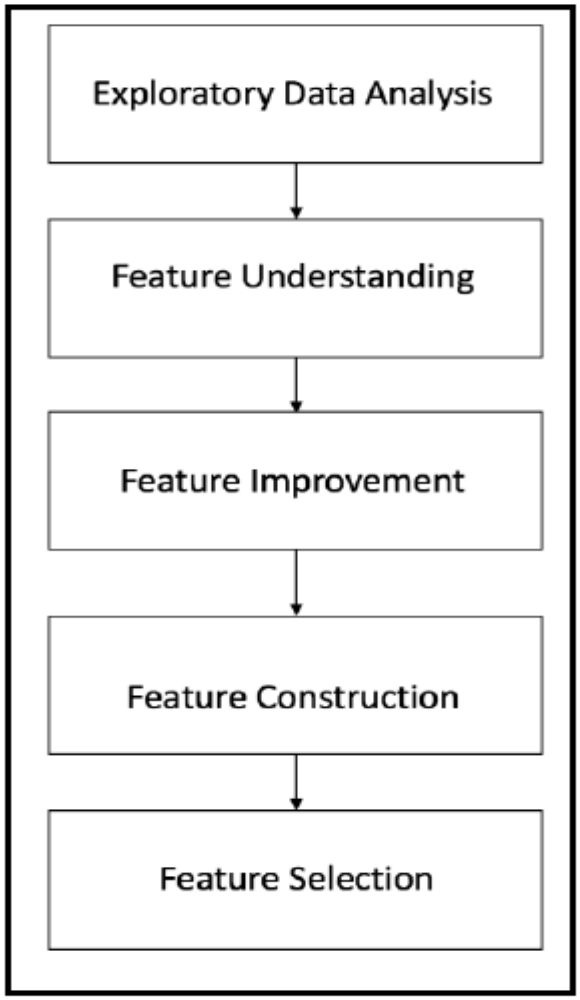
Feature Transformations

Feature transformation

- Feature transformation is simply a function that transforms features from one representation to another.
- But why would we transform our features? Well there are many reasons, such as:
  1. data types are not suitable to be fed into a machine learning algorithm, e.g. text, categories
  2. feature values may cause problems during the learning process, e.g. data represented in different scales
  3. we want to reduce the number of features to plot and visualize data, speed up training or improve the accuracy of a specific model

Feature transformation

Machine learning pipeline



Feature transformation

- **Feature Transformations**, a suite of algorithms designed to alter the internal structure of data to produce mathematically superior *super-columns*.
- Feature transformations are a set of matrix algorithms that will structurally alter our data and produce what is essentially a brand new matrix of data.
- **Feature Transformations** create a new set of features that explain the data-points, perhaps even better, with fewer columns.

Feature transformation

Dimension reduction – feature transformations versus feature selection versus feature construction

- The toughest part of feature transformations is the suspension of our belief that original feature space is the best.
- Must be open to the fact that there may be other mathematical axes and systems that describe our data just as well with fewer features, or possibly even better.
- Could squish datasets to have fewer columns to describe data in new ways.
- Feature selection processes are limited to only being able to select features from the original set of columns.
- Feature construction is again limited to constructing new columns using simple operations (addition, multiplication, and so on) between a few columns at a time.
- While feature transformation algorithms use these original columns and combine them in useful ways to create new columns that are better at describing the data than any single column from the original dataset.

Feature transformation

Dimension reduction – feature transformations versus feature selection versus feature construction

- Feature transformation methods create new columns using hidden structures in the original datasets to produce an entirely new, structurally different dataset.
- Create brand new columns that are so powerful that we only need a few of them to explain our entire dataset accurately.
- Feature Transformation works by producing new columns that capture the essence (variance) of the data.
- Utilize small bits of information from all columns in every new super-column.
- Feature transformation algorithms are able to *construct* new features by *selecting* best of all columns and combining this latent structure with a few brand new columns.

Dimensionality reduction

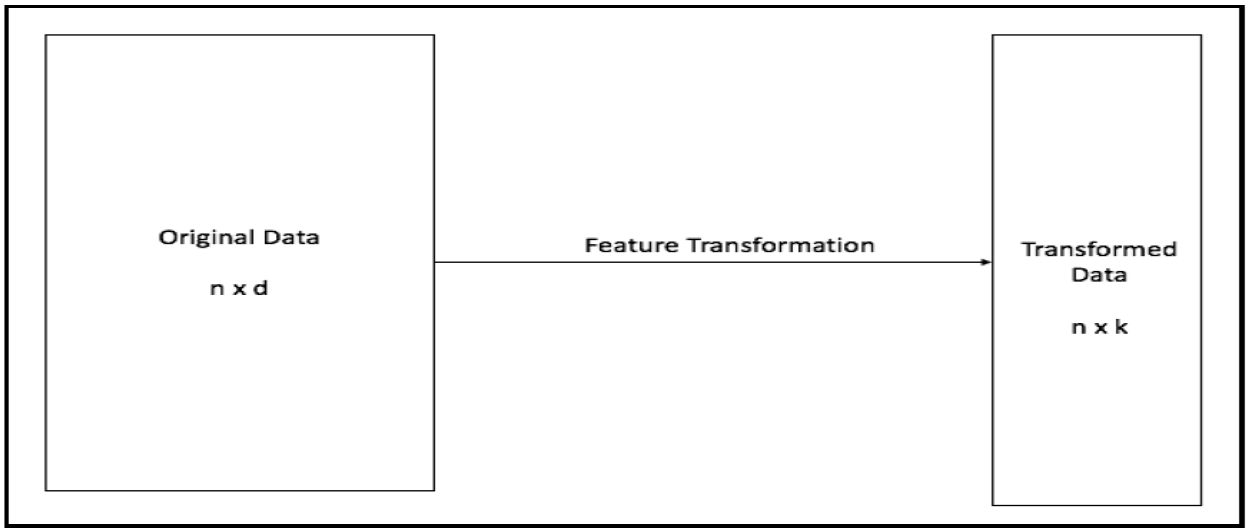
- **Dimensionality reduction**, or **dimension reduction**, is the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data.
- Working in high-dimensional spaces can be undesirable for many reasons;
  - raw data are often sparse as a consequence of the curse of dimensionality, and
  - analyzing the data is usually computationally intractable.

Principal Component Analysis

- Principal Component Analysis is a technique that takes datasets that have several correlated features and projects them onto a coordinate (axis) system that has fewer correlated features.
- These new, uncorrelated features (referred as super-columns) are called **principal components**.
- The principal components serve as an alternative coordinate system to the original feature space that requires fewer features and captures as much variance as possible.

Principal Component Analysis

- Goal of the PCA is to identify patterns and latent structures within datasets in order to create new columns and use these columns instead of original features.
- If we start with a data matrix of size *n x d* where *n* is number of observations and *d* is number of original features, we are projecting this matrix onto a matrix of **size *n* x *k* (where *k* < *d*).**
- Principal Components give rise to new columns that maximize the variance in our data
- Principal components are ordered by variance.



Principal Component Analysis

- PCA is itself an unsupervised task
- It does not utilize a response column in order to make projection/transformation.
- LDA supervised and will utilize the response variable in order to create super-columns in a different way that optimizes predictive tasks.

Need For Principal Component Analysis (PCA)

- Machine Learning in general works wonders when dataset for training is large.
- Good amount of data - better predictive model.
- However, biggest pitfall is the curse of dimensionality.
- It turns out that in large dimensional datasets,
  - lots of inconsistencies in the features or
  - lots of redundant features in the dataset,
  - which will only increase computation time and
  - make data processing and EDA more convoluted.
- To get rid of curse of dimensionality, a process called dimensionality reduction was introduced.
- Dimensionality reduction techniques can be used to filter only a limited no. of significant features and this is where PCA comes in.

What Is Principal Component Analysis (PCA)?

- Principal components analysis (PCA) is a dimensionality reduction technique that enables you to identify correlations and patterns in a data set so that it can be transformed into a data set of significantly lower dimension without loss of any important information.
- Main idea behind PCA is to figure out patterns and correlations among various features.
- On finding a strong correlation between different variables, a final decision is made about reducing dimensions of data in such a way that significant data is still retained.

Step By Step Computation Of PCA

1. Standardization of the data
2. Computing the covariance matrix
3. Calculating the eigenvectors and eigenvalues
4. Computing the Principal Components
5. Reducing the dimensions of the data set

Step 1: Standardization of the data

- Missing out on standardization will probably result in a biased outcome.
- Standardization is all about scaling data in such a way that all variables and their values lie within a similar range.
- Standardizing data into a comparable range is very important

$$Z = \frac{Variable\ value - mean}{Standard\ deviation}$$

- Post this step, all the variables in the data are scaled across a standard and comparable scale.

Step 2: Computing the covariance matrix

- PCA helps to identify correlation and dependencies among the features.
- A covariance matrix expresses the correlation between the different variables in the data set.
- It is essential to identify heavily dependent variables because they contain biased and redundant information.
- Mathematically, a covariance matrix is a p × p matrix, where p represents dimensions of the data set.
- Each entry in matrix represents covariance of corresponding variables.

Step 2: Computing the covariance matrix

- Consider a case where we have a 2-Dimensional data set with variables a and b, the covariance matrix is a 2×2 matrix as shown below:

$$\begin{bmatrix} Cov(a, a) & Cov(a, b) \\ Cov(b, a) & Cov(b, b) \end{bmatrix}$$

- In the above matrix:
  - Cov(a, a) represents the covariance of a variable with itself, which is nothing but the variance of the variable 'a'
  - Cov(a, b) represents the covariance of the variable 'a' with respect to the variable 'b'.
  - And since covariance is commutative, Cov(a, b) = Cov(b, a)

Here are key takeaways from the covariance matrix:

- The covariance value denotes how co-dependent two variables are with respect to each other
- If the covariance value is negative, it denotes the respective variables are indirectly proportional to each other
- A positive covariance denotes that the respective variables are directly proportional to each other

Step 3: Calculating the Eigenvectors and Eigenvalues

- Eigenvectors and eigenvalues are the mathematical constructs that must be computed from the covariance matrix in order to determine the principal components of the data set.
- **Eigenvalues** and **Eigenvectors** have following components:
  - The **eigenvector** is an array with n entries where n is the number of rows (or columns) of a square matrix. The **eigenvector** is represented as x.
  - **Eigenvalues is** a scalar associated with a given linear transformation of a vector space

What are Principal Components?

- **Principal Components** are the new set of variables that are obtained from the initial set of variables.
- The principal components are computed in such a manner that newly obtained variables are highly significant and independent of each other.
- The principal components compress and possess most of the useful information that was scattered among the initial variables.
- *If your data set is of 5 dimensions, then 5 principal components are computed, such that, the first principal component stores the maximum possible information and the second one stores the remaining maximum info and so on, you get the idea.*

What are Principal Components?

Now, where do Eigenvectors fall into this whole process?

- Eigenvectors and eigenvalues, are two algebraic formulations are always computed as a pair, i.e, for every eigenvector there is an eigenvalue.
- The dimensions in the data determine the number of eigenvectors to be calculated.
- Consider a 2-Dimensional data set, for which 2 eigenvectors (and their respective eigenvalues) are computed.
- The idea behind eigenvectors is to use the Covariance matrix to understand **where in the data there is the most amount of variance**.
- Since more variance in the data denotes more information about the data, eigenvectors are used to identify and compute Principal Components.
- Eigenvalues*, on the other hand, simply denote the* scalars *of the respective eigenvectors.*
- *Therefore, eigenvectors and eigenvalues will compute the* Principal Components *of the data set.*

Step 4: Computing the Principal Components

- Once computed the Eigenvectors and eigenvalues, order them in the descending order, where the eigenvector with the highest eigenvalue is the most significant and thus forms the first principal component.
- The principal components of lesser significances can thus be removed in order to reduce the dimensions of the data.
- The final step in computing the Principal Components is to form a matrix known as the feature matrix that contains all the significant data variables that possess maximum information about the data.

Step 5: Reducing the dimensions of the data set

- Last step in performing PCA is to re-arrange the original data with the final principal components which represent the maximum and the most significant information.
- In order to replace original data axis with newly formed Principal Components, simply multiply transpose of the original data set by the transpose of the obtained feature vector.

How PCA works

- PCA works by invoking a process called the **eigenvalue decomposition** of the covariance of a matrix.
- The mathematics behind this was first published in 1930s and involves of multivariable calculus and linear algebra.
- PCA may also work on correlation matrix.
- This process happens in four steps:
  1. Create the covariance matrix of the dataset
  2. Calculate the eigenvalues of the covariance matrix
  3. Keep the top *k* eigenvalues (sorted by the descending eigenvalues)
  4. Use the kept eigenvectors to transform new data-points

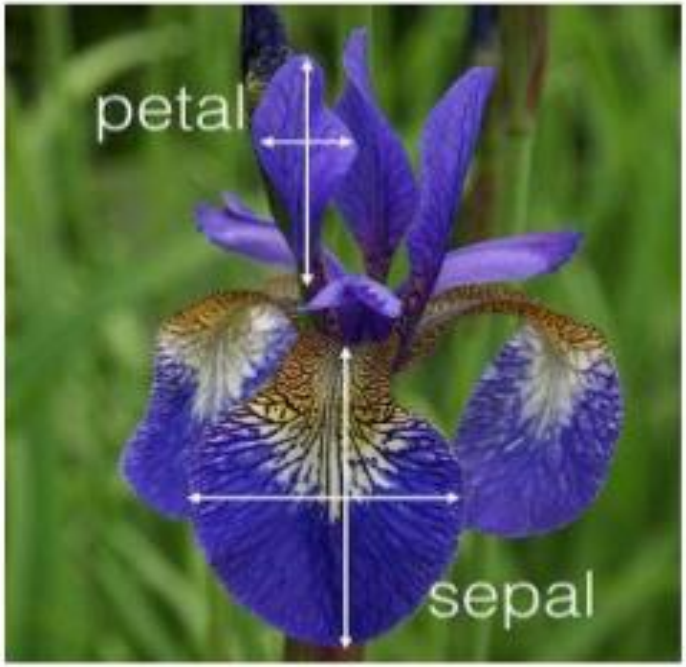
PCA with the Iris dataset

Data Set Characteristics:	Multivariate	**Number of Instances:**	150	**Area:**	Life
Attribute Characteristics:	Real	**Number of Attributes:**	4	**Date Donated**	1988-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	2331246

Attribute Information:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:

-- Iris Setosa
-- Iris Versicolour
-- Iris Virginica



PCA with the Iris dataset

https://github.com/PacktPublishing/Feature-Engineering-Made-Easy/blob/master/Chapter06/Ch_6.ipynb

Scikit-learn's PCA

1. We can import it from scikit-learn's decomposition module:

```
# scikit-learn's version of PCA
from sklearn.decomposition import PCA
```

2. To mimic the process we performed with the `iris` dataset, let's instantiate a `PCA` object with only two components:

```
# Like any other sklearn module, we first instantiate the class
pca = PCA(n_components=2)
```

3. Now, we can fit our PCA to the data:

```
# fit the PCA to our data
pca.fit(iris_X)
```

4. Let's take a look at some of the attributes of the PCA object to see if they match up with what we achieved in our manual process. Let's take a look at the `components_ attribute` of our object to see if this matches up without the `top_2_eigenvectors` variable:

```
pca.components_

array([[ 0.36158968, -0.08226889,  0.85657211,  0.35884393],
       [ 0.65653988,  0.72971237, -0.1757674 , -0.07470647]])

# note that the second column is the negative of the manual process
# this is because eignevectors can be positive or negative
# It should have little to no effect on our machine learning
pipelines
```

Scikit-learn's PCA

5. Our two components match, almost exactly, our previous variable, `top_2_eigenvectors`. We say almost because the second component is actually the negative of the eigenvector we calculated. This is fine because, mathematically, both eigenvectors are 100% valid and still achieve the primary goal of creating uncorrelated columns.
6. So far, this process is much less painful than what we were doing before. To complete the process, we need to use the transform method of the PCA object to project data onto our new two-dimensional plane:

```
pca.transform(iris_X)[:5,]

array([[-2.68420713,  0.32660731],
       [-2.71539062, -0.16955685],
       [-2.88981954, -0.13734561],
       [-2.7464372 , -0.31112432],
       [-2.72859298,  0.33392456]])

# sklearn PCA centers the data first while transforming, so these
numbers won't match our manual process.
```

Notice that our projected data does not match up with the projected data we got before at all. This is because the scikit-learn version of PCA automatically centers data in the prediction phase, which changes the outcome.

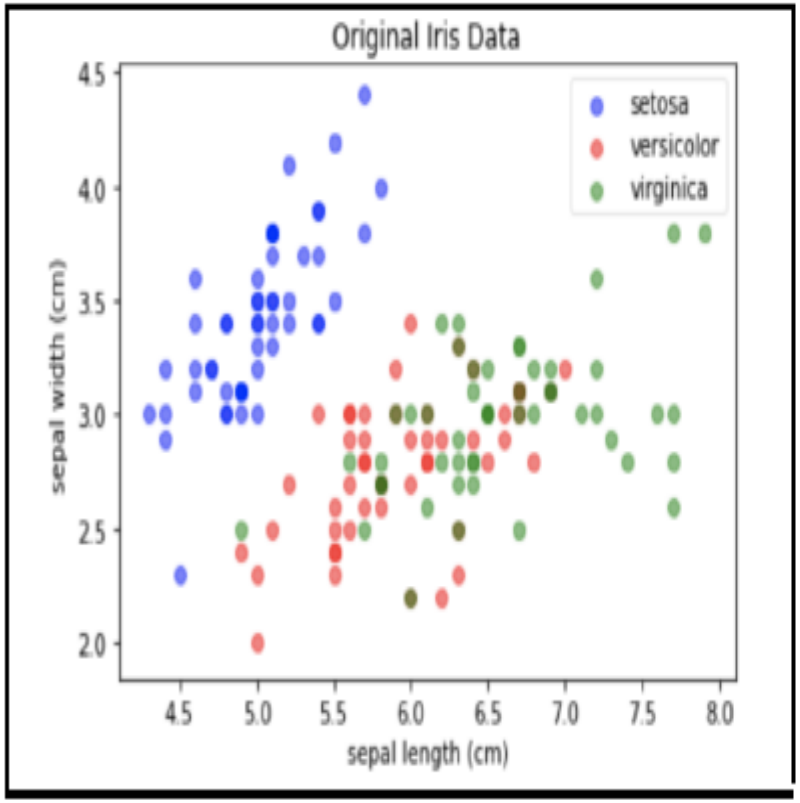
7. We can mimic this by altering a single line in our version to match:

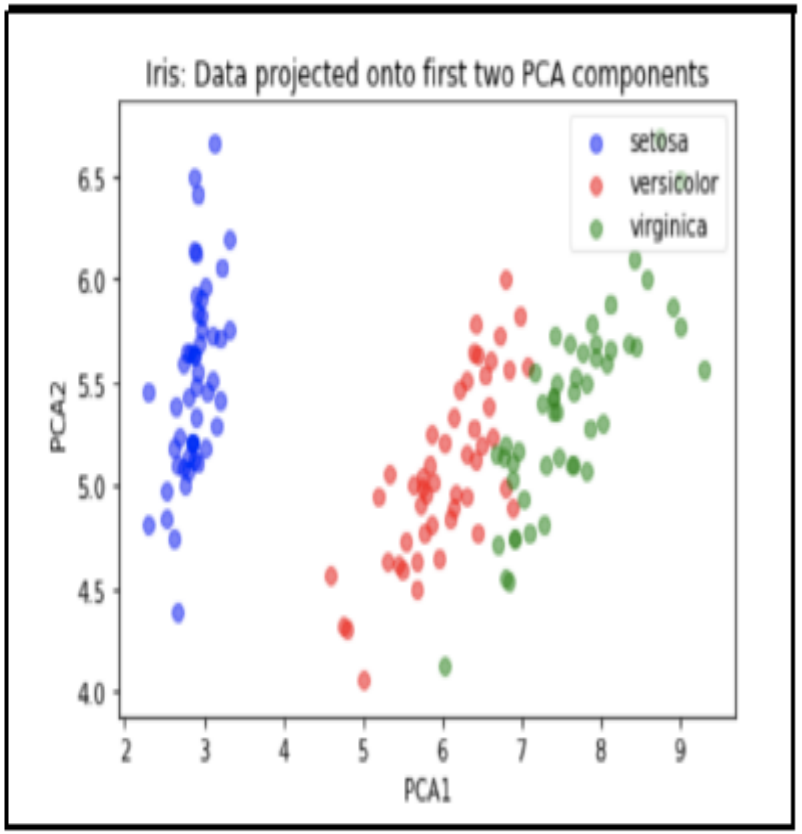
```
# manually centering our data to match scikit-learn's
implementation of PCA
np.dot(iris_X-mean_vector, top_2_eigenvectors.T)[:5,]

array([[-2.68420713, -0.32660731],
       [-2.71539062,  0.16955685],
       [-2.88981954,  0.13734561],
       [-2.7464372 ,  0.31112432],
       [-2.72859298, -0.33392456]])
```

Scikit-learn's PCA

Let's make a quick plot of projected iris data and compare what dataset looks like before and after projecting onto our new coordinate system:

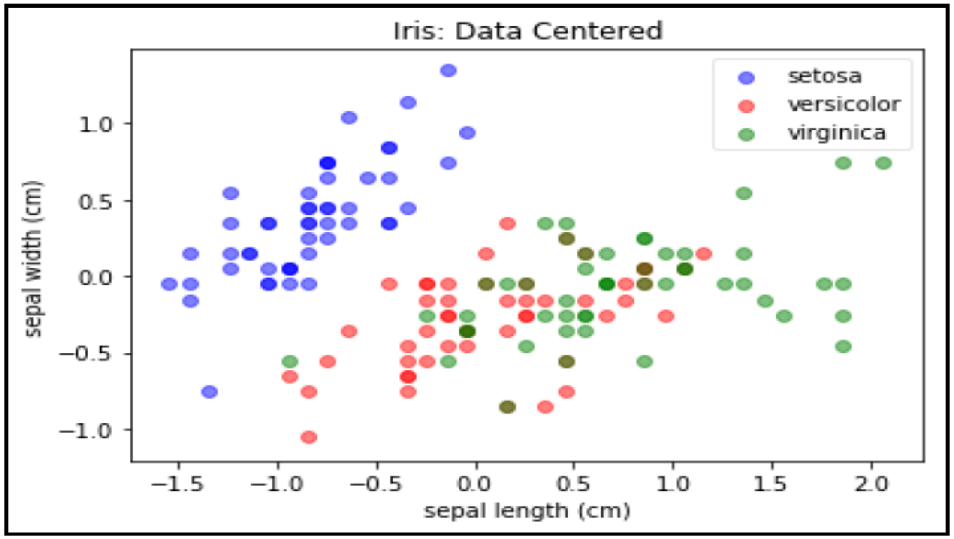


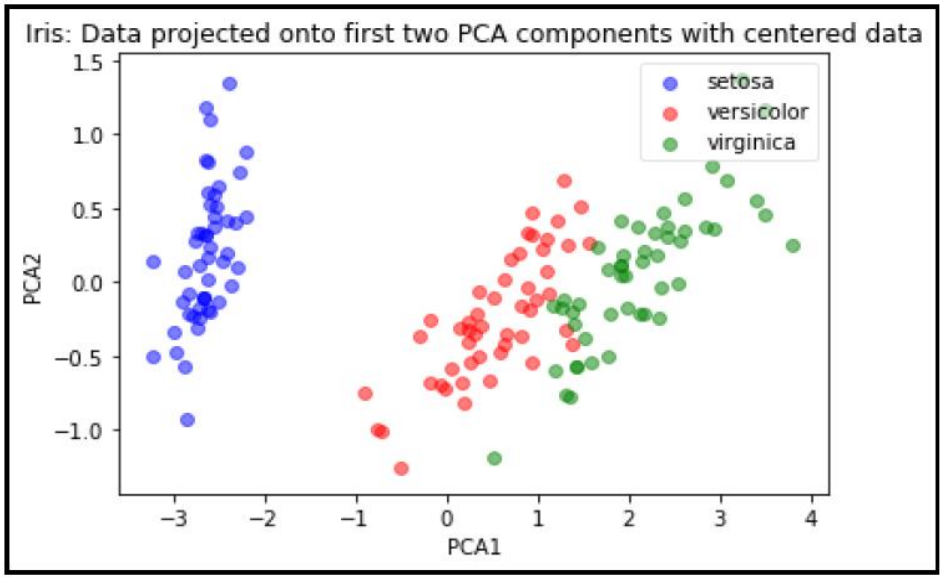


Scikit-learn's PCA

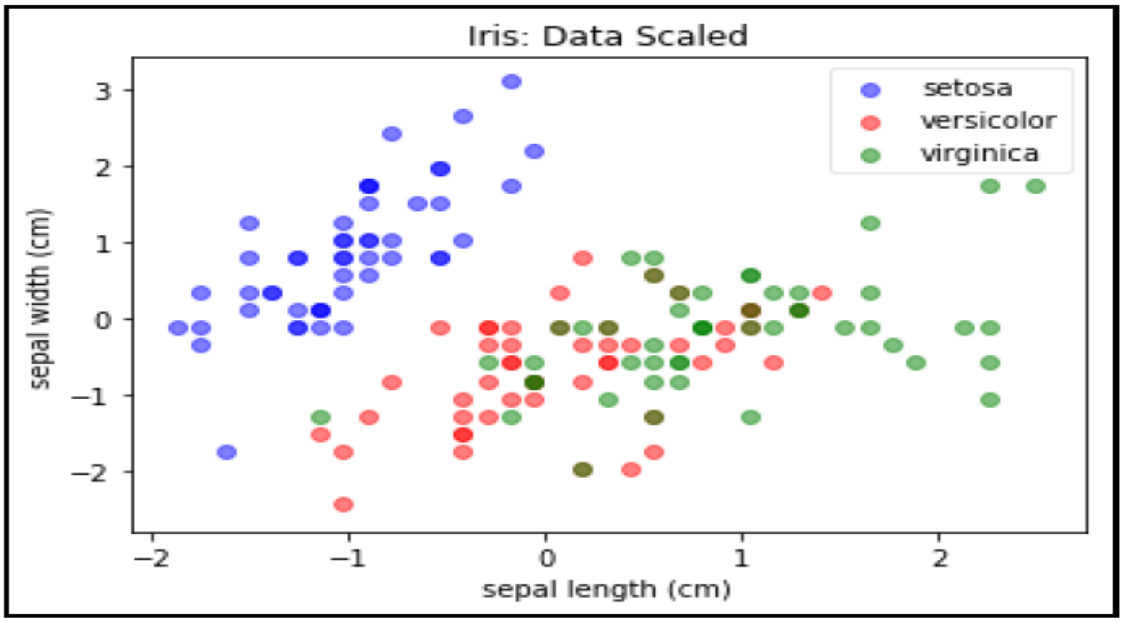
- One of the main benefits of PCA: de-correlating features.
- By nature, in the eigenvalue decomposition procedure, the resulting principal components are perpendicular to each other, meaning that they are linearly independent of one another.
- This is a major benefit because many machine learning models and preprocessing techniques make the assumption that inputted features are independent, and utilizing PCA ensures this for us.

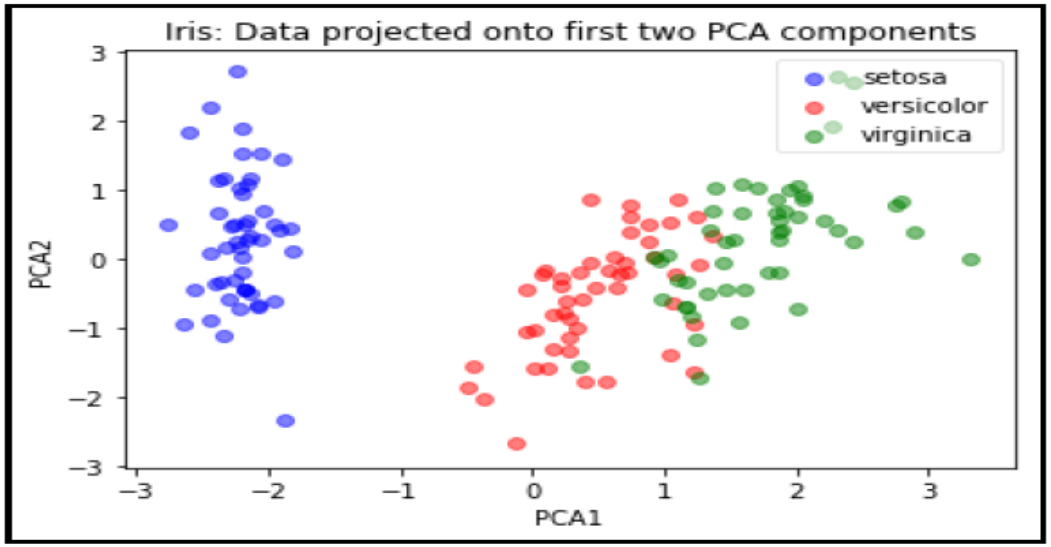
How centering and scaling data affects PCA



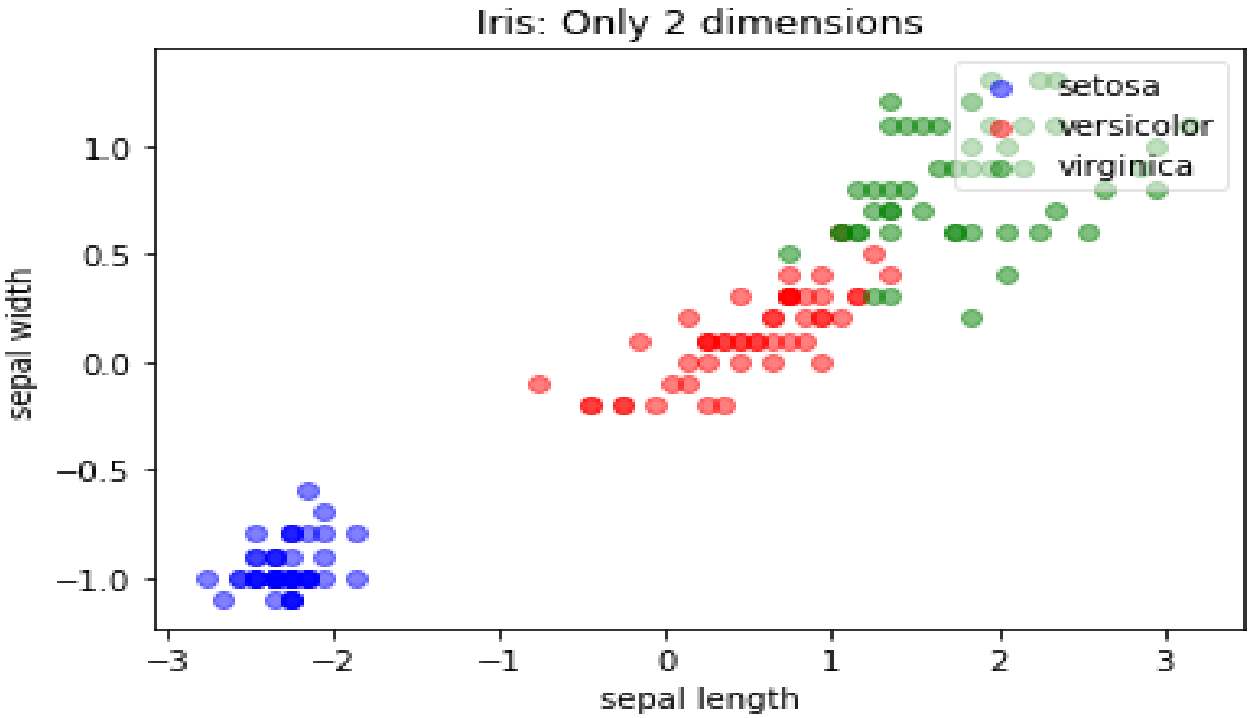


How centering and scaling data affects PCA



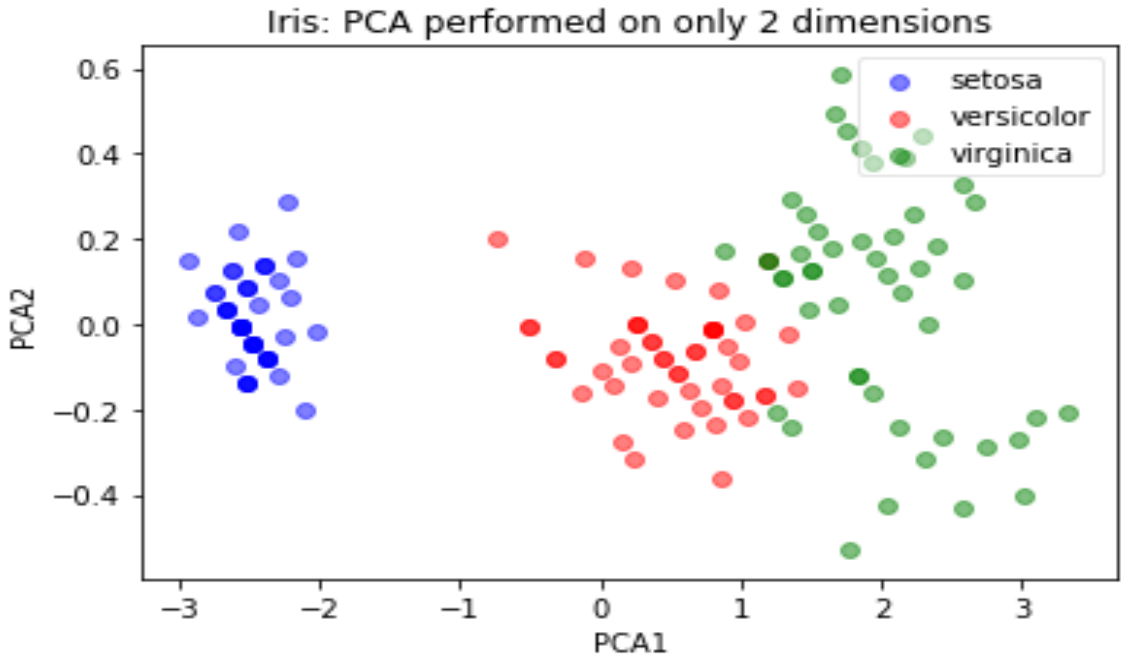


A deeper look into the principal components



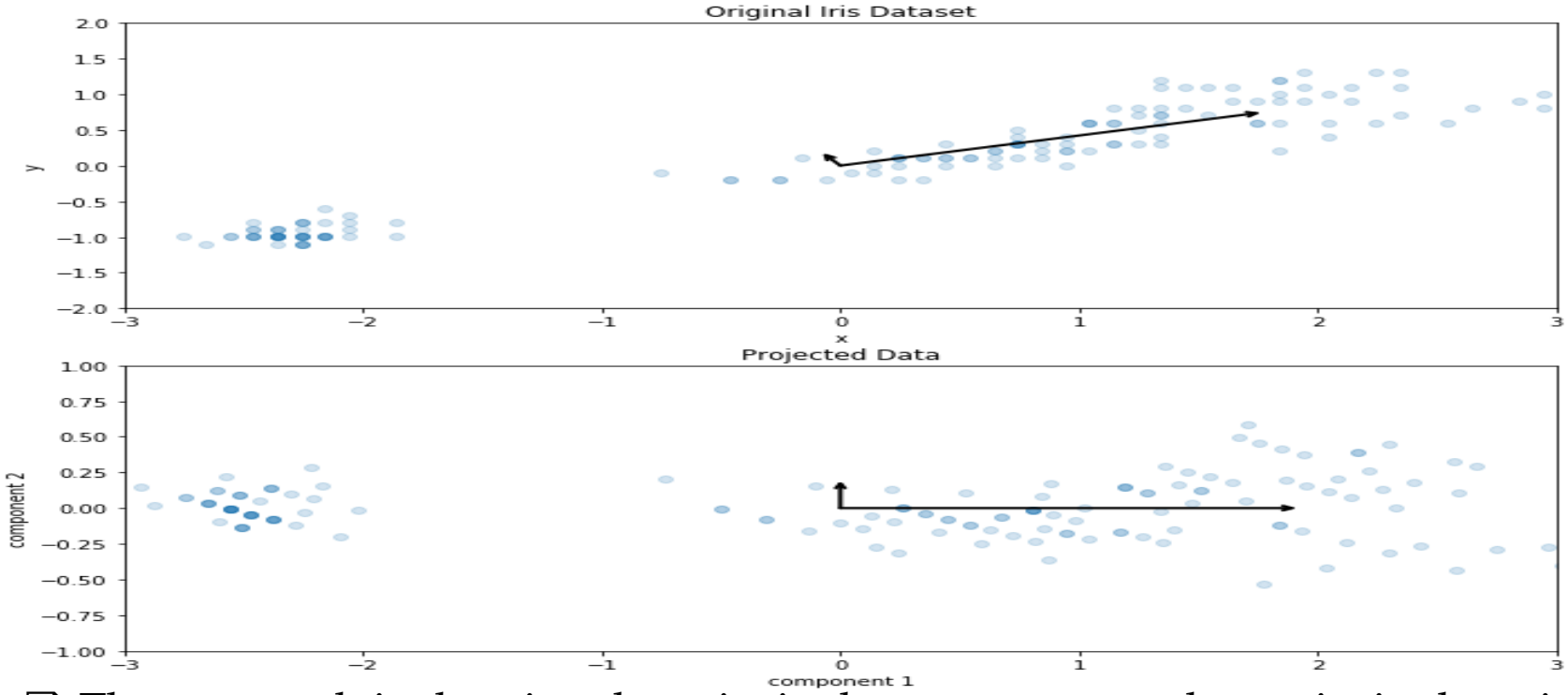
- Cluster of flowers (**setosas**) on the bottom left and a larger cluster of both **versicolor** and **virginicia** flowers on the top right.
- It appears obvious right away that the data, as a whole, is stretched along a diagonal line stemming from the bottom left to the top right.

A deeper look into the principal components



- **PCA 1**, first principal component, should be carrying the majority of the variance within it, which is why the projected data is spread out mostly across the new x axis.
- Notice how the scale of the x axis is between -3 and 3 while the y axis is only between -0.4 and 0.6.

A deeper look into the principal components



- The top graph is showing the principal components as they exist in the original data's axis system.
- They are not perpendicular and they are pointing in the direction that the data naturally follows.
- Bottom graph shows projected iris data onto these new components accompanied by same components, but acting as perpendicular coordinate systems.
- They have become the new *x* and *y* axes.

A deeper look into the principal components

- PCA is a feature transformation tool that allows us to construct brand new super features as linear combinations of previous features.
- These components carry maximum amount of variance within them, and act as new coordinate systems for data.

Linear Discriminant Analysis

- **Linear Discriminant Analysis (LDA)** is a feature transformation technique as well as a supervised classifier.
- It is commonly used as a preprocessing step for classification pipelines.
- The goal of LDA, like PCA, is to extract a new coordinate system and project datasets onto a lower-dimensional space.
- Main difference between LDA and PCA - instead of focusing on variance of the data as a whole like PCA, LDA optimizes the lower-dimensional space for best class separability.
- This means that the new coordinate system is more useful in finding decision boundaries for classification models, which is perfect for us when building classification pipelines.

Linear Discriminant Analysis

- The reason that LDA is extremely useful is that separating based on class separability helps us avoid overfitting in our machine learning pipelines.
- This is also known as preventing the curse of dimensionality.
- LDA also reduces computational costs.

Linear Discriminant Analysis

- Linear Discriminant Analysis (LDA) is most commonly used as dimensionality reduction technique in the pre-processing step for pattern-classification and machine learning applications.
- The goal is to project a dataset onto a lower-dimensional space with good class-separability in order avoid overfitting ("curse of dimensionality") and also reduce computational costs.
- Approach is very similar to a Principal Component Analysis
- In addition to finding the component axes that maximize the variance of our data (PCA), interested in the axes that maximize the separation between multiple classes (LDA).
- Will maintaining the class-discriminatory information.

PCA v/s LDA

- LDA and PCA are linear transformation techniques used for dimensionality reduction.
- PCA an “unsupervised” algorithm, and its goal is to find the directions (the so-called principal components) that maximize the variance in a dataset.
- In contrast to PCA, LDA is “supervised” and computes the directions (“linear discriminants”) that will represent the axes that that maximize the separation between multiple classes.
- Some times PCA tends to outperform LDA if the number of samples per class is relatively small
- Use both LDA and PCA in combination: E.g., PCA for dimensionality reduction followed by an LDA.

Linear Discriminant Analysis

What is a "god" feature subspace?

- Goal is to reduce the dimensions of a d-dimensional dataset by projecting it onto a k-dimensional subspace (where k<dk<d).
- How do we know what size we should choose for k
- Compute eigenvectors (the components) from our data set and collect them in a so-called scatter-matrices
- Each of these eigenvectors is associated with an eigenvalue
- If all eigenvalues have a similar magnitude, then this may be a good indicator that our data is already projected on a "good" feature space.
- If some of the eigenvalues are much larger than others, we might be interested in keeping only those eigenvectors with the highest eigenvalues
- They contain more information about our data distribution.
- Eigenvalues that are close to 0 are less informative and we might consider dropping those for constructing the new feature subspace.

Linear Discriminant Analysis

Summarizing the LDA approach in 5 steps

1. Compute the d-dimensional mean vectors for different classes from dataset.
2. Compute the scatter matrices (in-between-class and within-class scatter matrix).
3. Compute the eigenvectors ($e_1,e_2,...,e_d$) and corresponding eigenvalues ($\lambda_1,\lambda_2,...,\lambda_d$) for the scatter matrices.
4. Sort eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a d×k dimensional matrix W
5. Use this d×k eigenvector matrix to transform the samples onto the new subspace.

$$Y=X\times W$$

(where X is a n×d- dimensional matrix representing n samples, and y are transformed n×k-dimensional samples in new subspace).

Dataset for LDA

Data Set Characteristics:	Multivariate	**Number of Instances:**	150	**Area:**	Life
Attribute Characteristics:	Real	**Number of Attributes:**	4	**Date Donated**	1988-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	2331246

Attribute Information:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:

-- Iris Setosa
-- Iris Versicolour
-- Iris Virginica

Linear Discriminant Analysis

Step 1: Computing d-dimensional mean vectors

- Start off with a simple computation of the mean vectors m_i, (i=1,2,3) of the 3 different flower classes:

$$\boldsymbol{m}_i = \begin{bmatrix} \mu_{\omega_i}(\text{sepal length}) \\ \mu_{\omega_i}(\text{sepal width}) \\ \mu_{\omega_i}(\text{petal length}) \\ \mu_{\omega_i}(\text{petal width}) \end{bmatrix}, \text{ with } \quad i = 1, 2, 3$$

Linear Discriminant Analysis

Step 2: Computing the Scatter Matrices

- Compute the two 4x4-dimensional matrices:
  1. The within-class and
  2. the between-class scatter matrix.

Within-class scatter matrix S_W

The **within-class scatter** matrix S_W is computed by the following equation:

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

where

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

(scatter matrix for every class)

and $\boldsymbol{m}_i$ is the mean vector

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

Linear Discriminant Analysis

Step 2: Computing the Scatter Matrices

Between-class scatter matrix S_B

The **between-class scatter** matrix S_B is computed by the following equation:

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

where

$\boldsymbol{m}$ is the overall mean, and $\boldsymbol{m}_i$ and N_i are the sample mean and sizes of the respective classes.

Linear Discriminant Analysis

Step 3: Solving the generalized eigenvalue problem for the matrix $S_W^{-1}S_B$

- Solve the generalized eigenvalue problem for the matrix $\mathbf{S_W^{-1}S_B}$ to obtain linear discriminants.
- If v is an eigenvector of a matrix Σ, we have

  $\Sigma v = \lambda v$

  Here, λ is eigenvalue, and

  v is also an eigenvector
- A quick check that eigenvector-eigenvalue calculation is correct and satisfy equation:

  $Av=\lambda v$

  where

  $A=S_W^{-1}S_B$

  v=Eigenvector

  λ=Eigenvalue

Linear Discriminant Analysis

Step 4: Selecting linear discriminants for the new feature subspace

- Interested in merely projecting the data into a subspace that improves the class separability, but also reduces the dimensionality of our feature space
- Eigenvectors will form the axes of this new feature subspace
- In order to decide which eigenvector(s) we want to drop for our lower-dimensional subspace, we have to take a look at the corresponding eigenvalues of eigenvectors.
- Eigenvectors with lowest eigenvalues bear least information about the distribution of the data, and those are the ones we want to drop.
- The common approach is to rank the eigenvectors from highest to lowest corresponding eigenvalue and choose the top k eigenvectors.
- After sorting eigenpairs by decreasing eigenvalues, construct k×d-dimensional eigenvector matrix W.

Linear Discriminant Analysis

Step 5: Transforming the samples onto the new subspace

- Interested in merely projecting the data into a subspace that improves the class separability, but also reduces the dimensionality of our feature space
- Use matrix W to transform samples onto the new subspace via the equation

$$Y = X \times W$$

where X is a n×d-dimensional matrix representing n samples, and
Y are transformed n×k-dimensional samples in new subspace

LDA versus PCA – iris dataset

- Can try using both PCA and LDA in machine learning pipelines.
- Demonstrate the utility of both LDA and PCA as feature transformational pre-processing steps for supervised and unsupervised machine learning.
- The baseline accuracy to beat is 98.04%.
- LDA, which keeps only the most powerful component – accuracy is 0.96
- It seems that only using a single linear discriminant isn't enough to beat our baseline accuracy.
- Let us now try the PCA.
- Our hypothesis here is that the PCA will not outperform the LDA for the sole reason that the PCA is not trying to optimize for class separation as LDA.
- Accuracy with PCA - 0.89 - the worst so far
- LDA with two components – accuracy 0.98
- Want to do better than our baseline
- Check statistical feature selection would best our LDA module:
- 1 best feature has accuracy: 0.95, 2 best feature has accuracy:0.96, 3 best feature has accuracy: 0.97
- LDA is even better than the best selectkbest
- LDA with two components is so far winning
- Can use a combination of scaling, PCA, and LDA- accuracy near 99%

Thank You !!!

https://www.knowledgehut.com/blog/data-science/linear-discriminant-analysis-for-machine-learning

https://sebastianraschka.com/Articles/2014_python_lda.html

https://www.apsl.net/blog/2017/07/18/using-linear-discriminant-analysis-lda-data-explore-step-step/#:~:text=Linear%20Discriminant%20Analysis%20(LDA)%20is,classes%20of%20objects%20or%20events.

https://towardsdatascience.com/linear-discriminant-analysis-explained-f88be6c1e00b

https://www.edureka.co/blog/principal-component-analysis/

https://github.com/PacktPublishing/Feature-Engineering-Made-Easy/blob/master/Chapter06/Ch_6.ipynb

Dimension-reduction methods have the goal of using the correlation structure among the predictor variables to accomplish the following:

- To reduce the number of predictor items.
- To help ensure that these predictor items are independent.
- To provide a framework for interpretability of the results

https://www.edureka.co/blog/principal-component-analysis/

Principal Component Analysis

Principal Component Analysis (PCA) is a procedure that converts a set of observations from *m* to *n* dimensions *(m > n)*, after analyzing the correlated features of the variables. It is used to move the data from high to a low dimension for visualization or dimensionality reduction purposes. I will not go much into details since my goal here is to teach you how to apply it.

Applying PCA is no different than applying other estimators:

1.create an *estimator*,

2.fit it on the model to get a transformer,

3.apply the transformer to the data.

https://towardsdatascience.com/apache-spark-mllib-tutorial-7aba8a1dce6e