Ghazi Ben Henia

Report of the AI project
-Principal Component Analysis and Support Vector Machines-

I. Introduction :

In this project, we aim to understand Principal Component Analysis (PCA) and Support Vector Machines (SVM) using Python. Our goal is to sort out plant data about different types of flowers using a method called supervised machine learning. This means we have a labeled dataset (called flowerTrain_data.csv) that helps us. It shows how the shapes of flowers vary among three species: interior, versicolor, and convoluta. We have 48 examples for each species in the dataset. Each example has four measurements: the length and width of sepals, and the length and width of petals (all in cm).

1)Principal Component Analysis (PCA):

PCA is a statistical technique used for dimensionality reduction in data analysis. It aims to simplify complex datasets by transforming them into a new coordinate system, where the dimensions are ordered by their importance. This transformation is achieved by identifying the principal components, which are the directions in the data that capture the maximum variance. PCA is commonly employed for exploratory data analysis, visualization, and noise reduction, as well as for preparing data for subsequent machine learning tasks.

PCA consists of the following six steps:

We align *m* measures in a *n* dimensional space in a matrix $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \dots & \mathbf{x}_m \end{bmatrix}$

Each **x** has *n* dimensions, but these are too many!
We want to reduce to the *p principal* dimensions

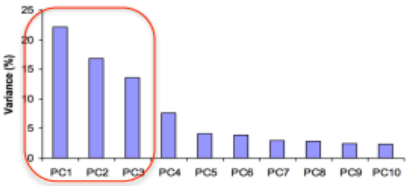
1) Compute the mean of all measures $\bar{\mathbf{x}} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}_i$

2) Shift matrix **X** by this mean (this is equivalent to translating the coordinate system to the location of the mean) $\bar{\mathbf{X}} = \begin{bmatrix} \mathbf{x}_1 - \bar{\mathbf{x}} & \dots & \mathbf{x}_m - \bar{\mathbf{x}} \end{bmatrix}$

3) Build covariance matrix $\mathbf{C} = \frac{1}{m-1}\bar{\mathbf{X}}\bar{\mathbf{X}}^{\mathrm{T}}$

4) Using Singular value decomposition, extract eigenvalues and normalized eigenvectors of **C**

$$\mathbf{C} = \mathbf{U}\Lambda\mathbf{V}^{\mathrm{T}}$$

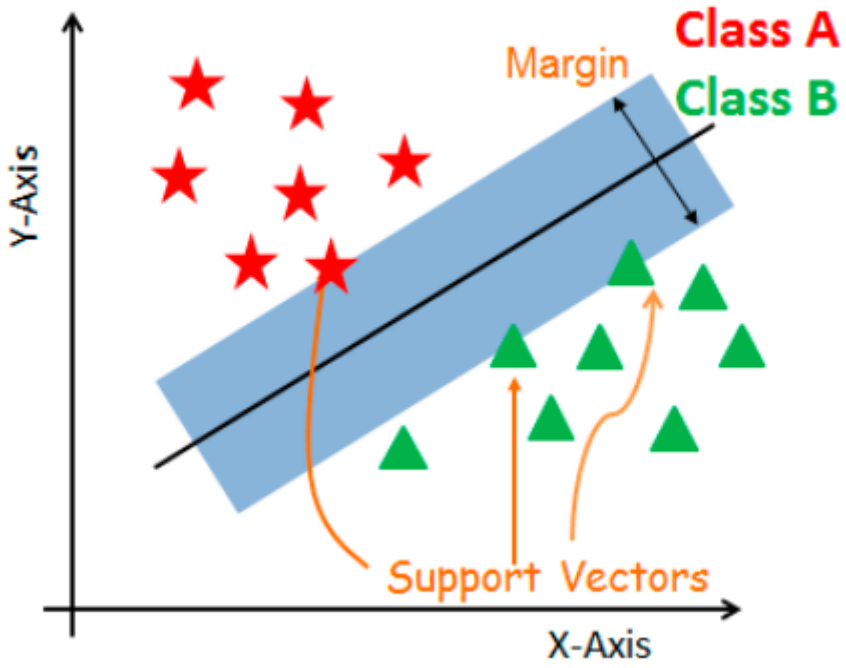


5) Choose the *p* largest eigenvalues and reduce **U** to the corresponding *p* eigenvectors (columns)

6) The data can now be reduced to: $\mathbf{s} = \mathbf{U}_p^{\mathrm{T}}\left(\mathbf{x} - \bar{\mathbf{x}}\right)$

2)Support Vector Machines (SVM):

SVM is a powerful supervised learning algorithm used for classification and regression tasks. It works by finding the hyperplane that best separates different classes in a high-dimensional feature space. The hyperplane is positioned to maximize the margin, which is the distance between the hyperplane and the closest data points (called support vectors) of each class. SVM is particularly effective in scenarios with complex decision boundaries and is widely used in various fields, including image classification, text classification, and bioinformatics. Additionally, SVM can be extended to handle non-linear classification tasks through the use of kernel functions, allowing it to capture more intricate relationships within the data.



3)Preparing the data:

We load the dataset 'flowerTrain_data.csv' into a Pandas DataFrame. Initially, the dataset is displayed to understand its structure.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.svm import SVC
import seaborn as sb
```

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount,
```

```
[ ] %cd /content/drive/MyDrive/dataflower
```

```
/content/drive/MyDrive/dataflower
```

```
[ ] dataset = pd.read_csv('/content/drive/MyDrive/dataflower/flowerTrain_dat
```

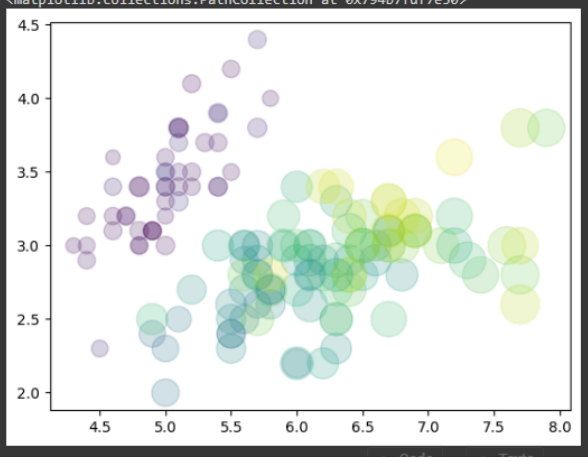
```
[ ] dataset.head()
```

	4.7	3.2	1.3	0.2	interior
0	4.6	3.1	1.5	0.2	interior
1	5.0	3.6	1.4	0.2	interior
2	5.4	3.9	1.7	0.4	interior
3	4.6	3.4	1.4	0.3	interior
4	5.0	3.4	1.5	0.2	interior

Following this, the dataset is reloaded with specified column names: 'sepal length', 'sepal width', 'petal length', 'petal width', and 'species'.
Subsequently, a scatter plot is created using the sepal length and sepal width as coordinates. The plot utilizes petal length and petal width to determine the size and color intensity of the plotted points, respectively, providing a visual representation of the data's characteristics.

```
import pandas as pd

dataset = pd.read_csv('/content/drive/MyDrive/dataflower/flowerTrain_data.csv',
                      names=['sepal length', 'sepal width', 'petal length', 'petal width', 'species'])
X = dataset.iloc[:,0:4]
plt.scatter(dataset.iloc[:,0],dataset.iloc[:,1],alpha = 0.2,
            s = 100 * dataset.iloc[:,2],c = 100 * dataset.iloc[:,3])
```

```
<matplotlib.collections.PathCollection at 0x794b7fdf7e50>
```



- ## II. Principal Component Analysis :

1)Practicing PCA on two examples:

The goal is to reduce the data from 2 dimensions to 1 dimension. I designed a function def PCA_func(X , reducedDim) to implement the six steps of PCA.

```python
def PCA_func(X, reducedDim):
    # mean of all measures
    mean_X = np.mean(X, axis=0)
    # Shifting X by this mean
    X_shift = X - mean_X
    # covariance matrix
    cov_matrix = np.cov(X_shift, rowvar=False)
    # Singular Value Decomposition to extract eigenvalues and normalized eigenvectors
    eig_val_U, eig_vect_U = np.linalg.eigh(cov_matrix)

    # eigenvalues and eigenvectors sorted in descending order
    sorted_indices = np.argsort(eig_val_U)[::-1]
    eig_val_U_sorted = eig_val_U[sorted_indices]
    eig_vect_U_sorted = eig_vect_U[:, sorted_indices]

    # Reduced U to the corresponding p eigenvectors
    # eig_val_U_reduced = eig_val_U_sorted[:reducedDim]
    eig_vect_U_reduced = eig_vect_U_sorted[:, :reducedDim]

    S = np.dot(X_shift, eig_vect_U_reduced)

    return S
```

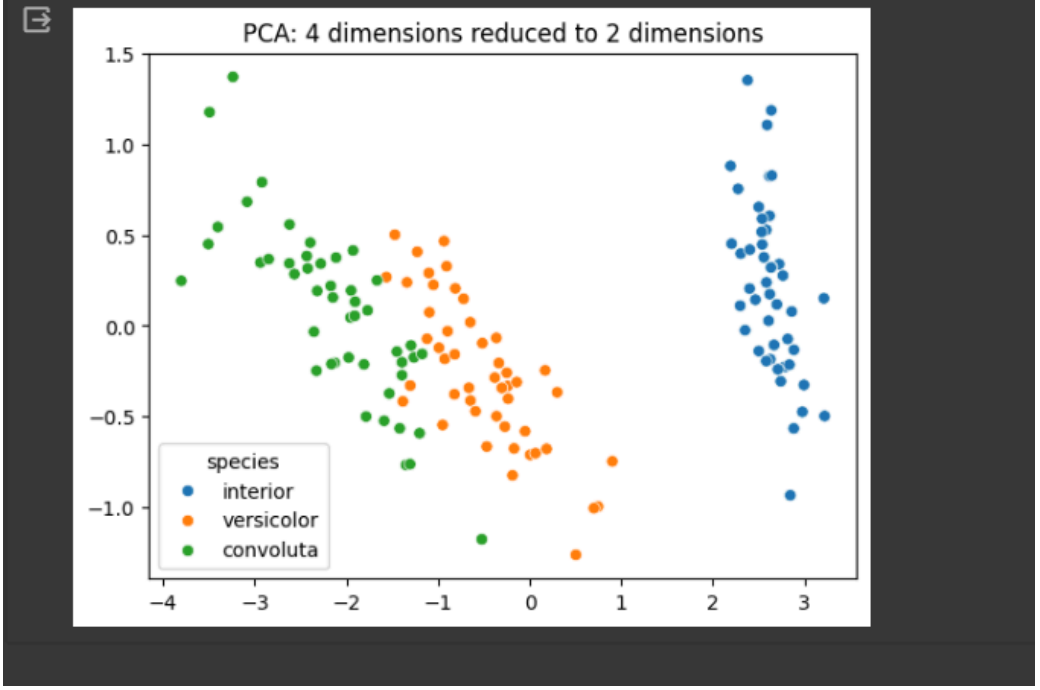
```python
# Calling the PCA Function
x1 = np.array([[1, 5, 3, 3], [4, 4, 3, 5]])
x2 = np.array([[1.268, 4.732, 3.5, 2.5], [3, 5, 3.134, 4.866]])
reducedDim = 2
print('Example 1 :\n',PCA_func(x1, reducedDim))
print('Example 2 :\n',PCA_func(x2, reducedDim))
```

```
Example 1 :
 [[-1.87082869e+00  1.11022302e-16]
 [ 1.87082869e+00 -1.11022302e-16]]
Example 2 :
 [[-1.48353969e+00 -7.34402892e-17]
 [ 1.48353969e+00 -3.70648921e-16]]
```

2)Applying PCA to the flower dataset:

I used the PCA_func(X, reducedDim) function on the flower dataset and generated a two-dimensional plot of the samples. This plot was created using the scatterplot function from the seaborn package. In this visualization, the clusters representing samples from the three species are significantly more distinct compared to the original four-dimensional space. Notably, there is a noticeable segregation between the interior species and the other two species.

```
reducedDim = 2
S = PCA_func(X, reducedDim)
sb.scatterplot(x=S[:, 0], y=S[:, 1], hue=dataset['species'])
plt.title(f'PCA: {X.shape[1]} dimensions reduced to {reducedDim} dimensions')
plt.show()
```

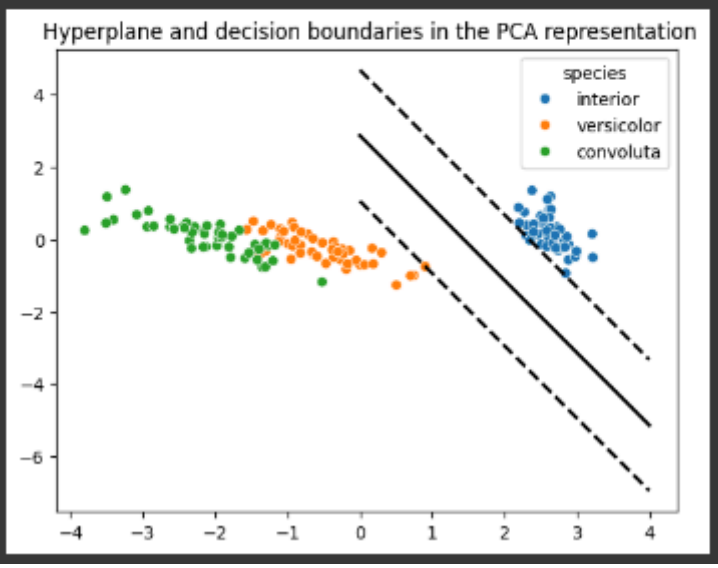


- ## III. Support Vector Machine :

In this section, we designed a linear classifier (support vector machine) for determining whether a sample corresponds to an interior flower or not. First we initialize a Support Vector Machine (SVM) classifier object. The SVC function from the scikit-learn library is used for this purpose. Inside the function, the kernel parameter is set to 'linear', indicating that a linear kernel will be used for classification. A linear kernel computes the decision boundary as a straight line in the feature space. Additionally, the C parameter is set to a very large value, specifically 1E10 (which is equivalent to 10 raised to the power of 10). This parameter controls the regularization strength of the SVM, where a larger value of C indicates less regularization, potentially leading to a more complex decision boundary that closely fits the training data.

```
[11] svm = SVC(kernel ='linear', C=1E10)
```

```
#Assigning the target feature
Target = dataset['species']

# PCA_Function to reduce the data to 2 dimensions
X_pca = PCA_func(X, reducedDim=2)

df0 = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
df1 = pd.DataFrame(Target)
df2 = pd.concat([df0, df1], axis = 1)


label=np.where(Target == 'interior',1,0)
# Training an SVM on the PCA-transformed data
svm.fit(df0, label)
w = svm.coef_[0]
a=-w[0]/w[1]
x=np.linspace(0,4,3)
y = a*x - (svm.intercept_[0]/w[1])
b = svm.support_vectors_[0]
y_down = a*x + (b[1] - a*b[0])
b = svm.support_vectors_[-1]
y_up = a * x + (b[1] - a*b[0])


plt.plot(x,y,linewidth = 2, color='black')
plt.plot(x,y_down,linewidth = 2, color='black', linestyle= '--')
plt.plot(x,y_up,linewidth = 2, color='black', linestyle='--')



# Plotting the data points
plt.title(' Hyperplane and decision boundaries in the PCA representation')
sb.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=dataset['species'])
plt.show()
```



- ## IV. Classifying new samples :

Once the flower classifier is constructed using the training data, we utilize it to evaluate new samples and ascertain whether they belong to the interior flower category or not. The predictions will be in the form of an array containing values between 0 and 1, where a value of 1 signifies an interior flower, while 0 indicates otherwise.

```
New_sample = np.array([[5.1, 3.5, 1.4, 0.2], [7.0, 3.2, 4.7, 1.4], [6.4, 3.2, 4.5, 1.5],
                       [6.3, 3.3, 6.0, 2.5], [5.8, 2.7, 5.1, 1.9], [4.9, 3.0, 1.4, 0.2]])
# Calling the PCA Function
reducedDim = 2
New_sample_pca = PCA_func(New_sample, reducedDim)
New_sample_df = pd.DataFrame(New_sample_pca, columns=['PC1', 'PC2'])
predictions = svm.predict(New_sample_df)
```

```
[16] predictions #1 means interior flower, 0 means not interior flower
```

```
array([1, 0, 0, 0, 0, 1])
```

To validate the previous work and results, I employed scikit-learn to train and evaluate a Support Vector Machine (SVM) classifier for flower species classification without using PCA_func(). Initially, the dataset is split into training and testing sets, and an SVM model is initialized and trained on the training data. Predictions are made on the test set to evaluate the model's accuracy. Additionally, I used the same new samples to predict the species using the trained SVM classifier. At the end we get the same results where only the first and last samples are interior.

```
[21] from sklearn.model_selection import train_test_split
     from sklearn.metrics import accuracy_score


     X_data = dataset.drop('species', axis=1)
     y_data = dataset['species']

     # Split the dataset into training and testing sets
     X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.2, random_state=5)

     model = SVC(kernel ='linear', C=1E10)

     # Fit the classifier to the training data
     model.fit(X_train, y_train)

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

     # Evaluate the accuracy
     accuracy = accuracy_score(y_test, y_pred)
     print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.97
```

```
# Features for new samples
new_samples = np.array([[5.1, 3.5, 1.4, 0.2], [7.0, 3.2, 4.7, 1.4], [6.4, 3.2, 4.5, 1.5],
                        [6.3, 3.3, 6.0, 2.5], [5.8, 2.7, 5.1, 1.9], [4.9, 3.0, 1.4, 0.2]])

# Feature names
feature_names = ['sepal length', 'sepal width', 'petal length', 'petal width']

# Create a DataFrame with feature names
new_samples_df = pd.DataFrame(new_samples, columns=feature_names)

print("New Samples DataFrame:")
print(new_samples_df)
```

```
New Samples DataFrame:
   sepal length  sepal width  petal length  petal width
0           5.1          3.5           1.4          0.2
1           7.0          3.2           4.7          1.4
2           6.4          3.2           4.5          1.5
3           6.3          3.3           6.0          2.5
4           5.8          2.7           5.1          1.9
5           4.9          3.0           1.4          0.2
```

```
[ ] predictions = model.predict(new_samples_df)
```

```
[ ] predictions
```

```
array(['interior', 'versicolor', 'versicolor', 'convoluta', 'convoluta',
       'interior'], dtype=object)
```

V. Conclusion :

In summary, this lab work has demonstrated the effective application of machine learning techniques, including PCA and SVM, for flower species classification. Through Python libraries such as scikit-learn and seaborn, we explored dimensionality reduction and supervised classification tasks. By leveraging labeled datasets and advanced algorithms, we successfully trained and evaluated a SVM classifier to distinguish between different flower species based on morphological features.