# Project Report

**Subject :** Pattern Processing using Artificial Intelligence

**Submitted By :**
*Sarthak Shukla* [2019UCO1732]

**Under:** Asst Prof. Rudresh Dwivedi

# Experiment -1

**Aim:** Write a python program to implement a chat bot

**Code:**

```python
from chatterbot import ChatBot
from chatterbot.trainers import ListTrainer
bot = ChatBot('MyBot')
trainer = ListTrainer(bot)
trainer.train([
    'Hi',
    'Hello',
    'How are you?',
    'I am fine, thank you.',
    'What is your name?',
    'My name is MyBot',
    'Who created you?',
    'I was created by a team of developers.'
])
response = bot.get_response('Hello')
print(response)
```

```
Training conversations.yml: [####################] 100%
You: Hi
Bot:  Hello
You: How are you ?
Bot:  i am doing great these days
You: I am good.
Bot:  Simple is better than complex.

You:
```

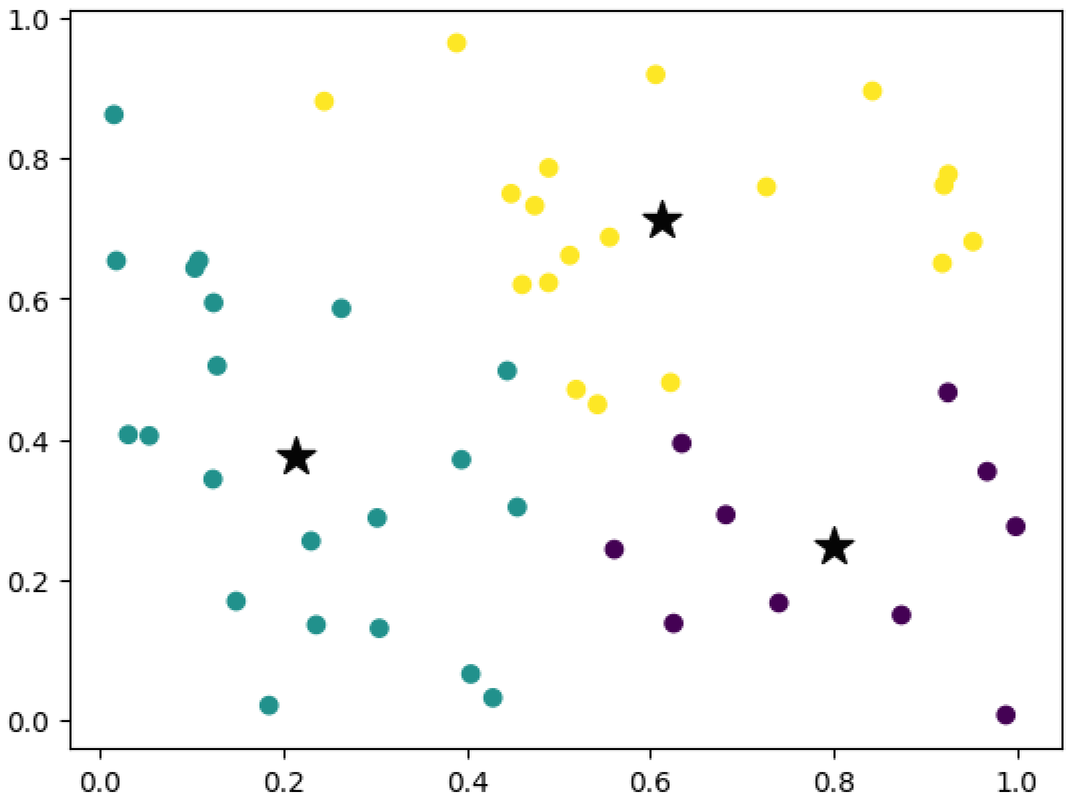
# Experiment - 2

**Aim:** Write a python program to implement kMeans from scratch

## Code:

```
import numpy as np
import matplotlib.pyplot as plt

X = np.random.rand(50, 2)
K = 3
centroids = X[np.random.choice(X.shape[0], K, replace=False)]
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2)**2))
def kMeans(X, K, num_iterations):
    N = X.shape[0]
    clusters = np.zeros(N)
    for i in range(num_iterations):
        # Assign each data point to its nearest centroid
        for j in range(N):
            distances = np.zeros(K)
            for k in range(K):
                distances[k] = euclidean_distance(X[j], centroids[k])
            clusters[j] = np.argmin(distances)
        # Update centroids to the mean of the points assigned to each
cluster
        for k in range(K):
            centroids[k] = np.mean(X[clusters == k], axis=0)
    return clusters
clusters = kMeans(X, K, 10)
fig, ax = plt.subplots()
ax.scatter(X[:,0], X[:,1], c=clusters, cmap='viridis')
ax.scatter(centroids[:,0], centroids[:,1], marker='*', s=200, c='#050505')
plt.show()
```

**Output:**




# Experiment - 3

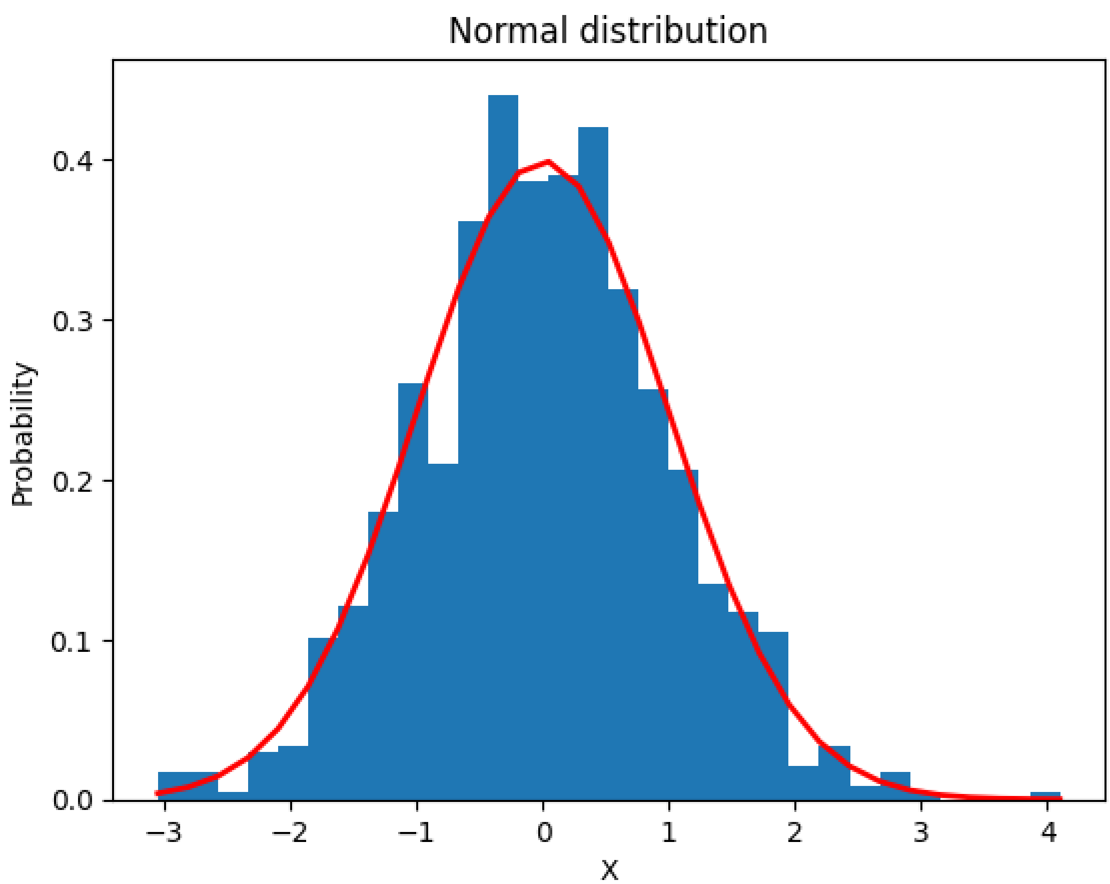
**Aim:** Generating samples of Normal distribution and plotting them

**Code:**

```
import numpy as np
import matplotlib.pyplot as plt


s = np.random.normal(mu, sigma, 1000)
```

```
count, bins, ignored = plt.hist(s, 30, density=True)

plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) *
         np.exp( - (bins - mu)**2 / (2 * sigma**2) ),
         linewidth=2, color='r')

plt.xlabel('X')
plt.ylabel('Probability')
plt.title('Normal distribution')

plt.show()
```

**Output:**



# Experiment - 4

**Aim:** Implement Decision tree and visualize it

## Implementation Code:

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

iris = load_iris()

X_train, X_test, y_train, y_test = train_test_split(iris.data,
iris.target, test_size=0.2, random_state=42)

dtc = DecisionTreeClassifier()

dtc.fit(X_train, y_train)

y_pred = dtc.predict(X_test)

score = dtc.score(X_test, y_test)

print(f"Accuracy: {score*100}")
```

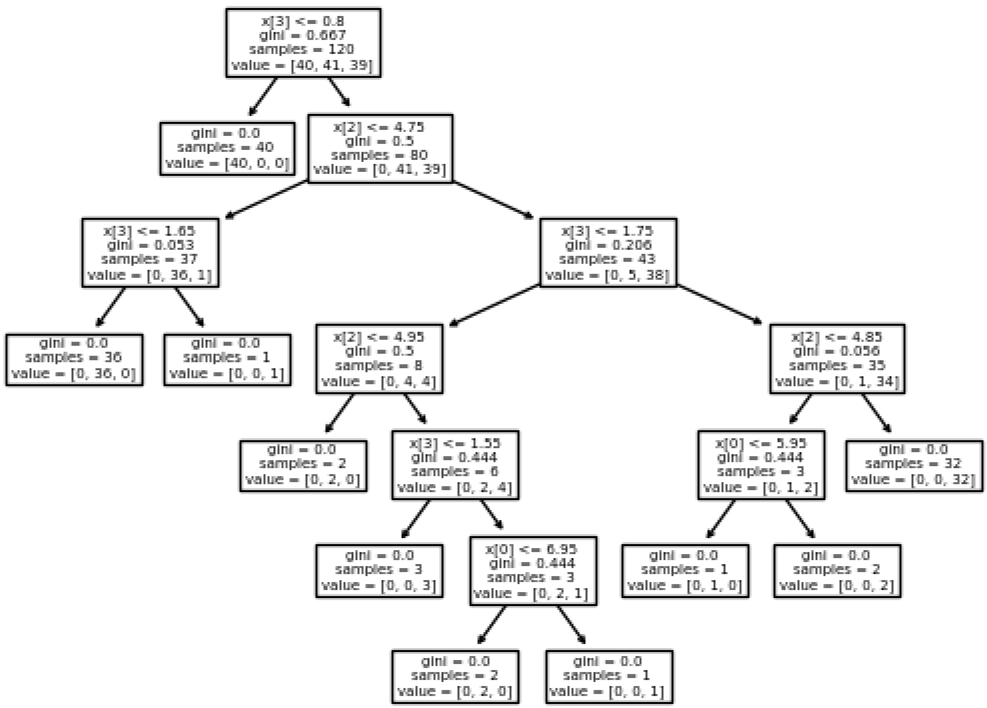
## Implementation output:

```
Accuracy: 100.0
```

## Visualizing code:

```
from sklearn.tree import plot_tree
plot_tree(dtc)
```

**Visualization output:**




# Experiment - 5

**Aim:** Implement SVM

## Code:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

iris = datasets.load_iris()

X_train, X_test, y_train, y_test = train_test_split(iris.data,
iris.target, test_size=0.3, random_state=42)

svm = SVC(kernel='linear')

svm.fit(X_train, y_train)

y_pred = svm.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy*100)
```

## Output:

**Accuracy: 100.0**

---

# Experiment - 6

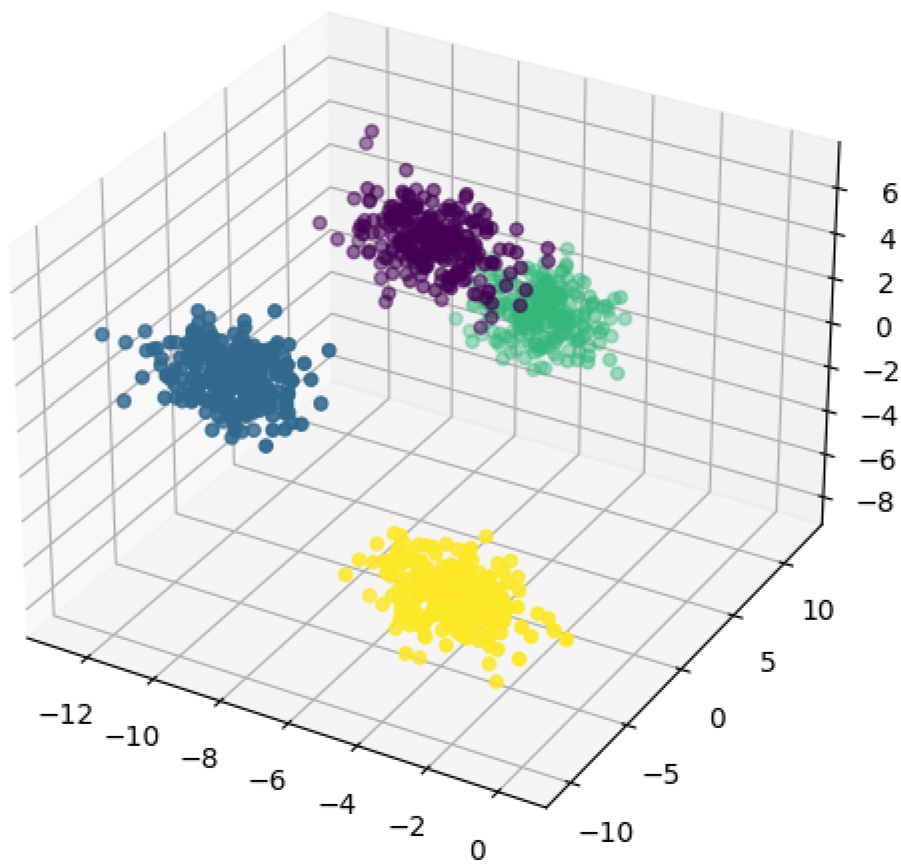
**Aim:** Implement Principal Components Analysis and use it for unsupervised learning.

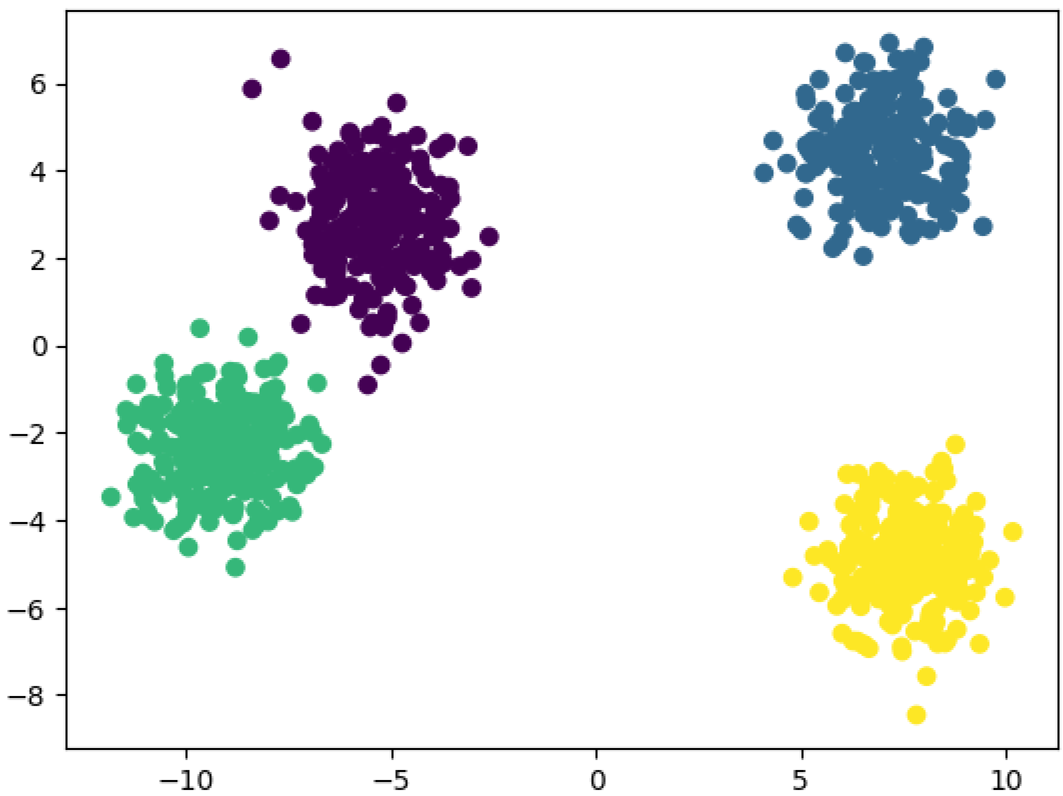
**Code:**

```
import numpy as np
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans
class PCA:

    def __init__(self, n_components):
        self.n_components = n_components
        self.components = None
        self.mean = None

    def fit(self, X):
        self.mean = np.mean(X, axis=0)
        X = X - self.mean

        cov = np.cov(X.T)

        eigenvalues, eigenvectors = np.linalg.eig(cov)

        eigenvectors = eigenvectors.T
        idxs = np.argsort(eigenvalues)[::-1]
        eigenvalues = eigenvalues[idxs]
        eigenvectors = eigenvectors[idxs]

        self.components = eigenvectors[0:self.n_components]

    def transform(self, X):
        X = X - self.mean

        return np.dot(X, self.components.T)

X, y = make_blobs(n_samples=1000, n_features=3, centers=4)
```

```
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[:,0], X[:,1], X[:,2], c=y)
plt.show()

pca = PCA(n_components=2)
pca.fit(X)
X_pca = pca.transform(X)

plt.scatter(X_pca[:,0], X_pca[:,1], c=y)
plt.show()

kmeans = KMeans(n_clusters=4)
kmeans.fit(X_pca)
y_pred = kmeans.predict(X_pca)

plt.scatter(X_pca[:,0], X_pca[:,1], c=y_pred)
plt.show()
```

**Output**:






# Experiment - 7

**Aim:** Implement Maximum likelihood estimation

**Code:**

```
import numpy as np
from scipy.optimize import minimize

def likelihood(params, data):
    mu, sigma = params
    n = len(data)
    log_likelihood = -(n/2)*np.log(2*np.pi*sigma**2) -
(1/(2*sigma**2))*np.sum((data-mu)**2)
```

```
    return -log_likelihood

np.random.seed(123)
data = np.random.normal(0, 1, 100)

initial_guess = [0, 1]
result = minimize(likelihood, initial_guess, args=data,
method='Nelder-Mead')
mu_MLE, sigma_MLE = result.x

print("Maximum likelihood estimates:")
print("mu = {:.2f}".format(mu_MLE))
print("sigma = {:.2f}".format(sigma_MLE))
```

### Output:

```
Maximum likelihood estimates:
mu = 0.03
sigma = 1.13
```

# Experiment - 8

**Aim**: Implement Agglomerative Hierarchical Clustering

### Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import AgglomerativeClustering

X, y = make_blobs(n_samples=100, centers=3, random_state=42)
```

```
agg_clustering = AgglomerativeClustering(n_clusters=3)
agg_clustering.fit(X)

plt.scatter(X[:, 0], X[:, 1], c=agg_clustering.labels_, cmap='viridis')
plt.show()
```

**Output:**