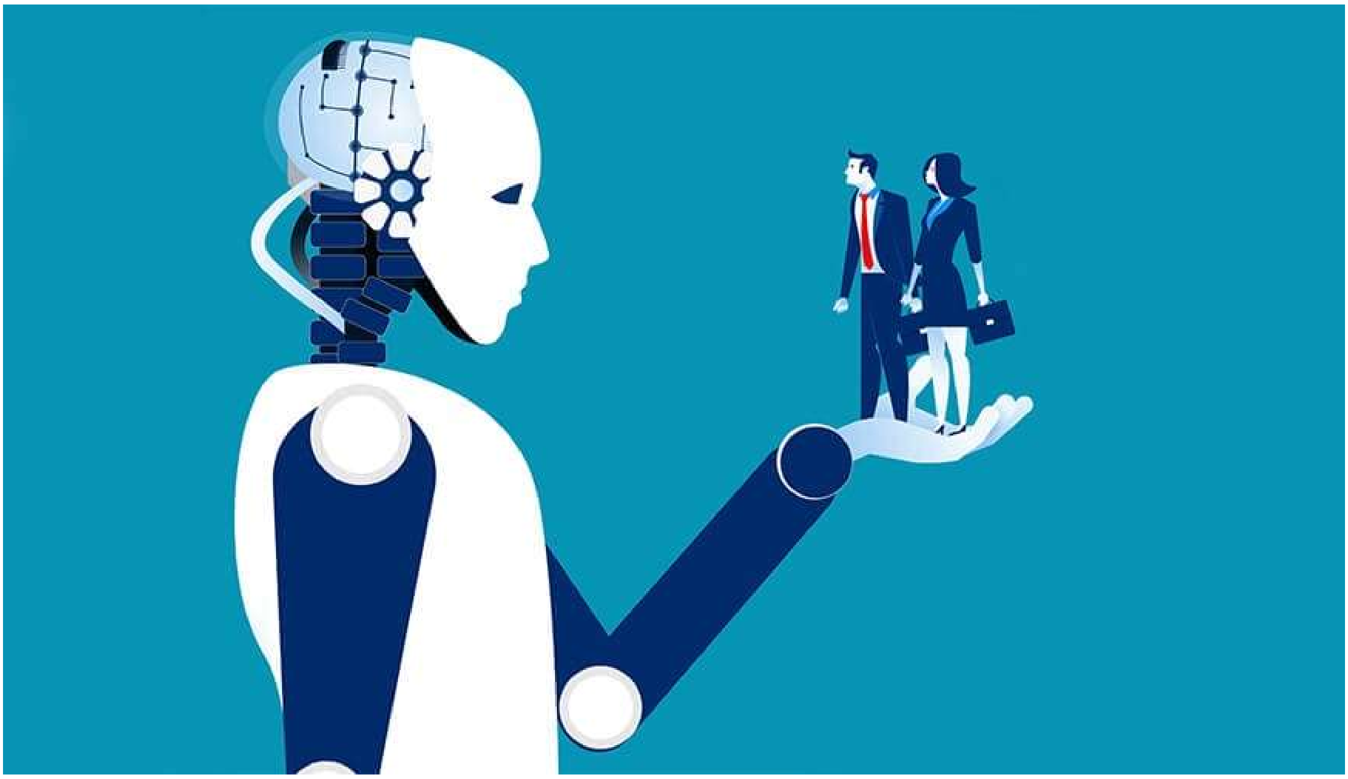
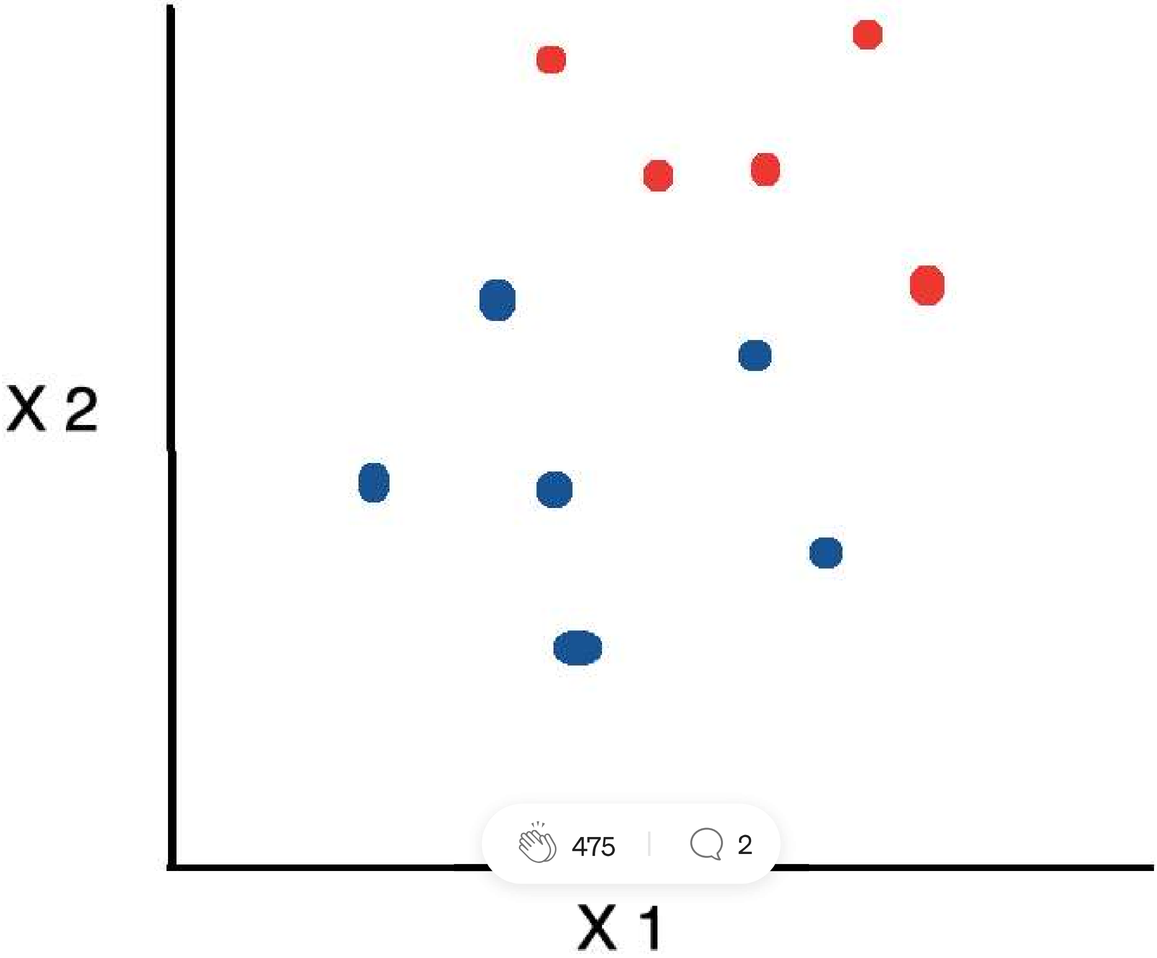
Published in Towards Data Science

Cory Maklin Follow
Aug 4, 2019 · 7 min read ★ · Listen

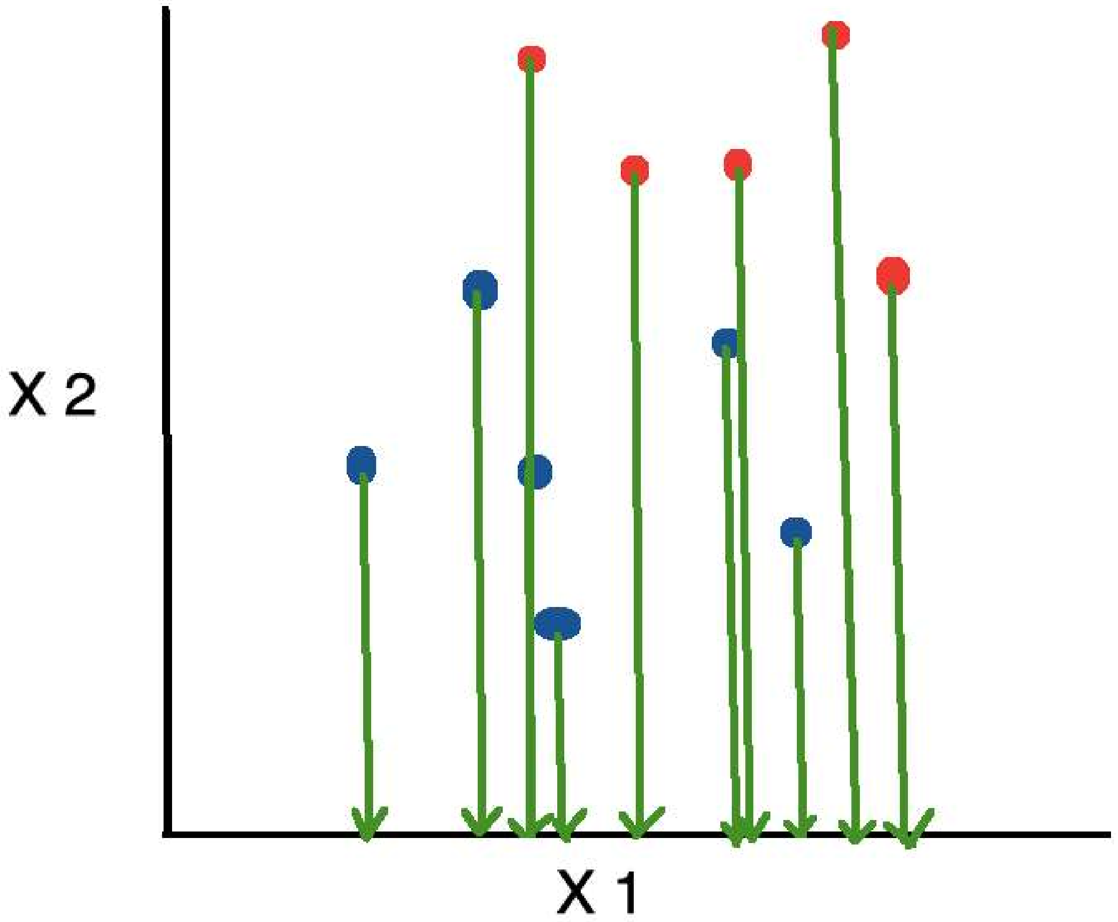


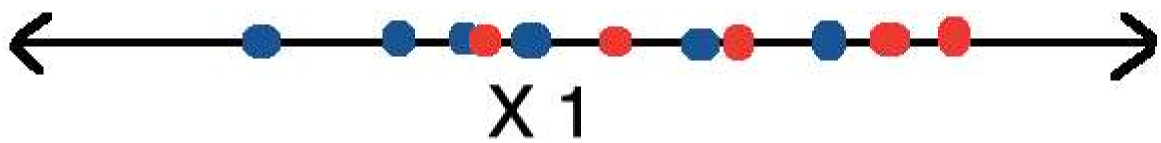
Linear Discriminant Analysis In Python

Linear Discriminant Analysis (LDA) is a dimensionality reduction technique. As the name implies dimensionality reduction techniques reduce the number of dimensions (i.e. variables) in a dataset while retaining as much information as possible. For instance, suppose that we plotted the relationship between two variables where each color represent a different class.

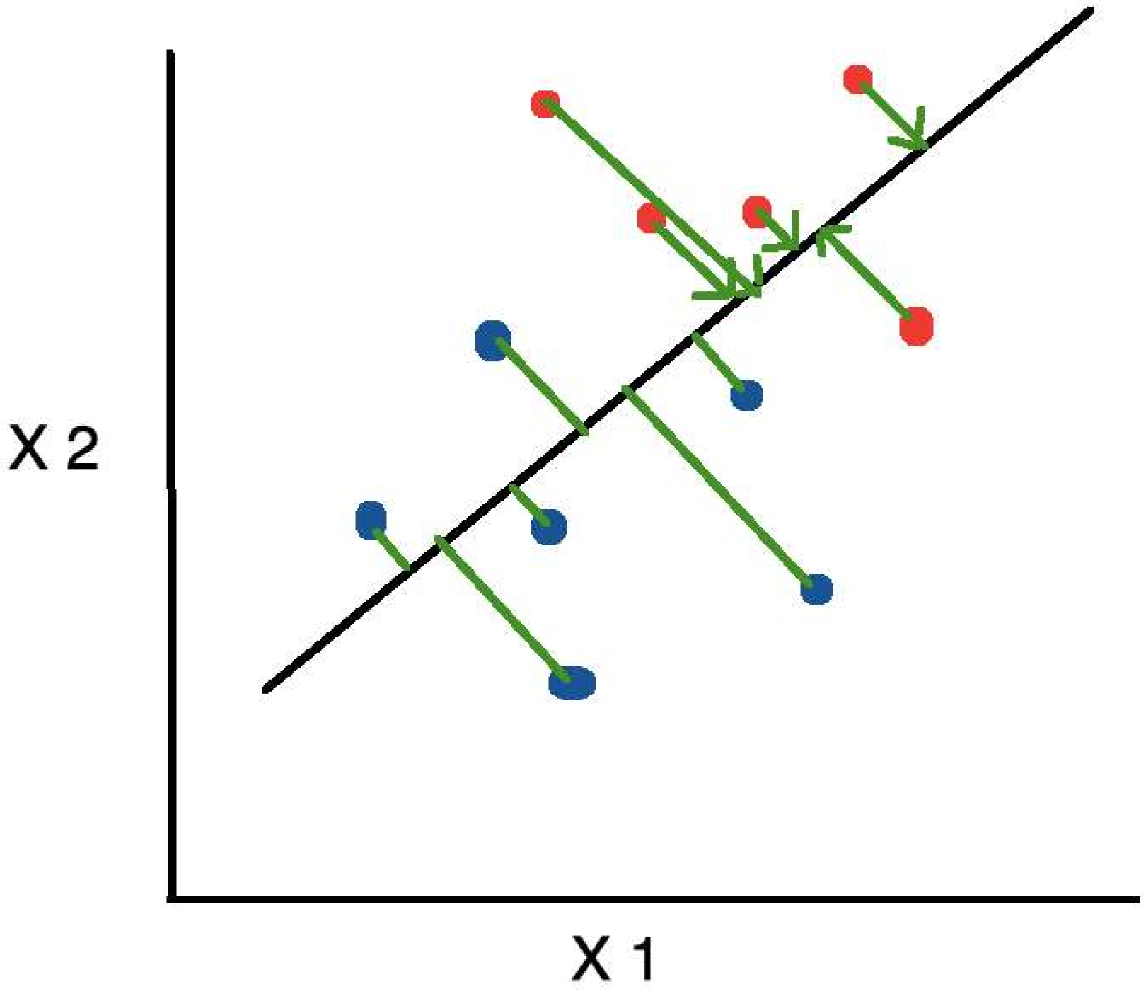


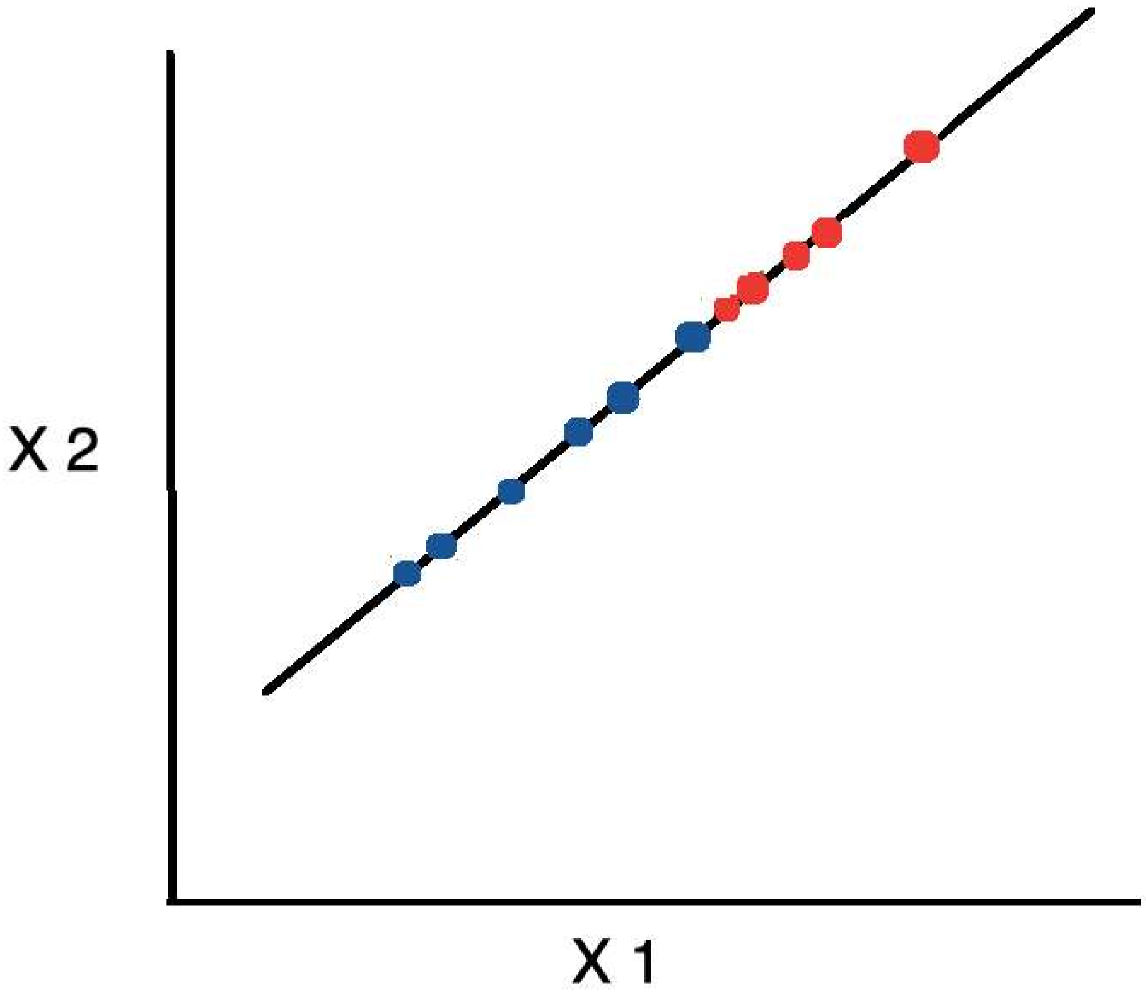
If we'd like to reduce the number of dimensions down to 1, one approach would be to project everything on to the x-axis.

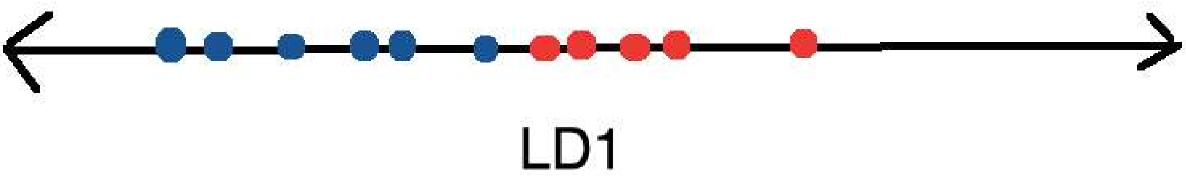




This is bad because it disregards any useful information provided by the second feature. On the other hand, Linear Discriminant Analysis, or LDA, uses the information from both features to create a new axis and projects the data on to the new axis in such a way as to minimizes the variance and maximizes the distance between the means of the two classes.


X 2
X 1





Code

Let’s see how we could go about implementing Linear Discriminant Analysis from scratch using Python. To start, import the following libraries.

```
np.set_printoptions(precision=4)
from matplotlib import pyplot as plt
import seaborn as sns
sns.set()
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

In the proceeding tutorial, we'll be working with the wine dataset which can be obtained from the UCI machine learning repository. Fortunately, the `scitkit-learn` library provides a wrapper function for downloading and

```
wine = load_wine()

X = pd.DataFrame(wine.data, columns=wine.feature_names)
y = pd.Categorical.from_codes(wine.target, wine.target_names)
```

The dataset contains 178 rows of 13 columns each.

```
X.shape
```

```
(178, 13)
```

The features are composed of various characteristics such as the magnesium and alcohol content of the wine.

```
X.head()
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0

```
wine.target_names
```

```
array(['class_0', 'class_1', 'class_2'], dtype='<U7')
```

We create a DataFrame containing both the features and classes.

```
df = X.join(pd.Series(y, name='class'))
```

Linear Discriminant Analysis can be broken up into the following steps:

1. Compute the within class and between class scatter matrices
2. Compute the eigenvectors and corresponding eigenvalues for the scatter matrices
3. Sort the eigenvalues and select the top ***k***
4. Create a new matrix containing eigenvectors that map to the ***k*** eigenvalues
5. Obtain the new features (i.e. LDA components) by taking the dot product of the data and the matrix from step 4

Within Class Scatter Matrix

We calculate the *within class scatter matrix* using the following formula.

$$S_W = \sum_{i=1}^{c} S_i$$

where ***c*** is the total number of distinct classes and

$$S_i = \sum_{\boldsymbol{x} \in D_i}^{n} (\boldsymbol{x} - \boldsymbol{m}_i)\,(\boldsymbol{x} - \boldsymbol{m}_i)^T$$

where ***x*** is a sample (i.e. row) and ***n*** is the total number of samples with a given class.

For every class, we create a vector with the means of each feature.

```
class_feature_means = pd.DataFrame(columns=wine.target_names)

for c, rows in df.groupby('class'):
    class_feature_means[c] = rows.mean()

class_feature_means
```

	class_0	class_1	class_2
alcohol	13.744746	12.278732	13.153750
malic_acid	2.010678	1.932676	3.333750
ash	2.455593	2.244789	2.437083
alcalinity_of_ash	17.037288	20.238028	21.416667
magnesium	106.338983	94.549296	99.312500
total_phenols	2.840169	2.258873	1.678750
flavanoids	2.982373	2.080845	0.781458
nonflavanoid_phenols	0.290000	0.363662	0.447500
proanthocyanins	1.899322	1.630282	1.153542
color_intensity	5.528305	3.086620	7.396250
hue	1.062034	1.056282	0.682708
od280/od315_of_diluted_wines	3.157797	2.785352	1.683542
proline	1115.711864	519.507042	629.895833

Then, we plug the mean vectors (***mi***) into the equation from before in order to obtain the within class scatter matrix.

```
within_class_scatter_matrix = np.zeros((13,13))

for c, rows in df.groupby('class'):

rows = rows.drop(['class'], axis=1)
```

Open in app Get started

```
        s += (x - mc).dot((x - mc).T)

    within_class_scatter_matrix += s
```

Between Class Scatter Matrix

Next, we calculate the *between class scatter matrix* using the following formula.

$$S_B = \sum_{i=1}^{c} N_i(\boldsymbol{m}_i - \boldsymbol{m})(\boldsymbol{m}_i - \boldsymbol{m})^T$$

where

$$\boldsymbol{m}_i = \frac{1}{n_i} \sum_{\boldsymbol{x} \in D_i}^{n} \boldsymbol{x}_k$$

$$m = \frac{1}{n} \sum_{i}^{n} x_i$$

```
feature_means = df.mean()

between_class_scatter_matrix = np.zeros((13,13))

for c in class_feature_means:
    n = len(df.loc[df['class'] == c].index)

    mc, m = class_feature_means[c].values.reshape(13,1),
feature_means.values.reshape(13,1)

    between_class_scatter_matrix += n * (mc - m).dot((mc - m).T)
```

Then, we solve the generalized eigenvalue problem for to obtain the linear discriminants.

```
eigen_values, eigen_vectors =
np.linalg.eig(np.linalg.inv(within_class_scatter_matrix).dot(between
_class_scatter_matrix))
```

The eigenvectors with the highest eigenvalues carry the most information about the distribution of the data. Thus, we sort the eigenvalues from highest to lowest and select the first ***k*** eigenvectors. In order to ensure that the eigenvalue maps to the same eigenvector after sorting, we place them in a temporary array.

```
pairs = [(np.abs(eigen_values[i]), eigen_vectors[:,i]) for i in
range(len(eigen_values))]

pairs = sorted(pairs, key=lambda x: x[0], reverse=True)

for pair in pairs:
    print(pair[0])
```

```
9.081739435042472
4.128469045639484
8.881784197001252e-16
7.41949604398113e-16
7.41949604398113e-16
6.57104310784389e-16
6.57104310784389e-16
2.9039090283069212e-16
2.9039090283069212e-16
2.58525572226227e-16
6.126103277916086e-17
6.126103277916086e-17
4.86945776983596e-17
```

Just looking at the values, it's difficult to determine how much of the variance is explained by each component. Thus, we express it as a percentage.

```
eigen_value_sums = sum(eigen_values)
```

```
Explained Variance
Eigenvector 0: 0.6874788878860784
Eigenvector 1: 0.31252111211392164
Eigenvector 2: 6.723424698398662e-17
Eigenvector 3: 5.616486715430028e-17
Eigenvector 4: 5.616486715430028e-17
Eigenvector 5: 4.9742160522698054e-17
Eigenvector 6: 4.9742160522698054e-17
Eigenvector 7: 2.1982310366664338e-17
Eigenvector 8: 2.1982310366664338e-17
Eigenvector 9: 1.957013567229342e-17
Eigenvector 10: 4.637400906181487e-18
Eigenvector 11: 4.637400906181487e-18
Eigenvector 12: 3.686132415666904e-18
```

First, we create a matrix ***W*** with the first two eigenvectors.

```
w_matrix = np.hstack((pairs[0][1].reshape(13,1), pairs[1]
[1].reshape(13,1))).real
```

Then, we save the dot product of ***X*** and ***W*** into a new matrix ***Y***.

$$Y = X \cdot W$$

where ***X*** is a ***n×d*** matrix with ***n*** samples and ***d*** dimensions, and ***Y*** is a ***n×k*** matrix with ***n*** samples and ***k*** (k<n) dimensions. In other words, ***Y*** is composed of the LDA components, or said yet another way, the new feature space.

```
X_lda = np.array(X.dot(w_matrix))
```

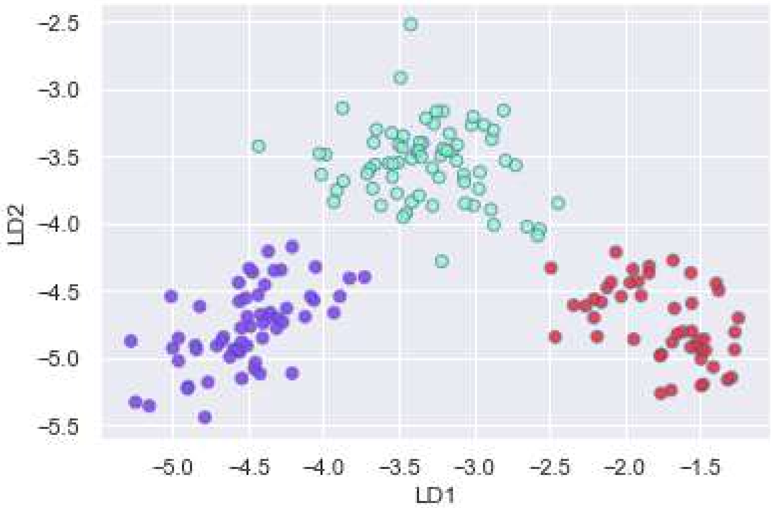
`matplotlib` can't handle categorical variables directly. Thus, we encode every class as a number so that we can incorporate the class labels into our plot.

```
le = LabelEncoder()

y = le.fit_transform(df['class'])
```

```
plt.xlabel('LD1')
plt.ylabel('LD2')

plt.scatter(
    X_lda[:,0],
    X_lda[:,1],
    c=y,
    cmap='rainbow',
    alpha=0.7,
    edgecolors='b'
)
```



Rather than implementing the Linear Discriminant Analysis algorithm from scratch every time, we can use the predefined `LinearDiscriminantAnalysis` class made available to us by the `scikit-learn` library.

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

lda = LinearDiscriminantAnalysis()

X_lda = lda.fit_transform(X, y)
```

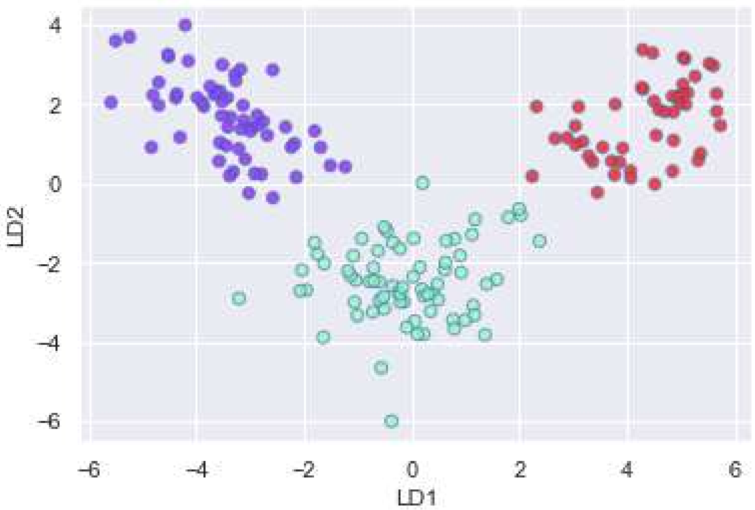
We can access the following property to obtain the variance explained by each component.

```
array([0.6875, 0.3125])
```

Just like before, we plot the two LDA components.

```
plt.xlabel('LD1')
plt.ylabel('LD2')

plt.scatter(
    X_lda[:,0],
    X_lda[:,1],
    c=y,
    cmap='rainbow',
    alpha=0.7,
    edgecolors='b'
)
```



Next, let's take a look at how LDA compares to Principal Component Analysis or PCA. We start off by creating and fitting an instance of the `PCA` class.

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)

X_pca = pca.fit_transform(X, y)
```

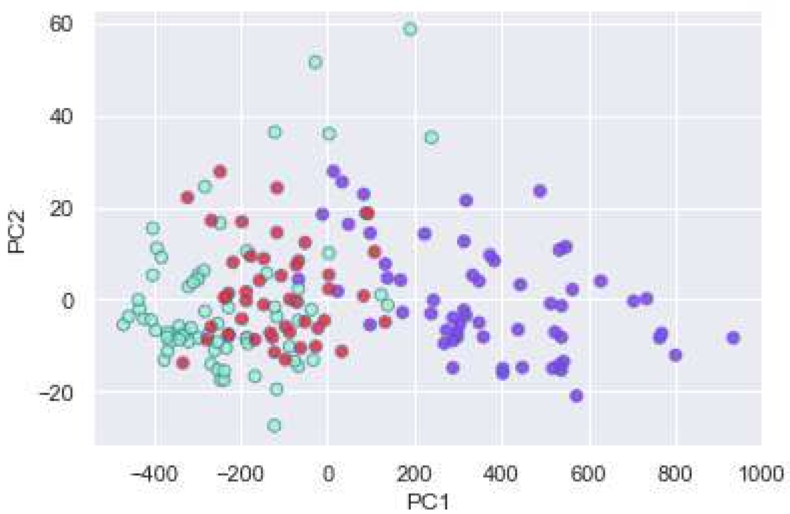
```
pca.explained_variance_ratio_
```

```
array([0.9981, 0.0017])
```

As we can see, PCA selected the components which would result in the highest spread (retain the most information) and not necessarily the ones which maximize the separation between classes.

```
plt.xlabel('PC1')
plt.ylabel('PC2')

plt.scatter(
    X_pca[:,0],
    X_pca[:,1],
    c=y,
    cmap='rainbow',
    alpha=0.7,
    edgecolors='b'
)
```



Next, let's see whether we can create a model to classify the using the LDA components as features. First, we split the data into training and testing sets.

Then, we build and train a Decision Tree. After predicting the category of each sample in the test set, we create a confusion matrix to evaluate the model's performance.

```
dt = DecisionTreeClassifier()

dt.fit(X_train, y_train)

y_pred = dt.predict(X_test)

confusion_matrix(y_test, y_pred)
```

```
array([[18,  0,  0],
       [ 0, 17,  0],
       [ 0,  0, 10]])
```

As we can see, the Decision Tree classifier correctly classified everything in the test set.