoint: Last Monday at 7:16 PM (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [1]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.datasets import load_iris
```

In [2]:
```
# Load the iris dataset
iris = load_iris()
X = iris.data
y = iris.target
```

In [3]:
```
# Create a LDA object and fit the data
lda = LinearDiscriminantAnalysis()
lda.fit(X, y)
```

Out[3]:
```
▾ LinearDiscriminantAnalysis
LinearDiscriminantAnalysis()
```

In [4]:
```
# Predict the class labels for new data
new_data = [[5.1, 3.5, 1.4, 0.2], [6.2, 2.8, 4.8, 1.8]]
print(lda.predict(new_data))
```

```
[0 2]
```

Implementation of Linear Regression in Python

Home Page - Select or create a n | Linear Regression - Jupyter Note | Linear Regression - Jupyter Note

localhost:8890/notebooks/Linear%20Regression.ipynb

jupyter Linear Regression Last Checkpoint: a few seconds ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

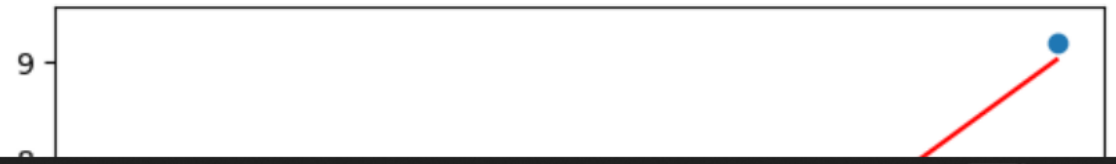
Run Code

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

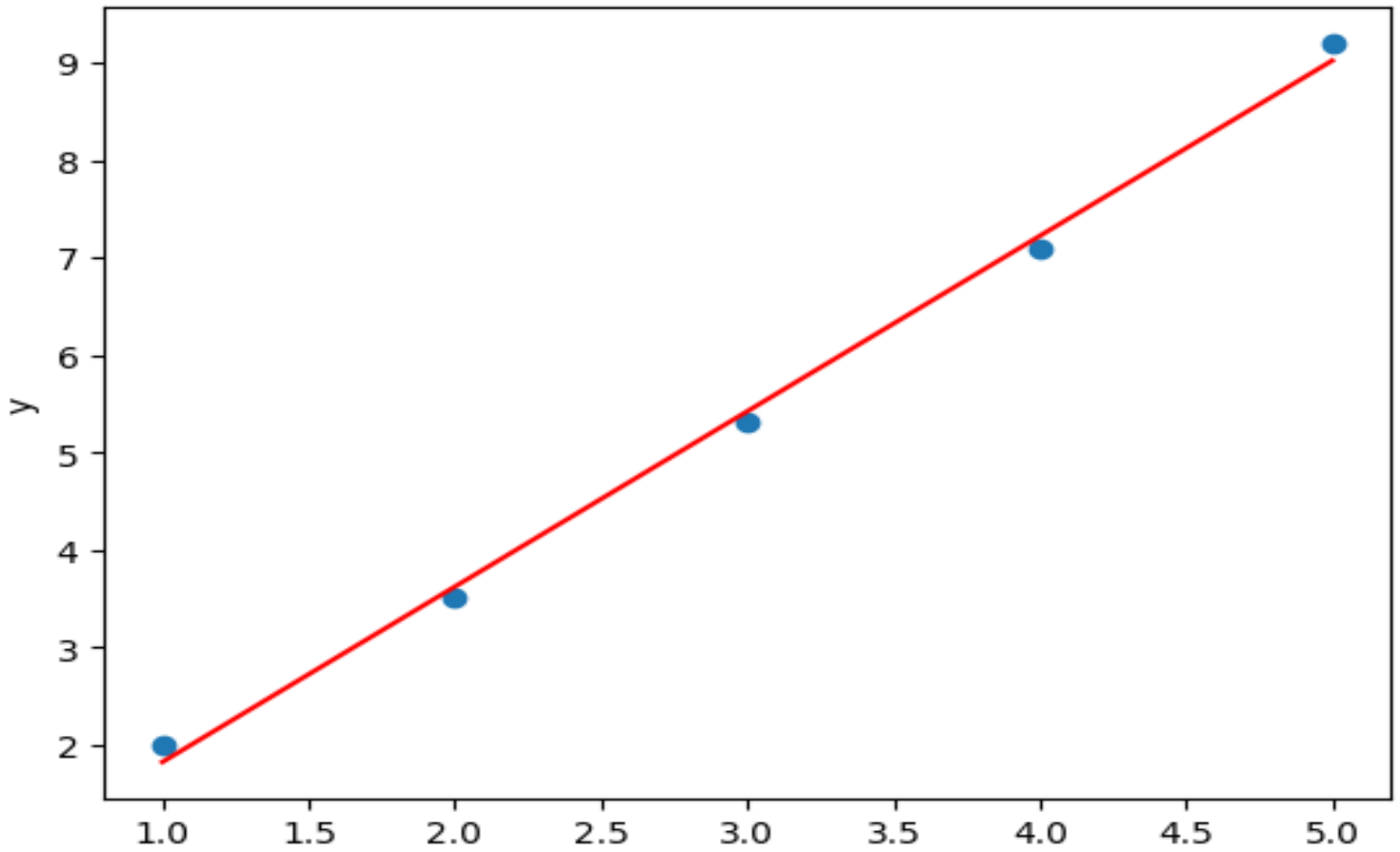
In [2]:
```
# Generate some random data
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 3.5, 5.3, 7.1, 9.2])
```

In [3]:
```
# Calculate the slope and y-intercept using linear algebra
X = np.vstack([x, np.ones(len(x))]).T
m, c = np.linalg.lstsq(X, y, rcond=None)[0]
```

In [4]:
```
# Plot the data points and the regression line
plt.scatter(x, y)
plt.plot(x, m*x + c, 'r')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



Search ENG IN 21:50 28-04-2023



In [5]:
```
# Print the slope and y-intercept
print('Slope:', m)
print('Y-Intercept:', c)
```

```
Slope: 1.7999999999999998
Y-Intercept: 0.01999999999999983
```

Implementation of Logistic Regression in Python

localhost:8890/notebooks/Logistic%20Regression.ipynb

jupyter Logistic Regression Last Checkpoint: 3 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [1]:
```
# Import necessary libraries
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [2]:
```
# Load dataset
iris = load_iris()
X = iris.data
y = iris.target
```

In [3]:
```
# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [4]:
```
# Create logistic regression model
log_reg = LogisticRegression()
```

In [5]:
```
# Train the model
log_reg.fit(X_train, y_train)
```

Out[5]:
```
▾ LogisticRegression
LogisticRegression()
```

localhost:8890/notebooks/Logistic%20Regression.ipynb

jupyter Logistic Regression Last Checkpoint: 3 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

Out[5]:

```
LogisticRegression
LogisticRegression()
```

In [6]:

```
# Make predictions on test set
y_pred = log_reg.predict(X_test)
```

In [7]:

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
```

In [8]:

```
# Print the accuracy
print("Accuracy:", accuracy)
```

```
Accuracy: 1.0
```

Implementation of Naïve Bayes Classifier in Python

localhost:8890/notebooks/Naive%20Bayes%20Classifier.ipynb

jupyter Naive Bayes Classifier Last Checkpoint: 5 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [1]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
```

In [2]:
```
# Define the data
texts = ['this is a positive sentence', 'this is a negative sentence', 'this is a positive review', 'this is a negative review']
labels = ['positive', 'negative', 'positive', 'negative']
```

In [3]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(texts, labels, test_size=0.2, random_state=42)
```

In [4]:
```
# Convert the text data into a bag-of-words representation
vectorizer = CountVectorizer()
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
```

In [5]:
```
# Train the Naive Bayes classifier
clf = MultinomialNB()
clf.fit(X_train, y_train)
```

Out[5]:
```
▾ MultinomialNB
MultinomialNB()
```

localhost:8890/notebooks/Naive%20Bayes%20Classifier.ipynb

jupyter Naive Bayes Classifier Last Checkpoint: 6 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [6]:

```
# Make predictions on the test set
y_pred = clf.predict(X_test)
```

In [7]:

```
# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.0
```

In [8]:

```
# Print the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:\n', cm)
```

```
Confusion Matrix:
 [[0 1]
 [0 0]]
```

Implementation of Principal Component Analysis (PCA)

localhost:8890/notebooks/PCA%20(Principle%20Component%20Analysis).ipynb

jupyter PCA (Principle Component Analysis) Last Checkpoint: 13 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3 (ipykernel)

Run Code

In [1]:
```
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
```

In [2]:
```
# Load the iris dataset
iris = load_iris()
X = iris.data
```

In [3]:
```
# Create a PCA object and fit the data
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
```

In [4]:
```
# Print the explained variance ratios
print('Explained variance ratio:', pca.explained_variance_ratio_)
```

```
Explained variance ratio: [0.92461872 0.05306648]
```

In [5]:
```
# Plot the transformed data
import matplotlib.pyplot as plt
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=iris.target)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```

localhost:8890/notebooks/PCA%20(Principle%20Component%20Analysis).ipynb

jupyter PCA (Principle Component Analysis) Last Checkpoint: 11 minutes ago (autosaved)

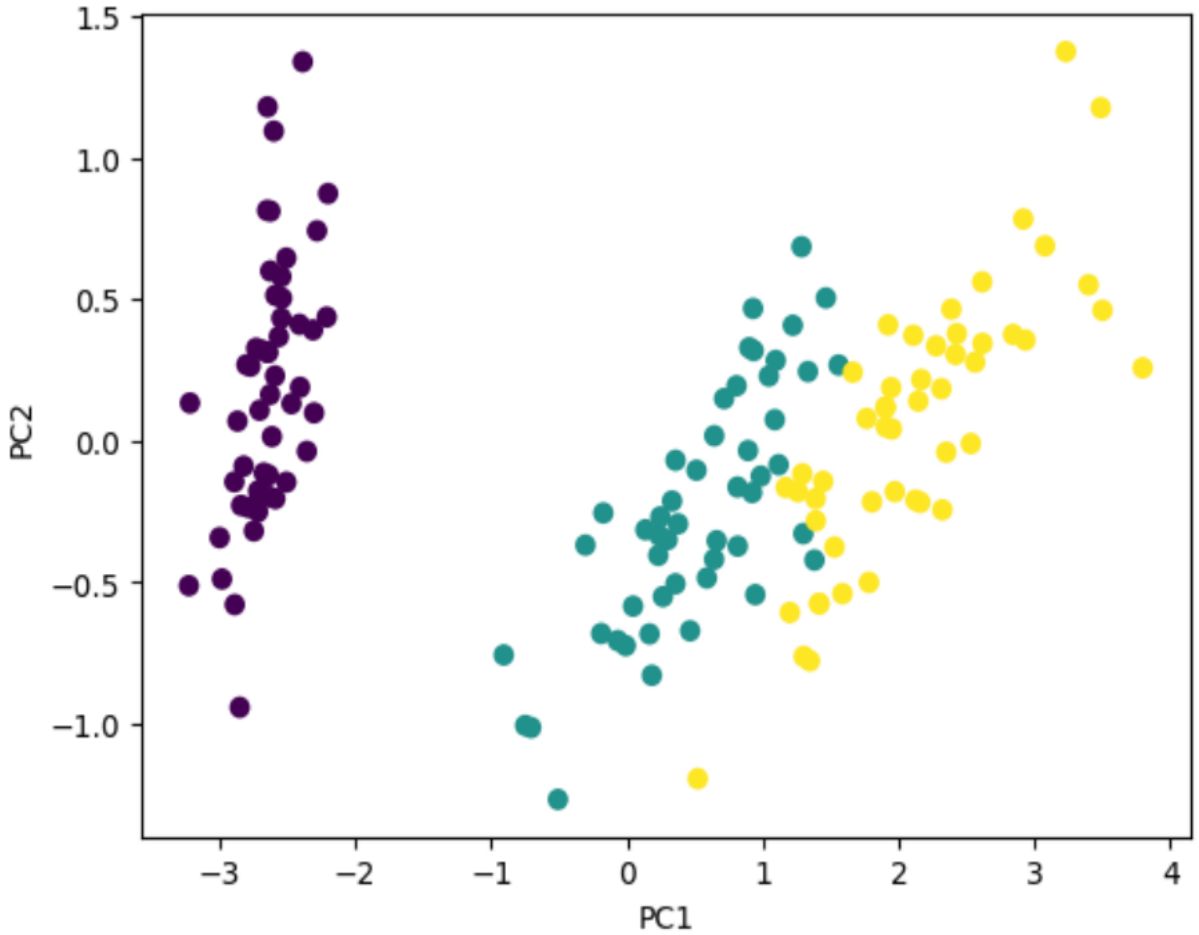
Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3 (ipykernel)

Run Code

```
plt.show()
```



Implement Support Vector Machine in Python

localhost:8890/notebooks/Support%20Vector%20Machine%20(SVM).ipynb

jupyter Support Vector Machine (SVM) Last Checkpoint: 39 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [11]:
```
# Import necessary libraries
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

In [12]:
```
# Load dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

In [13]:
```
# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [14]:
```
# Create SVM classifier
svm = SVC(kernel='linear', C=1)
```

In [15]:
```
# Train the classifier
svm.fit(X_train, y_train)
```

Out[15]:
```
SVC
SVC(C=1, kernel='linear')
```

localhost:8890/notebooks/Support%20Vector%20Machine%20(SVM).ipynb

jupyter Support Vector Machine (SVM) Last Checkpoint: 40 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

Out[15]:

```
SVC
SVC(C=1, kernel='linear')
```

In [16]:

```
# Make predictions on test set
y_pred = svm.predict(X_test)
```

In [17]:

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
```

In [18]:

```
# Print the accuracy
print("Accuracy:", accuracy)
```

```
Accuracy: 1.0
```

Implement k-mean Clustering in Python

jupyter K means Clustering Last Checkpoint: 3 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [4]:
```
def closest_centroid(X, centroids):
    """ Return an array containing the index of the closest centroid to each data point in X """
    distances = np.sqrt(((X - centroids[:, np.newaxis])**2).sum(axis=2))
    return np.argmin(distances, axis=0)

def move_centroids(X, closest, centroids):
    """ Update the centroids based on the mean of the points closest to them """
    return np.array([X[closest == k].mean(axis=0) for k in range(centroids.shape[0])])

def k_means(X, k, max_iter=100):
    centroids = initialize_centroids(X, k)
    for i in range(max_iter):
        closest = closest_centroid(X, centroids)
        new_centroids = move_centroids(X, closest, centroids)
        if np.all(centroids == new_centroids):
            break
        centroids = new_centroids
    return centroids, closest
```

localhost:8890/notebooks/K%20means%20Clustering.ipynb

jupyter K means Clustering Last Checkpoint: 3 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [5]:
```
# Example usage
X = np.array([[1, 2], [1, 4], [1, 0],
              [4, 2], [4, 4], [4, 0]])
k = 2

centroids, closest = k_means(X, k)
```

In [6]:
```
# Plot the results
colors = ['r', 'g', 'b', 'c', 'm', 'y', 'k']
for i in range(k):
    plt.scatter(X[closest == i, 0], X[closest == i, 1], s=30, c=colors[i])
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=200, c='k')
plt.show()
```

localhost:8890/notebooks/K%20means%20Clustering.ipynb

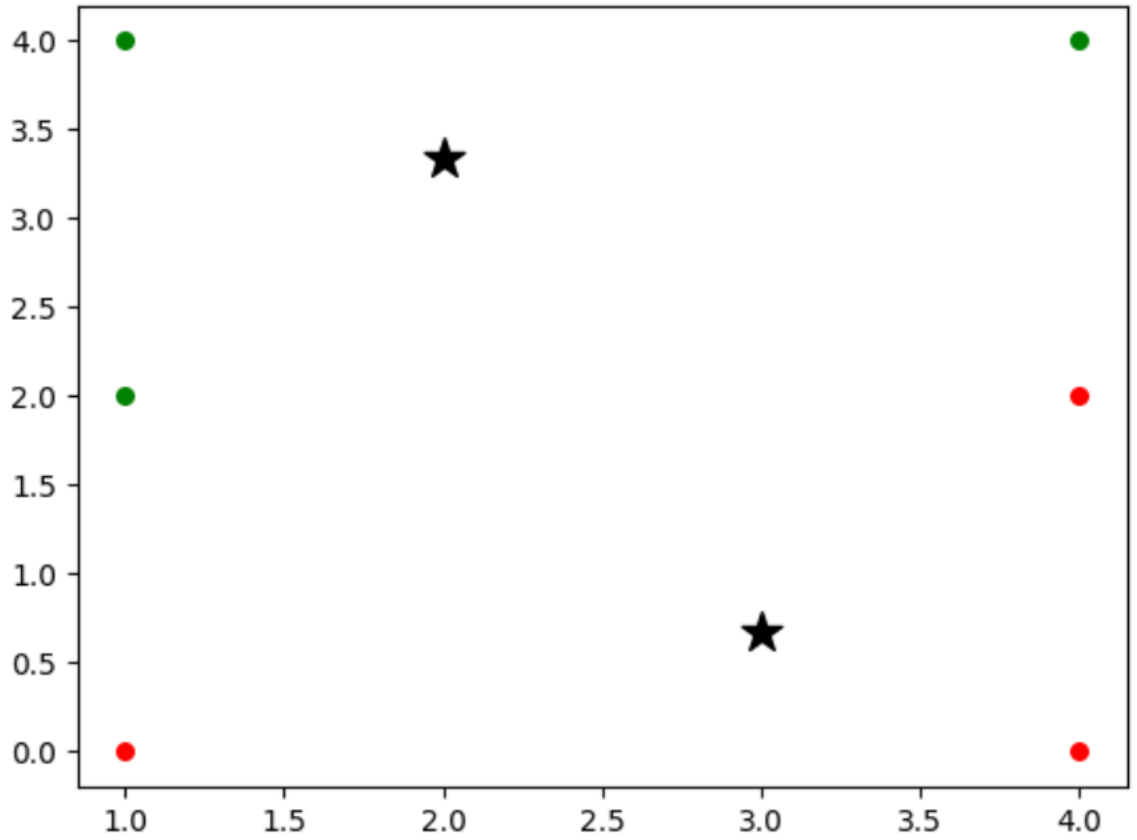
jupyter K means Clustering Last Checkpoint: 4 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

```
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=200, c='k')
plt.show()
```



Implement Decision Tree in Python

localhost:8890/notebooks/Decision%20Tree.ipynb

jupyter Decision Tree Last Checkpoint: a minute ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [1]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [2]:
```
# Create a decision tree classifier
clf = DecisionTreeClassifier()
```

In [3]:
```
# Train the classifier on a dataset
X = [[0, 0], [0, 1], [1, 0], [1, 1]]
y = [0, 1, 1, 0]
clf.fit(X, y)
```

Out[3]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [4]:
```
# Use the classifier to make predictions
X_test = [[0, 0], [0, 1], [1, 0], [1, 1]]
y_pred = clf.predict(X_test)

print(y_pred)
```

```
[0 1 1 0]
```