CSCI2202: Lecture 11
Machine Learning

Finlay Maguire (finlay.maguire@dal.ca)
TA: Ehsan Baratnezhad (ethan.b@dal.ca)
TA: Precious Osadebamwen (precious.osadebamwen@dal.ca)

Overview

- Machine Learning
- Traditional Machine Learning in Python (Scikit-Learn)
- Deep Learning in Python (PyTorch) - **not covered**

- Supervised Learning
  - Logistic Regression

- Unsupervised Learning
  - K-means clustering
  - t-SNE embedding/projection

What is Machine Learning?

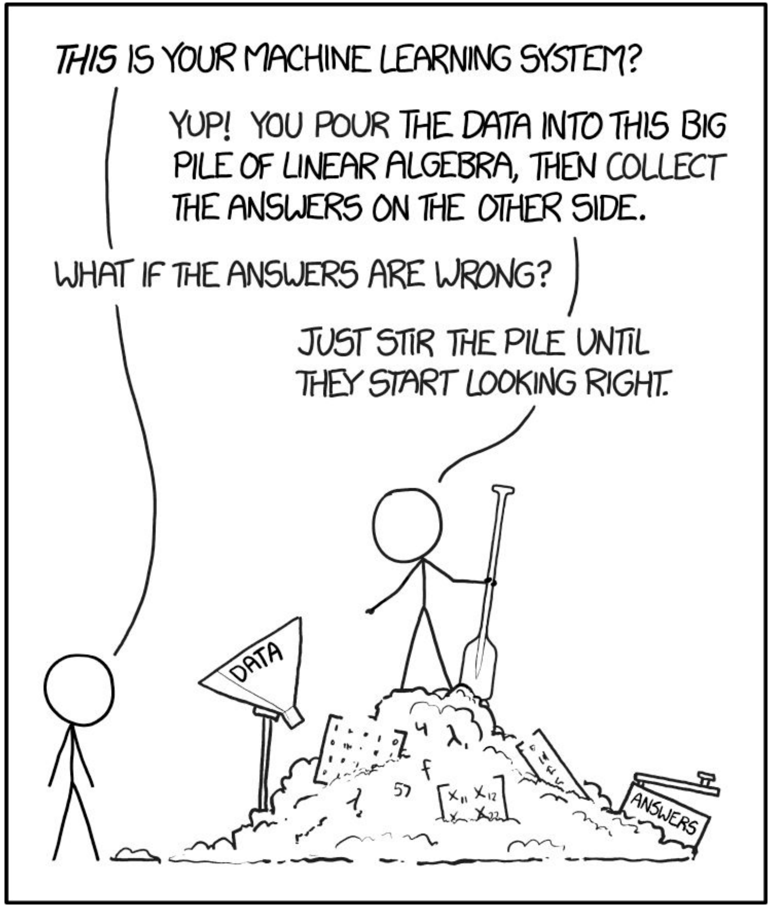
What is Machine Learning?

- “Machine Learning is the field of study that gives the computer the ability to learn without being explicitly programmed”

- “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.”

- Experience (data):
    - games played by the program (with itself)
- Performance measure:
    - winning rate

- *Training models which identify patterns in data*


THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

https://xkcd.com/1838/

Types of Machine Learning

- SUPERVISED - predict y from x (classification/regression)
  - Labeled classes
  - Minimise
  - Feedback: information about labeling is used to train classifier

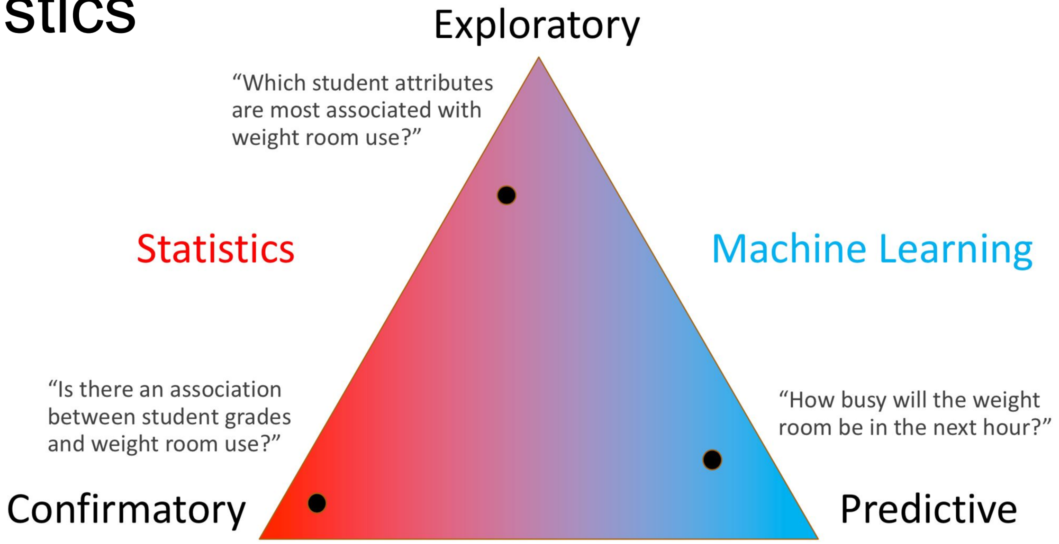
- UNSUPERVISED - find groups in x (clustering/dimensionality reduction)
  - Classes may be labeled or unlabelled
  - Classifier develops the classification/clustering scheme independently from class labels

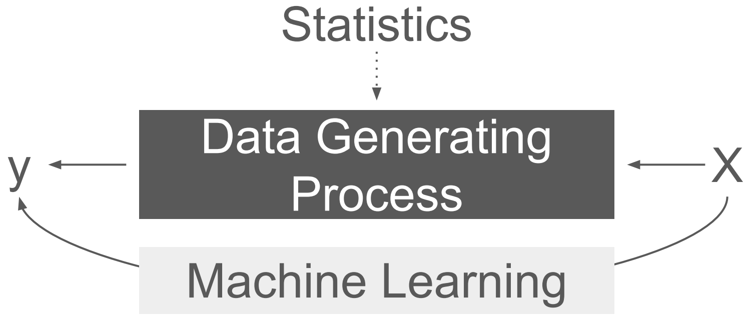
- SEMI-SUPERVISED - blend of the above
- REINFORCEMENT - Identify optimal moves / strategies in a search space

Machine Learning vs Statistics

- Many shared methods
- Difference in focus/priorities/culture
- Statistics ~ tries to understand how outcome was generated by data
- ML infers/learns A process for linking data to outcome
- Alternative framing: Data Modelling vs Algorithmic Modelling

- Pitfalls (ML can be):
  - Less rigorous/principled
  - Prone to reinventing the wheel
- Benefits (can be):
  - More flexible
  - Less prescriptive/intimidating

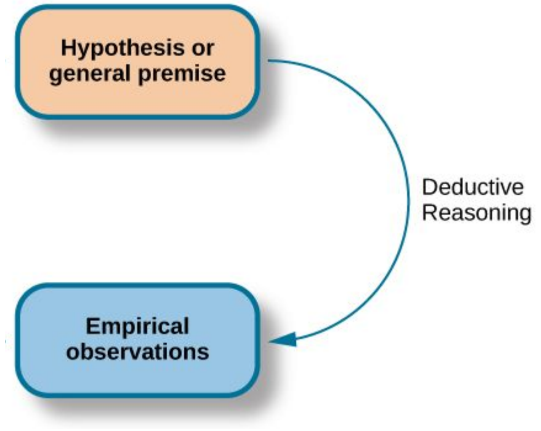

Exploratory
"Which student attributes are most associated with weight room use?"
Statistics
Machine Learning
"Is there an association between student grades and weight room use?"
"How busy will the weight room be in the next hour?"
Confirmatory
Predictive



But Machine Learning can be used to create hypotheses!

Deductive:

- "Condition X, causes Y"
- Collect data
- Perform (typically) frequentist statistical tests
- Reject or confirm null hypothesis



https://opened.cuny.edu/courseware/lesson/14/student/?task=3

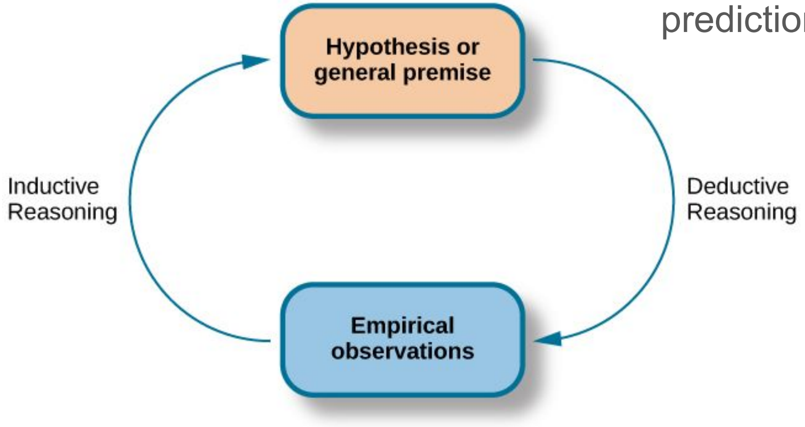
But Machine Learning can be used to create hypotheses!

Deductive:

- “Condition X, causes Y”
- Collect data
- Perform (typically) frequentist statistical tests
- Reject or confirm null hypothesis

Inductive:

- Collect data
- Identify patterns in the data
- Observe X and Y seem connected somehow
- Quantify strength of association e.g., prediction performance

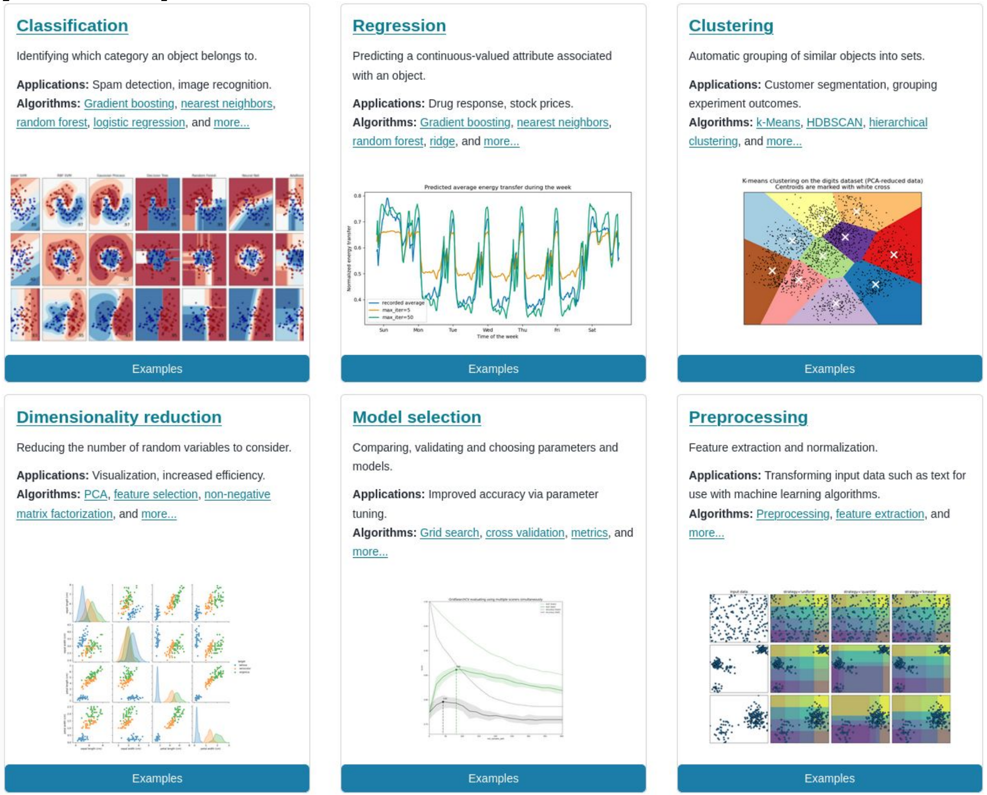


https://opened.cuny.edu/courseware/lesson/14/student/?task=3

Traditional Machine Learning in Python

- Scikit-learn:
  - Very widely used
  - Gold-standard traditional ML package
  - Fantastic documentation ->
  - Relatively fast (numpy)
  - Simple model
  - Many compatible contribution packages
  - Limited neural network support

```
from sklearn.MODULE import CLASSIFIER

model = CLASSIFIER()

model.fit(x, y)

# just x for unsupervised

y_pred = model.predict(x)

performance = model.score(x, y)
```



Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: Gradient boosting, nearest neighbors, random forest, logistic regression, and more...

Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, stock prices.
Algorithms: Gradient boosting, nearest neighbors, random forest, ridge, and more...

Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, grouping experiment outcomes.
Algorithms: k-Means, HDBSCAN, hierarchical clustering, and more...

Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, increased efficiency.
Algorithms: PCA, feature selection, non-negative matrix factorization, and more...

Examples

Model selection

Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tuning.
Algorithms: Grid search, cross validation, metrics, and more...

Examples

Preprocessing

Feature extraction and normalization.

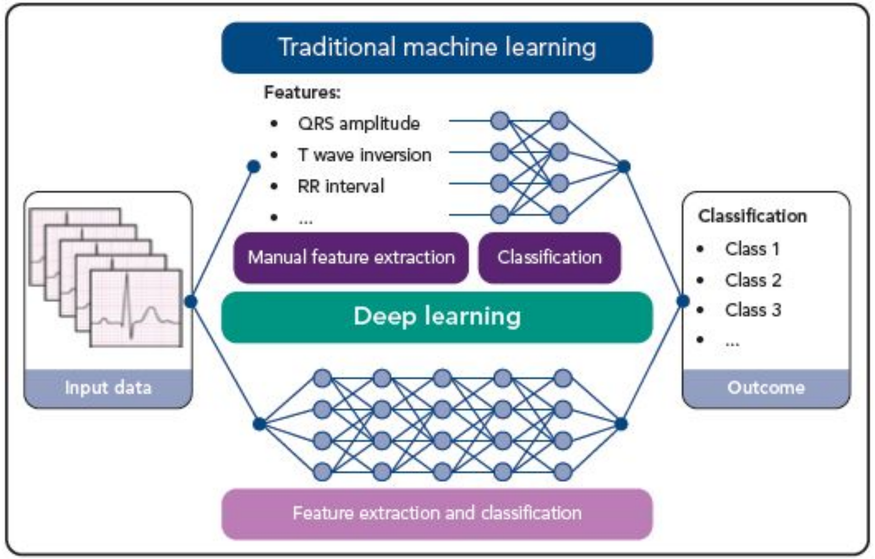
Applications: Transforming input data such as text for use with machine learning algorithms.
Algorithms: Preprocessing, feature extraction, and more...

Examples

Predicted average energy transfer during the week
K-means clustering on the digits dataset (PCA-reduced data)
Centroids are marked with white cross

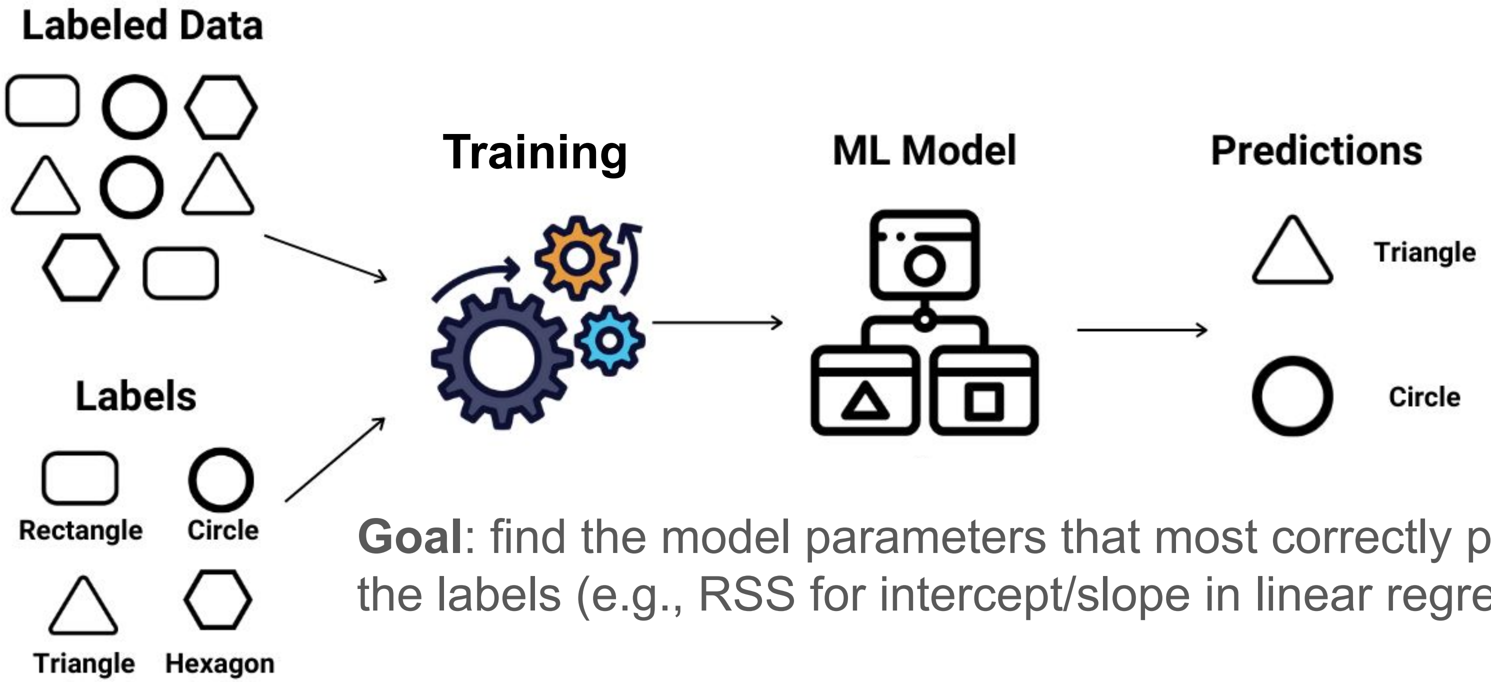
Deep Learning in Python - **not covered**

- PyTorch
  - Popular in latest research
  - More python-like and dynamic graphs
  - Originally Facebook/Meta
- TensorFlow
  - Popular in product/industry
  - More verbose (although Keras API now)
  - Originally Google
- Many others: Keras, Theano, Caffe,
  - Generally slower and/or legacy libraries



Supervised Learning

Predicting Labels (Classification) or Values (Regression)



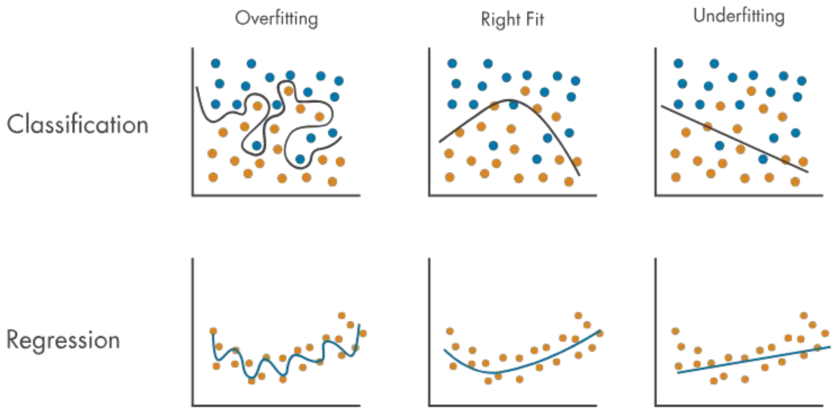
Goal: find the model parameters that most correctly predict the labels (e.g., RSS for intercept/slope in linear regression)

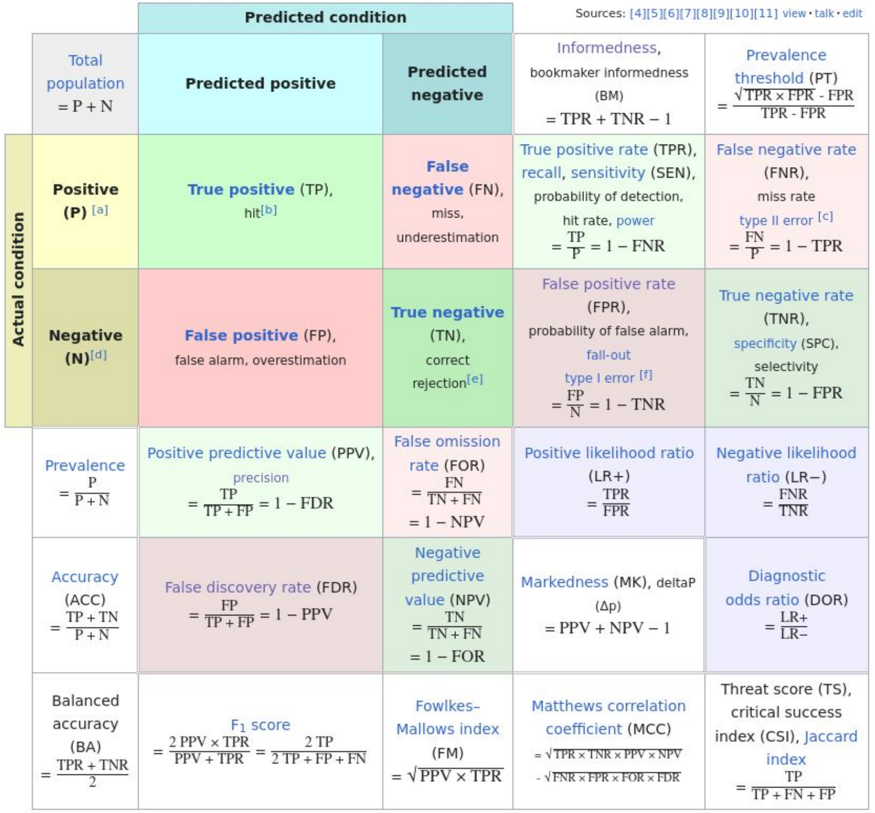
https://medium.com/@dhara732002/supervised-machine-learning-a-beginners-guide-9ac0b07eccbb

Goal is a model that predicts class in a generalizable way

Many ways to assess “correctness”

We want model to generalise to new date (i.e., not overfit to training data)





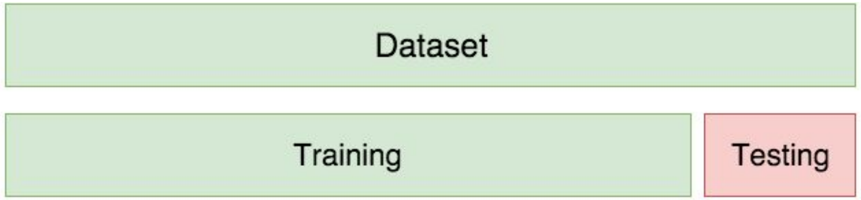
		Predicted condition			Sources: [4][5][6][7][8][9][10][11] view · talk · edit
	Total population $= P + N$	Predicted positive	Predicted negative	Informedness, bookmaker informedness (BM) $= TPR + TNR - 1$	Prevalence threshold (PT) $= \frac{\sqrt{TPR \times FPR} - FPR}{TPR - FPR}$
Actual condition	Positive (P) [a]	True positive (TP), hit[b]	False negative (FN), miss, underestimation	True positive rate (TPR), recall, sensitivity (SEN), probability of detection, hit rate, power $= \frac{TP}{P} = 1 - FNR$	False negative rate (FNR), miss rate type II error [c] $= \frac{FN}{P} = 1 - TPR$
	Negative (N)[d]	False positive (FP), false alarm, overestimation	True negative (TN), correct rejection[e]	False positive rate (FPR), probability of false alarm, fall-out type I error [f] $= \frac{FP}{N} = 1 - TNR$	True negative rate (TNR), specificity (SPC), selectivity $= \frac{TN}{N} = 1 - FPR$
	Prevalence $= \frac{P}{P + N}$	Positive predictive value (PPV), precision $= \frac{TP}{TP + FP} = 1 - FDR$	False omission rate (FOR) $= \frac{FN}{TN + FN} = 1 - NPV$	Positive likelihood ratio (LR+) $= \frac{TPR}{FPR}$	Negative likelihood ratio (LR−) $= \frac{FNR}{TNR}$
	Accuracy (ACC) $= \frac{TP + TN}{P + N}$	False discovery rate (FDR) $= \frac{FP}{TP + FP} = 1 - PPV$	Negative predictive value (NPV) $= \frac{TN}{TN + FN} = 1 - FOR$	Markedness (MK), deltaP (Δp) $= PPV + NPV - 1$	Diagnostic odds ratio (DOR) $= \frac{LR+}{LR-}$
	Balanced accuracy (BA) $= \frac{TPR + TNR}{2}$	F_1 score $= \frac{2\,PPV \times TPR}{PPV + TPR} = \frac{2\,TP}{2\,TP + FP + FN}$	Fowlkes–Mallows index (FM) $= \sqrt{PPV \times TPR}$	Matthews correlation coefficient (MCC) $= \sqrt{TPR \times TNR \times PPV \times NPV} - \sqrt{FNR \times FPR \times FOR \times FDR}$	Threat score (TS), critical success index (CSI), Jaccard index $= \frac{TP}{TP + FN + FP}$

Holdout part of data to evaluate generalised performance

Training set: Used to train the model (typically 70-80% of the data)

Testing set: Used to evaluate the model's performance on unseen data (typically 20-30%)

1. Randomly shuffle the dataset
2. Split the data into training and testing portions
3. Train the model using only the training data
4. Evaluate the model's performance on the testing data



```
from sklearn.model_selection import train_test_split
from sklearn.MODULE import CLASSIFIER
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = CLASSIFIER()
model.fit(X_train, y_train)
performance = model.score(x_test, y_test)
```

Logistic Regression

100 patients with surgical site infections.

We've measured how many bacteria are present in the wound (bacterial load)

Each wound is treated with the same amount of cefoxitin (antibiotic)

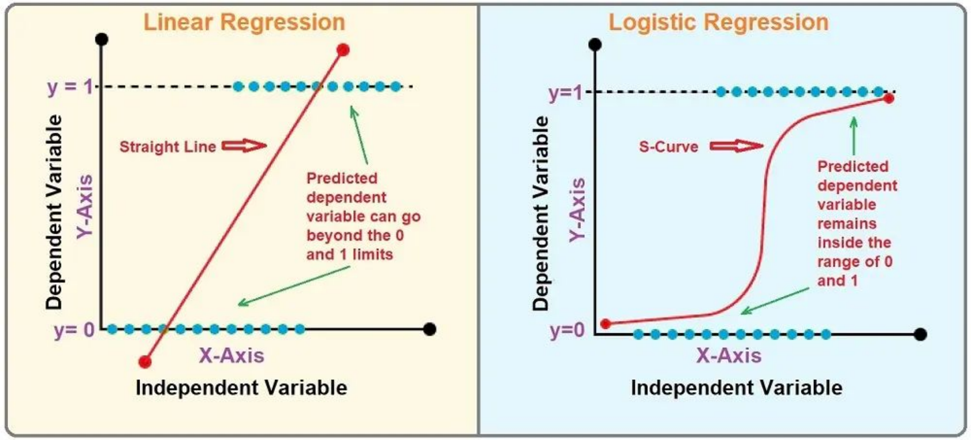
We want to predict whether treatment is successful or not (y=1 or y=0) based on bacterial load (x)

Linear regression not appropriate:

Predicts y < 0 and y > 1

Heteroscedasticity

Solution: **Logistic Regression**



$$\hat{y} = S(\beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2 + \ldots)$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

```
sigmoid = lambda z: 1 / (1 + np.exp(-z))
```

$\hat{y}$ now predicts a probability: $P(Y = 1|X)$

We can round $\hat{y}$ to get our 1 or 0 prediction

Logistic Regression

```
def log_loss(y_true, y_pred):

        return -(1/len(y_true)) * np.sum(\

                y_true * np.log(y_pred)

                + (1 - y_true) * np.log(1 - y_pred))

learning_rate = 0.1

for i in range(num_iterations):

        model = np.dot(X, slope) + intercept

        y_pred = sigmoid(linear_model)

        ds = (1/m) * np.dot(X.T, (y_pred - y))

        di = (1/m) * np.sum(y_pred - y)

        slope = slope - learning_rate * dw

        intercept = intercept - learning_rate * db

        cost = log_loss(y, y_pred)
```

Our linear regression loss/cost needs updated:

$$L = \frac{SSE}{n} = \frac{1}{n}\sum_{i}^{n}([b_0 + b_1 * x(i)] - y(i))^2$$

Use log-loss instead:

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}(y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i))$$

Fit LR using gradient descent:

$$\frac{\partial L}{\partial \beta} = \frac{1}{n}\sum_{i}^{n} x(i)(\hat{y}_i - y(i))$$

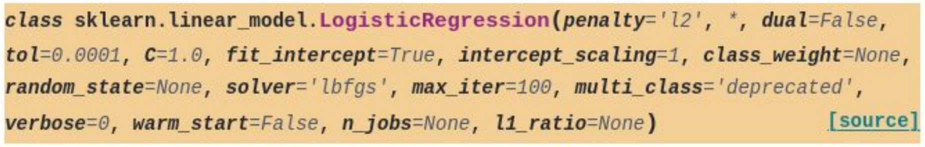
$$= (1/n) * X^T \cdot (\hat{Y} - Y)$$

Scikit-Learn makes this very simple!

```python
def log_loss(y_true, y_pred):
    return -(1/len(y_true)) * np.sum(\
        y_true * np.log(y_pred)
        + (1 - y_true) * np.log(1 - y_pred))
learning_rate = 0.1
for i in range(num_iterations):
    model = np.dot(X, slope) + intercept
    y_pred = sigmoid(linear_model)
    ds = (1/m) * np.dot(X.T, (y_pred - y))
    di = (1/m) * np.sum(y_pred - y)
    slope = slope - learning_rate * dw
    intercept = intercept - learning_rate * db
    cost = log_loss(y, y_pred)
```

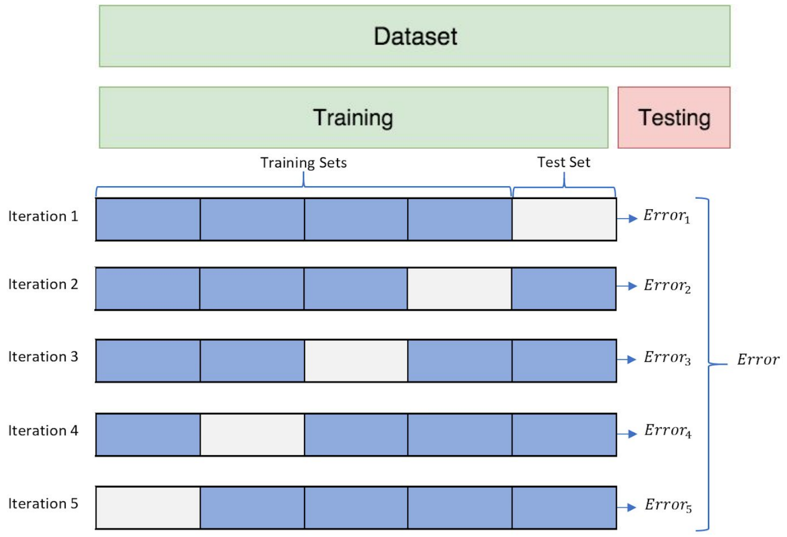
```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
X_train, X_test,
y_train, y_test = train_test_split(X, y, test_size=0.2,
                                   random_state=42)

lr = LogisticRegression()
lr.fit(X_train, y_train)
performance = lr.score(x_test, y_test)
```

Many different options e.g., regularisation



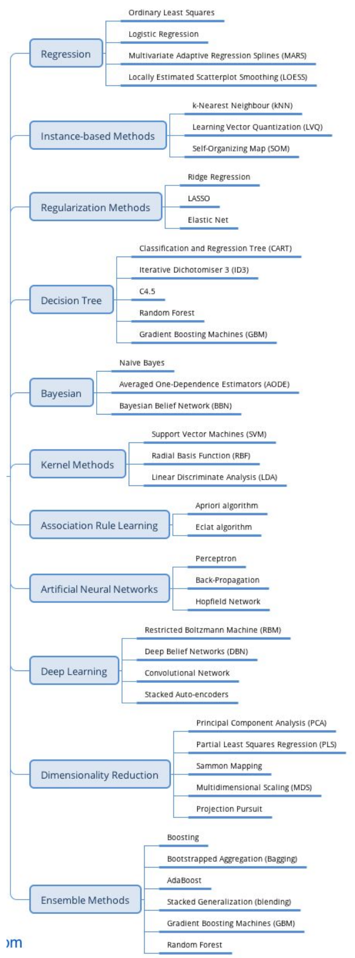
```
class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False,
tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None,
random_state=None, solver='lbfgs', max_iter=100, multi_class='deprecated',
verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)
```
[source]

Many model choices => comparing and tuning



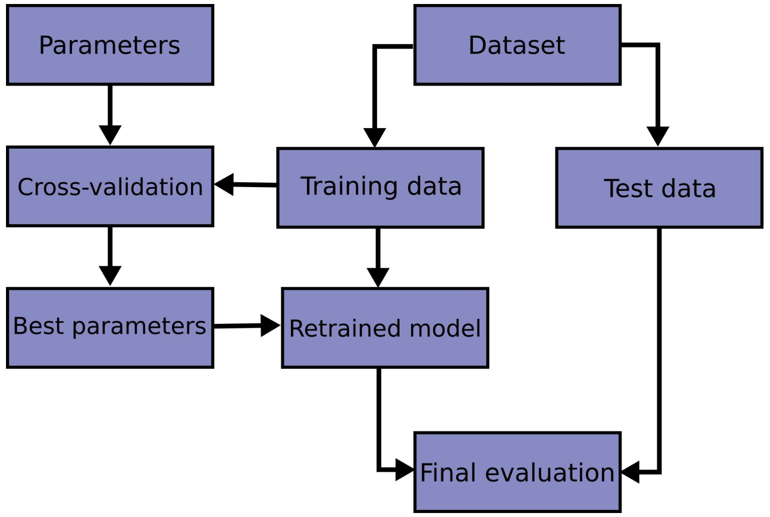
Using test set to tune/compare will lead to overfitting

Cross-validation: split training into pieces and train on ⅘ and compare on ⅕ (repeat for mean/variance estimate)

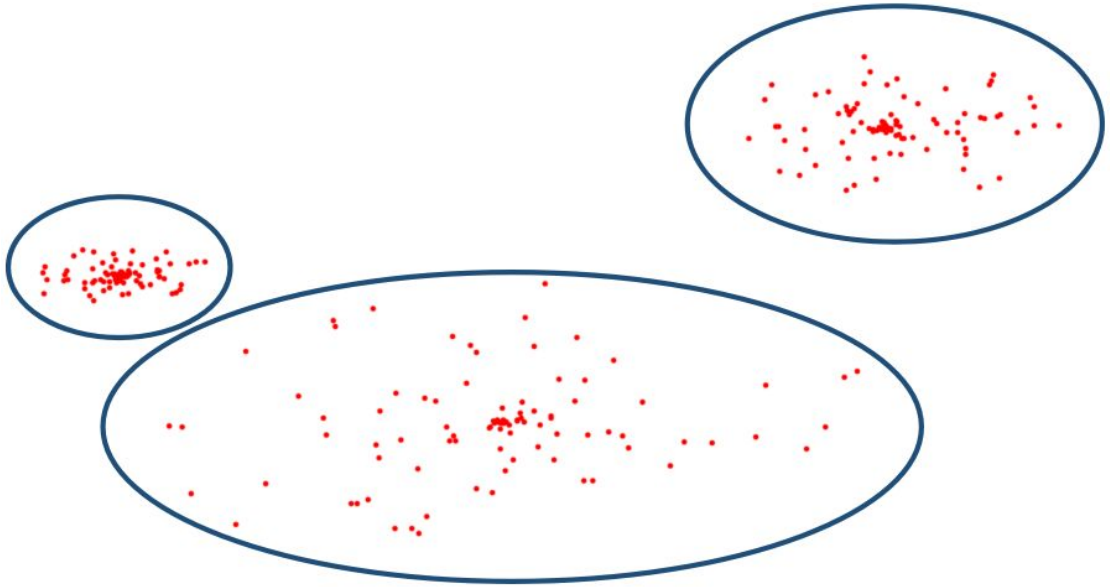

Regression
Ordinary Least Squares
Logistic Regression
Multivariate Adaptive Regression Splines (MARS)
Locally Estimated Scatterplot Smoothing (LOESS)
Instance-based Methods
k-Nearest Neighbour (kNN)
Learning Vector Quantization (LVQ)
Self-Organizing Map (SOM)
Regularization Methods
Ridge Regression
LASSO
Elastic Net
Decision Tree
Classification and Regression Tree (CART)
Iterative Dichotomiser 3 (ID3)
C4.5
Random Forest
Gradient Boosting Machines (GBM)
Bayesian
Naive Bayes
Averaged One-Dependence Estimators (AODE)
Bayesian Belief Network (BBN)
Kernel Methods
Support Vector Machines (SVM)
Radial Basis Function (RBF)
Linear Discriminate Analysis (LDA)
Association Rule Learning
Apriori algorithm
Eclat algorithm
Artificial Neural Networks
Perceptron
Back-Propagation
Hopfield Network
Deep Learning
Restricted Boltzmann Machine (RBM)
Deep Belief Networks (DBN)
Convolutional Network
Stacked Auto-encoders
Dimensionality Reduction
Principal Component Analysis (PCA)
Partial Least Squares Regression (PLS)
Sammon Mapping
Multidimensional Scaling (MDS)
Projection Pursuit
Ensemble Methods
Boosting
Bootstrapped Aggregation (Bagging)
AdaBoost
Stacked Generalization (blending)
Gradient Boosting Machines (GBM)
Random Forest
›m

Machine Learning Cross-Validation

```
from sklearn.linear_model import LogisticRegressionCV

from sklearn.model_selection import train_test_split


X_train, X_test,
y_train, y_test = train_test_split(X, y, test_size=0.2,
                                    random_state=42)

lr = LogisticRegressionCV(cv=5, random_state=0)

lr.fit(X_train, y_train)

performance = lr.score(x_test, y_test)
```



Unsupervised Learning: Clustering



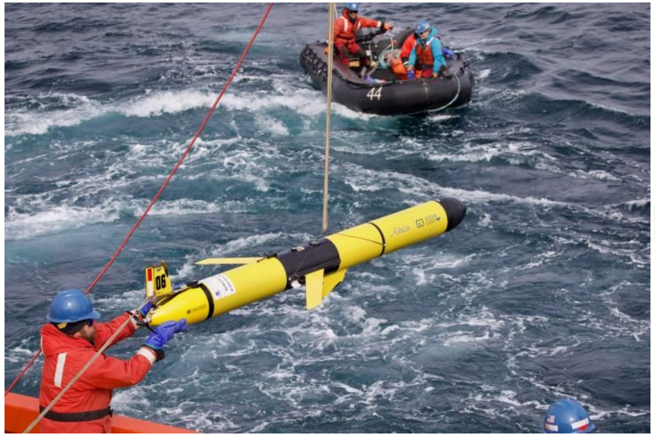
Clustering as an optimization problem

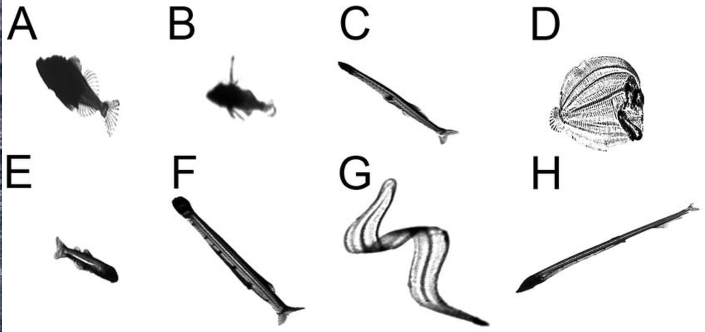
Using a glider with a shadowgraph camera we've taken images of lots of fish and then measured their lengths and widths.

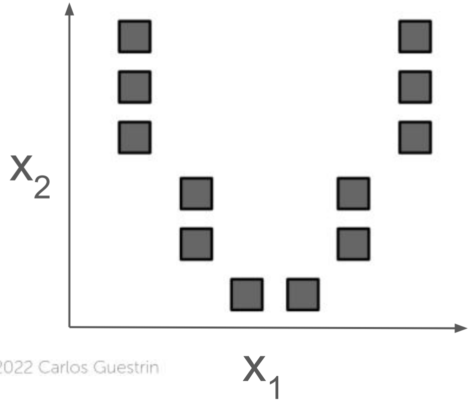
Now we want to group these fish into size categories to explore trophic sizes

Find k-centroids (cluster centers) that minimise the total distances from n data points

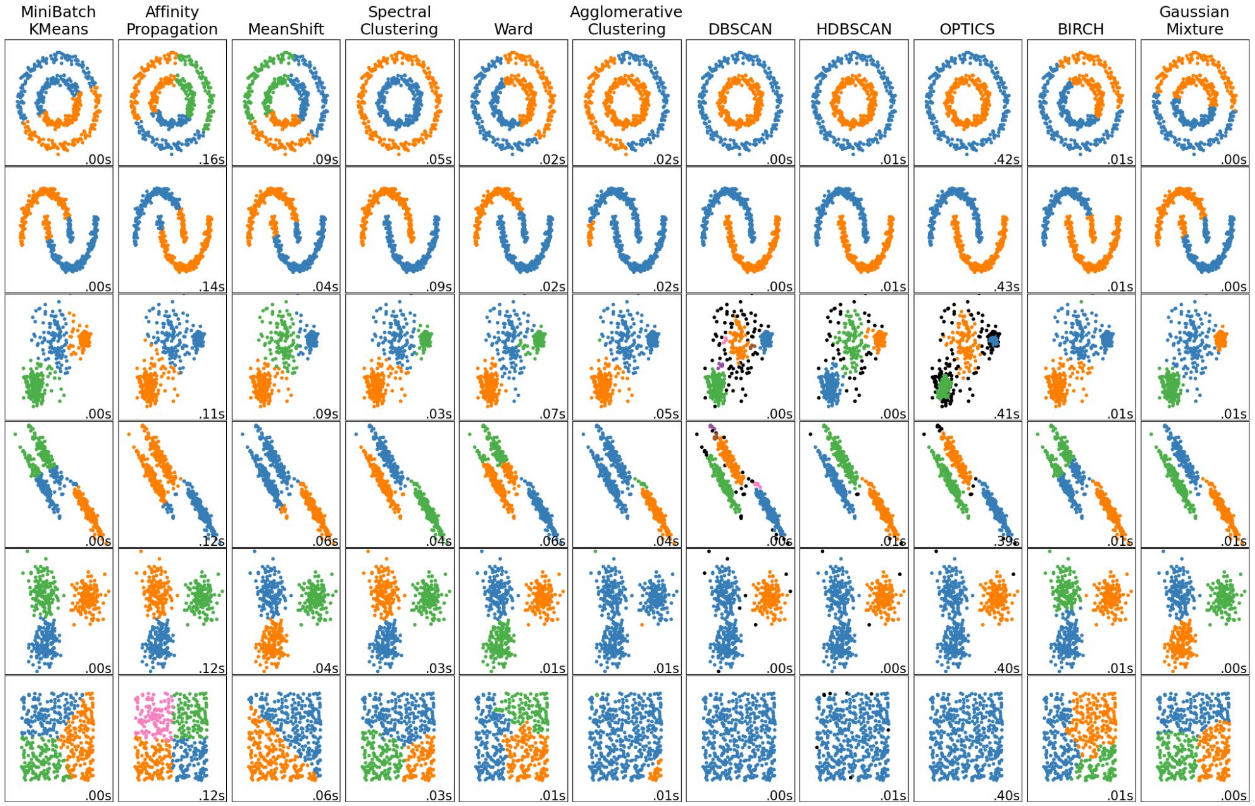
```
import numpy as np

X = np.column_stack((fish_lengths,

                     fish_widths))
```







Many different clustering algorithms



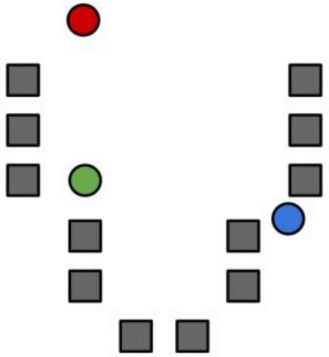
https://scikit-learn.org/stable/modules/clustering.html

Clustering as an optimization problem: k-means

```
rng = np.random.default_rng(42)

k = 3

centroids = np.random.choice(X.shape[0],

                            k,

                           replace=False)

centroids = x[centroids]
```

0. Initialize cluster centers

$$\mu_1, \mu_2, \ldots, \mu_k$$



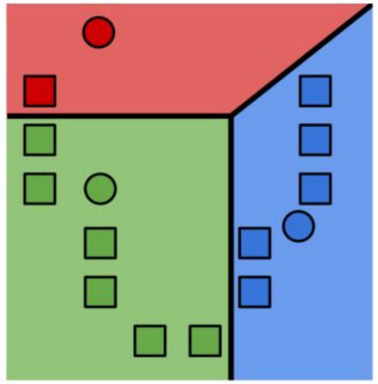
Clustering as an optimization problem: k-means

```python
def dist(point1, point2):
    return np.sqrt(np.sum((point1 - point2) ** 2))

while True:
    centroid_hist = [ centroids ]
    clusters = [[] for _ in centroids]
    for fish in X:
        i = np.argmin([dist(fish, c) for c in centroids])
        clusters[i].append(fish)
```

0. Initialize cluster centers
1. Assign observations to closest cluster center

$$z_i \leftarrow \arg\min_j ||\mu_j - \mathbf{x}_i||_2^2$$

Inferred label for obs i, whereas supervised learning has given label y_i

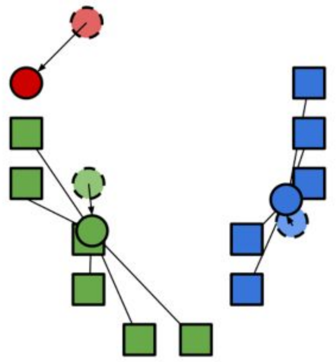


Clustering as an optimization problem: k-means

```
while True:
        centroid_hist = [ centroids ]
        clusters = [[] for _ in centroids]
        for fish in X:
                i = np.argmin([dist(fish, c) for c in centroids])
                clusters[i].append(fish)
        for ix, cluster in enumerate(clusters):
                centroids[ix] = np.mean(cluster, axis=1)
```

0. Initialize cluster centers
1. Assign observations to closest cluster center
2. Revise cluster centers as mean of assigned observations

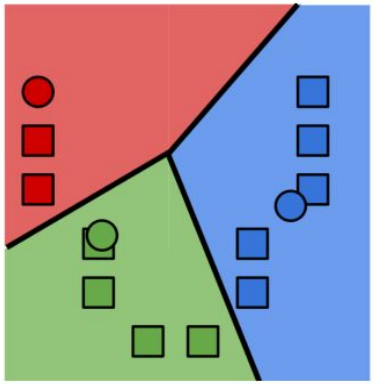
$$\mu_j = \frac{1}{n_j} \sum_{i:z_i=j} \mathbf{x}_i$$



Clustering as an optimization problem: k-means

```
while True:
    centroid_hist = [ centroids ]
    clusters = [[] for _ in centroids]
    for fish in X:
        i = np.argmin([dist(fish, c) for c in centroids])
        clusters[i].append(fish)
    for ix, cluster in enumerate(clusters):
        centroids[ix] = np.mean(cluster, axis=1)
    if centroids == centroid_hist[-1]:
        break
    centroid_hist.append([centroids])
```

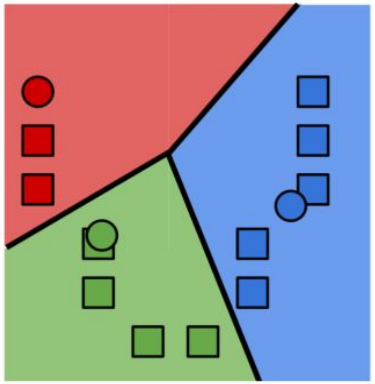
0. Initialize cluster centers
1. Assign observations to closest cluster center
2. Revise cluster centers as mean of assigned observations
3. Repeat 1.+2. until convergence



Clustering as an optimization problem: k-means

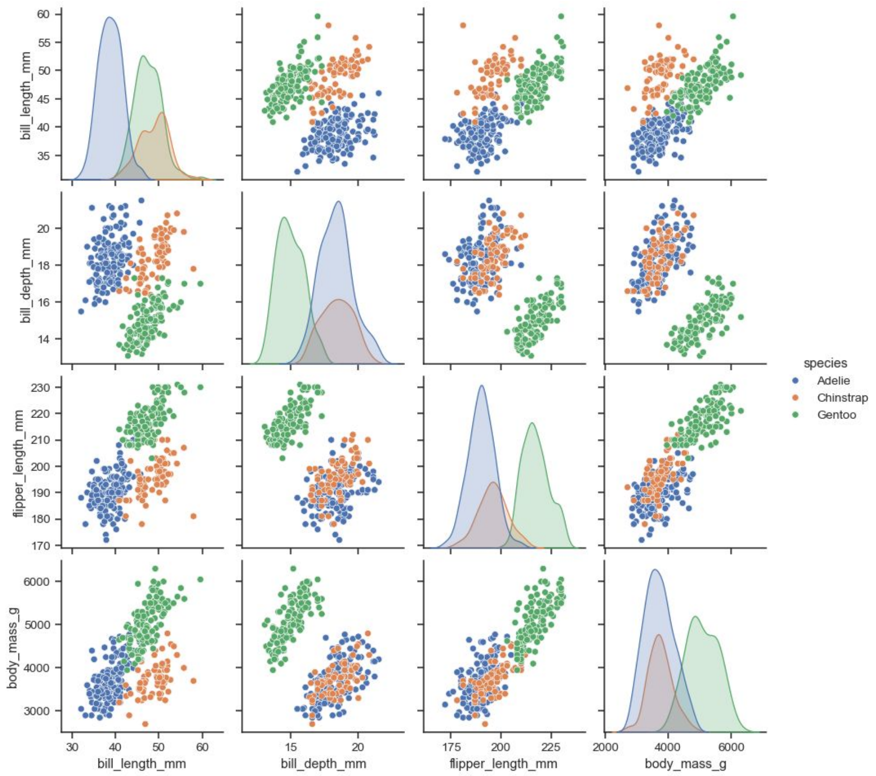
```
kmeans = KMeans(n_clusters=3, n_init="auto")

kmeans.fit(X)

kmeans.labels_

array([1, 1, 1, 0, 0, 2], dtype=int32)



kmeans.predict([[0, 0], [12, 3]])

array([1, 0], dtype=int32)



kmeans.cluster_centers_

array([[10.,  2.], [ 1.,  2.], [ 3., 4.]])
```

0. Initialize cluster centers
1. Assign observations to closest cluster center
2. Revise cluster centers as mean of assigned observations
3. Repeat 1.+2. until convergence

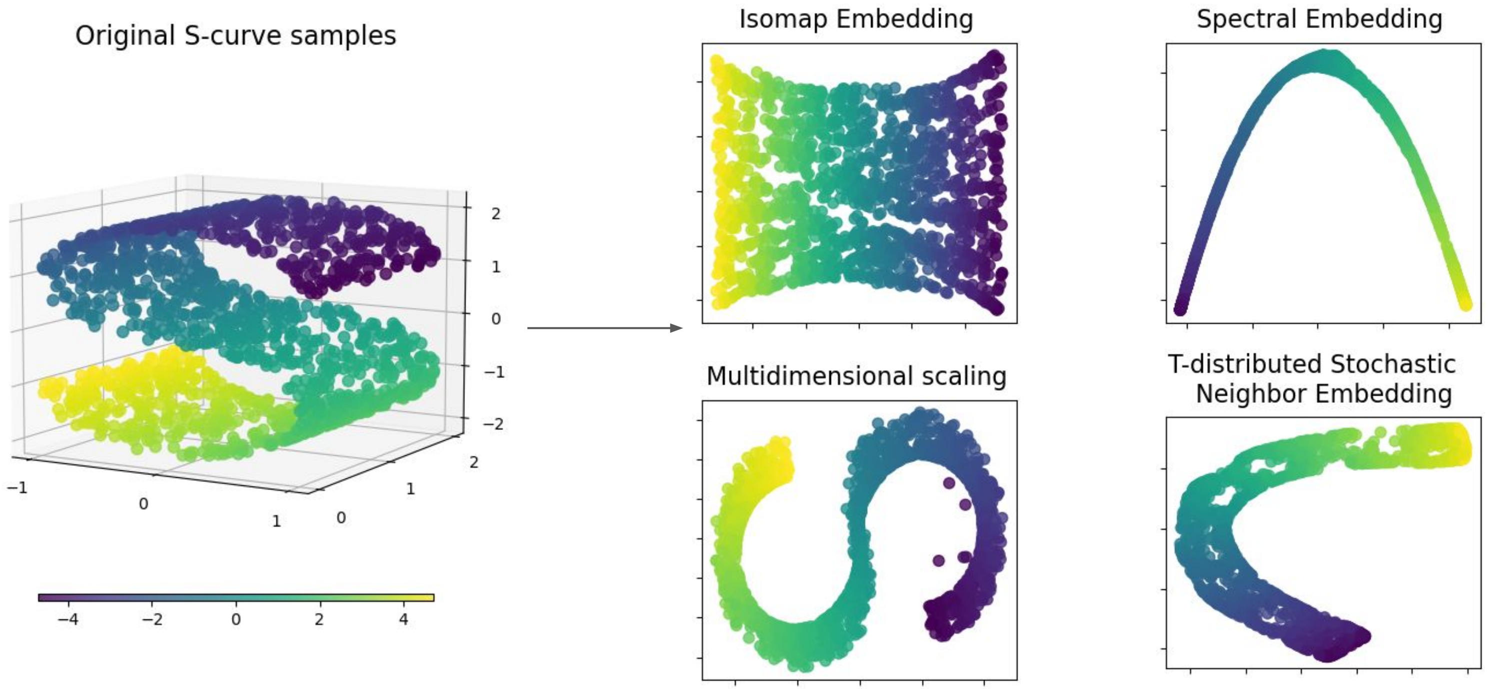


Looking at really high-dimensional data?

Pairplots useful but only pairwise so miss complex shapes

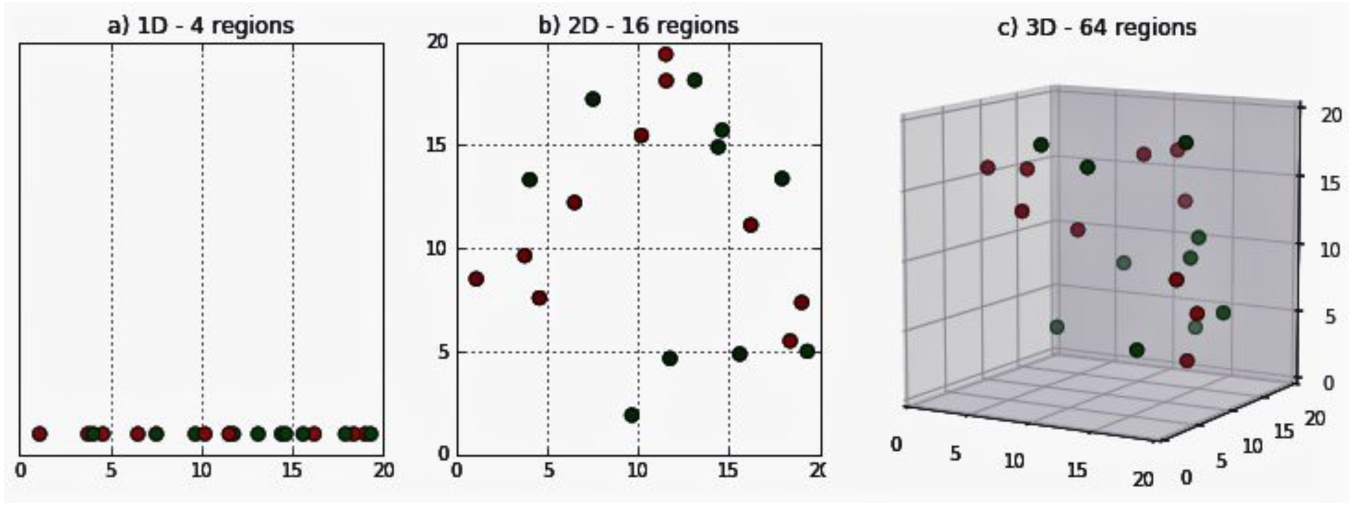


Many dimensions to few: Manifold learning, Ordination, Decomposition, Dimensionality reduction



Why is this hard?

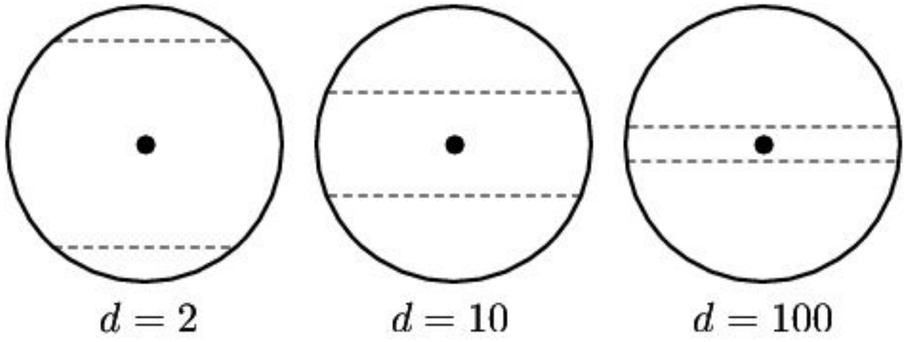
High dimensional data is sparse

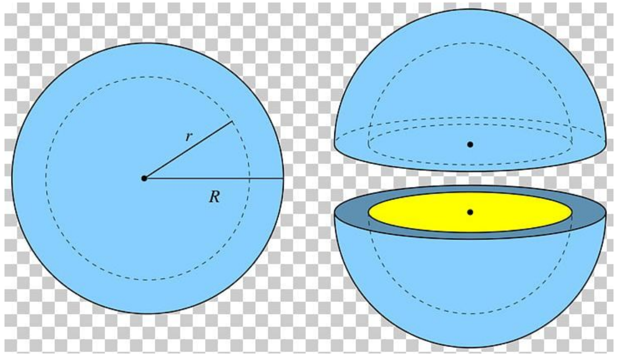


https://medium.com/analytics-vidhya/the-curse-of-dimensionality-and-its-cure-f9891ab72e5c

High dimensional space is counterintuitive

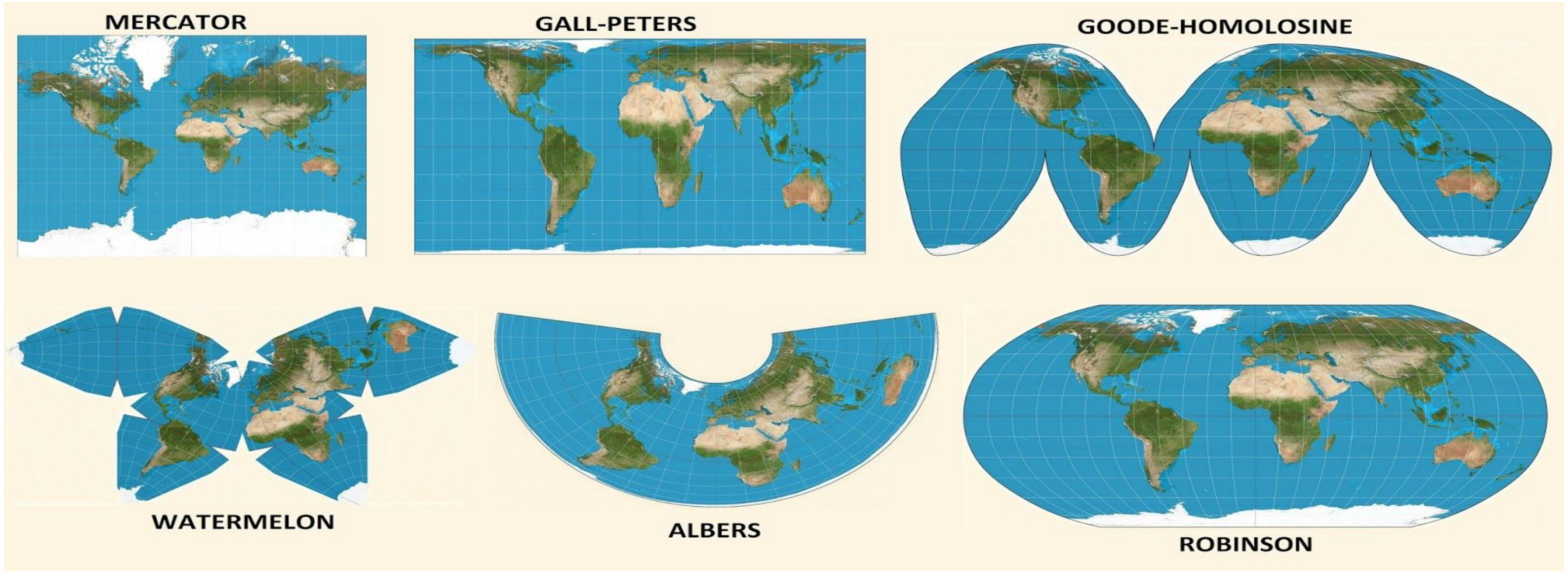
Orthogonality -> Band-size to capture 99% of the volume of a sphere:





Mass becomes increasingly “shell-like”

No representation is perfect

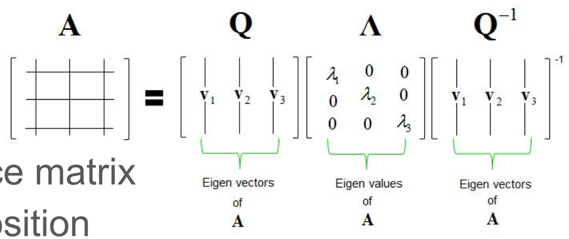


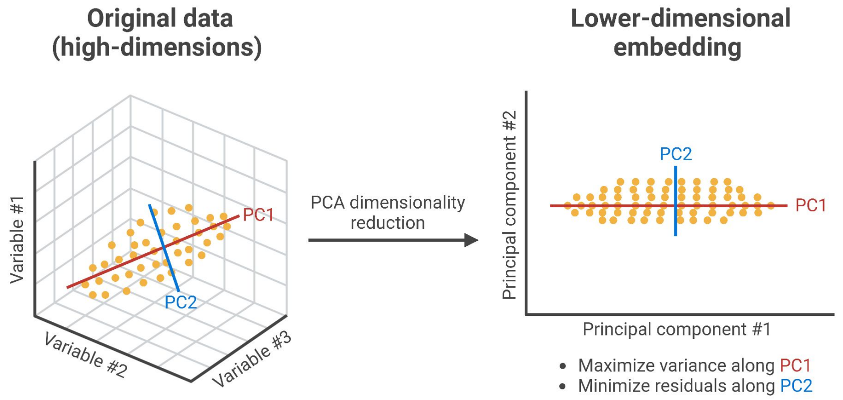
So, how can we do it?

Principal Component Analysis - Simplest Method

Reorient the data in the direction of maximal variance

1. Center the data
2. Calculate the covariance matrix
3. Perform eigendecomposition
4. Sort and select n principal components
5. Project the data onto the reduced space

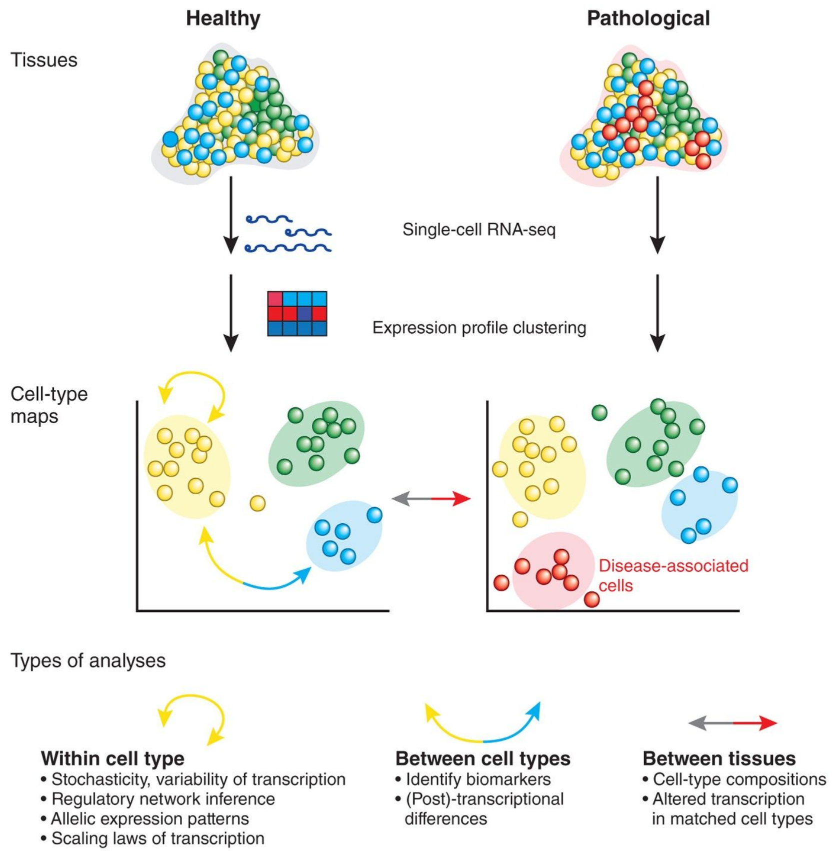



Original data (high-dimensions)
Lower-dimensional embedding
Variable #1
Variable #2
Variable #3
PC1
PC2
PCA dimensionality reduction
Principal component #2
Principal component #1
• Maximize variance along PC1
• Minimize residuals along PC2

```
X_centered = X - np.mean(X, axis=0)

cov_matrix = np.cov(X_centered, rowvar=False)

eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)

idx = np.argsort(eigenvalues)[::-1]

components = eigenvectors[:, idx[:n_components]]

X_reduced = X_centered @ components



from sklearn.decomposition import PCA

pca = PCA(n_components=2)

X_reduced = pca.fit_transform(X)
```

https://www.biorender.com/template/principal-component-analysis-pca-transformation

Trying to conserve global and local structure


Healthy
Pathological
Tissues
Single-cell RNA-seq
Expression profile clustering
Cell-type maps
Disease-associated cells
Types of analyses
Within cell type
• Stochasticity, variability of transcription
• Regulatory network inference
• Allelic expression patterns
• Scaling laws of transcription
Between cell types
• Identify biomarkers
• (Post)-transcriptional differences
Between tissues
• Cell-type compositions
• Altered transcription in matched cell types

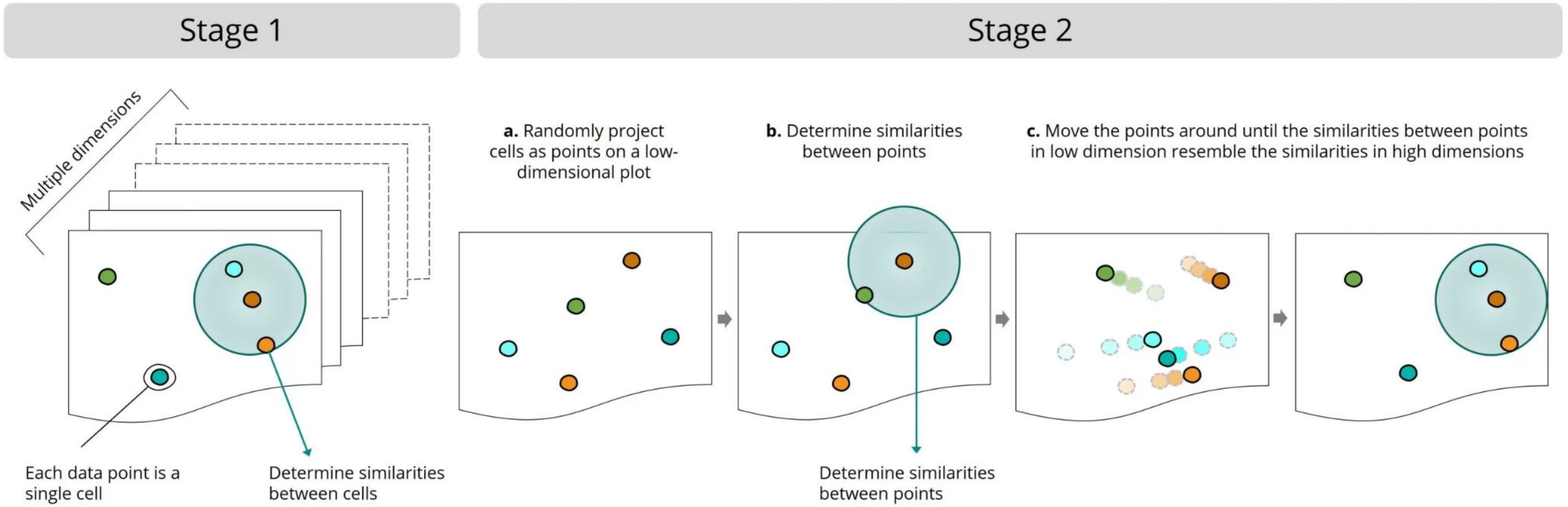
https://www.nature.com/articles/nmeth.2764

- Single-cell RNA-seq tells us how much each of millions of cells are expressing 10,000s of genes

	gene1,	gene2,	gene3,	gene4…
Cell 1 = [	0.2,	0.5,	1.0,	0.01…]
Cell 2 = [	1.0,	0.5,	0.2,	0.91…]

- Lots of types of cells and lots of variability in what cells are doing
- Don't know what each type of cell is during sequencing
- Need to cluster/project all this noisy data to lower dimensions to identify patterns

t-SNE (stochastic neighbour embedding) and UMAP


Stage 1
Stage 2
Multiple dimensions
a. Randomly project cells as points on a low-dimensional plot
b. Determine similarities between points
c. Move the points around until the similarities between points in low dimension resemble the similarities in high dimensions
Each data point is a single cell
Determine similarities between cells
Determine similarities between points

- Pairwise probability distribution in all dimensions
- Pairwise probability distribution in few dimensions
- Stochastic minimisation of KL divergence between distributions

```
from sklearn.manifold import TSNE
X = np.array([[0, 0, 0], [0, 1, 1], [1, 0, 1], [1, 1, 1]])
model = TSNE(n_components=2, learning_rate='auto', init='random', perplexity=3)
X_embedded = model.fit_transform(X)
X_embedded.shape
[4, 2]
```

https://www.scdiscoveries.com/blog/knowledge/what-is-t-sne-plot/

Summary

- Machine Learning: training models with label
- Scikit-Learn easy to use with great documentation/tutorials
- Supervised Learning: predicting output label (number or class) from data
  - Logistic Regression - linear regression with a sigmoid function and gradient descent
  - Split data into training and test data to evaluate generalisability of model
  - Cross-validation is used to tune a model/compare models without overfitting to test data
- Unsupervised Learning: finding structure in data without using labels
  - Clustering - inferring clusters in your dataset
    - K-means - pick k random points as “centroids” and move them to minimise the average distance of all points from these centroids.
  - Embeddings/Projections - finding a lower dimensional representation of the original data
    - t-SNE - move points around randomly to minimise difference between multivariate probability distribution in original dimension and lower dimensional embedding