# Week 12 - Clustering, Dimensionality, and Dimensionality Reduction

- https://towardsdatascience.com/k-means-clustering-and-principal-component-analysis-in-10-minutes-2c5b69c36b6b

# Supervised / Unsupervised / Semi Supervised (Self-Training) Learning

Labels or no labels

Unsupervised learning is a type of algorithm that learns patterns from untagged data. The hope is that through mimicry, which is an important mode of learning in people, the machine is forced to build a compact internal representation of its world and then generate imaginative content from it. In contrast to supervised learning where data is tagged by an expert, e.g. as a "ball" or "fish", unsupervised methods exhibit self-organization that captures patterns as probability densities or a combination of neural feature preferences. The other levels in the supervision spectrum are reinforcement learning where the machine is given only a numerical performance score as guidance, and semi-

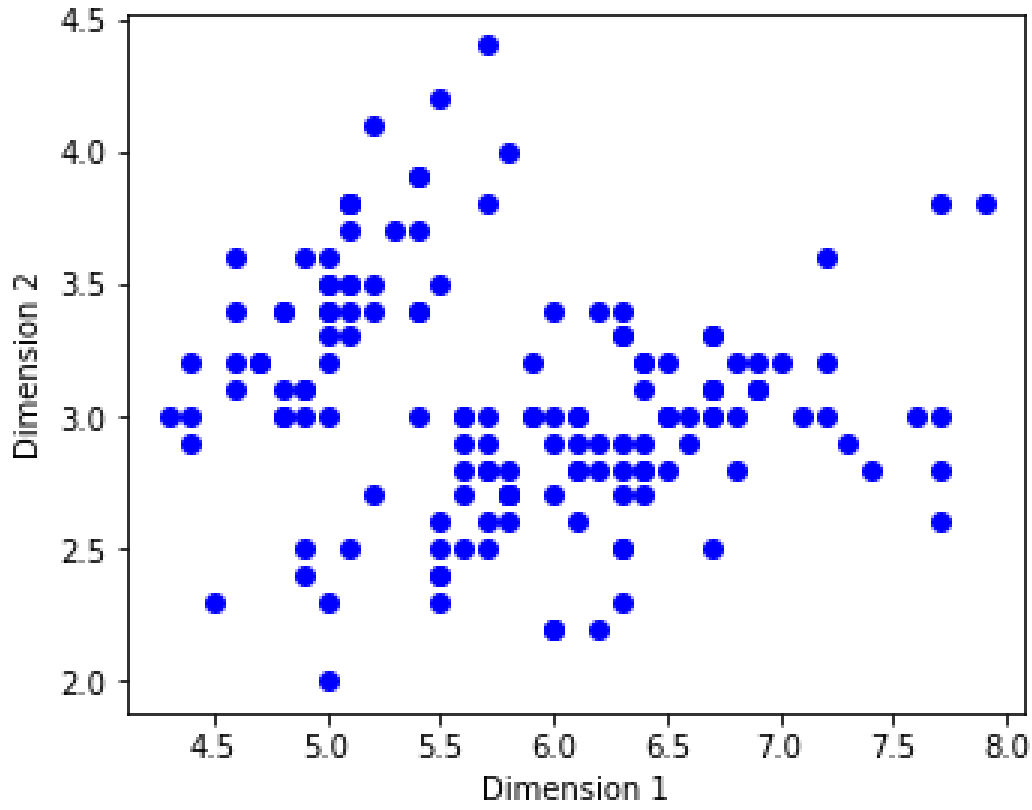
supervised learning where a smaller portion of the data is tagged.

https://en.wikipedia.org/wiki/Unsupervised_learning

## Dimensionality

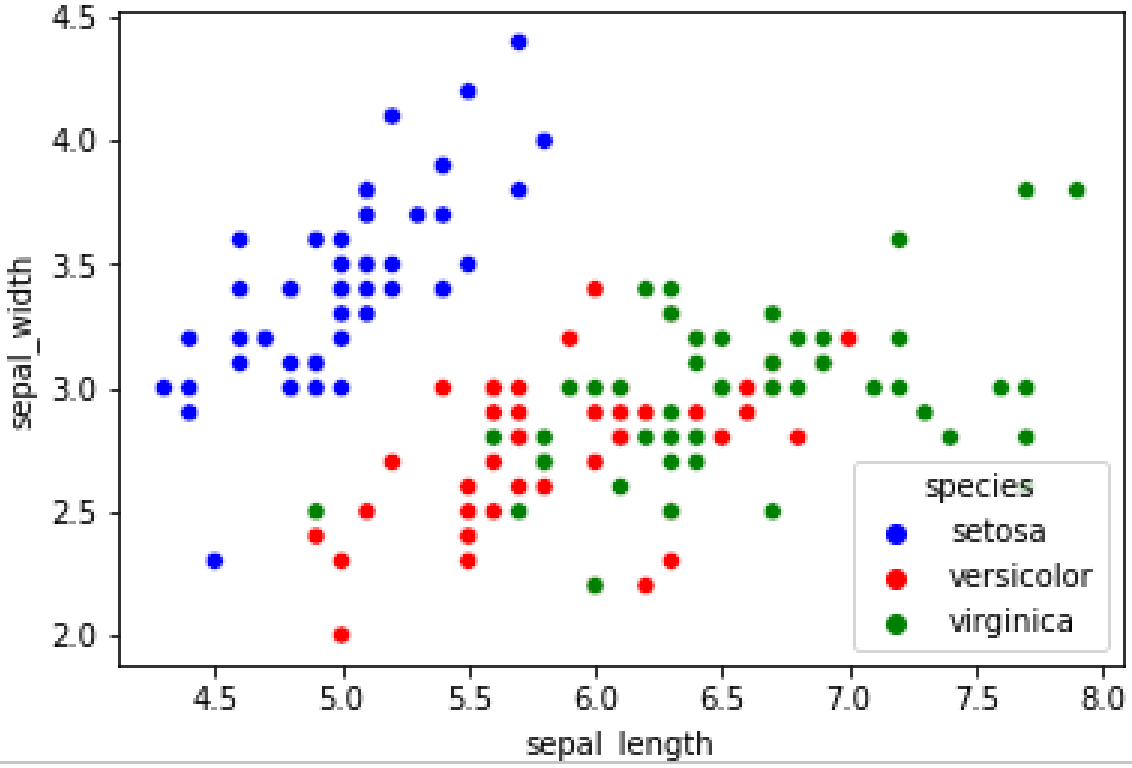
```
# plot 2 dimensions, https://scipy-lectures.org/packa
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.datasets import load_iris

iris = load_iris()
df = pd.DataFrame(data= np.c_[iris['data'], iris['tar
                     columns= iris['feature_names'] +

print(df.head())

# this formatter will label the colorbar with the cor
formatter = plt.FuncFormatter(lambda i, *args: iris.t

plt.figure(figsize=(5, 4))
plt.scatter(df['sepal length (cm)'], df['sepal width
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')

plt.tight_layout()
plt.show()
```

```
   sepal length (cm)  sepal width (cm)  petal length (
0                5.1               3.5
1                4.9               3.0
2                4.7               3.2
3                4.6               3.1
4                5.0               3.6

   target
0     0.0
1     0.0
2     0.0
3     0.0
4     0.0
```




```
# add hue as a third dimension, https://www.educba.co
import seaborn as sns
import matplotlib.pyplot as plt

iris_data = sns.load_dataset("iris")
sns.scatterplot(x="sepal_length", y="sepal_width", hu
```

```
<AxesSubplot:xlabel='sepal_length',
ylabel='sepal_width'>
```




```
# !pip install plotly
```

```
# check version
import plotly
plotly.__version__
```

```
'5.6.0'
```

```
# plot 3 dimensions
import plotly.express as px

df = px.data.iris()

fig = px.scatter_3d(df, x = 'sepal_width',
                    y = 'sepal_length',
                    z = 'petal_width',
                    color = 'species')
```

```
    fig.show()
```

```
    # 5 dimensions, https://www.geeksforgeeks.org/3d-scat
    import plotly.express as px

    df = px.data.iris()
```

```
fig = px.scatter_3d(df, x = 'sepal_width',
                    y = 'sepal_length',
                    z = 'petal_width',
                    color = 'species',
                    size='petal_length',
                    size_max = 20,
                    opacity = 0.5)

fig.show()
```

## Dimensionality Reduction

https://towardsdatascience.com/11-dimensionality-reduction-techniques-you-should-know-in-2021-

dcb9500d388b

- 1 Dimension: Point
- 2 Dimensions: Line / Plane
- 3 Dimensions: Cube

## The Curse of Dimensionality

The curse of dimensionality refers to various phenomena that arise when analyzing and organizing data in high-dimensional spaces that do not occur in low-dimensional settings such as the three-dimensional physical space of everyday experience. The expression was coined by Richard E. Bellman when considering problems in dynamic programming. Dimensionally cursed phenomena occur in domains such as numerical analysis, sampling, combinatorics, machine learning, data mining and databases. The common theme of these problems is that when the dimensionality increases, the volume of the space increases so fast that the available data become sparse. In order to obtain a reliable result, the amount of data needed often grows exponentially with the dimensionality. Also, organizing and searching data often relies on detecting areas where objects form groups with similar properties; in high dimensional data, however, all objects appear to be sparse and dissimilar in many ways, which prevents common data organization strategies from being efficient.

https://en.wikipedia.org/wiki/Curse_of_dimensionality

# 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, ideally close to its intrinsic dimension. 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 (hard to control or deal with). Dimensionality reduction is common in fields that deal with large numbers of observations and/or large numbers of variables, such as signal processing, speech recognition, neuroinformatics, and bioinformatics.

https://en.wikipedia.org/wiki/Dimensionality_reduction

# Principal Component Analysis - PCA (unsupervised)

PCA is considered unsupervised while LDA is supervised because it uses the dependent variable.

- Represent data with a smaller set of features while keeping most of the variance
- Discovers the correlation between variables (least squares)

- https://setosa.io/ev/principal-component-analysis/
- Good for multicollinearity
- Train (fit) on X

PCA is used in exploratory data analysis and for making predictive models. It is commonly used for dimensionality reduction by projecting each data point onto only the first few principal components to obtain lower-dimensional data while preserving as much of the data's variation as possible. The first principal component can equivalently be defined as a direction that maximizes the variance of the projected data.

https://en.wikipedia.org/wiki/Principal_component_analysis

More readings:

- https://towardsdatascience.com/principal-component-analysis-pca-from-scratch-in-python-7f3e2a540c51
- https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html
- https://towardsdatascience.com/machine-learning-part-15-dimensionality-reduction-with-principal-component-analysis-a5b3bb7353bc
- https://www.analyticsvidhya.com/blog/2016/03/pca-practical-guide-principal-component-analysis-python/
- https://stackoverflow.com/questions/22984335/recovering-features-names-of-explained-variance-ratio-in-pca-with-sklearn

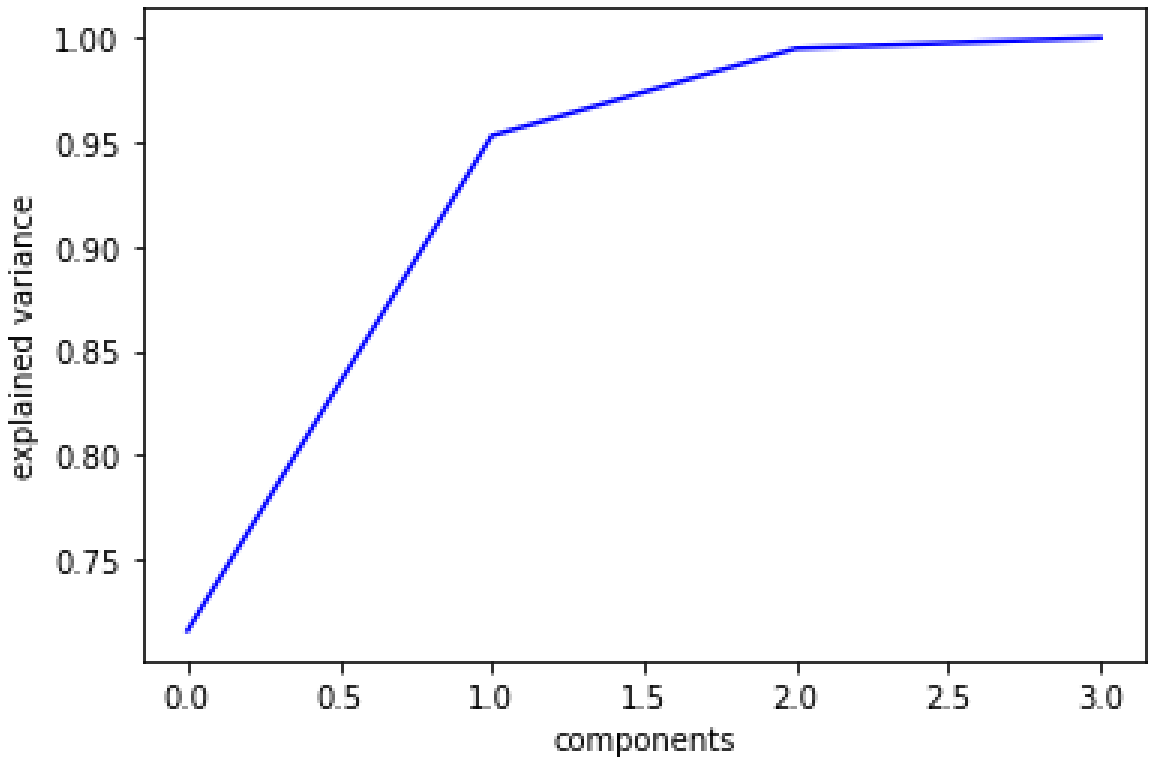
```
# get data
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.featur
df['target'] = iris.target

# train test split
X_train, X_test, y_train, y_test = train_test_split(d
columns = X_train.columns
print(X_train.head())
```

```
     sepal length (cm)  sepal width (cm)  petal length
4                  5.0               3.6
32                 5.2               4.1
142                5.8               2.7
85                 6.0               3.4
86                 6.7               3.1
```

```
# scale data
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
# plot explained variance
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

model = PCA(n_components = 4)
```

```
model.fit_transform(X_train)
plt.plot(np.cumsum(model.explained_variance_ratio_))
plt.xlabel('components')
plt.ylabel('explained variance');
```




```
# reduce dimensions
import pandas as pd
from sklearn.decomposition import PCA

model = PCA(n_components = 2)
X_train = model.fit_transform(X_train) # unsupervised
X_test = model.transform(X_test)

# feature importance or influence (weights)
print(pd.DataFrame(model.components_, columns=columns
```

```
      sepal length (cm)  sepal width (cm)  petal lengt
PC-1           0.522766         -0.245027           0.
PC-2          -0.368870         -0.928071          -0.
```

Features 1, 3, and 4 are important for PC-1 and feature 2 has the highest value for PC-2

```
# PVE proportion of variance explained
print(model.explained_variance_ratio_)
```

```
[0.71581797 0.23720827]
```

Read the following for another look at PCA on a dataset with more features and how to choose components.

https://towardsdatascience.com/principal-component-analysis-pca-with-scikit-learn-1e84a0c731b0

```
# build and evaluate supervised classification model
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accurac

model = LogisticRegression(random_state=42)
model.fit(X_train, y_train)
predictions = model.predict(X_test)

print(confusion_matrix(y_test, predictions))
print(f'Training Score: {model.score(X_train, y_train
print(f'Test Score: {accuracy_score(y_test, predictio
```

```
[[15  0  0]
 [ 0  9  2]
 [ 0  1 11]]
Training Score: 0.9107142857142857
Test Score: 0.9210526315789473
```

# Linear Discriminant Analysis - LDA

- Finds component axes
- Maximizes the separation between multiple classes
- Train (fit) on X, y

Linear discriminant analysis (LDA), normal discriminant analysis (NDA), or discriminant function analysis is a generalization of Fisher's linear discriminant, a method used in statistics and other fields, to find a linear combination of features that characterizes or separates two or more classes of objects or events. The resulting combination may be used as a linear classifier, or, more commonly, for dimensionality reduction before later classification.

LDA is closely related to analysis of variance (ANOVA) and regression analysis, which also attempt to express one dependent variable as a linear combination of other features or measurements. However, ANOVA uses categorical independent variables and a continuous dependent variable, whereas discriminant analysis has continuous independent variables and a categorical dependent variable (i.e. the class label). Logistic regression and probit regression are more similar to LDA than ANOVA is, as they also explain a categorical variable by the values of continuous independent variables. These other methods are preferable in applications where it is not reasonable to assume that the independent

variables are normally distributed, which is a fundamental assumption of the LDA method.

LDA is also closely related to principal component analysis (PCA) and factor analysis in that they both look for linear combinations of variables which best explain the data. LDA explicitly attempts to model the difference between the classes of data. PCA, in contrast, does not take into account any difference in class, and factor analysis builds the feature combinations based on differences rather than similarities. Discriminant analysis is also different from factor analysis in that it is not an interdependence technique: a distinction between independent variables and dependent variables (also called criterion variables) must be made.

https://en.wikipedia.org/wiki/Linear_discriminant_analysis

```
# get data
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.featur
df['target'] = iris.target

# train test split
X_train, X_test, y_train, y_test = train_test_split(d
columns = X_train.columns
print(X_train.head())
```

```
     sepal length (cm)  sepal width (cm)  petal length
4                  5.0               3.6
```

```
32                  5.2                 4.1
142                 5.8                 2.7
85                  6.0                 3.4
86                  6.7                 3.1
```

```
# code example
from sklearn.discriminant_analysis import LinearDiscr

lda = LDA(n_components = 2)
X_train = lda.fit_transform(X_train, y_train) # super
X_test = lda.transform(X_test)
```

```
# create model
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accurac

model = LogisticRegression(random_state=42)
model.fit(X_train, y_train)
predictions = model.predict(X_test)

print(confusion_matrix(y_test, predictions))
print(f'Training Score: {model.score(X_train, y_train
print(f'Test Score: {accuracy_score(y_test, predictio
```

```
[[15  0  0]
 [ 0 11  0]
 [ 0  0 12]]
Training Score: 0.9821428571428571
Test Score: 1.0
```

## Singular Value Decompostion

- Used for dimensionality reduction, image compression, denoising data, image recovery, removing background noise
- Matrix can be represented as the product of three matrices - $A_{nn} = U_{np} S_{np} V_{pp}^T$
- where n = rows and p = dimensions
- S is the diagonal matrix of singular values, importance values of each of the features
- Eigenvetors of a matrix are directions of maximum spread or variance of data

Readings:

- https://www.analyticsvidhya.com/blog/2019/08/5-applications-singular-value-decomposition-svd-data-science/
- https://towardsdatascience.com/singular-value-decomposition-example-in-python-dab2507d85a0
- https://machinelearningmastery.com/dimensionality-reduction-algorithms-with-python/
- https://stackabuse.com/dimensionality-reduction-in-python-with-scikit-learn/

## t-distributed Stochastic Neighbor Embedding (t-SNE)

- Constructs a probability distribution for high-dimensional samples

- Projects high-dimensional data points into 2D/3D by inducing the projected data to have a similar distribution as the original dataset
- Similar samples have a high likelihood of being picked
- Dissimilar points have and extremely small likelihood of being picked

Readings

- https://plotly.com/python/t-sne-and-umap-projections/
- https://towardsdatascience.com/t-sne-python-example-1ded9953f26
- https://towardsdatascience.com/dimension-reduction-techniques-with-python-f36ca7009e5c
- https://www.learndatasci.com/tutorials/applied-dimensionality-reduction-techniques-using-python/

```
import plotly.express as px

df = px.data.iris()
features = ["sepal_width", "sepal_length", "petal_wid
fig = px.scatter_matrix(df, dimensions=features, colo
fig.show()
```

```
from sklearn.manifold import TSNE
import plotly.express as px

df = px.data.iris()

features = df.loc[:, :'petal_width']
```

```
tsne = TSNE(n_components=2, random_state=42, init='ra
projections = tsne.fit_transform(features)

fig = px.scatter(
    projections, x=0, y=1,
    color=df.species, labels={'color': 'species'}
)
fig.show()
```

```
from sklearn.manifold import TSNE
import plotly.express as px

df = px.data.iris()

features = df.loc[:, :'petal_width']
```

```
tsne = TSNE(n_components=3, random_state=42, init='ra
projections = tsne.fit_transform(features, )

fig = px.scatter_3d(
    projections, x=0, y=1, z=2,
    color=df.species, labels={'color': 'species'}
)
fig.update_traces(marker_size=8)
fig.show()
```

## Isomap Embedding

A technique that combines several different algorithms, enabling it to use a non-linear way to reduce dimensions while preserving local neighborhoods

https://towardsdatascience.com/isomap-embedding-an-awesome-approach-to-non-linear-dimensionality-reduction-fc7efbca47a0

## Locally Linear Embedding

Tries to reduce n-Dimensions while preserving the geometric features of the original non-linear feature structure or relationships between neighborhoods

https://docs.google.com/document/d/1J9rjTzv3VKB2Qq10P3Oq2mV_K8pJOSdaeJ3IImPszBA/edit

## Modified Locally Linear Embedding

An extension fo Locally Linear Embedding that creates multipl weighting vectors for each neighborhood

https://machinelearningmastery.com/dimensionality-reduction-algorithms-with-python/

## Kernel PCA

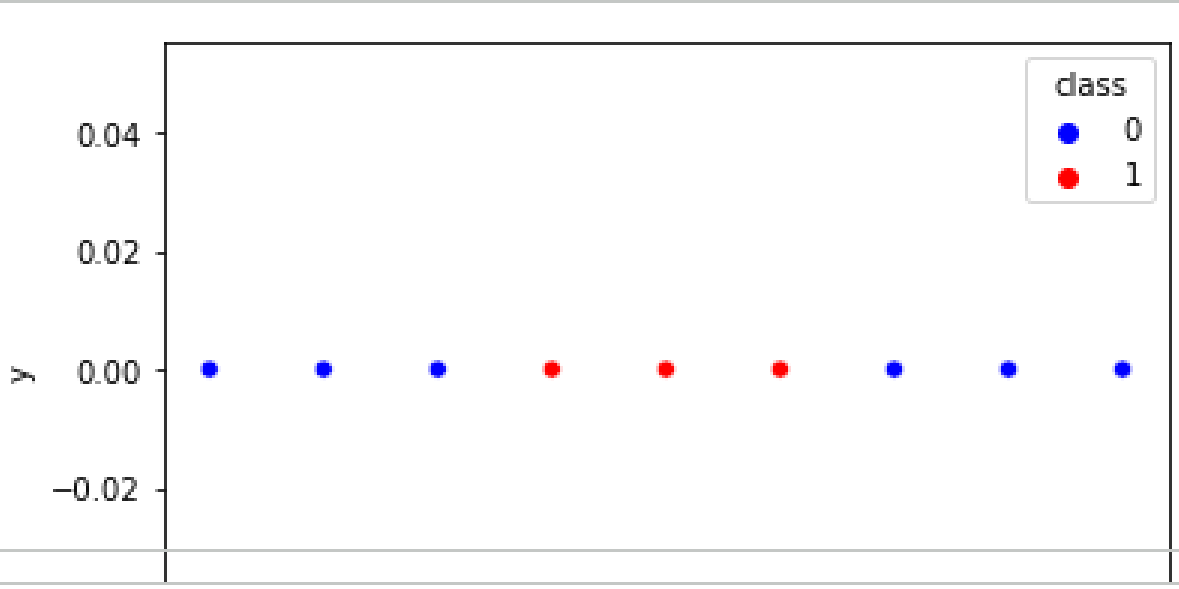
- Creates a new space of dimensions
- Non-linear dimensionality reduction through the use of kernels

- https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.KernelPCA.html

## SVC with Kernel Trick

- https://miro.medium.com/max/1400/1*mCwnu5kXot6buL7jeIafqQ.png (https://medium.com/@zxr.nju/what-is-the-kernel-trick-why-is-it-important-98a98db0961d)
- Non linear
- Maps classes to different dimensions
- https://vitalflux.com/machine-learning-svm-kernel-trick-example/
- Example - exponential curve can be used to make something linearly separable
- The kernel trick

```
# svm kernel trick review
import pandas as pd
import seaborn as sns

x = [-4, -3, -2, -1, 0, 1, 2, 3, 4]
y = [0, 0, 0, 0, 0, 0, 0, 0, 0]
target = [0, 0, 0, 1, 1, 1, 0, 0, 0]
d = {'x': x, 'y': y, 'class': target}
df = pd.DataFrame(d)
sns.scatterplot(data=df, x='x', y='y', hue='class');
```




```
# y = x**2
import numpy as np

df['kernel_trick'] = df['x']**2
sns.scatterplot(data=df, x='x', y='kernel_trick', hue
plt.axhline(y=2.5, color='k')
plt.show()
```