

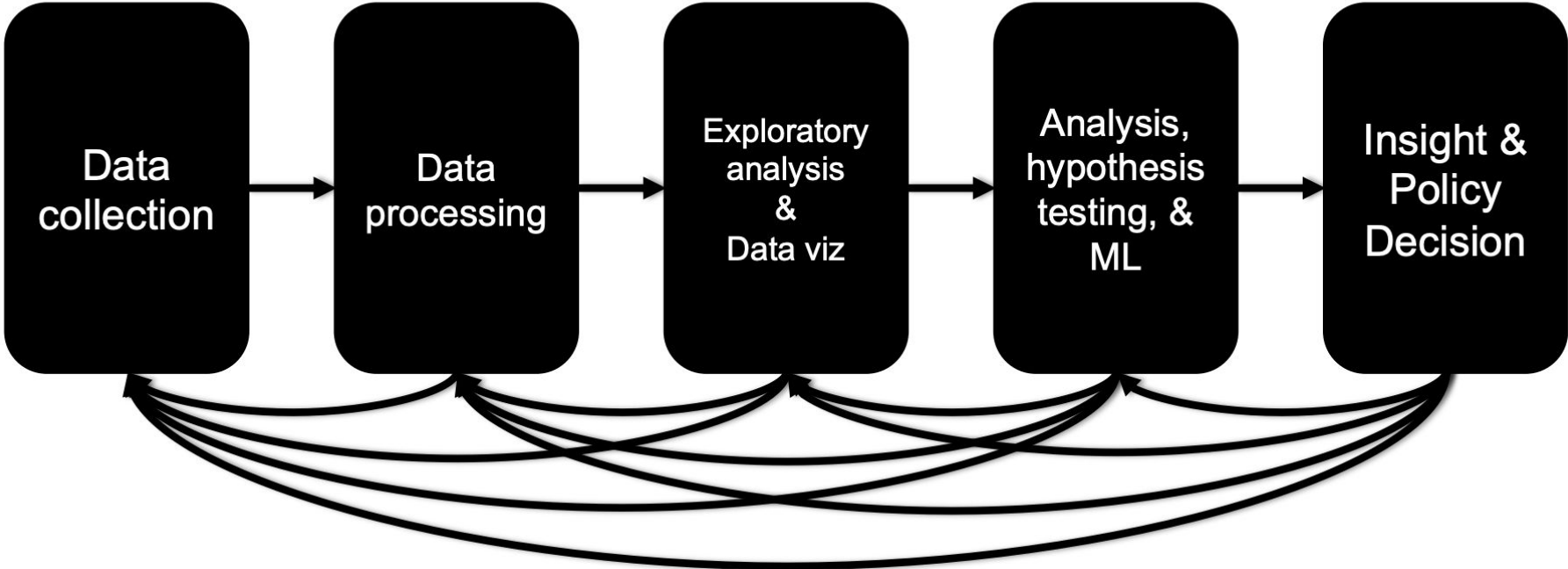


# Data Preprocessing +

## CSC 380 - Principles of Data Science

### Lecture 7.1

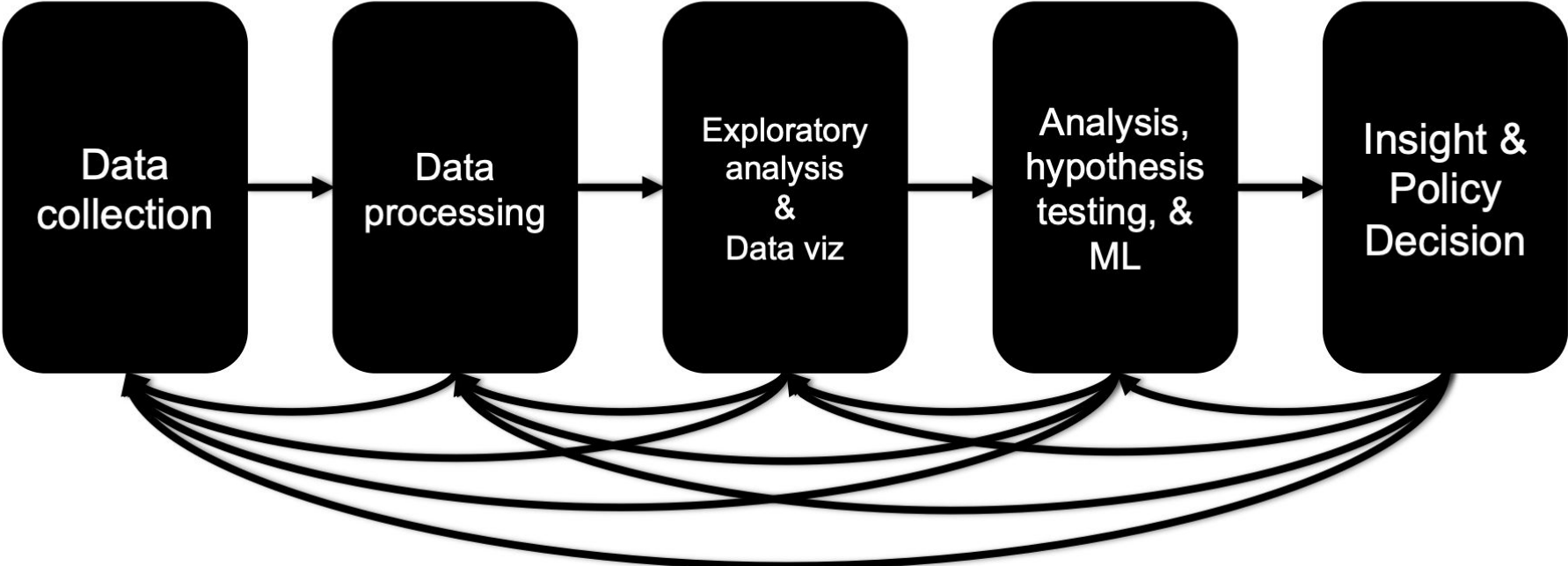
# So far in the course



# Data Collection

- Direct download and load from local storage
- Generate locally via downloaded code (e.g., simulation)
- Query data from a database.
- Query an API from the intra/internet
- Scrape data from a webpage

# So far in the course



# 1. Data Processing

Data Cleaning:

- Missing values, duplicates, type conversions, modification

Visited One ML Algorithm to understand why data is so very important.

# 1.1 Why

- **Accuracy:** To check whether the data entered is correct or not.
- **Completeness:** To check whether the data is available or not recorded.
- **Consistency:** To check whether the same data is kept in all the places that do or do not match.
- **Timeliness:** The data should be updated correctly.
- **Believability:** The data should be trustable.
- **Interpretability:** The understandability of the data.

## 1.2 Common Tasks

1. Cleaning
2. Integration
3. Transformation
4. Reduction
5. Discretization
6. Normalization

# 1.2 Common Tasks

## 1. Cleaning

a. Missing Data

b. Noisy Data

2. Integration

3. Transformation

4. Reduction

5. Discretization

6. Normalization



# 1.2.1 Cleaning

## a. Missing Data

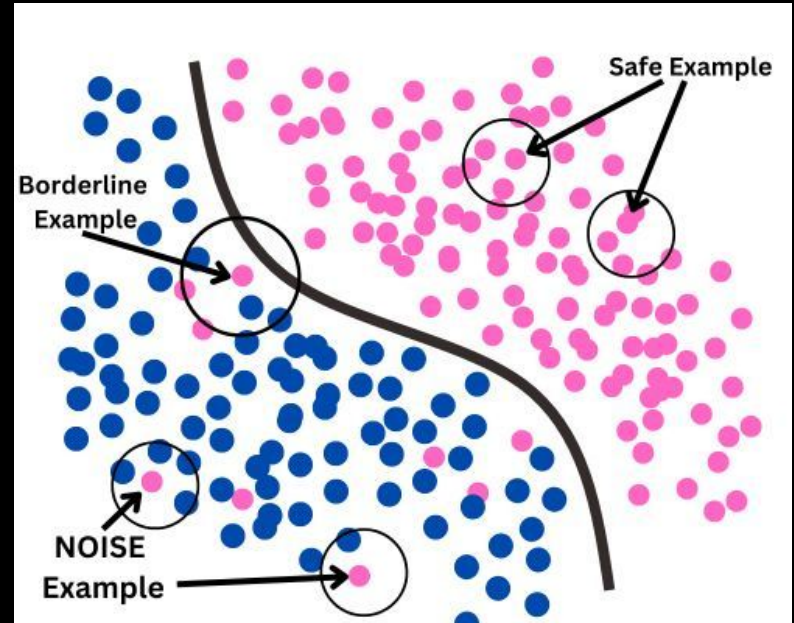
- Ignore that data point
- Fill the Missing values
  - Fill with “Not Available” or “NA”
  - Manually ( not recommended for large datasets)
  - Mean
  - Most probable ( esp regression or decision tree)

**Notebook**

## 1.2.1 Cleaning

### b. Noisy Data :

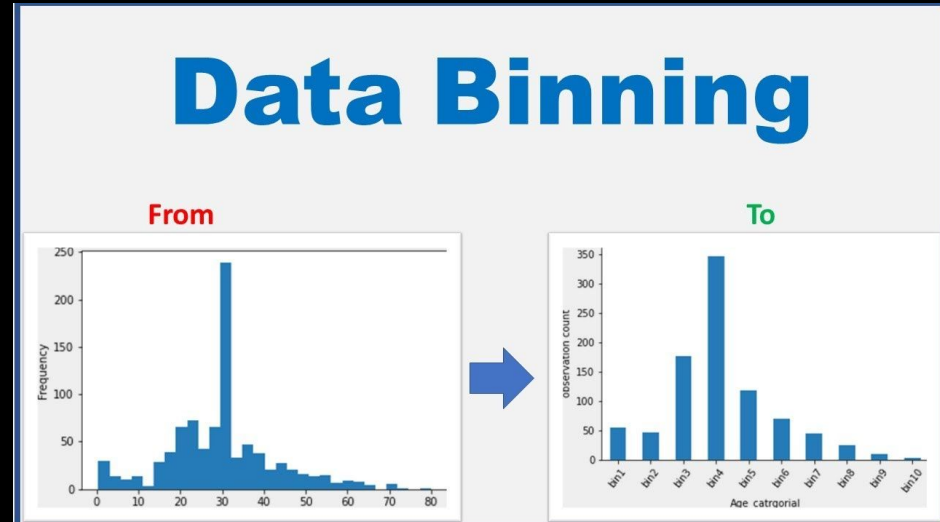
- Binning
- Regression
- Clustering



## 1.2.1 Cleaning

### b. Noisy Data :

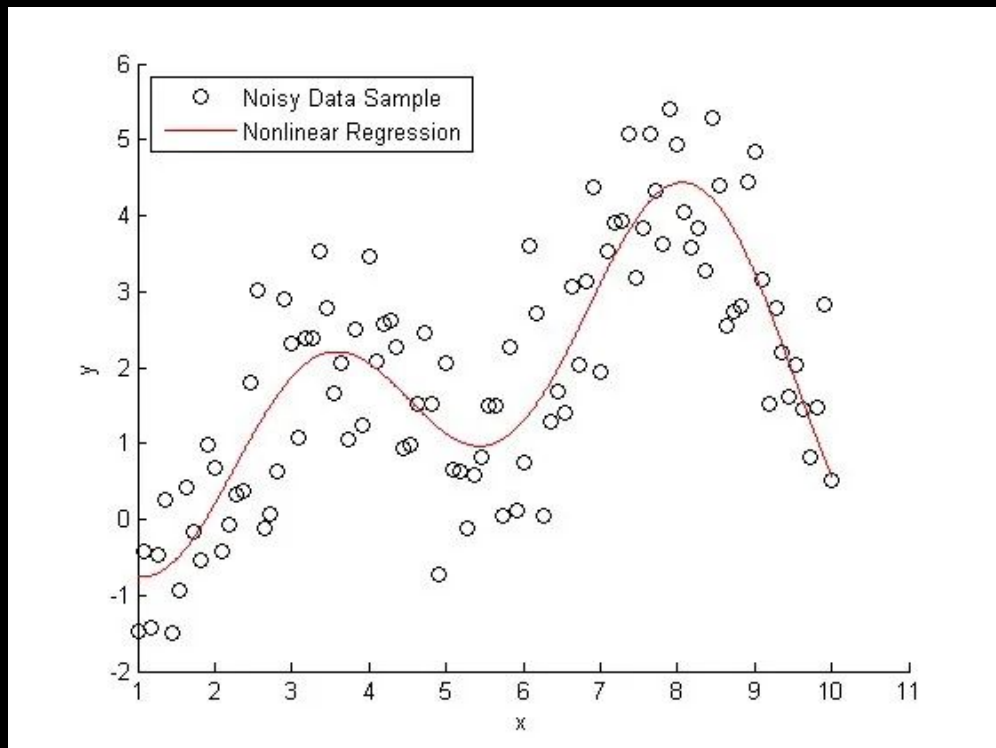
- **Binning:**
  - By mean, median and boundary
  - Regression
  - Clustering



## 1.2.1 Cleaning

### b. Noisy Data :

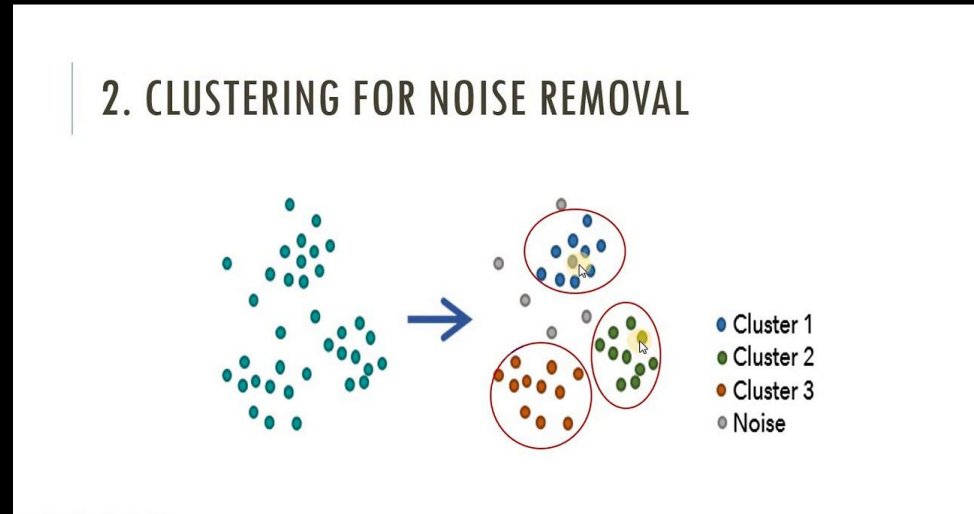
- Binning:
  - By mean, median and boundary
- **Regression**
- Clustering



# 1.2.1 Cleaning

## b. Noisy Data :

- Binning:
  - By mean, median and boundary
- Regression
- **Clustering**



# 1.2 Common Tasks

1. Cleaning
  - a. Missing Data
  - b. Noisy Data

## 2. Integration

3. Transformation
4. Reduction
5. Discretization
6. Normalization

## 1.2.2 Integration

Combining multiple sources into a single dataset.

- **Schema integration:** Integrates metadata (a set of data that describes other data) from different sources.
- **Entity identification problem:** Identifying entities from multiple databases.
- **Detecting and resolving data value concepts:** The data taken from different databases while merging may differ. The attribute values from one database may differ from another database. For example, the date format may differ, like “MM/DD/YYYY” or “DD/MM/YYYY”.



# 1.2 Common Tasks

1. Cleaning
  - a. Missing Data
  - b. Noisy Data
2. Integration

## 3. Transformation

4. Reduction
5. Discretization
6. Normalization

## 1.2.3 Transformation

The change made in the format or the structure of the data is called data transformation.

Discretization and Normalisation may also be seen as a type of transformation.

# 1.2 Common Tasks

1. Cleaning
  - a. Missing Data
  - b. Noisy Data

2. Integration

3. Transformation

## 4. Reduction

5. Discretization

6. Normalization

## 1.2.4 Data Reduction

Reducing the size of the dataset while preserving the important information.

*Curse of dimensionality*

*As the number of dimensions or features increases, the amount of data needed to generalize the machine learning model accurately increases exponentially*

[Excellent article with examples](#)

## 1.2.4 Data Reduction

Reducing the size of the dataset while preserving the important information.

- Feature Selection
- Feature Extraction
- Sampling
- Clustering
- Compression

## 1.2.4 Data Reduction

- Feature Selection
  - correlation analysis
  - mutual information
  - principal component analysis (PCA).
- Feature Extraction
- Sampling
- Clustering
- Compression

## 1.2.4 Data Reduction

- Feature Selection
  - correlation analysis
  - mutual information
  - principal component analysis (PCA)
- Dimensionality reduction, Numerosity Reduction, Data compression

# 1.2.4 Data Reduction

- Feature Selection

- correlation analysis:

Features with high correlation are more linearly dependent and hence have almost the same effect on the dependent variable. So, when two features have high correlation, we can drop one of the two features.

- mutual information

- principal component analysis (PCA).

- Feature Extraction

- Sampling

- Clustering

- Compression



# 1.2.4 Data Reduction

- **Feature Selection**
  - correlation analysis
  - **mutual information**
  - principal component analysis (PCA).
- Feature Extraction
- Sampling
- Clustering
- Compression

# Mutual Information

measures the amount of information one can obtain from one random variable given another.

$$MI(i,j) = \sum_{a,b} P(a_i, b_j) \cdot \log \left( \frac{P(a_i, b_j)}{P(a_i) \cdot P(b_j)} \right)$$

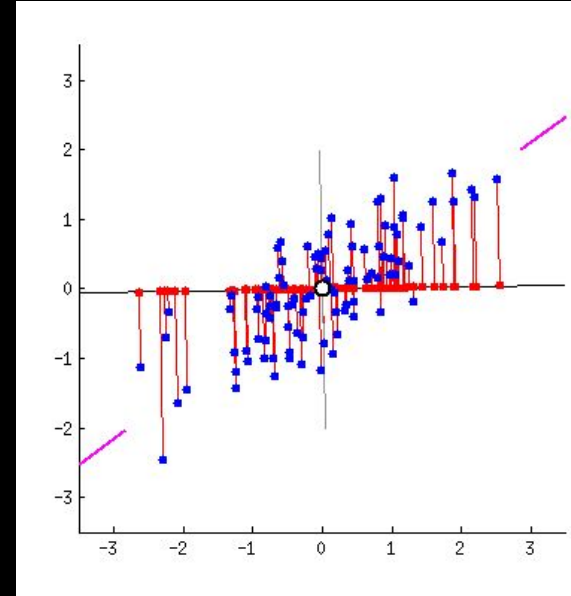
Sklearn.feature\_selection

- mutual\_info\_classif
- Mutual\_info\_regression

Fun easy to understand example: [https://www.youtube.com/watch?v=eJIp\\_mgVLwE](https://www.youtube.com/watch?v=eJIp_mgVLwE)

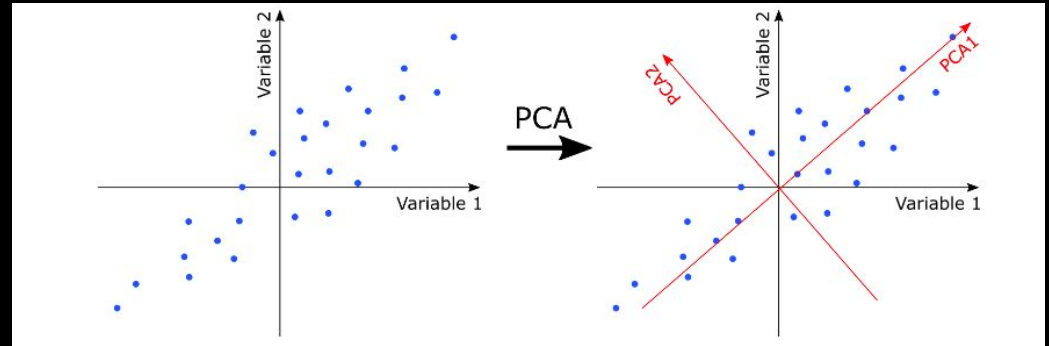
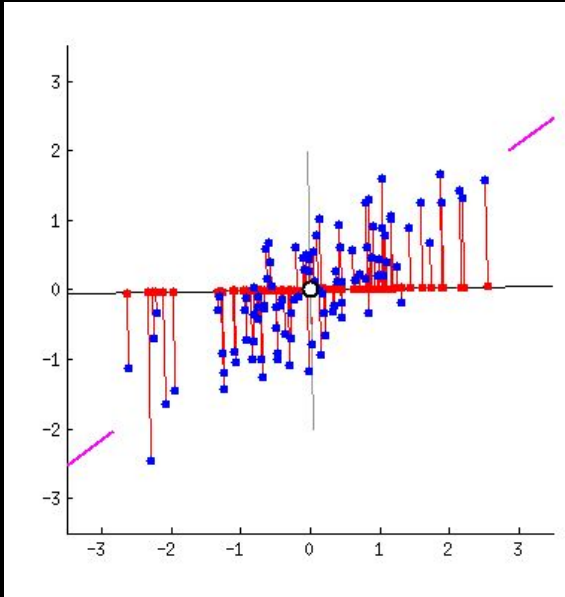
# 1.2.4 Data Reduction

- Feature Selection
  - correlation analysis
  - mutual information
  - **principal component analysis (PCA).**
- Feature Extraction
- Sampling
- Clustering
- Compression



# Principal component analysis (PCA)

Aim : find the directions of maximum variance in high-dimensional data and projects the data onto a new subspace with equal or fewer dimensions than the original one.



# Principal component analysis (PCA).

1. Take the whole dataset consisting of  $d+1$  dimensions and ignore the labels such that our new dataset becomes  $d$  dimensional
2. Compute the mean for every dimension of the whole dataset.
3. Compute the covariance matrix of the whole dataset.
4. Compute eigenvectors and the corresponding eigenvalues.
5. Sort the eigenvectors by decreasing eigenvalues and choose  $k$  eigenvectors with the largest eigenvalues to form a  $d \times k$  dimensional matrix  $W$ .
6. Use this  $d \times k$  eigenvector matrix to transform the samples onto the new subspace.

# Principal component analysis (PCA).

1. Take the whole dataset consisting of  $d+1$  dimensions and ignore the labels such that our new dataset becomes  $d$  dimensional

Student	Math	English	Art
1	90	60	90
2	90	90	30
3	60	60	60
4	60	60	90
5	30	30	30

$$\mathbf{A} = \begin{bmatrix} 90 & 60 & 90 \\ 90 & 90 & 30 \\ 60 & 60 & 60 \\ 60 & 60 & 90 \\ 30 & 30 & 30 \end{bmatrix}$$

# Principal component analysis (PCA).

2. Compute the mean for every dimension of the whole dataset.

$$\mathbf{A} = \begin{bmatrix} 90 & 60 & 90 \\ 90 & 90 & 30 \\ 60 & 60 & 60 \\ 60 & 60 & 90 \\ 30 & 30 & 30 \end{bmatrix}$$

$$\bar{\mathbf{A}} = [ 66 \quad 60 \quad 60 ]$$

# Principal component analysis (PCA).

3. Compute the covariance matrix of the whole dataset.

$$\mathbf{A} = \begin{bmatrix} 90 & 60 & 90 \\ 90 & 90 & 30 \\ 60 & 60 & 60 \\ 60 & 60 & 90 \\ 30 & 30 & 30 \end{bmatrix}$$

	<i>Math</i>	<i>English</i>	<i>Art</i>
<i>Math</i>	504	360	180
<i>English</i>	360	360	0
<i>Art</i>	180	0	720



# Principal component analysis (PCA).

4. Compute eigenvectors and the corresponding eigenvalues.

$$\mathbf{A} = \begin{bmatrix} 90 & 60 & 90 \\ 90 & 90 & 30 \\ 60 & 60 & 60 \\ 60 & 60 & 90 \\ 30 & 30 & 30 \end{bmatrix}$$

$$\begin{pmatrix} -3.75100... \\ 4.28441... \\ 1 \end{pmatrix}, \begin{pmatrix} -0.50494... \\ -0.67548... \\ 1 \end{pmatrix}, \begin{pmatrix} 1.05594... \\ 0.69108... \\ 1 \end{pmatrix}$$

# Principal component analysis (PCA).

5. Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a  $d \times k$  dimensional matrix W.

$$\begin{pmatrix} -3.75100... \\ 4.28441... \\ 1 \end{pmatrix}, \begin{pmatrix} -0.50494... \\ -0.67548... \\ 1 \end{pmatrix}, \begin{pmatrix} 1.05594... \\ 0.69108... \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} 910.06995 \\ 629.11039 \\ 44.81966 \end{pmatrix}$$

# Principal component analysis (PCA).

5. Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a  $d \times k$  dimensional matrix W.

$$\begin{pmatrix} 910.06995 \\ 629.11039 \\ 44.81966 \end{pmatrix}$$

$$W = \begin{bmatrix} 1.05594 & -0.50494 \\ 0.69108 & -0.67548 \\ 1 & 1 \end{bmatrix}$$

# Principal component analysis (PCA).

6. Use this  $d \times k$  eigenvector matrix ( $W$ ) to transform the samples onto the new subspace.

$y = W' \times x$  where  $W'$  is the transpose of the matrix  $W$ .

## sklearn.decomposition.PCA

```
class sklearn.decomposition.PCA(n_components=None, *, copy=True, whiten=False, svd_solver='auto', tol=0.0,
iterated_power='auto', n_oversamples=10, power_iteration_normalizer='auto', random_state=None) \[source\]
```

Principal component analysis (PCA).

Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space. The input data is centered but not scaled for each feature before applying the SVD.

It uses the LAPACK implementation of the full SVD or a randomized truncated SVD by the method of Halko et al. 2009, depending on the shape of the input data and the number of components to extract.

It can also use the `scipy.sparse.linalg` ARPACK implementation of the truncated SVD.

Notice that this class does not support sparse input. See [TruncatedSVD](#) for an alternative with sparse data.

Read more in the [User Guide](#).

<https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>

## 1.2.4 Data Reduction

- Feature Selection
  - correlation analysis
  - mutual information
  - principal component analysis (PCA).
- Feature Extraction
- Sampling
- Clustering
- Compression

# 1.2.4 Data Reduction

- Feature Selection
  - correlation analysis
  - mutual information
  - principal component analysis (PCA).
- **Feature Extraction**
- Sampling
- Clustering
- Compression

# Feature Extraction

- transforming the data into a lower-dimensional space while preserving the important information
  - PCA
  - linear discriminant analysis (LDA)
  - non-negative matrix factorization (NMF).

## sklearn.discriminant\_analysis.LinearDiscriminantAnalysis

```
class sklearn.discriminant_analysis.LinearDiscriminantAnalysis(solver='svd', shrinkage=None, priors=None, n_components=None, store_covariance=False, tol=0.0001, covariance_estimator=None) [source]
```

Linear Discriminant Analysis.

A classifier with a linear decision boundary, generated by fitting class conditional densities to the data and using Bayes' rule.

The model fits a Gaussian density to each class, assuming that all classes share the same covariance matrix.

The fitted model can also be used to reduce the dimensionality of the input by projecting it to the most discriminative directions, using the `transform` method.

New in version 0.17: `LinearDiscriminantAnalysis`.

Read more in the [User Guide](#).

## sklearn.decomposition.NMF

```
class sklearn.decomposition.NMF(n_components=None, *, init=None, solver='cd', beta_loss='frobenius', tol=0.0001, max_iter=200, random_state=None, alpha_W=0.0, alpha_H='same', l1_ratio=0.0, verbose=0, shuffle=False) [source]
```

Non-Negative Matrix Factorization (NMF).

Find two non-negative matrices, i.e. matrices with all non-negative elements, (W, H) whose product approximates the non-negative matrix X. This factorization can be used for example for dimensionality reduction, source separation or topic extraction.

The objective function is:

$$\begin{aligned} L(W, H) = & 0.5 * \|X - WH\|_{loss}^2 \\ & + \alpha_W * l1\_ratio * n\_features * \|\text{vec}(W)\|_1 \\ & + \alpha_H * l1\_ratio * n\_samples * \|\text{vec}(H)\|_1 \\ & + 0.5 * \alpha_W * (1 - l1\_ratio) * n\_features * \|W\|_{Fro}^2 \\ & + 0.5 * \alpha_H * (1 - l1\_ratio) * n\_samples * \|H\|_{Fro}^2 \end{aligned}$$



## 1.2.4 Data Reduction

- Feature Selection
  - correlation analysis
  - mutual information
  - principal component analysis (PCA).
- Feature Extraction
- **Sampling : selecting a subset of data points from the dataset**
- Clustering
- Compression

# 1.2.4 Data Reduction

- Feature Selection
  - correlation analysis
  - mutual information
  - principal component analysis (PCA).
- Feature Extraction
- Sampling
- **Clustering**
- Compression

# 1.2.4 Data Reduction

- Feature Selection
  - correlation analysis
  - mutual information
  - principal component analysis (PCA).
- Feature Extraction
- Sampling
- Clustering
- **Compression**

# 1.2 Common Tasks

1. Cleaning
  - a. Missing Data
  - b. Noisy Data
2. Integration
3. Transformation
4. Reduction
- 5. Discretization**
6. Normalization

## 1.2.5 Discretisation

- dividing continuous data into discrete categories or intervals.
- equal width binning, equal frequency binning, and clustering.

### sklearn.preprocessing.KBinsDiscretizer

```
class sklearn.preprocessing.KBinsDiscretizer(n_bins=5, *, encode='onehot', strategy='quantile', dtype=None, subsample='warn', random_state=None) \[source\]
```

Bin continuous data into intervals.

Read more in the [User Guide](#).

# 1.2 Common Tasks

1. Cleaning
  - a. Missing Data
  - b. Noisy Data
2. Integration
3. Transformation
4. Reduction
5. Discretization
- 6. Normalization**

## 1.2.6 Normalisation

- This involves scaling the data to a common range, such as between 0 and 1 or -1 and 1.
- Handle data with different units and scales.

### `sklearn.preprocessing.StandardScaler`

```
class sklearn.preprocessing.StandardScaler(*, copy=True, with_mean=True, with_std=True) \[source\]
```

Standardize features by removing the mean and scaling to unit variance.

### `sklearn.preprocessing.MinMaxScaler`

```
class sklearn.preprocessing.MinMaxScaler(feature_range=(0, 1), *, copy=True, clip=False) \[source\]
```

Transform features by scaling each feature to a given range.

# 1.2 Common Tasks

1. Cleaning
  - a. Missing Data
  - b. Noisy Data
2. Integration
3. Transformation
4. Reduction
5. Discretization
6. Normalization
- 7. Encoding**



## 1.2.6 Encoding

Technique of converting categorical variables into numerical values so that it could be easily fitted to a machine learning model.


- Ordinal Categorical variables
- Nominal categorical variables:
  - One hot encoding
  - Label Encoding

# Ordinal Categorical variables

Original Encoding	Ordinal Encoding
Poor	1
Good	2
Very Good	3
Excellent	4

[Source](#)

# Nominal - Label Encoding



State (Nominal Scale)
Maharashtra
Tamil Nadu
Delhi
Karnataka
Gujarat
Uttar Pradesh

State (Label Encoding)
3
4
0
2
1
5

`sklearn.preprocessing.LabelEncoder`

```
class sklearn.preprocessing.LabelEncoder
```

[\[source\]](#)

Encode target labels with value between 0 and `n_classes-1`.

[Source](#)

# Nominal - One hot encoding

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50



One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

[Source](#)

## sklearn.preprocessing.OneHotEncoder

```
class sklearn.preprocessing.OneHotEncoder(*, categories='auto', drop=None, sparse='deprecated',  
sparse_output=True, dtype=<class 'numpy.float64'>, handle_unknown='error', min_frequency=None, max_categories=None,  
feature_name_combiner='concat')
```

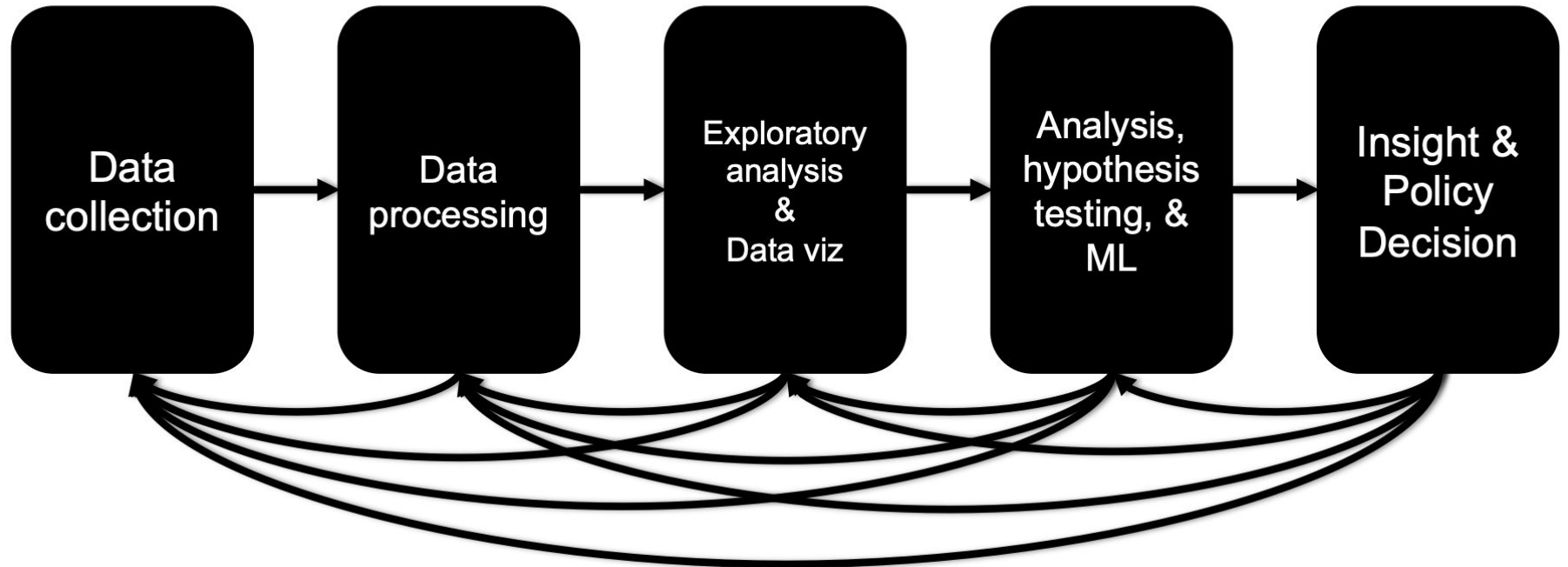
[\[source\]](#)

Encode categorical features as a one-hot numeric array.

## Resources:

- [Analytics Vidya : Data Preprocessing in Data Mining – A Hands On Guide \(Updated 2023\)](#)
- [Geeks for Geeks : Data Preprocessing in Data Mining](#)

# Data Science Process

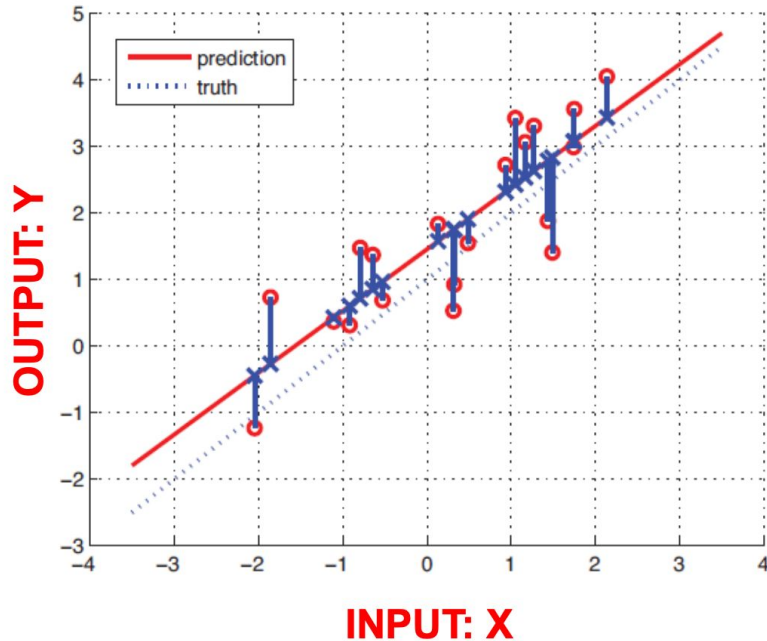


# Supervised Machine Learning Models

## Linear models

Slides adapted from Prof. Jason Pachecos slides : CSC 380 Fall 2021

# Linear Regression



**Regression** Learn a function that predicts outputs from inputs,

$$y = f(x)$$

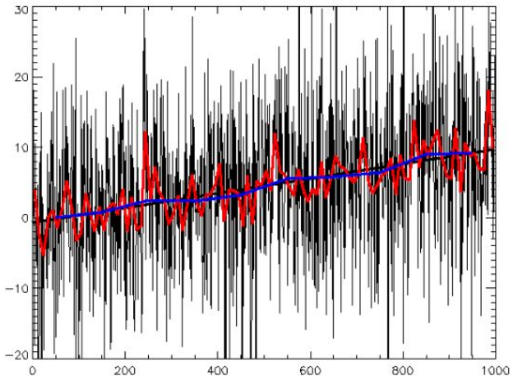
Outputs  $y$  are real-valued

**Linear Regression** As the name suggests, uses a *linear function*:

$$y = w^T x + b$$



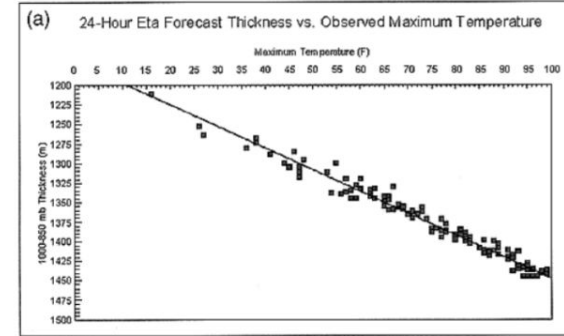
# Where is linear regression useful?



Trendlines

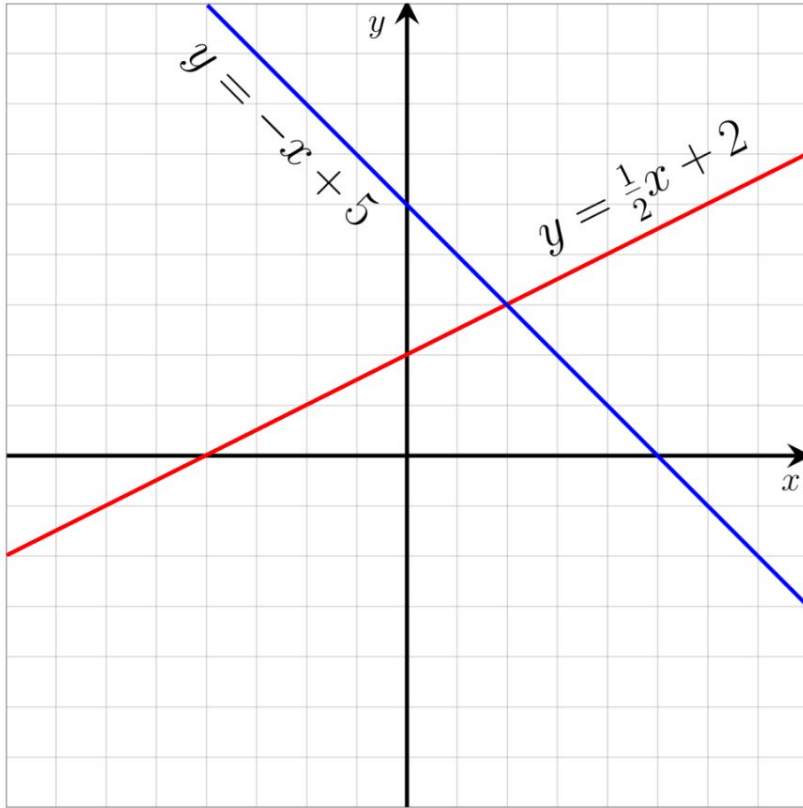


Stock Prediction



Climate Models  
*Massie and Rose (1997)*

*Used anywhere a linear relationship is assumed  
between continuous inputs / outputs*



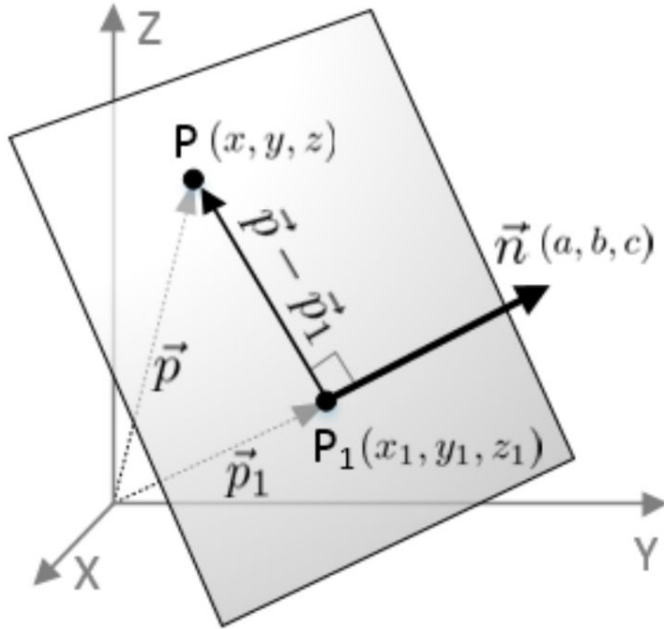
Recall the equation for a line has a *slope* and an *intercept*,

$$y = w \cdot x + b$$

**Slope**      **Intercept**

- Intercept (b) indicates where line crosses y-axis
- Slope controls angle of line
- Positive slope (w) → Line goes up left-to-right
- Negative slope → Line goes down left-to-right

In higher dimensions Line  $\rightarrow$  Plane



Multiple ways to define a plane, we will use:

$$n^T (p - p_1) = 0$$

**Normal Vector**  
(controls orientation)

**In-Plane Vector**  
(handles offset)

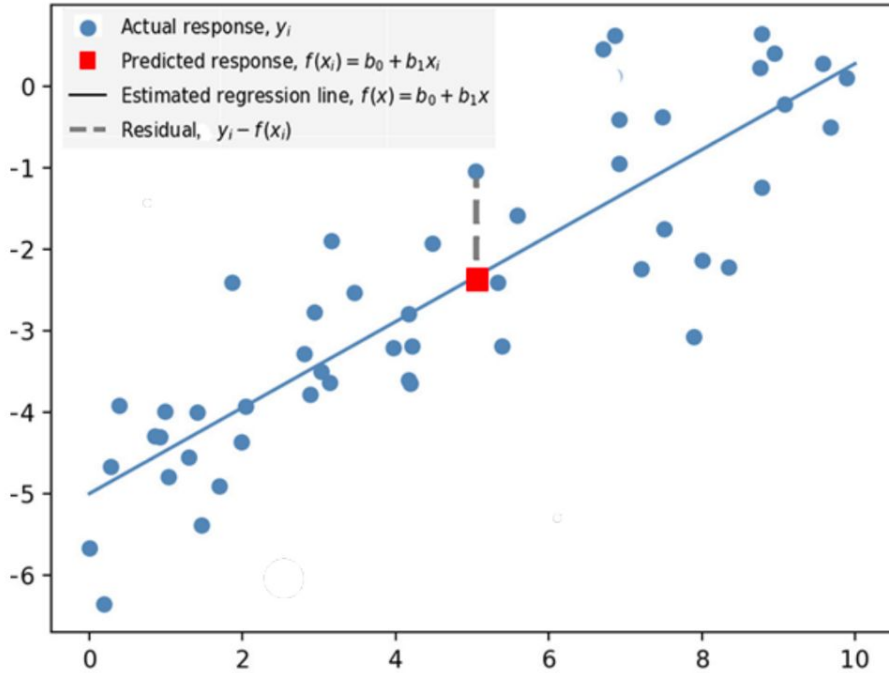
*Regression weights will take place of normal vector*

## There are several ways to think about fitting regression:

- **Intuitive** Find a plane/line that is close to data
- **Functional** Find a line that minimizes the *least squares* loss
- **Estimation** Find maximum likelihood estimate of parameters

*They are all the same thing...*

---



**Intuition** Find a line that is as *close as possible* to every training data point

The distance from each point to the line is the **residual**

$$y - w^T x$$

Training Output      Prediction

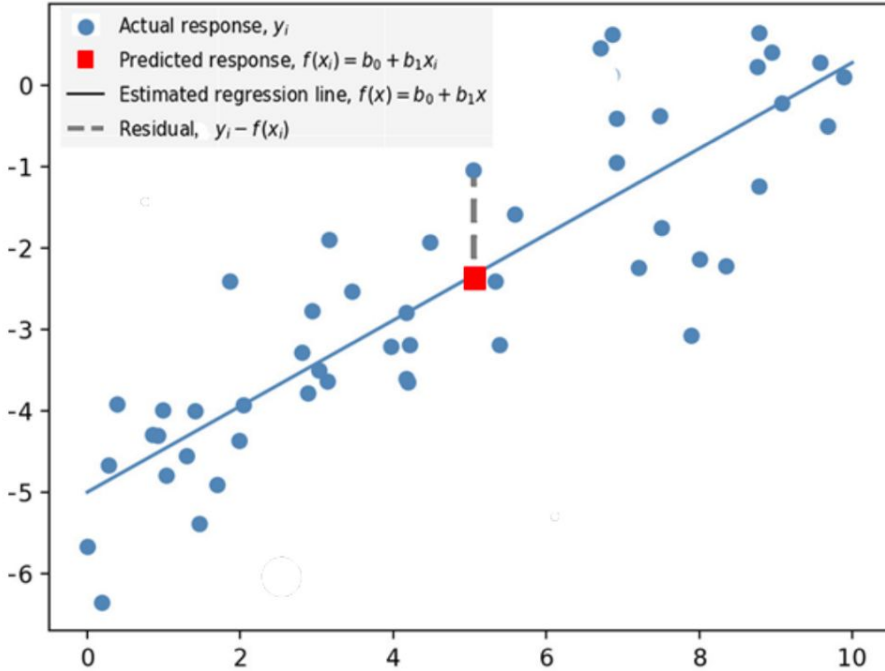
**Functional** Find a line that minimizes the sum of squared residuals

$$w^* = \arg \min_w \sum_{i=1}^N (y_i - w^T x_i)^2$$

Over training all the data,

$$\{(x_i, y_i)\}_{i=1}^N$$

**Least squares regression**



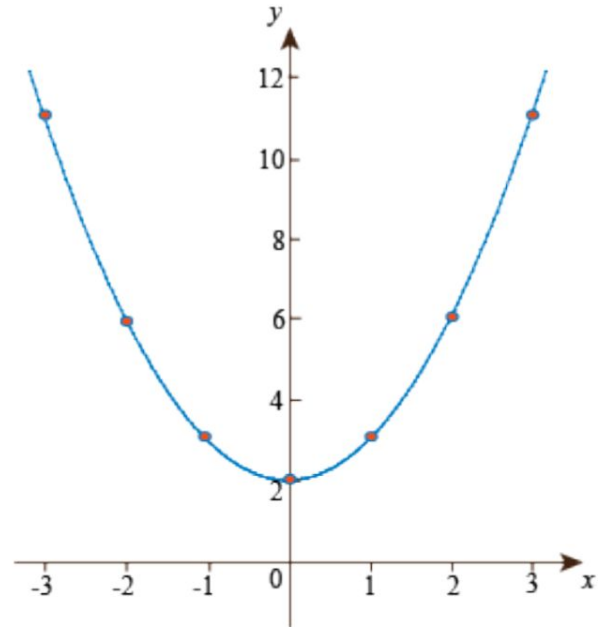
$$\min_w \sum_{i=1}^N (y_i - w^T x_i)^2$$

**This is just a quadratic function...**

- *Convex*, unique minimum
- Minimum given by zero-derivative
- Can find a closed-form solution

Let's see for scalar case with no bias,

$$y = wx$$



---

## There are several ways to think about fitting regression:

- **Intuitive** Find a plane/line that is close to data
- **Functional** Find a line that minimizes the *least squares* loss
- **Estimation** Find maximum likelihood estimate of parameters



# Linear Regression Summary

1. Definition of linear regression model,

$$y = w^T x + \epsilon \quad \text{where} \quad \epsilon \sim \mathcal{N}(0, \sigma^2 I)$$

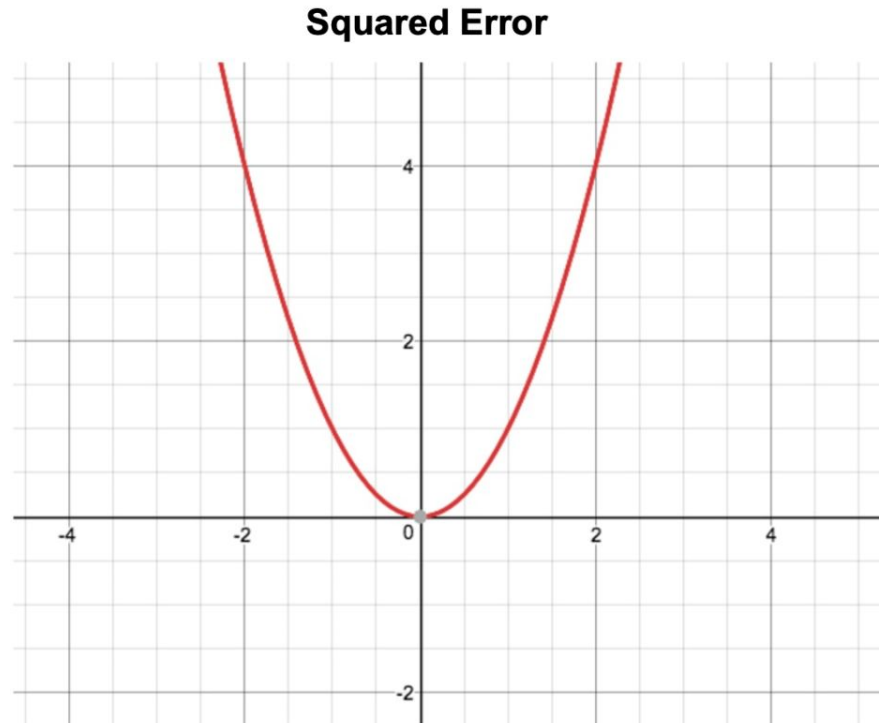
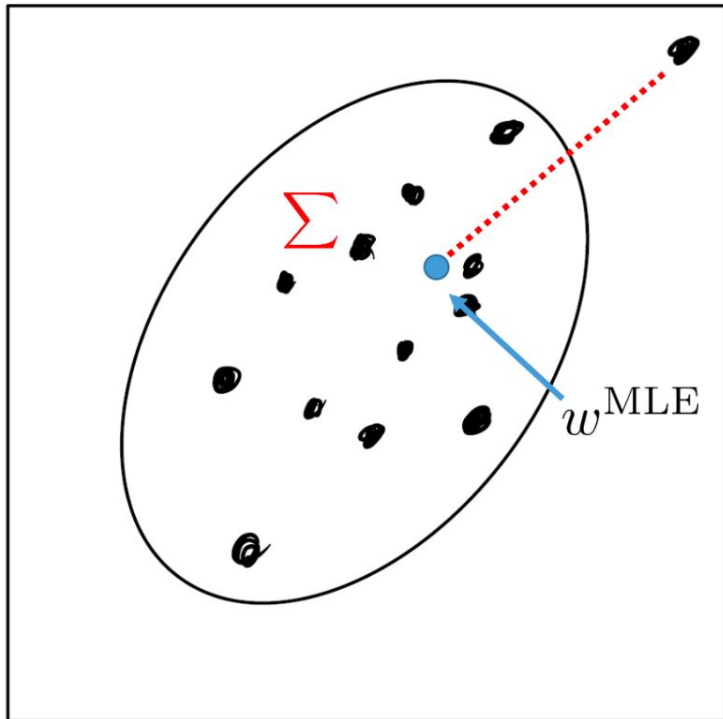
2. For N iid training data fit using least squares,

$$w^{\text{OLS}} = \arg \min_w \sum_{i=1}^N (y_i - w^T x_i)^2$$

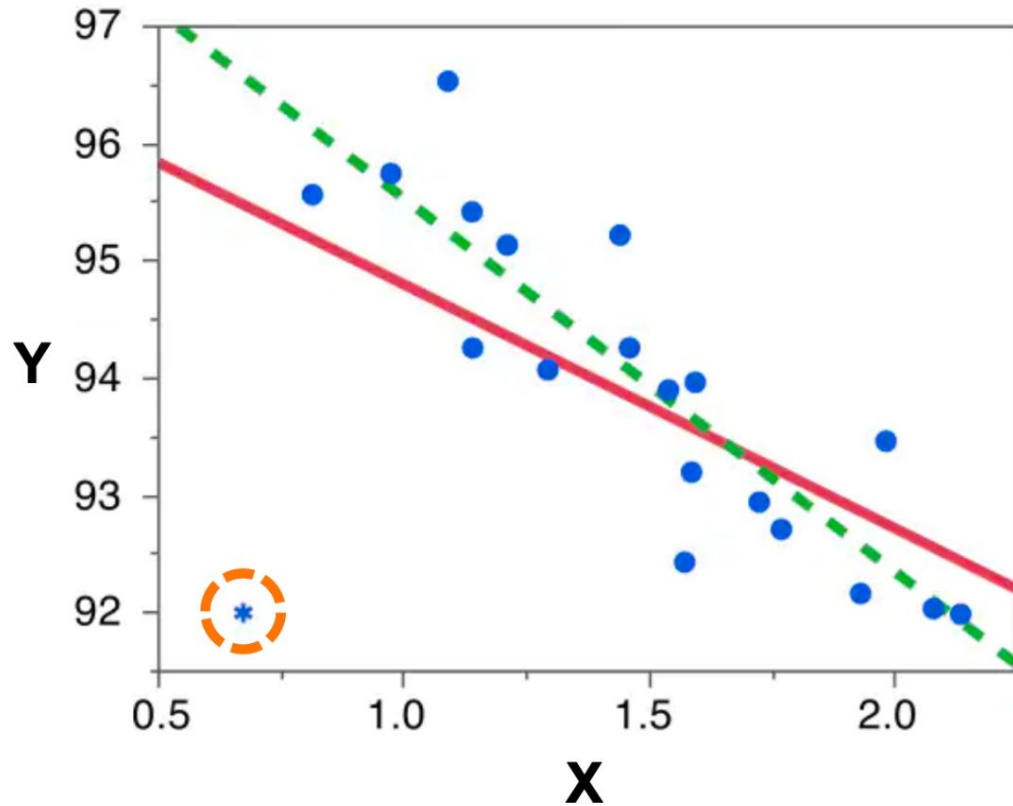
3. Equivalent to maximum likelihood solution

# Outliers

*How does an outlier affect the estimator?*



# Outliers in Linear Regression



Outlier “pulls”  
regression line away  
from inlier data

Need a way to *ignore* or  
to *down-weight* impact  
of outlier

# Dealing with Outliers

*Too many outliers can indicate many things: non-Gaussian (heavy-tailed) data, corrupt data, bad data collection, ...*

A few ways to handle outliers...

1. Use a heavy-tailed noise distribution (Student's T)

*Fitting regression becomes difficult*

2. Identify outliers and discard them

*NP-Hard and throwing away data is generally bad*

3. Penalize large weights to avoid overfitting (Regularization)

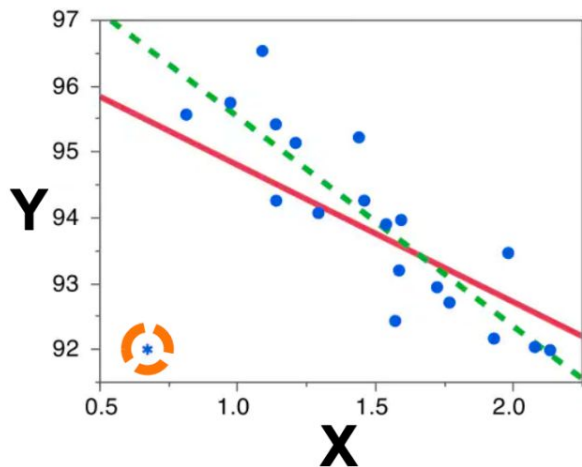
# Regularisation

- Reducing Model Complexity:
  - L1/L2 Regularisation
  - Dropout
  - Early stopping
- Data Augmentation

# Regularization

*Recall, regularization helps avoid overfitting training data...*

$$\text{Model} = \min_{\text{model}} \text{Loss}(\text{Model}, \text{Data}) + \lambda \cdot \text{Regularizer}(\text{Model})$$



**Regularization  
Strength**

**Regularization Penalty**

**Red** model is without regularization

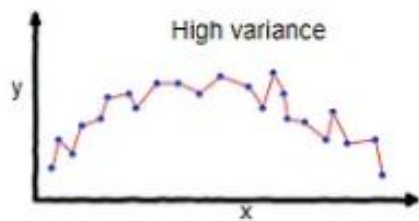
**Green** model includes regularization

# Bias-Variance Tradeoff

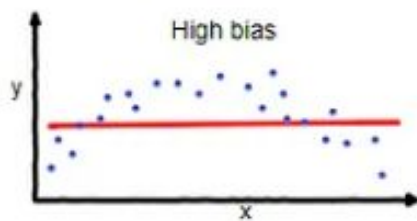
Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.

Variance is the variability of model prediction for a given data point or a value which tells us spread of our data. Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before. As a result, such models perform very well on training data but has high error rates on test data.

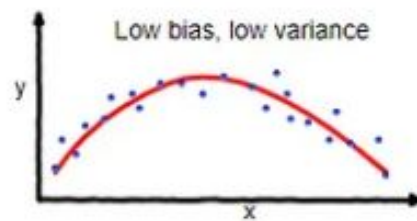
[Source : Towards Data Science - Understanding the Bias-Variance Tradeoff](#)



**overfitting**



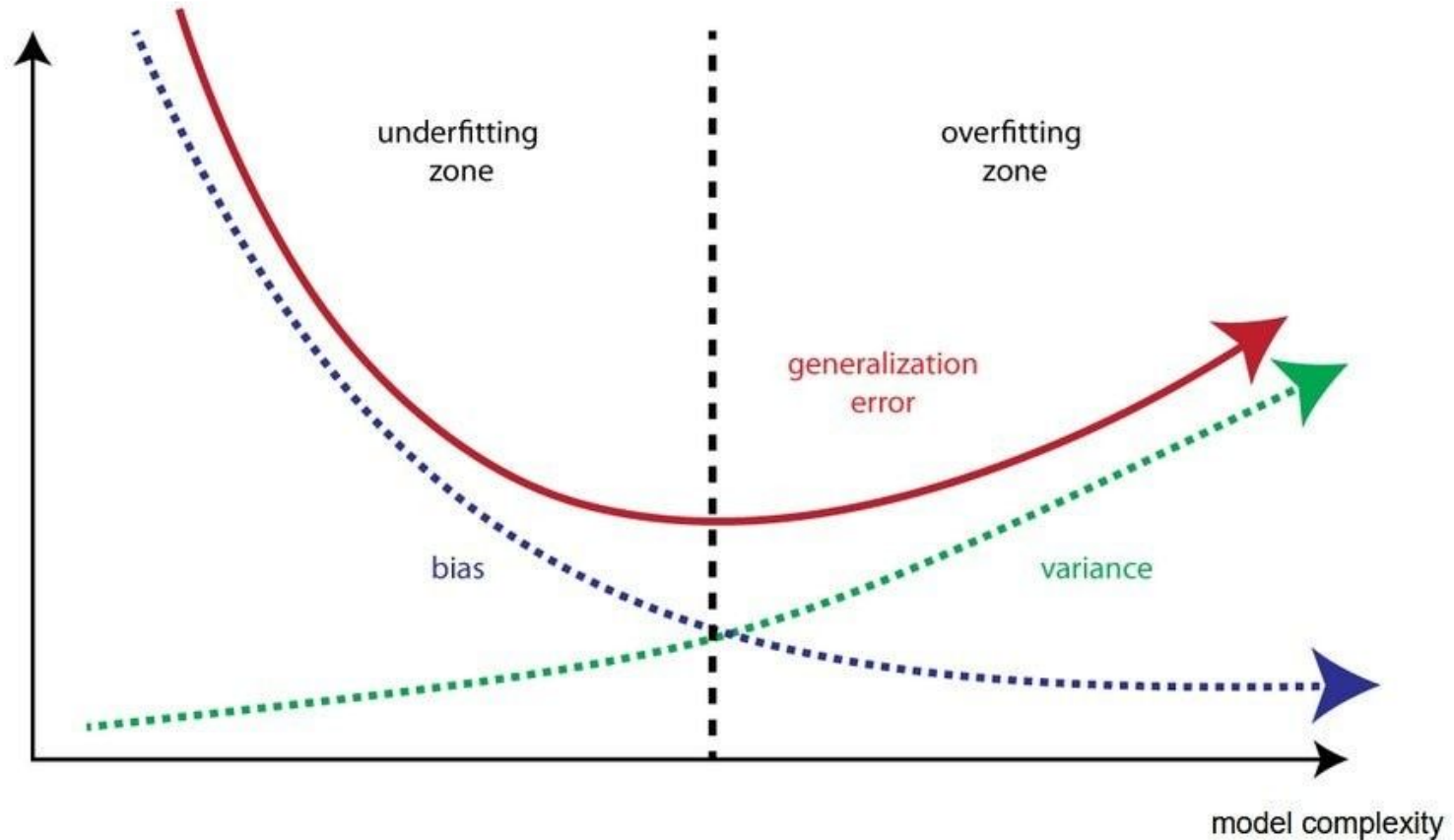
**underfitting**



**Good balance**



## the bias vs. variance trade-off



# Choosing Regularization Strength

*We need to tune regularization strength to avoid over/under fitting...*

$$w^{\text{L2}} = \arg \min_w \sum_{i=1}^N (y_i - w^T x_i)^2 + \frac{\lambda}{2} \|w\|^2$$

# Evaluation

- MSE
- $R^2$  - Coefficient of Determination ( I said last week, we will revisit this )

**Residual Sum-of-squared Errors** The total squared residual error on the held-out validation set,

$$\text{RSS} = \sum_{i=1}^N (y_i - w^T x_i)^2$$

**Coefficient of Determination** Also called R-squared or  $R^2$ .  
Fraction of variation explained by the model.

*Model selection metrics are known as “goodness of fit” measures*

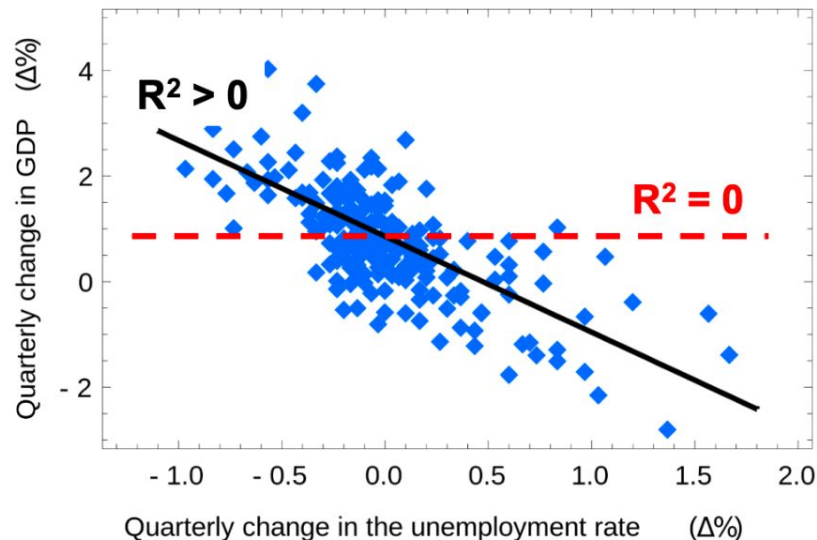
# Coefficient of Determination $R^2$

$$R^2 = 1 - \frac{\text{RSS}}{\text{SS}} = 1 - \frac{\sum_{i=1}^N (y_i - w^T x_i)^2}{\sum_{i=1}^N (y_i - \bar{y})^2}$$

Maximum value  $R^2=1.0$  means model explains *all variation* in the data

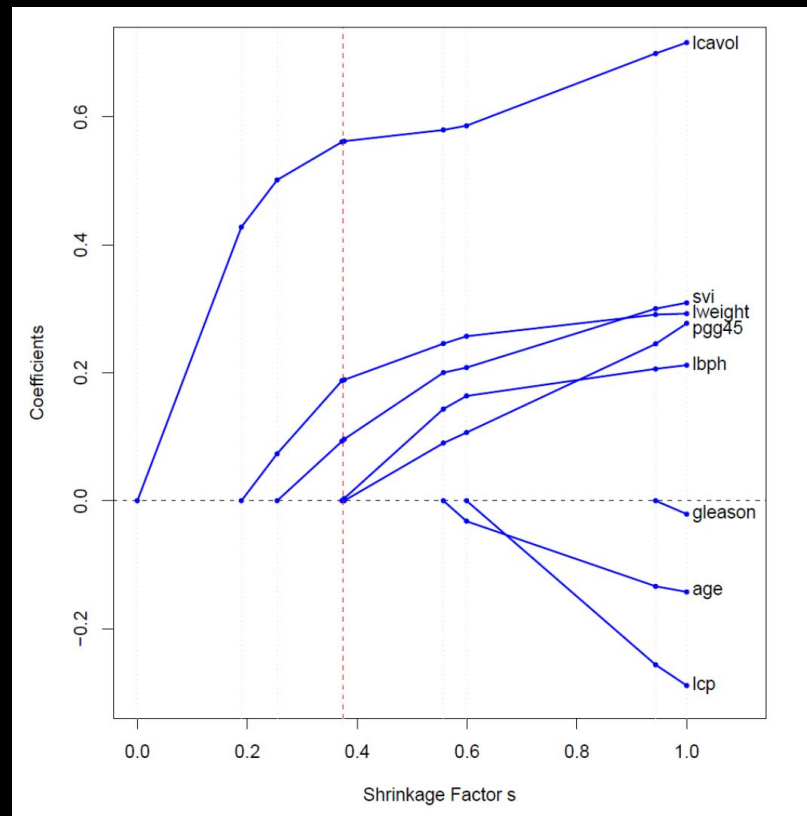
Maximum value  $R^2=0$  means model is as good as predicting average response

$R^2 < 0$  means model worse than predicting average output

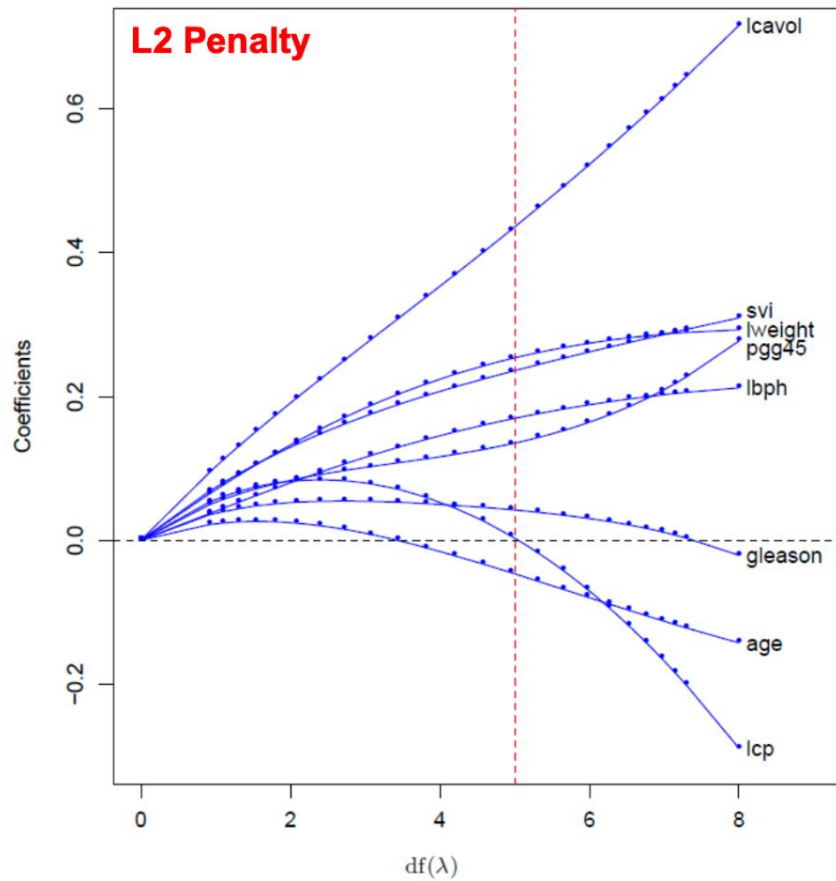
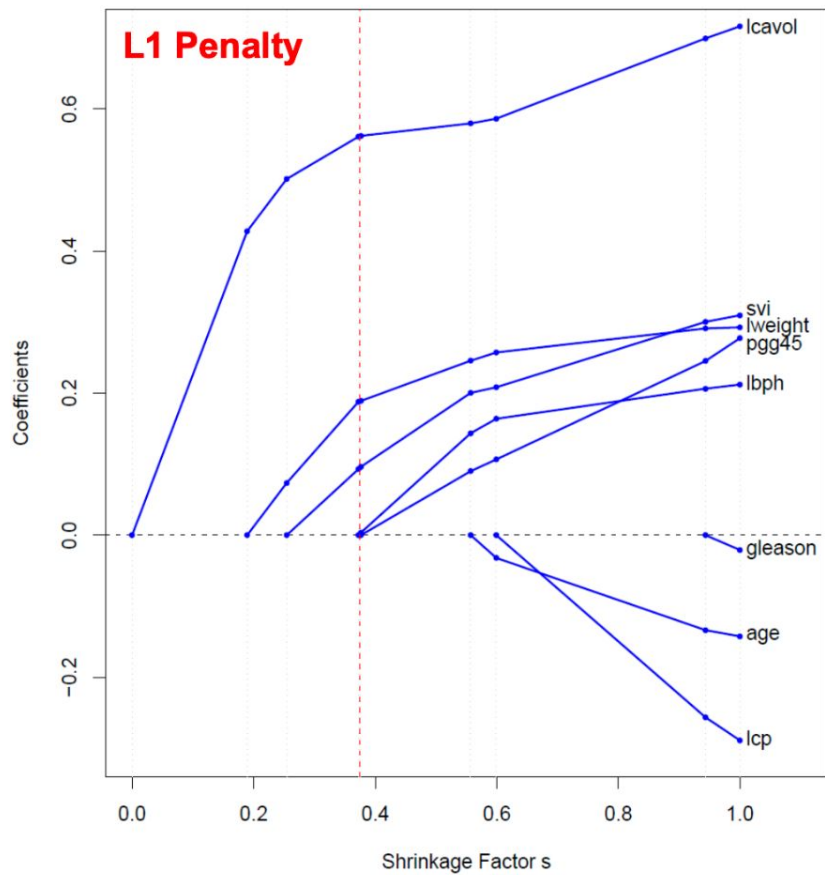


# Using Regularisation for Feature Selection

Term	LS	Ridge	Lasso
Intercept	2.465	2.452	2.468
lcavol	0.680	0.420	0.533
lweight	0.263	0.238	0.169
age	-0.141	-0.046	0.000
lbph	0.210	0.162	0.002
svi	0.305	0.227	0.094
lcp	-0.288	0.000	0.000
gleason	-0.021	0.040	0.000
pgg45	0.267	0.133	0.000



# Feature Weight Profiles



# Best-Subset Selection

*L1 / L2 shrinkage offer approximate feature selection...*

The optimal strategy for  $p$  features looks at models over *all possible combinations* of features,

```
For k in 1, ..., p:  
    subset = Compute all subset of k-features (p-choose-k)  
    For kfeat in subset:  
        model = Train model on kfeat features  
        score = Evaluate model using cross-validation  
Choose the model with best cross-validation score
```



# Feature Selection: Prostate Cancer Dataset

Best subset has highest test accuracy (lowest variance) with just 2 features

Term	LS	Best Subset	Ridge	Lasso
Intercept	2.465	2.477	2.452	2.468
lcavol	0.680	0.740	0.420	0.533
lweight	0.263	0.316	0.238	0.169
age	-0.141		-0.046	
lbph	0.210		0.162	0.002
svi	0.305		0.227	0.094
lcp	-0.288		0.000	
gleason	-0.021		0.040	
pgg45	0.267		0.133	
Test Error	0.521	0.492	0.492	0.479
Std Error	0.179	0.143	0.165	0.164

# Sequential Feature Selection

Forward Sequential Selection

Backward Sequential Selection

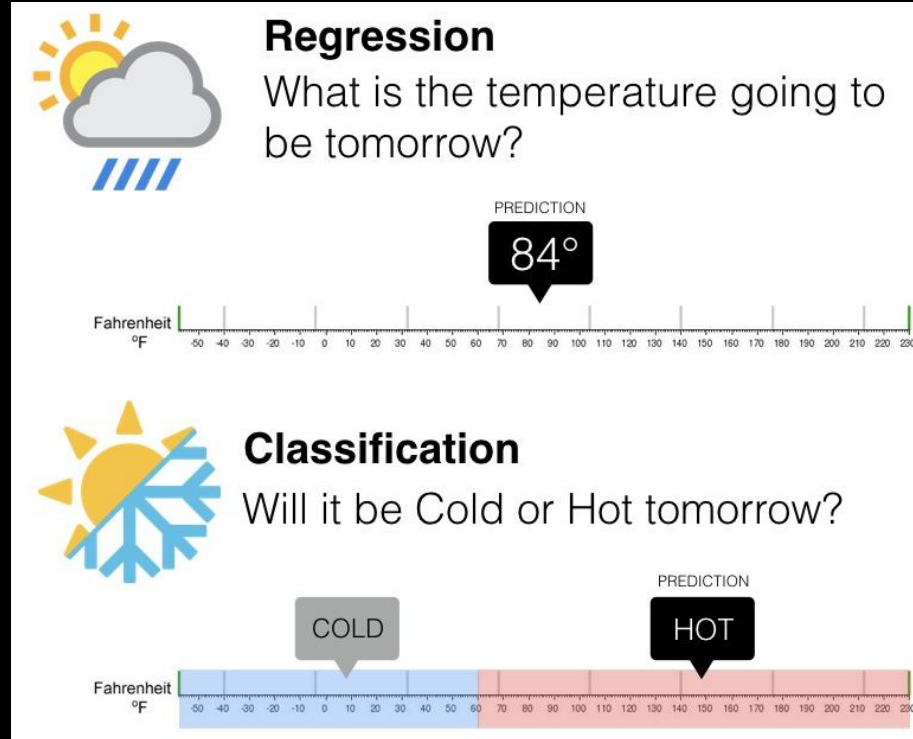
## `sklearn.feature_selection.SequentialFeatureSelector`

```
class sklearn.feature_selection.SequentialFeatureSelector(estimator, *, n_features_to_select='auto', tol=None, direction='forward', scoring=None, cv=5, n_jobs=None)
```

[\[source\]](#)

Transformer that performs Sequential Feature Selection.

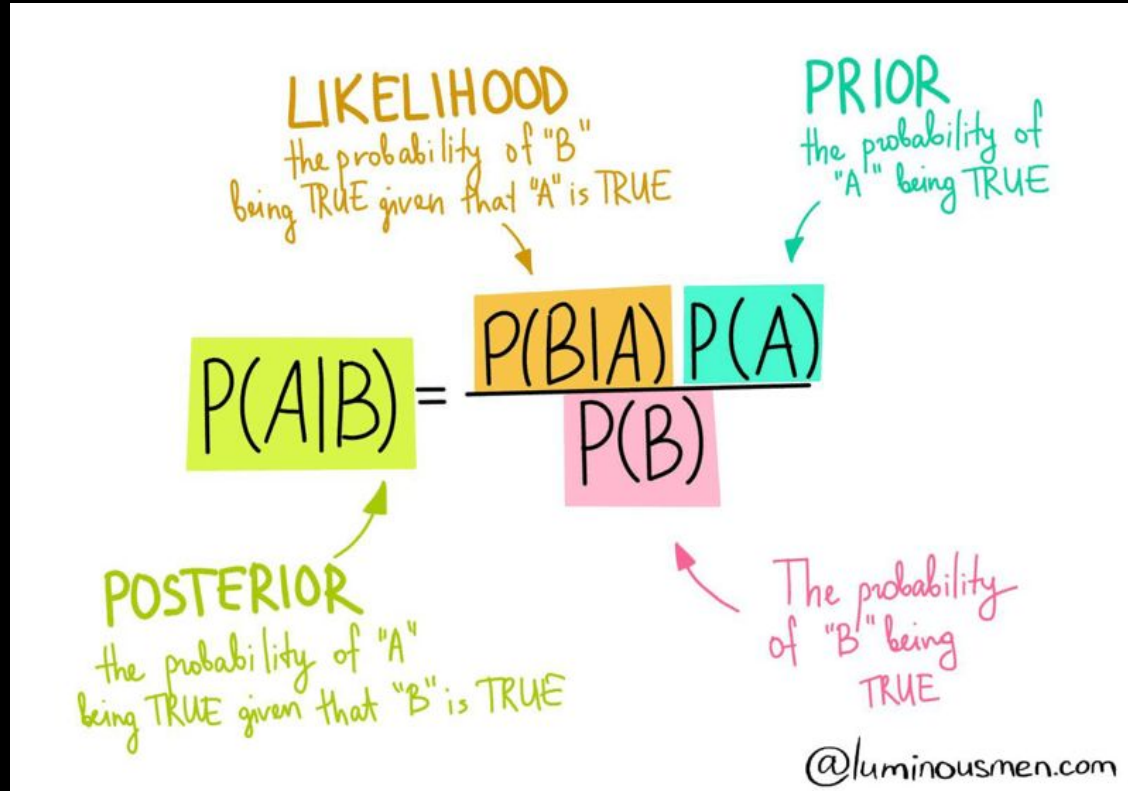
# Regression versus Classification



# Supervised Machine Learning Models

## Non-Linear models

# Naïve Bayes classifiers



Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

$P(x | c) = P(\text{Sunny} | \text{Yes}) = 3 / 9 = 0.33$

Frequency Table		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	
	Overcast	4	0	
	Rainy	2	3	

→

Likelihood Table		Play Golf		
		Yes	No	
Outlook	Sunny	3/9	2/5	5/14
	Overcast	4/9	0/5	4/14
	Rainy	2/9	3/5	5/14
		9/14	5/14	

$P(x) = P(\text{Sunny}) = 5 / 14 = 0.36$

$P(c) = P(\text{Yes}) = 9 / 14 = 0.64$

Posterior Probability:  $P(c | x) = P(\text{Yes} | \text{Sunny}) = 0.33 \times 0.64 \div 0.36 = 0.60$

$P(x | c) = P(\text{Sunny} | \text{No}) = 2 / 5 = 0.4$

Frequency Table		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
		9	5	14

$P(x) = P(\text{Sunny}) = 5 / 14 = 0.36$

$P(c) = P(\text{No}) = 5 / 14 = 0.36$

Posterior Probability:  $P(c | x) = P(\text{No} | \text{Sunny}) = 0.40 \times 0.36 \div 0.36 = 0.40$

### Frequency Table

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3



### Likelihood Table

		Play Golf	
		Yes	No
Outlook	Sunny	3/9	2/5
	Overcast	4/9	0/5
	Rainy	2/9	3/5

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1



		Play Golf	
		Yes	No
Humidity	High	3/9	4/5
	Normal	6/9	1/5

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1



		Play Golf	
		Yes	No
Temp.	Hot	2/9	2/5
	Mild	4/9	2/5
	Cool	3/9	1/5

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3



		Play Golf	
		Yes	No
Windy	False	6/9	2/5
	True	3/9	3/5

Outlook	Temp	Humidity	Windy	Play
Rainy	Cool	High	True	?

$$P(\text{Yes} | X) = P(\text{Rainy} | \text{Yes}) \times P(\text{Cool} | \text{Yes}) \times P(\text{High} | \text{Yes}) \times P(\text{True} | \text{Yes}) \times P(\text{Yes})$$

$$P(\text{Yes} | X) = 2/9 \times 3/9 \times 3/9 \times 3/9 \times 9/14 = 0.00529 \rightarrow 0.2 = \frac{0.00529}{0.02057 + 0.00529}$$

$$P(\text{No} | X) = P(\text{Rainy} | \text{No}) \times P(\text{Cool} | \text{No}) \times P(\text{High} | \text{No}) \times P(\text{True} | \text{No}) \times P(\text{No})$$

$$P(\text{No} | X) = 3/5 \times 1/5 \times 4/5 \times 3/5 \times 5/14 = 0.02057 \rightarrow 0.8 = \frac{0.02057}{0.02057 + 0.00529}$$



[Prev](#) [Up](#) [Next](#)

**scikit-learn 1.3.0**  
[Other versions](#)

Please [cite us](#) if you use the software.

- 1.9. Naive Bayes**
- [1.9.1. Gaussian Naive Bayes](#)
- [1.9.2. Multinomial Naive Bayes](#)
- [1.9.3. Complement Naive Bayes](#)
- [1.9.4. Bernoulli Naive Bayes](#)
- [1.9.5. Categorical Naive Bayes](#)
- [1.9.6. Out-of-core naive Bayes model fitting](#)

## 1.9. Naive Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. Bayes' theorem states the following relationship, given class variable  $y$  and dependent feature vector  $\mathbf{x}_1$  through  $\mathbf{x}_n$  :

$$P(y | \mathbf{x}_1, \dots, \mathbf{x}_n) = \frac{P(y)P(\mathbf{x}_1, \dots, \mathbf{x}_n | y)}{P(\mathbf{x}_1, \dots, \mathbf{x}_n)}$$

Using the naive conditional independence assumption that

$$P(\mathbf{x}_i | y, \mathbf{x}_1, \dots, \mathbf{x}_{i-1}, \mathbf{x}_{i+1}, \dots, \mathbf{x}_n) = P(\mathbf{x}_i | y),$$

for all  $i$ , this relationship is simplified to

$$P(y | \mathbf{x}_1, \dots, \mathbf{x}_n) = \frac{P(y) \prod_{i=1}^n P(\mathbf{x}_i | y)}{P(\mathbf{x}_1, \dots, \mathbf{x}_n)}$$

Since  $P(\mathbf{x}_1, \dots, \mathbf{x}_n)$  is constant given the input, we can use the following classification rule:

$$P(y | \mathbf{x}_1, \dots, \mathbf{x}_n) \propto P(y) \prod_{i=1}^n P(\mathbf{x}_i | y)$$

$$\Downarrow$$

$$\hat{y} = \arg \max_y P(y) \prod_{i=1}^n P(\mathbf{x}_i | y),$$

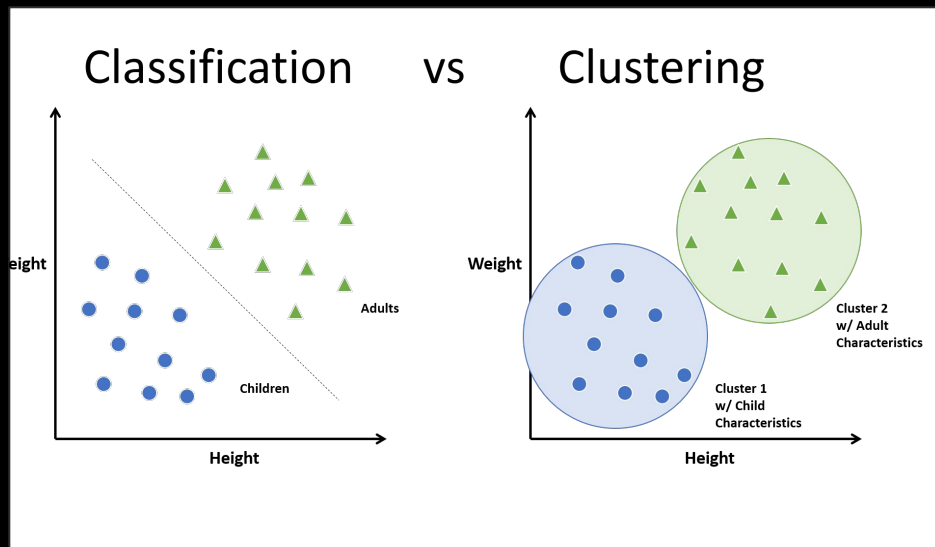
and we can use Maximum A Posteriori (MAP) estimation to estimate  $P(y)$  and  $P(\mathbf{x}_i | y)$ ; the former is then the relative frequency of class  $y$  in the training set.

The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of  $P(\mathbf{x}_i | y)$ .

# Unsupervised Machine Learning Models

Slides adapted from Prof. Jason Pachecos slides : CSC 380 Fall 2021

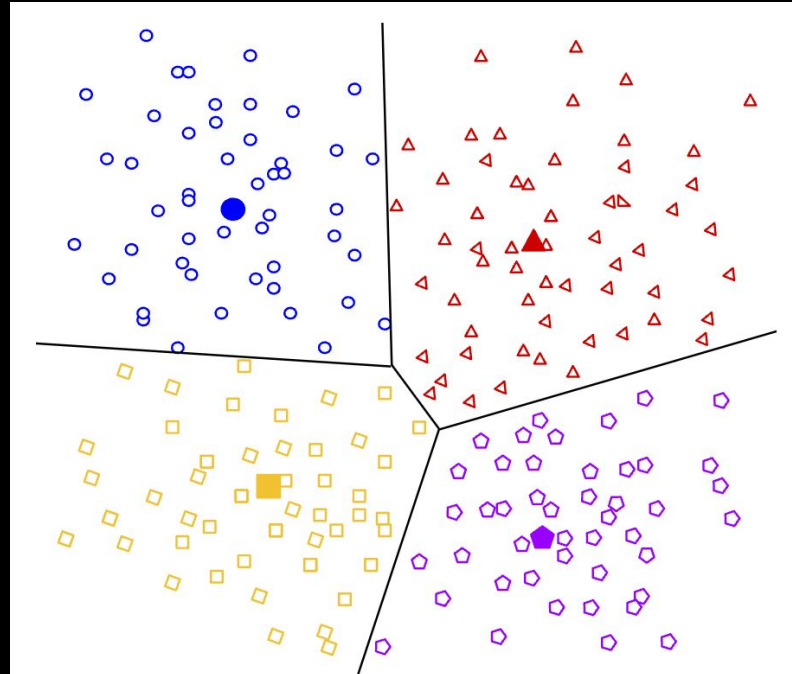
# Clustering



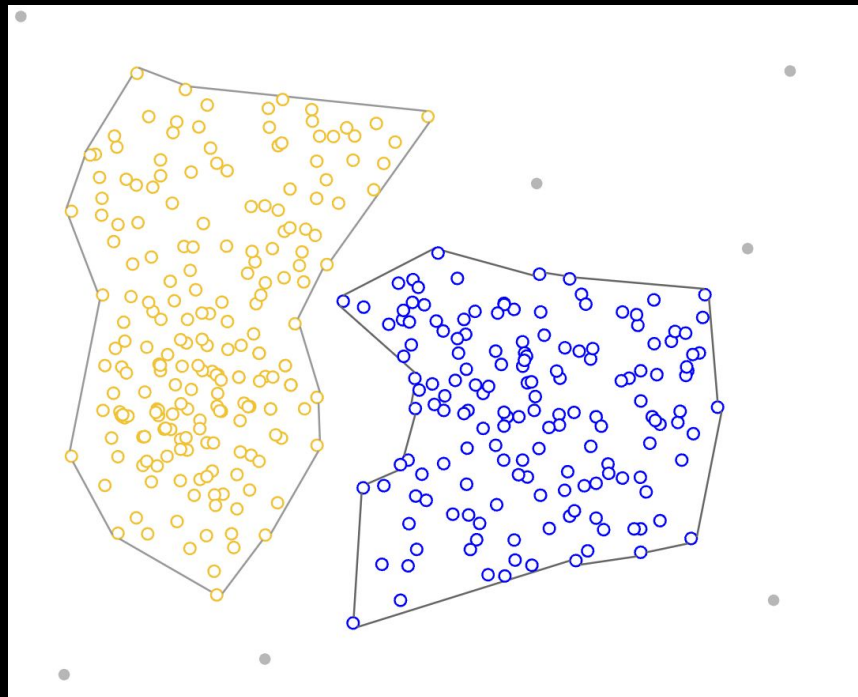
# Types

- Centroid-based Clustering
- Density-based Clustering
- Distribution-based Clustering
- Hierarchical Clustering

# Centroid-based Clustering

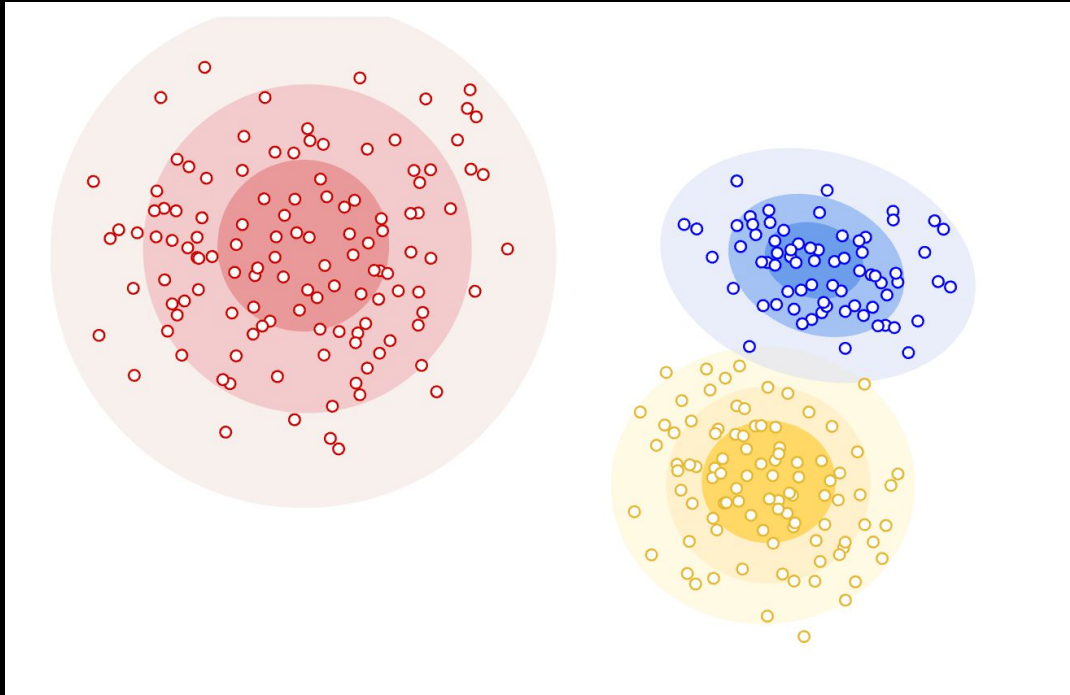


# Density-based Clustering

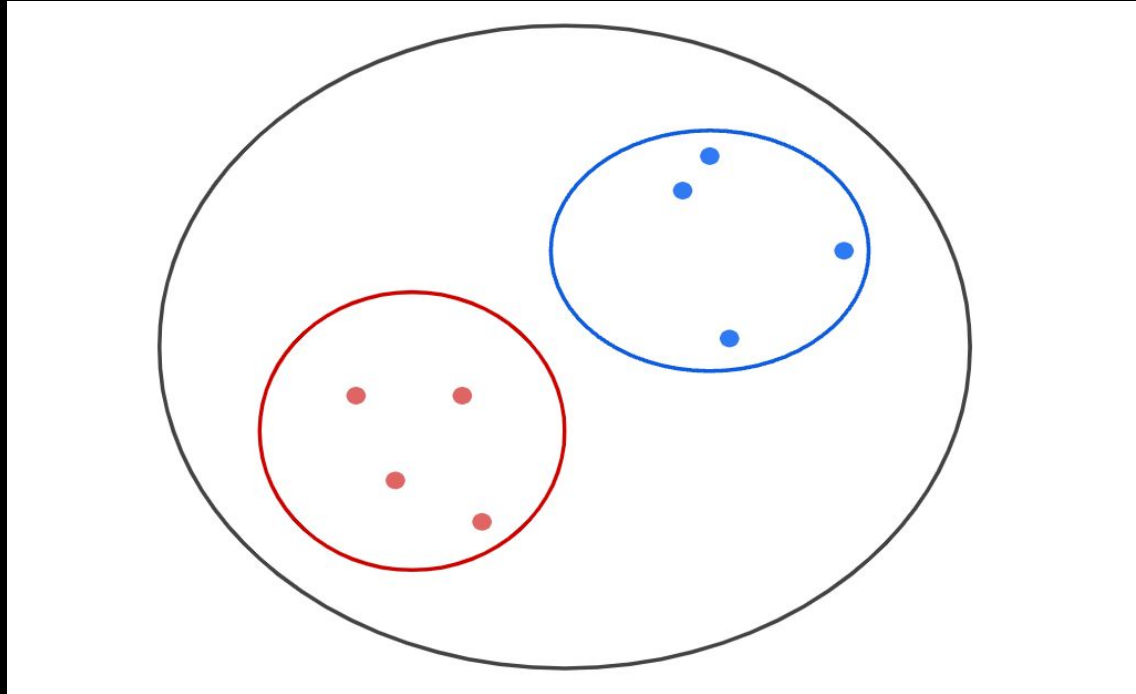


Source : <https://developers.google.com/machine-learning/clustering/clustering-algorithms>

# Distribution-based Clustering



# Hierarchical Clustering





# K-means

1. First, we need to provide the number of clusters, K, that need to be generated by this algorithm.
2. Next, choose K data points at random and assign each to a cluster. Briefly, categorize the data based on the number of data points.
3. The cluster centroids will now be computed.
4. Iterate the steps below until we find the ideal centroid, which is the assigning of data points to clusters that do not vary.
  - a. The sum of squared distances between data points and centroids would be calculated first.
  - b. At this point, we need to allocate each data point to the cluster that is closest to the others (centroid).
  - c. Finally, compute the centroids for the clusters by averaging all of the cluster's data points.



## sklearn.cluster.KMeans

```
class sklearn.cluster.KMeans(n_clusters=8, *, init='k-means++', n_init='warn', max_iter=300, tol=0.0001, verbose=0, random_state=None, copy_x=True, algorithm='lloyd')
```

[\[source\]](#)

K-Means clustering.

Read more in the [User Guide](#).

### Parameters:

**n\_clusters** : *int*, **default=8**

The number of clusters to form as well as the number of centroids to generate.

**init** : {'k-means++', 'random'}, *callable or array-like of shape (n\_clusters, n\_features)*, **default='k-means++'**

Method for initialization:

'k-means++' : selects initial cluster centroids using sampling based on an empirical probability distribution of the points' contribution to the overall inertia. This technique speeds up convergence. The algorithm implemented is "greedy k-means++". It differs from the vanilla k-means++ by making several trials at each sampling step and choosing the best centroid among them.

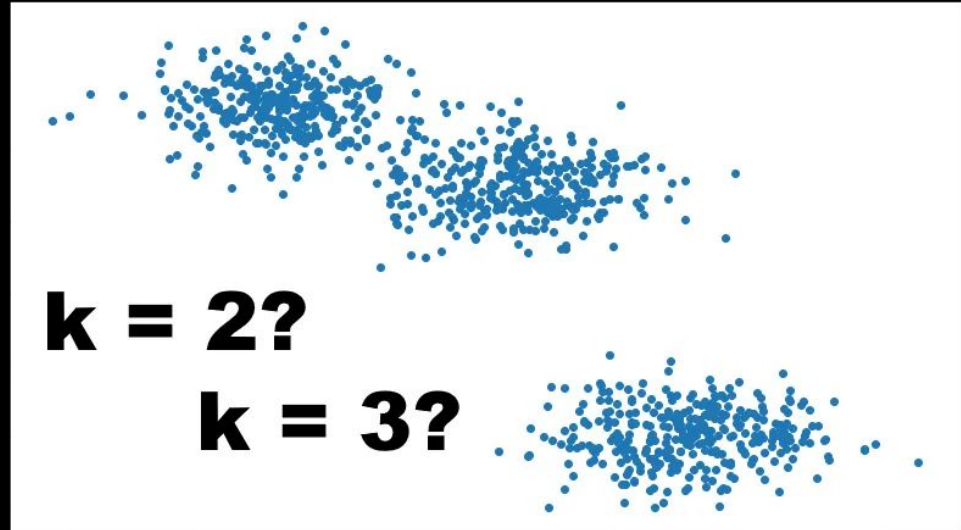
'random': choose `n_clusters` observations (rows) at random from data for the initial centroids.

If an array is passed, it should be of shape (n\_clusters, n\_features) and gives the initial centers.

If a callable is passed, it should take arguments X, n\_clusters and a random state and return an initialization.

# How to choose k?

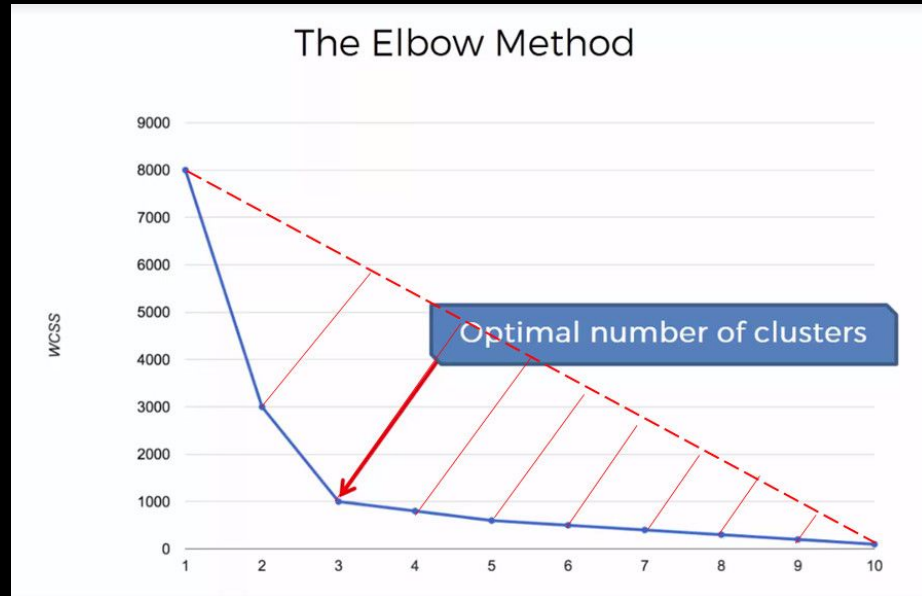
- The Elbow Method
- The Silhouette Method



# The Elbow Method

- WCSS (within-cluster sum of squares ):

The sum of square distances between the centroids and each points.



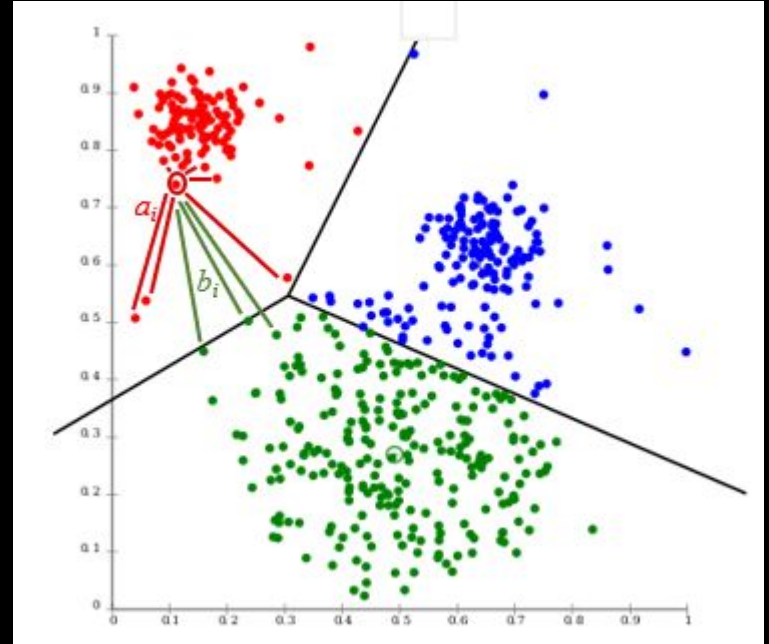
Source :  
<https://www.linkedin.com/pulse/finding-optimal-number-clusters-k-means-through-elbow-asanka-perera/>

**inertia\_** : *float*

Sum of squared distances of samples to their closest cluster center, weighted by the sample weights if provided.

# The Silhouette Method

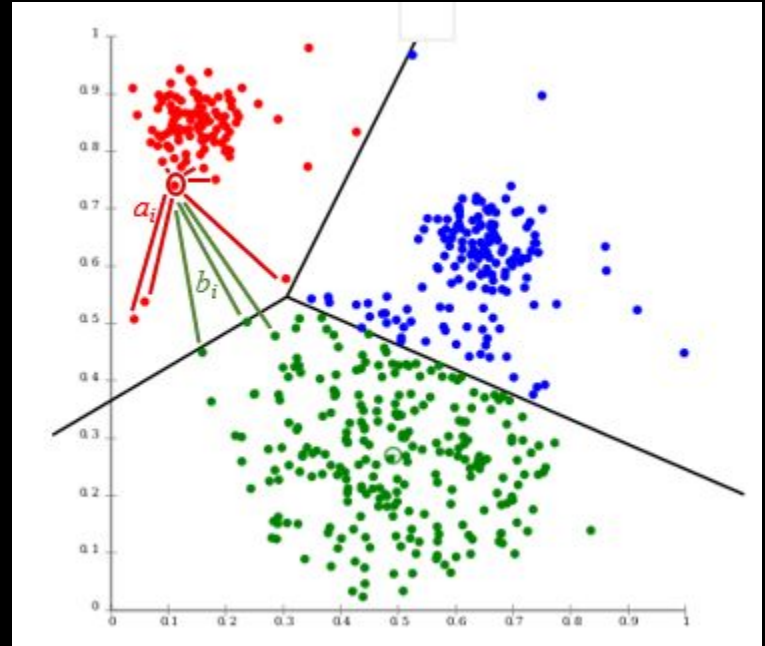
- Measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation).
- The silhouette ranges from  $-1$  to  $+1$ , where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters.



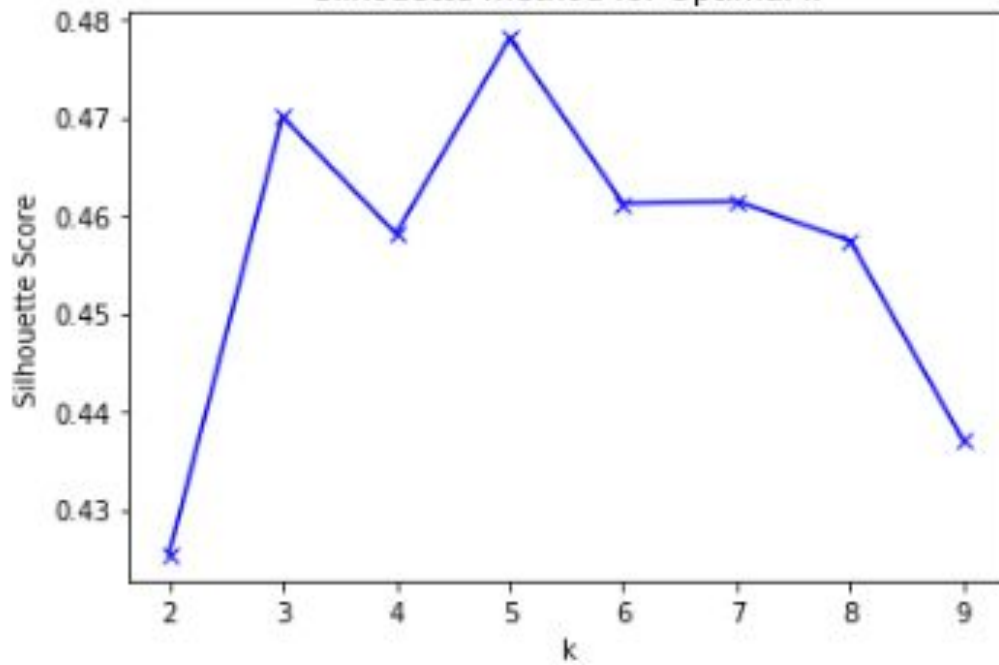
$$s(i) = \frac{b(i) - a(i)}{\text{larger of } b(i) \text{ and } a(i)}$$

$a(i)$  = average distance inside cluster

$b(i)$  = average distance nearest other cluster



Silhouette method for Optimal k





## sklearn.metrics.silhouette\_score

```
sklearn.metrics.silhouette_score(X, labels, *, metric='euclidean', sample_size=None, random_state=None, **kwargs)
```

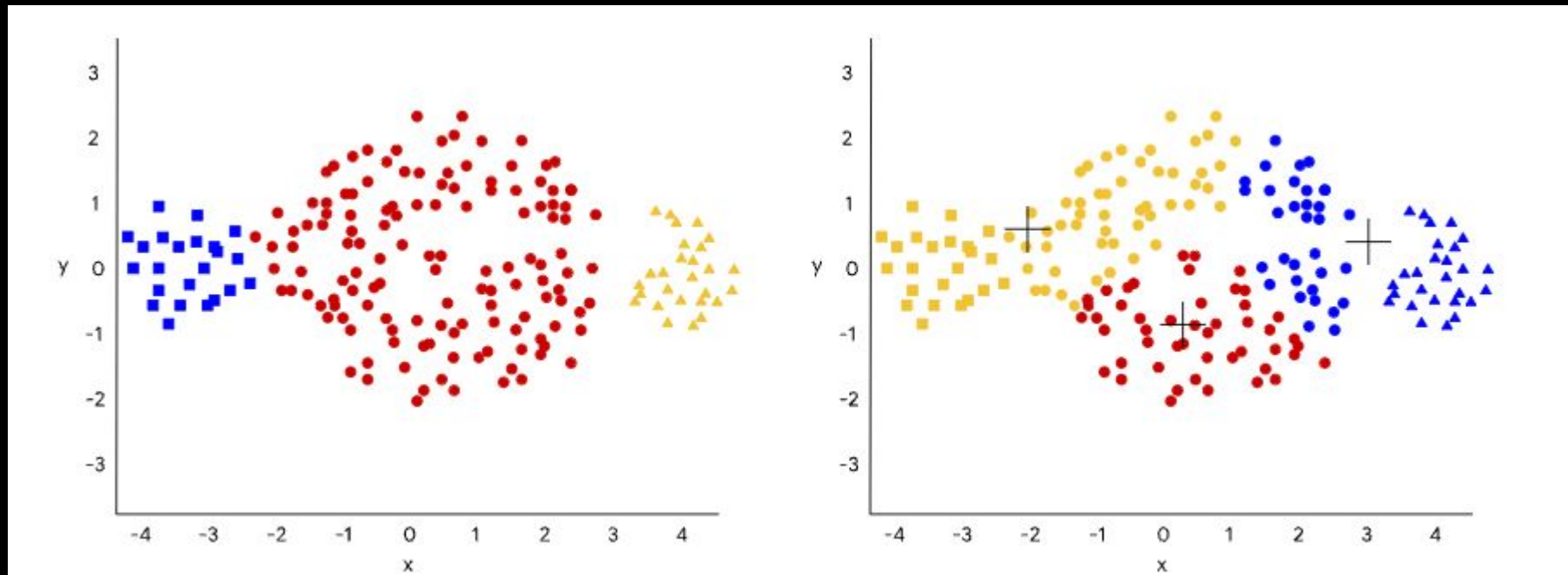
[\[source\]](#)

Compute the mean Silhouette Coefficient of all samples.

The Silhouette Coefficient is calculated using the mean intra-cluster distance ( $a$ ) and the mean nearest-cluster distance ( $b$ ) for each sample. The Silhouette Coefficient for a sample is  $(b - a) / \max(a, b)$ . To clarify,  $b$  is the distance between a sample and the nearest cluster that the sample is not a part of. Note that Silhouette Coefficient is only defined if number of labels is  $2 \leq n\_labels \leq n\_samples - 1$ .

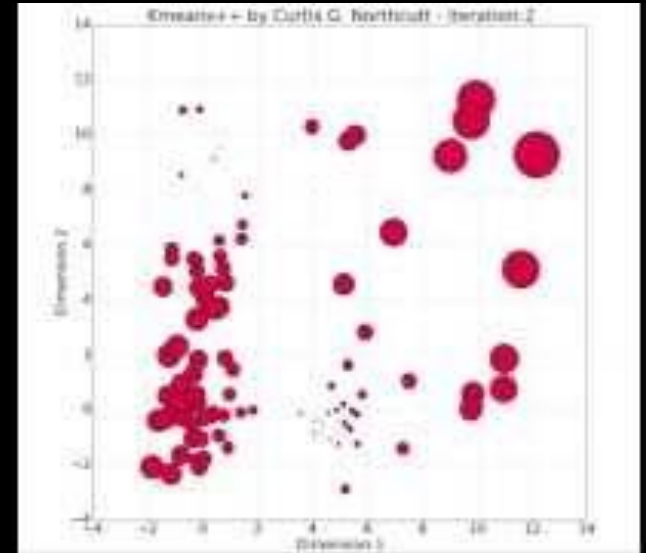
This function returns the mean Silhouette Coefficient over all samples. To obtain the values for each sample, use [silhouette\\_samples](#).

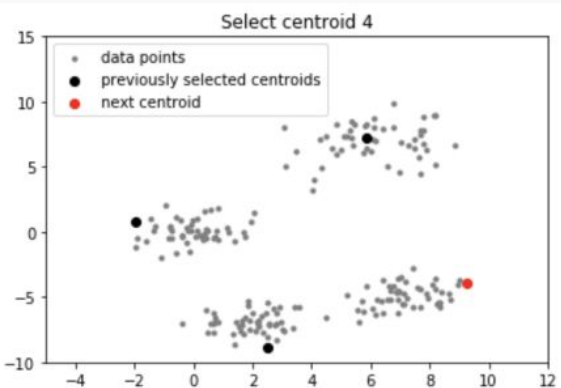
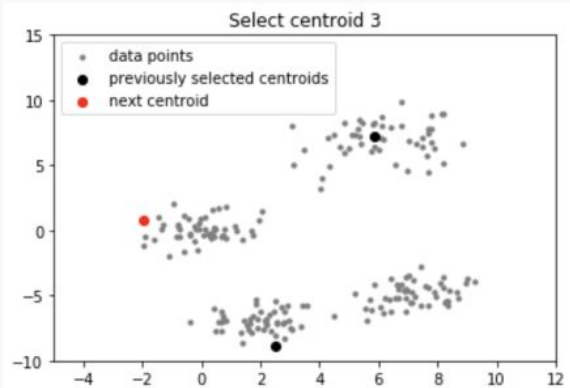
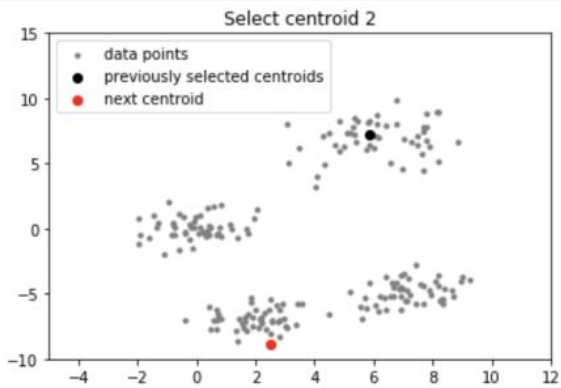
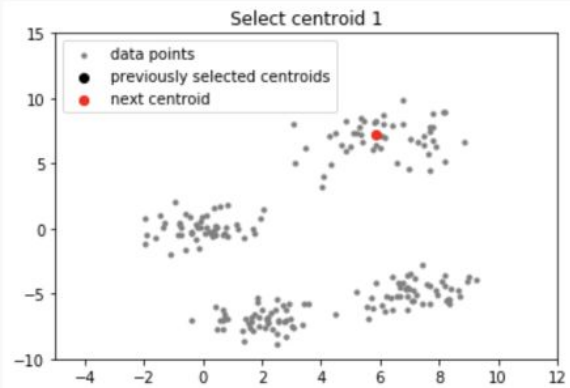
# Problem with kmeans



# How to choose the initial centroids? - Kmeans++

1. Randomly select the first centroid from the data points.
2. For each data point compute its distance from the nearest, previously chosen centroid.
3. Select the next centroid from the data points such that the probability of choosing a point as centroid is directly proportional to its distance from the nearest, previously chosen centroid. (i.e. the point having maximum distance from the nearest centroid is most likely to be selected next as a centroid)
4. Repeat steps 2 and 3 until k centroids have been sampled





## sklearn.cluster.KMeans

```
class sklearn.cluster.KMeans(n_clusters=8, *, init='k-means++', n_init='warn', max_iter=300, tol=0.0001, verbose=0, random_state=None, copy_x=True, algorithm='lloyd')
```

[\[source\]](#)

K-Means clustering.

Read more in the [User Guide](#).

### Parameters:

**n\_clusters** : *int*, **default=8**

The number of clusters to form as well as the number of centroids to generate.

**init** : {'k-means++', 'random'}, *callable or array-like of shape (n\_clusters, n\_features)*, **default='k-means++'**

Method for initialization:

'k-means++' : selects initial cluster centroids using sampling based on an empirical probability distribution of the points' contribution to the overall inertia. This technique speeds up convergence. The algorithm implemented is "greedy k-means++". It differs from the vanilla k-means++ by making several trials at each sampling step and choosing the best centroid among them.

'random': choose `n_clusters` observations (rows) at random from data for the initial centroids.

If an array is passed, it should be of shape (n\_clusters, n\_features) and gives the initial centers.

If a callable is passed, it should take arguments X, n\_clusters and a random state and return an initialization.