Functions Derivatives and their applications

Function27 function is it a gathering or what is it?

A function is a given process or rule, connecting two quantities such that an output

> This is like a machine which has an input and produces an output and combines the output in relation to the input.

$$\text{Function}(x) = \text{output}$$
$$\text{Plug } x \text{ into the machine}$$

Source: Math Club

Examples of functions

```
y = x^2
y = 2x + 3
y = sin(x)  # y is function which depends the value of x
```

```
import math
print(math.sin(0))
print(math.sin(math.pi/2))
print(math.sin(math.pi/2) + 2*math.sin(0))
print(math.sin(math.pi/2) + 2*math.sin(0) + 3)
```

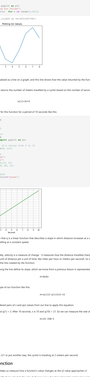
```
0.0
1.0
1.0
4.0
```

```
import matplotlib as mpl
mpl.rcParams['text.usetex'] = True
mpl.rcParams['text.latex.preamble'] = r'\usepackage{siunitx}'
import numpy as np
import matplotlib.pyplot as plt

# Generate data
x = np.linspace(0, 10, 100)
y = np.sin(x)

# Create the plot
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Plotting Sin Function')

# Show the plot
plt.show()
```



Rate of Change

Functions are often visualized as a line on a graph, and the line shows how the value returned by the function changes based on changes to the input value.

Imagine a function that returns the number of meters travelled by a cyclist based on the number of seconds that the cyclist has been cycling.

$$y(x) = 5x + 2$$

We can plot this function for the function for a period of 10 seconds:

```
import numpy as np
x = np.arange(0, 10)
y = 5*x + 2
print(x, y)
print(x.shape, y.shape)
# Plot the data
plt.plot(x, y, 'o')
# Add labels and title
plt.xlabel('x')
plt.ylabel('y')
plt.title('Plot of y = 5x + 2')
# Show the plot
plt.show()
```

```
[0 1 2 3 4 5 6 7 8 9] [ 2 7 12 17 22 27 32 37 42 47]
(10,) (10,)
```

It's clear from the graph that as time travels forward that there has a slope in which distance increases at a constant rate over time. In other words, the function is cycling at a constant speed.

But what speed?

Speed, or more formally, velocity is a measure of change. It measures how the function's dependent changes over time (which is why we plot time on the x axis). Velocity is the change in the function's output value over time. For a linear function, it can be calculated by dividing the change in output value by the change in input value.

The change in output along the line within the graph above between 2 points/ time is represented by this:

$$y_2 - y_1$$

We can calculate the slope of our function like:

$$y_2 - y_1 / x_2 - x_1 = y_2 - y_1 / x_2 - x_1$$

So we just need two ordered pairs of x and y values from our line to apply the calculation.

When $x_2 - x_1$ and $y_2 - y_1$ is the same for any point on the line, $y(x) = 5x + 2$ is an slope between the rate of change like this:

$$y_2 - y_1 / x_2 - x_1 = 5$$

This is the same as:

$$y_2 - y_1 = 5$$

Where simplified to:

$$y_2 - y_1 = 5$$

So we can rate of change $y = 5$ x and another way, the cyclist's travelling at an average velocity of 5 meters per second over the 10 second period.

```
import numpy as np
x = np.arange(0, 10)
y = 5*x + 2
print(x, y)
print(x.shape, y.shape)
# Plot the data
plt.plot(x, y, 'o')
# Add labels and title
plt.xlabel('x')
plt.ylabel('y')
plt.title('Plot of y = 5x + 2')
# Show the plot
plt.show()
# Calculate the slope
slope = (y[1] - y[0]) / (x[1] - x[0])
print('Slope:', slope)
# Add the slope to the plot
plt.text(5, 30, 'Slope: {:.2f}'.format(slope))
plt.show()
```

```
[0 1 2 3 4 5 6 7 8 9] [ 2 7 12 17 22 27 32 37 42 47]
(10,) (10,)
Slope: 5.0
```

Importance of derivatives in data science

Derivatives are used by Machine learning for solving optimization problems.

Gradient algorithms such as gradient descent are used to determine the values of parameters in a function or decrease the weights in order to minimize the cost function.

Data Scientists use calculus for almost every machine learning algorithm. Some of applications of calculus in Data Science

- Gradient descent
- First and second order derivatives and use in Image Processing

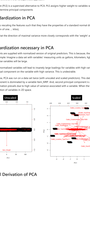
Gradient Descent

A gradient measures how much the output of a function changes if you change the input a little bit.

A machine learning model will use gradient descent to change the error in our model.

The cost function is used to monitor the error in predictions of an ML model. By minimizing this, basically means getting to the lowest error in our model, we can make our predictions as accurate as possible.

Therefore, we minimize the error by tweaking our existing weights when reading the parameters of our model.



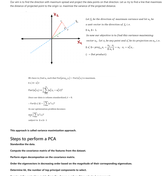
Research Paper Analysis

Introduction to Principal Component Analysis

> *Too much of anything is good for nothing!*

- This is the main common situation when working on large data sets, we have to work data rich in dimensions (features) to extract the most meaningful attributes and capture the main ...
- This always leads to the ambiguity with what items to analyze a specifiers which to linger-ing, i.e. ... ...

Our aim is to find the direction with maximum spread and project the data points on that direction. Let us try to find a line that maximizes the distance of projected points on the line. Let us maximize the variance of the projected distances.



Let $\mathbf{x}_i$ be the elements of standardized variables and let $\mathbf{u}_1$ be a unit vector in the direction of $\mathbf{x}_1$ i.e.

$$\mathbf{u}_1 = \begin{bmatrix} u_{11} \\ u_{12} \\ \vdots \\ u_{1p} \end{bmatrix}$$

Our aim is to find $\mathbf{u}_1$ such that the variance maximizing

$$\text{Var}(\mathbf{u}_1^T \mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{u}_1^T \mathbf{x}_i)^2 = \frac{1}{n} \sum_{i=1}^{n} \mathbf{u}_1^T \mathbf{x}_i \mathbf{x}_i^T \mathbf{u}_1 = \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1$$

$$\text{Var}(\mathbf{u}_1^T \mathbf{x}) = \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1$$

Where $\mathbf{S}$ is the covariance matrix of $\mathbf{x}$ and $\text{Var}(\mathbf{u}_1^T \mathbf{x})$ is the variance of $\mathbf{u}_1^T \mathbf{x}$.

Our aim is to maximize $\text{Var}(\mathbf{u}_1^T \mathbf{x})$ subject to $\mathbf{u}_1^T \mathbf{u}_1 = 1$

$$\text{Maximize}(\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1)$$

$$\text{Subject to } \mathbf{u}_1^T \mathbf{u}_1 = 1$$

$$\text{Lagrangian is } L = \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 - \lambda(\mathbf{u}_1^T \mathbf{u}_1 - 1)$$

$$\text{Differentiating with respect to } \mathbf{u}_1 \text{ we get}$$

$$\frac{\partial L}{\partial \mathbf{u}_1} = 2\mathbf{S}\mathbf{u}_1 - 2\lambda \mathbf{u}_1$$

$$\text{Equating to zero we get}$$

$$\mathbf{S}\mathbf{u}_1 = \lambda \mathbf{u}_1$$

This represents an eigenvalue-eigenvector problem.

Steps to perform a PCA

Standardize the data

Compute the covariance matrix of the features from the dataset

Perform eigen decomposition on the covariance matrix.

Order the eigenvalues in decreasing order based on the magnitude of their corresponding eigenvalues.

Determine, the, the number of top principal components to be retained.

Construct the projection matrix from the chosen number of top principal components.

Compute the new lo-dimensional feature space.

PCA on a Data Set

Importing the needed libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.ensemble import HistGradientBoostingClassifier
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
from sklearn.metrics import roc_auc_score, roc_curve
import scipy.stats as stats
import warnings
warnings.filterwarnings('ignore')
```

Reading the Data Set

```
df = pd.read_csv("bitcoin.csv")
```

```
df.head()
```

	symbol	year	month	open	high	low	close	volume_btc	volume_currency	weighted_price
0	BTCUSD	2014	9	394.79999	394.79999	394.79999	394.79999	0.01000	3.94800	394.79999
1	BTCUSD	2014	10	386.44000	386.44000	386.44000	386.44000	0.01000	3.86440	386.44000
2	BTCUSD	2014	11	356.99999	356.99999	356.99999	356.99999	0.01000	3.56999	356.99999
3	BTCUSD	2014	12	377.99000	377.99000	377.99000	377.99000	0.01000	3.77990	377.99000
4	BTCUSD	2015	1	314.25999	314.25999	314.25999	314.25999	0.01000	3.14260	314.25999

```
df.tail()
```

Output: 125 columns

Data Dictionary

Variable	Description
YEAR	
CARD_CVV	Card Verification Value (CVV)
STREET_CITY	Street Name/Number
TOTAL_UPDATES_24M	Total number of Updates for power change (p.c.)
AVG_LATENCY_MONTHLY	Average latency for last value for p.c.
TOTAL_ORDERS_365	Number of Orders
TOT_ACCOUNT	Number of Accounts
TOT_TICK_PAID	Number of Paids
TOT_TICK_BADD	Number of Frauds
TOT_KWH	Volume
TOT_REVENUE	Total Revenue
MAX_DAILY	Total Demand
DIRECT_COST	Direct Distance
LAST_DATE	Latest Contract
CUSTOMER_STATUS	Customer Status
TARIFF_NAME	Billing Plans
DIST_CHANNEL_NAME	Driving Plans
TOT_ON_DATE_MIN	On Duty Hours
TOT_ON_DATE_MAX	Off Duty Hours
DIST_CHANNEL_NAME	Dispatch Rules
PERC_ON_DATE	Percentage Hours
AVG_VOLAGE_MONTH	Weekly Utilization Percentage
AVG_VOLAGE_YEAR	Annual Utilization Percentage
MAX_VOLAGE_MONTH	Weight Utilization Percentage

Getting the shape and information about the Data

```
df.shape
```

```
(2918, 125)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2918 entries, 0 to 2917
Columns: 125 entries, YEAR to MAX_VOLAGE_YEAR
dtypes: int64(3), object(122)
memory usage: 2.8+ MB
```

Plotting part plots to understanding the complexity of analyzing variables

```
sns.pairplot(df)
```

From the above it is clearly evident that no the Data Set contains a lot of analyzing variables (125 to be exact).

Cleaning the data as needed for PCA

```
df.drop(['YEAR', 'CARD_CVV', 'STREET_CITY', 'TOTAL_UPDATES_24M', 'AVG_LATENCY_MONTHLY', 'TOTAL_ORDERS_365', 'TOT_ACCOUNT', 'TOT_TICK_PAID', 'TOT_TICK_BADD', 'TOT_KWH', 'TOT_REVENUE', 'MAX_DAILY', 'DIRECT_COST', 'LAST_DATE', 'CUSTOMER_STATUS', 'TARIFF_NAME', 'DIST_CHANNEL_NAME', 'TOT_ON_DATE_MIN', 'TOT_ON_DATE_MAX', 'DIST_CHANNEL_NAME', 'PERC_ON_DATE', 'AVG_VOLAGE_MONTH', 'AVG_VOLAGE_YEAR', 'MAX_VOLAGE_MONTH', 'MAX_VOLAGE_YEAR'], axis=1)
```

```
df.shape
```

```
(2918, 100)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2918 entries, 0 to 2917
Columns: 100 entries, TOT_CARD_VALIDATE_24M to TOT_CARD_VALIDATE_365
dtypes: int64(3), object(97)
memory usage: 2.2+ MB
```

```
df.isnull.sum()
```

```
TOT_CARD_VALIDATE_24M    0
TOT_CARD_VALIDATE_48M    0
TOT_CARD_VALIDATE_90M    0
TOT_CARD_VALIDATE_180M   0
TOT_CARD_VALIDATE_365    0
dtype: int64
```

Data Standardization

Standardization refers to shifting the distribution of each attribute to have a mean of zero and a standard deviation of one (unit variance). It requires rescaling the distribution of values.

Standardization can be helpful in cases where the data follows a Gaussian distribution.

```
sc = StandardScaler()
X = sc.fit_transform(df)
```

Understanding Variance, Covariance and its importance

Variance measures the variation of a single random variable (like the height of a person in a population), whereas covariance is a measure of how much two random variables vary together (like the height of a person and the weight of a person in a population). This concept is very important for data science.

$$\sigma^2(x) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2$$

where μ is the number of samples (e.g. the number of people) and μ is the mean of the random variable (μ represented as a mean). The covariance $\sigma(x, y)$ of two random variables x and y is given by

$$\sigma(x, y) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)(y_i - \mu)$$

with n samples. The variance σ^2_x of a random variable x can be also expressed as the covariance with itself $\sigma(x, x)$

Covariance Matrix

Will be convenient we can calculate covariance of the covariance matrix, which is a matrix containing covariance $\sigma(x, y)$ for all possible x and y. The covariance matrix is a square matrix of dimension $d \times d$ (where d is the number of dimensions, e.g. the number of features). The covariance matrix is symmetric, since $\sigma(x, y) = \sigma(y, x)$ and the diagonal entries of the covariance matrix are the variances of the features, because $\sigma(x, x) = \sigma^2(x)$. Covariance matrix is a convenient way to collect the pairwise covariances. The calculation for the covariance matrix is with the expression

$$\sigma = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)(x_i - \mu)^T$$

where σ is the covariance matrix, x_i is the number of people and d is the dimension of the random variable (d represented as a mean). The consequence $\sigma(x, y)$ of two random variables x and y is given by

$$\sigma(x, y) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)(y_i - \mu)$$

Let now a data matrix X be of size $N \times D$, where N stands for the number of samples and D is the number of features in each sample. We assume that we have subtracted the mean from each feature and the data matrix is now centered. We can then express the covariance matrix of the features in this sample as $\frac{1}{N} X^T X$.

Eigen Values

Let $\mathbf{v}$ be a vector such that $\mathbf{v}$ is an eigenvector of σ then λ is an eigenvalue of σ if $\sigma \mathbf{v} = \lambda \mathbf{v}$.

Let σ be a square matrix. λ is an eigenvalue of σ if $\det(\lambda I - \sigma) = 0$ where I is an identity matrix of the same dimensions as σ.

Let σ be a square matrix. λ is an eigenvalue of σ if there is a non-zero vector, $\mathbf{v}$ called an eigenvector if it satisfies the generalized equation

$$\sigma \mathbf{v} = \lambda \mathbf{v}$$

Let σ be a square matrix. λ is an eigenvalue of σ if there is a non-zero vector, $\mathbf{v}$ called an eigenvector if it satisfies the quadratic equation

$$\mathbf{v}^T \sigma \mathbf{v} = \lambda$$

Let σ be a square matrix. λ is an eigenvalue of σ corresponding to an eigenvector $\mathbf{v}$, if $\sigma \mathbf{v}$ is a scalar multiple of $\mathbf{v}$.

Let σ be a square matrix. λ is an eigenvalue of σ if there is a vector, $\mathbf{v}$, in the eigenspace of σ.

Let σ be a square matrix. λ is an eigenvalue of σ if the eigenspace of σ, λ is a non-zero vector.

Let σ be a square matrix. λ is an eigenvalue of σ if there is a vector, $\mathbf{v}$, in the eigenspace of σ, λ is an eigenvalue of σ.

Let σ be a square matrix. λ is an eigenvalue of σ if there is a vector, $\mathbf{v}$, in the eigenspace of σ, λ is an eigenvalue of σ.

Eigenvalue Example

In the image depicting the two orange distortions between the pink and blue lines, the blue line is the eigenvector because it does not change direction. Also, the length of the line is not changed.

Image from Bayes

Calculating Eigen Values and Vectors

```
eig_val, eig_vec = np.linalg.eig(cov_mat)
print('Eigenvectors \n%s', %(eig_vec))
print('Eigenvalues \n%s', %(eig_val))
```

```
Eigenvectors
[[ 0.14669811 -0.49073038  0.62620532  0.58789161]
 [ 0.24089319 -0.47931378 -0.77578566  0.33545954]
 [ 0.63097056  0.77030963  0.02267499  0.09072122]
 [ 0.73708687 -0.08758388  0.12239074 -0.66267215]]
Eigenvalues
[2.93035378 0.92740362 0.15059276 0.05091483]
```

Computing Eigen Pairs

```
eig_pairs = [(np.abs(eig_val[i]), eig_vec[:,i]) for i in range(len(eig_val))]
# Sort the (eigenvalue, eigenvector) tuples from highest to low
eig_pairs.sort(key = lambda x: x[0], reverse= True)
print('Everything ok!')
# Confirmation
print('Eigenvalues in descending order:')
for i in eig_pairs:
    print(i[0])
```

```
Everything ok!
Eigenvalues in descending order:
2.93035378
0.92740362
0.15059276
0.05091483
```

Explained Variance

After computing eigenvalues, we can compute in "Pure Linear Algebra" components we are going to choose for our new feature subspace? It's useful because it is the so called "explained variance", which can be calculated from the eigenvalues. The explained variance tells us how much information (variance) can be attributed to each of the principal components.

```
cum_sum = np.cumsum(pca.explained_variance_)
print("Cumulative explained variance: ", cum_sum)
```

```
Cumulative explained variance:  [2.93035378 3.85775741 4.01835017 4.07326499]
```

```
pca = PCA(n_components=4)
principal_components = pca.fit_transform(x_train, x_test)
principal_components.shape
```

```
(2918, 4)
```

```
pca.fit(X)
```

```
PCA(copy=True, iterated_power='auto', n_components=None, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
```

Plotting Principal Components

```
fig, axes = plt.subplots(1,2,figsize=(7,3))
print('Explained Variance\n', pca.explained_variance_)
print('Components\n', pca.components_)
print('Mean\n', pca.mean_)
print('Number of components\n', pca.n_components_)
print('Shape of the data\n', principal_components.shape)
```

The pair plot above shows that the maximum variance (explained variance) is captured by the first principal component alone.

The second third and fourth components capture close to 0% of variance in data.

We can also filter the components in the data set. We will change

Projection Matrix

Projection matrix is a matrix that when used that will be used to transform the Human resource analytics data onto the new feature subspace.

Transpose the matrix of principal component (these the eigenvectors ordered by their eigenvalues of importance) we get PCA Matrix we can use to plot other components. Here we are reducing the 7-dimensional feature space to a 2-dimensional feature subspace, by choosing the "top 2" eigenvectors with the highest eigenvalues to construct our 7-dimensional eigenspace matrix W.

```
matrix_w = np.hstack(eig_pairs[0][1].reshape(4,1),
                     eig_pairs[1][1].reshape(4,1))
print('Matrix W:\n', matrix_w)
```

```
Matrix W:
 [[0.14669811]
 [0.24089319]
 [0.63097056]
 [0.73708687]]
```

Projection

Projection Data the New Feature Space is the last step we will use the 7-dimensional projection matrix W to transform our sample onto the new subspace via the equation $Y = X \times W$.

```
Y = X.dot(matrix_w)
print(Y[0:5,:])
```

```
[[ 0.71956835 -0.35548474]
 [ 1.633141  -0.384916]
 [ 2.18451   -0.492637]
 [ 0.87503   -0.670444]
 [ 0.49314   -0.318274]]
```

PCA Using Python

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principal_components = pca.fit_transform(x_train, x_test)
principal_components.shape
```

```
(2918, 2)
```

```
pca.fit(X)
```

```
PCA(copy=True, iterated_power='auto', n_components=2,
  random_state=None, svd_solver='auto', tol=0.0, whiten=False)
```

```
fig, axes = plt.subplots(1,2,figsize=(7,3))
print('Explained Variance\n', pca.explained_variance_)
print('Components\n', pca.components_)
print('Mean\n', pca.mean_)
print('Number of components\n', pca.n_components_)
print('Shape of the data\n', principal_components.shape)
```


The above plot shows about 60% variance by the first 5 components. Therefore we can drop rest of the components.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=5)
principal_components = pca.fit_transform(x_train, x_test)
principal_components.shape
```

```
(2918, 5)
```

```
pca.fit(X)
```

```
PCA(copy=True, iterated_power='auto', n_components=5,
  random_state=None, svd_solver='auto', tol=0.0, whiten=False)
```

```
fig, axes = plt.subplots(1,2,figsize=(7,3))
print('Explained Variance\n', pca.explained_variance_)
print('Components\n', pca.components_)
print('Mean\n', pca.mean_)
print('Number of components\n', pca.n_components_)
print('Shape of the data\n', principal_components.shape)
```

The Principal Component Analysis is used to extract the expected features from the dataset, without losing much information.These features are two dimensional in nature.The first component has the highest variance followed by second, third and so on.

Principal Component Analysis is used to make data easy to explore and visualize. Because with higher dimensions, it becomes increasingly difficult to make interpretations from the samples out of data.

References

https://www.kaggle.com/code/ankur1mishra/pca-on-iris-dataset#eda-and

https://towardsdatascience.com/principal-component-analysis-explained-49fc7f45f99a

https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60

https://datascienceplus.com/understanding-the-covariance-matrix/#:~:text=where%20is%20the%20data%20matrix,%20%2C%20%2C%20%2C