Yale Biomedical Data Summer Course
Day 4

Sponsored by:
Yale Center for Biomedical Data Science
Yale Center for Medical Informatics

Machine Learning

Lecturers:
Prashant Emani
Jonathan Warrell
from the Gerstein lab in the Dept. of Molecular Biophysics and Biochemistry
Slide Design: Jonathan Warrell, Prashant Emani, Declan Clarke

Date: July 25th, 2019

Outline

1. Introduction to Machine Learning: basic concepts and intro to scikit-learn Python package

2. Unsupervised Learning:
   - PCA, Hierarchical Clustering, K-means Clustering

3. ”Linear” Supervised learning:
   - Linear Regression, Logistic Regression, Regularization, Random Forest/Tree-based methods

4. “Non-linear” Supervised Learning:
   - SVMs, Neural Network Models

5. Model Evaluation:
   - Cross-validation, AUC, Interpretation

Machine Learning: Definition and Basic concepts

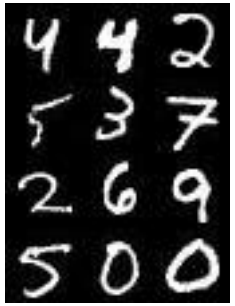
Machine learning is the process of finding patterns in data, as outsourced to computers.

In other words, we deem it "machine learning" when a computer, under certain guiding principles, finds a representation of the structure in data such that it can subsequently approximate the complex phenomenon in similar settings.

- "guiding principles": Human-directed rules and/or optimization goals
- "representation, approximate": Never an exact representation of the data, due to noise and/or heuristic nature of model
- "similar": Generalizability depends on degree of shared structure

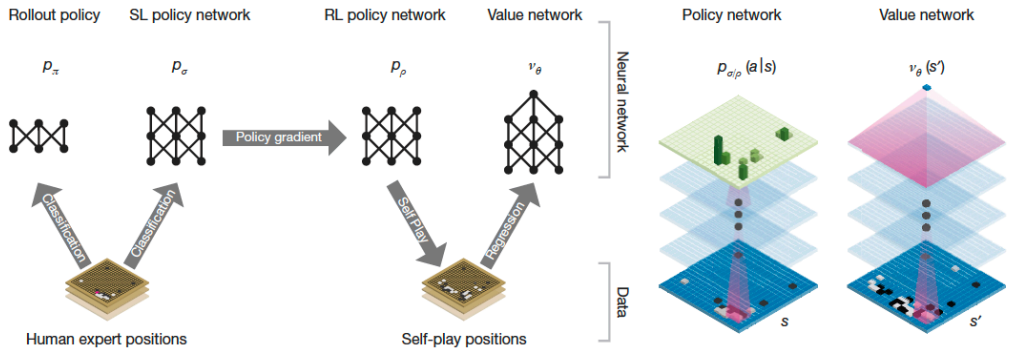
Examples of Machine Learning (ML) in practice

MNIST dataset: Classification of handwritten digits



http://yann.lecun.com/exdb/mnist/
https://github.com/teavanist/MNIST-JPG

AlphaGo: Teaching a computer to win at Go



Silver et al. 2016, *Nature* 529, Pgs. 484-489.

Broad categories of ML algorithms
(based on degree of human supervision)

I. Supervised Learning:

- Training a model based on a set of "gold standard" {input, target output} pairs
- Guiding principles: Structure is reflected in target output for a given input, so need to infer it

II. Unsupervised Learning:

- Discovering structure in data in the absence of curated set of target output
- Guiding Principles: Optimize (maximize or minimize) some target metric that tells you about the data samples and/or the features most important to structure determination

III. Reinforcement Learning:

- Discovering the rules of a task without curated data, but with some sense of reward
- Guiding Principles: Choose best sequence of steps to maximize long-term reward

Examples of the three categories

I. Supervised:

- Classify blood samples into cancer/non-cancer
- Predict chances of rain based on historical patterns, related to temperature and pressure
- Find text descriptions of images
- Netflix recommendations

II. Unsupervised:

- Build an evolutionary tree of finches based on similarity of beak shapes, body size, and other physiological features
- Cluster cells together based on gene expression, to yield groups of similar brain cell types

III. Reinforcement learning:

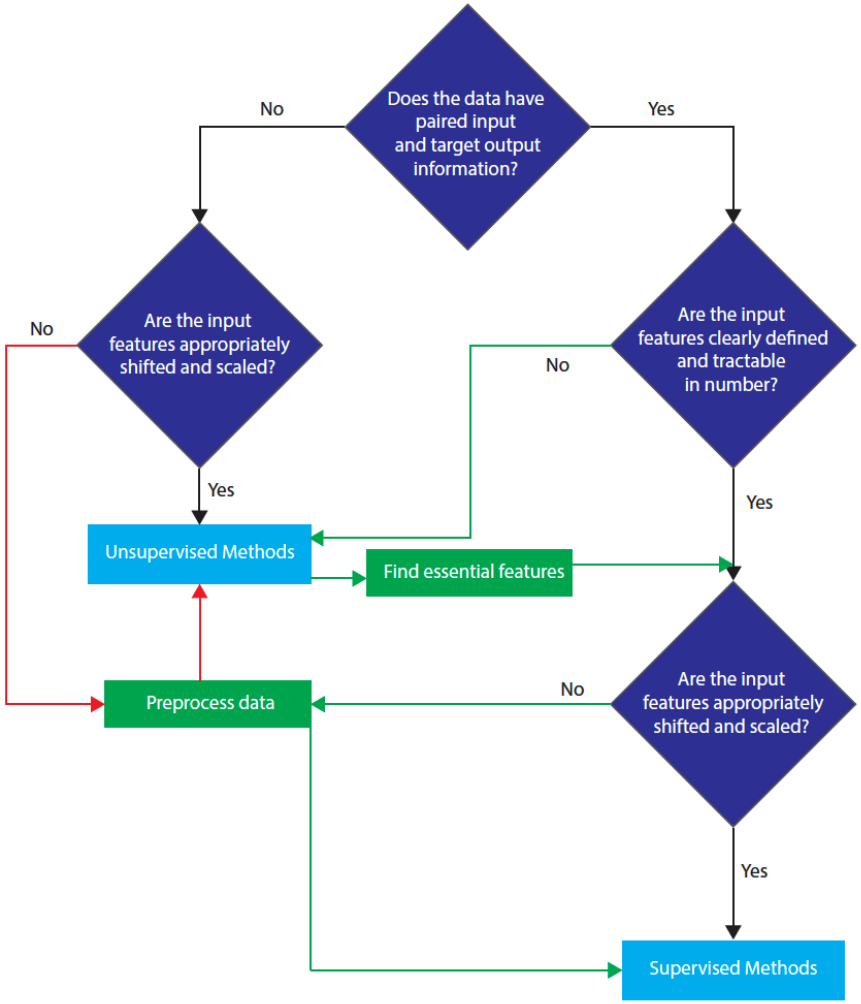
- AlphaGo learning to play Go, where the reward is winning
- Self-driving cars learning to avoid obstacles, where the reward is successfully arriving at the destination with no accidents

General questions before applying ML

1. What is the goal of your analysis?
   - Are you categorizing samples?
   - Are you predicting a quantity?
   - Are you looking for some structure in the data, without knowing what form it may take?

2. Will the dataset allow you to reach your intended goal?
   - Do you have labeled data from different categories for training?
   - Do you have **appropriate** controls?
   - Are there any confounding factors?
   - Are your datasets balanced?
   - Do you have appropriate statistical measures in place?
   - Is the data appropriately processed?

General strategy for choosing ML approaches



General strategy for choosing ML approaches(contd.)

I. Unsupervised:

a. **Principal Component Analysis (PCA):** Dimensional reduction, clustering similar samples, prioritizing features
b. **Hierarchical Clustering:** Inferring relative distance of samples to build trees
c. **K-means Clustering:** Finding cluster centroids

II. Supervised:

a. **Linear Regression:** Predict level of a continuous variable based on linear combination of other variables
b. **Classifiers:** Predict level of a binary/categorical variable based on other variables
    i. Logistic Regression
    ii. Random Forest
    iii. Support Vector Machines (SVMs)
c. **Neural Network Models:** General-purpose classifiers/predictors built from layers of simpler models

Avoiding bias, Part I:
Balancing datasets and accounting for confounding factors

- Imagine designing a classifier differentiating between patients with high blood pressure (BP) and normal-range BP using features of diet, smoking, exercise, medicine info:
  - Consider that your cohort includes 90 patients with high BP and 10 with normal-range BP, and your test set reflects the same.
  - Your classifier has 89% accuracy! Yay?
  - No! Nein! A classifier that randomly picks samples should give you a classification of high BP 90% of the time!

- You need to improve upon a random classifier. Usually, if you have sufficient samples, you can subsample to choose a more balanced cohort. Also, choose performance metrics other than accuracy.

- Equally essential, are "confounding factors":
  - These are potentially unobserved variables that mimic the target effect.
  - Eg: Family and Population structure in Genome-Wide Association Studies (GWAS)

Avoiding bias, Part II:
Overfitting

- Could potentially increase model complexity by increasing the number of parameters *ad infinitum*:
  - This can lead to '**overfitting**', where you customize your model perfectly to your training set, but it generalizes poorly
  - You end up fitting even the noise, and not just the essential aspects

- Overfitting arises when parameter contributions can grow in magnitude arbitrarily

- One solution is 'Regularization':
  - In addition to the usual cost function, you add a term that penalizes large magnitudes for the parameter contributions
  - Eg.: $Regularized\ Cost\ function = \frac{1}{2}\sum_{i=1}^{N}\left(y_i^{target} - \sum_{j=1}^{M} w_j x_{ij}\right)^2 + \frac{1}{2}\lambda\|\vec{w}\|^2$
  - This forces many parameter contributions to be small or 0 and so naturally controls the inflation associated with overfitting

Avoiding bias, Part III:
Validation, Cross-validation and model selection

- Methods like regularization add more parameters (like the λ parameter from the equation on the previous slide)

- Also, sometimes you would like to choose between multiple possible models (say, choosing the highest order of the polynomial to be considered in your model)

- Need to estimate these so-called "hyperparameters", i.e. parameters beyond the model parameters

- Validation and cross-validation:
  - Choose a subset of the dataset, in addition to training and testing, and run models with different choices of hyperparameters:

$$train \rightarrow validate\ models \rightarrow test$$

  - **Validation:** Split dataset
  - **Cross-validation:** n-fold; leave-one-out (LOOCV)

Avoiding bias, Part IV:
Avoiding prejudiced models

- There is a myth that ML is agnostic and unprejudiced because the human element is removed.

- But ML algorithms are only as free from bias as the people who program them.

- Sources of bias:
  - **Datasets not reflective of diversity:** Eg., lack of ethnic diversity in many, many genomics studies.
  - **Choice of parameters:** The assumption that the only parameters that matter are those that can be easily measured is potentially problematic.
  - **Confirmation bias:** With preponderance of methods, could cherry-pick the one that fulfills preconceived notions

- Extremely important when these are used to make medical or policy decisions that could affect many lives.

Structure of each ML section

- Definition: Qualitative Description
- Context of use with examples: When to use the method, potential limitations
- Mathematical formulation: Mostly linear algebra and basic calculus
- Cost function (or Guiding Principle): Every ML method is based on optimizing a certain quantity that significantly affects the results
- Relevant scikit-learn code: This code will be relevant for the lab sections

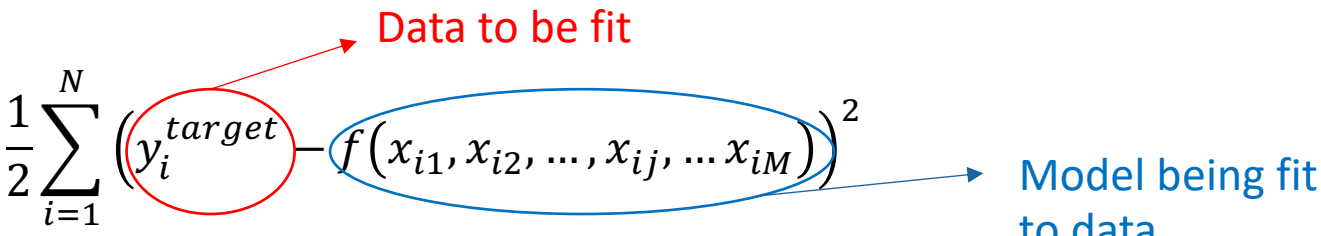
Cost function

- This is the quantity that is maximized or minimized in finding the best model

Example:

For a first time visit to a new place, you are going to find the best route to get there. Here, 'best' means the route that minimizes the distance and/or time to get from start to finish.

- Common choice of cost function:



= Distance between target data and model

Q&A

Introduction to scikit-learn

1. Import Libraries and Functions

```
import pandas as pd
from sklearn import preprocessing,decomposition
import plotnine as p9
from sklearn import cluster
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
```

2. Create objects of relevant classes: Object-oriented programming

```
#### Create a PCA object with 5 components (an object with no data at first)
pca = decomposition.PCA(n_components=5)
```

3. For an object, call necessary functions

```
#### Now use "fit_transform" to run a PCA on the normalized gene expression data and project it onto the first
#### 5 components; this is the reduced description
gene_reduced = pca.fit_transform(gene_expr_norm)
```

Additional coding structures

4. Read files into pandas dataframes

```
gene_expr = pd.read_csv(
    "/content/gdrive/My Drive/[YCMI_CBDS Summer Course] Data/bioinformatics/TEP_RNAs
                        sep="\t",
                        header=None)
```

5. Array subsetting

```
X = train[1:,1:]
y = train[1:,0]
geneIds = train[0,1:]
X2 = test[1:,1:]
y2 = test[1:,0]
```

6. For loops

```
for i in range(10):
    clf = LogisticRegression(C=10**(1-i), penalty='l2', solver='liblinear')
    sc = cross_val_score(clf, X, y, cv=5)
    scores.append(sc.mean())
```

Statistical terms

1. Sample Mean:

$$Mean\ \bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

2. Sample Standard Deviation:

$$Standard\ Deviation\ \sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

3. Expectation values:

$$\mathbb{E}[f(x)] = \int dx\ P(x)\ f(x)$$

for known, continuous distributions

$$\mathbb{E}[f(x)] \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

when samples are drawn from the underlying distribution

Statistical terms (contd.)

4. Covariance between two variables:

$$Cov(X,Y) = \mathbb{E}[(X - \bar{X})(Y - \bar{Y})] = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})$$

5. Correlation between two variables:

$$Corr(X,Y) = \frac{Cov(X,Y)}{\sigma_X \sigma_Y} = \frac{\mathbb{E}[(X - \bar{X})(Y - \bar{Y})]}{\sigma_X \sigma_Y}$$

References

1. *Pattern Recognition and Machine Learning*, Christopher M. Bishop, Springer 2006.

Unsupervised Learning I:
Principal Component Analysis (PCA)

- **Definition:** PCA is the construction of a set of new variables — a linear combination of the original variables (or equivalently, a rotation of the original variables) — where the covariance matrix is: (a) diagonalized; and (b) ranked in descending order of variance.

- In short, find the directions along which the data is most "spread-out".

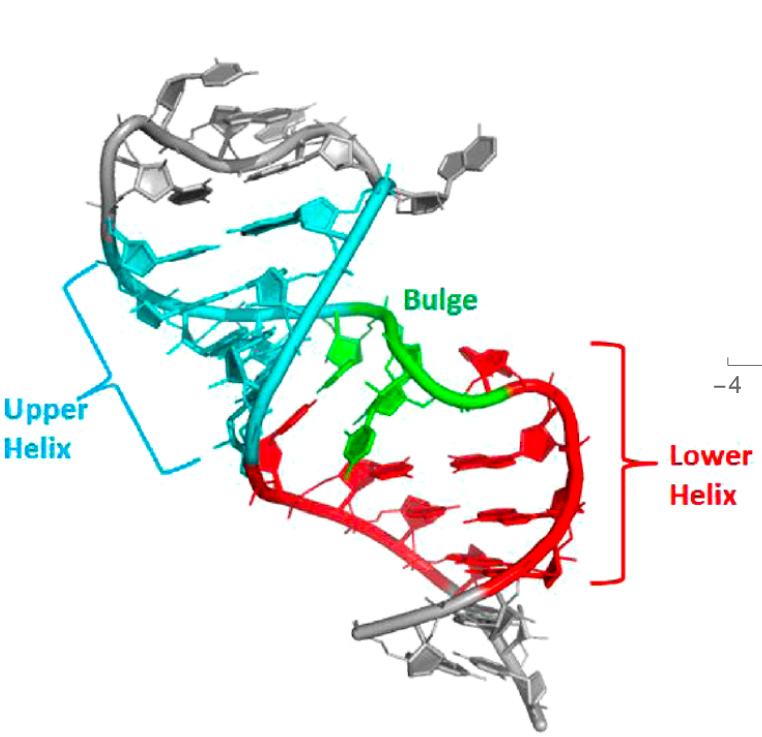
- **Context of use with examples:**

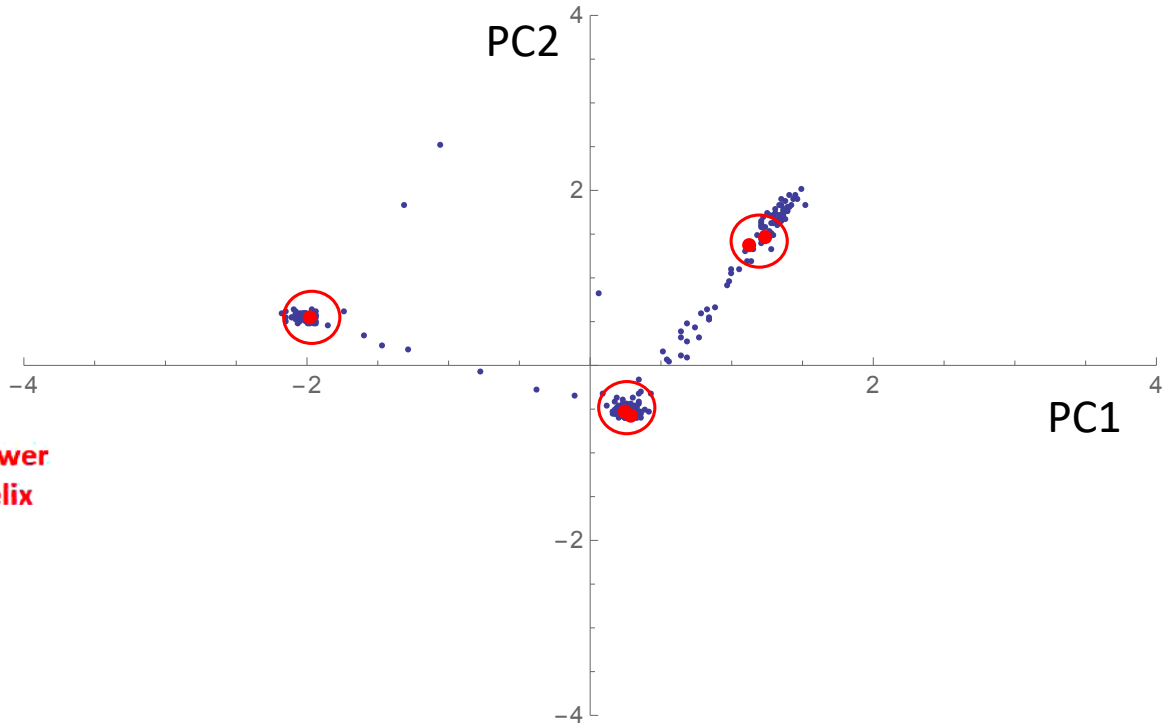
  PCA is used for
  - Reducing the number of dimensions to the most important ones
  - Discover structure, by identifying relevant variables

  Example:

  Torsion angles (~70 different ones) along the backbone of RNA can be used to identify clusters of RNA conformation

Principal Component Analysis (PCA) (contd.)





Emani et al **2014**, *J. Phys. Chem. B*, 118, Pgs. 1726-1742

Principal Component Analysis (PCA) (contd.)

- **Mathematical formulation:**

$$\text{Construct } \overleftrightarrow{X}^T.\overleftrightarrow{X} = \overleftrightarrow{A}_{M\times M} = \begin{pmatrix} (\sigma_{x_1})^2 & \cdots & Cov(x_1, x_M) \\ \vdots & \ddots & \vdots \\ Cov(x_M, x_1) & \cdots & (\sigma_{x_M})^2 \end{pmatrix}$$

$$\text{Find } t_k = \sum_{m=1}^{M} w_m^{(k)}.x_m \text{ such that the new variables } \{t_k\}_{k=1,\ldots,M} \text{ capture}$$

the most variance, the secondmost variance, etc.

The result turns out to be the eigenvalue decomposition:

$$\overleftrightarrow{X}^T.\overleftrightarrow{X} = \overleftrightarrow{W}.\overleftrightarrow{\Lambda}.\overleftrightarrow{W}^T$$

$$\overleftrightarrow{W} = (\vec{w}^{(1)} \quad \vec{w}^{(2)} \ldots \quad \vec{w}^{(M)}) \; and \; \overleftrightarrow{\Lambda} = \begin{pmatrix} (\sigma_{t_1})^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & (\sigma_{t_M})^2 \end{pmatrix}$$

Principal Component Analysis (PCA) (contd.)

- **Cost function/Guiding Principle:** Choose weights of new variables such that variance is maximized in decreasing order of contribution to the total variance

- **scikit-learn Code:**

1. Import libraries:

```
from sklearn import decomposition
from sklearn.decomposition import PCA
```

2. Define objects and number of components:

```
pca = decomposition.PCA(n_components=5)
```

3. Project your data onto the top 'n_components'; this is the dimensional reduction step (data_matrix with features as columns and samples as rows):

```
data_reduced = pca.fit_transform(data_matrix)
```

Unsupervised Learning II:
Hierarchical Clustering

- **Definition:** Hierarchical Clustering is the process of successively clustering (*agglomerative clustering*) or dividing (*divisive clustering*) groups of samples using measures of the relative (dis)similarity between them, until all the samples have been merged or separated, respectively.
- In short, find the "ancestral" tree of samples, where each generation involves the merging/dividing of two subclusters.

- **Agglomerative:** Keep merging clusters until all samples are in the same unified cluster
- **Divisive:** Keep dividing clusters until all samples are separate

- **Context of use with examples:**

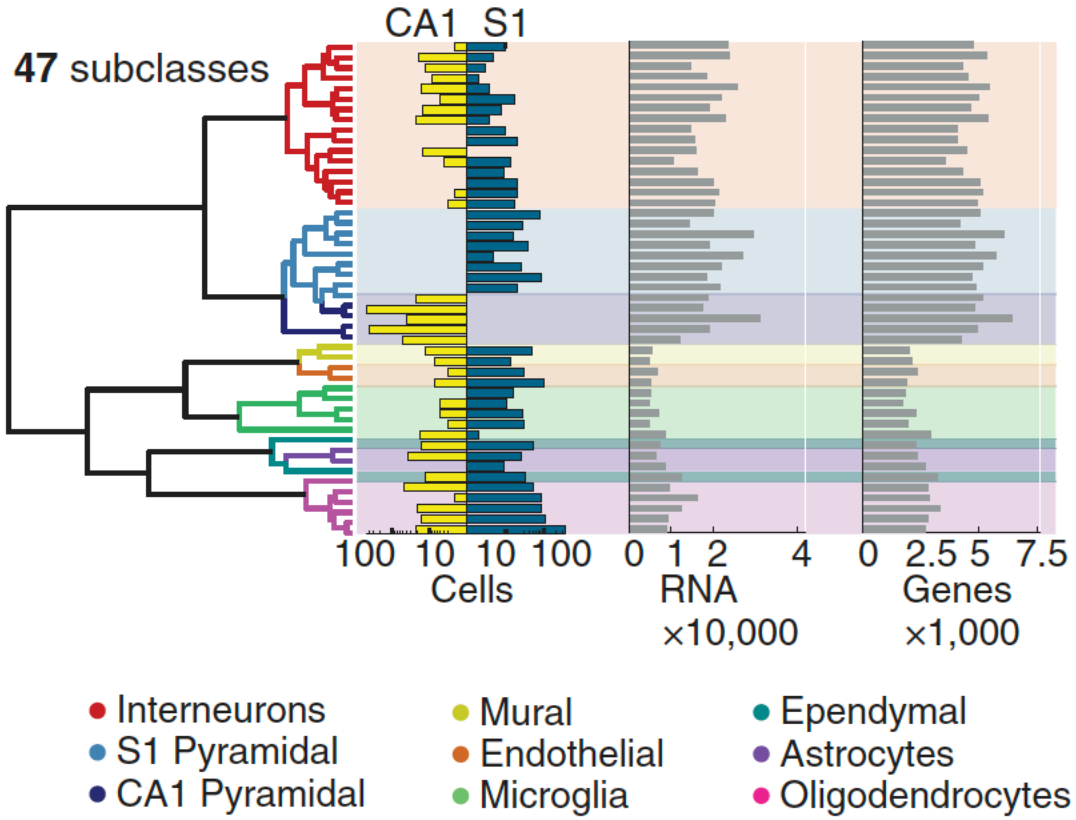
  Hierarchical Clustering is used for

  - Finding clusters in the data, by identifying relative relatedness between samples and larger patterns of grouping.

  Example:

  Clustering cells from the mouse brain based on gene expression patterns

Hierarchical Clustering (contd.): Example Dendrogram



Zeisel et al **2015**, *Science*, 347(6226), Pgs. 1138-1141

Hierarchical Clustering (contd.)

- **Mathematical formulation:**

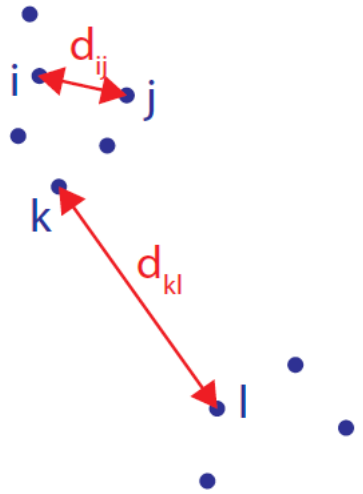
Distance metric:

$$\text{Euclidean } d_E(\vec{a},\vec{b}) = \sqrt{\sum_{i=1}^{N}(a_i - b_i)^2}$$

$$\text{Manhattan } d_M(\vec{a},\vec{b}) = \sum_{i=1}^{N}|a_i - b_i|$$

$$\text{Correlation } d_{Corr}(\vec{a},\vec{b}) = \frac{\sum_{i=1}^{N}(a_i - \bar{a})(b_i - \bar{b})}{\sigma_a \sigma_b}$$

$$\text{Cosine } d_{Cos}(\vec{a},\vec{b}) = \frac{\sum_{i=1}^{N} a_i b_i}{\sqrt{\sum_{i=1}^{N}(a_i)^2}\sqrt{\sum_{i=1}^{N}(b_i)^2}}$$



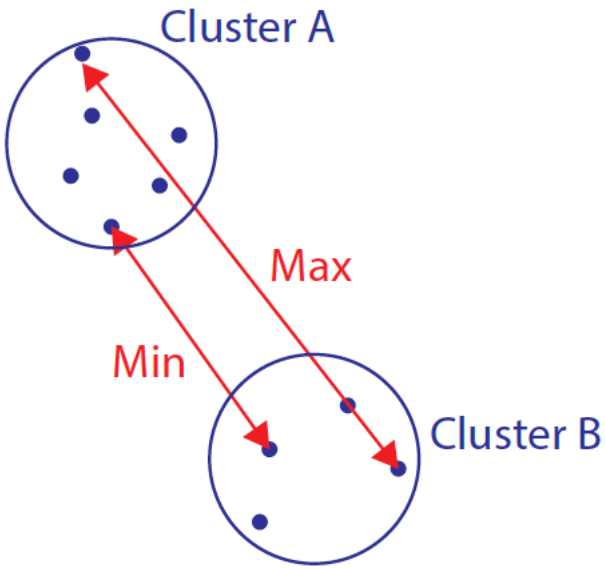
Hierarchical Clustering (contd.)

- **Linkage metric**:
  Metric to calculate "*distance*" between clusters

  Maximum or **complete** linkage L(A, B)
  $= \max\{d(\vec{a}, \vec{b}), \forall \vec{a} \in A, \vec{b} \in B\}$

  Minimum or **single** linkage L(A, B)
  $= \min\{d(\vec{a}, \vec{b}), \forall \vec{a} \in A, \vec{b} \in B\}$

  Average $L(A, B) = \text{mean}\{d(\vec{a}, \vec{b}), \forall \vec{a} \in A, \vec{b} \in B\}$



- **Agglomerative:** At every iteration, look for most similar clusters by linkage and merge
- **Divisive:** At every iteration look for most dissimilar clusters and separate

Hierarchical Clustering (contd.)

- **Cost function/Guiding Principle:** Choose the merge/division sequence based on maximizing similarity/dissimilarity.

- **scikit-learn Code:** Two different methods

Note: Will use scipy's 'linkage' for the sake of plotting dendrograms

1. Import libraries:

```
from sklearn import cluster
from scipy.cluster.hierarchy import dendrogram, linkage
```

2. *Find linkage matrix*/Define objects and number of clusters:

```
linked_matrix = linkage(data_matrix, metric ='euclidean',method='average')
cluster_object = cluster.AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='complete')
```

3. *Plot dendrogram*/Do a fit and predict cluster assignments:

```
dendrogram(linked_complete,orientation='top',labels=class_labels,distance_sort='descending',
           leaf_font_size=8.5,color_threshold=5000)
cluster_assignments = cluster_object.fit_predict(data_matrix)
```

Unsupervised Learning III:
K-means Clustering

- **Definition:** K-means Clustering is the assignment of samples to a pre-defined number (K) of clusters, by defining a set of cluster means such that the sum of the squared distance of points to their assigned means is minimized.

- In short, find the optimal cluster assignments for points, and the optimal set of cluster reference points.

- **Context of use with examples:**

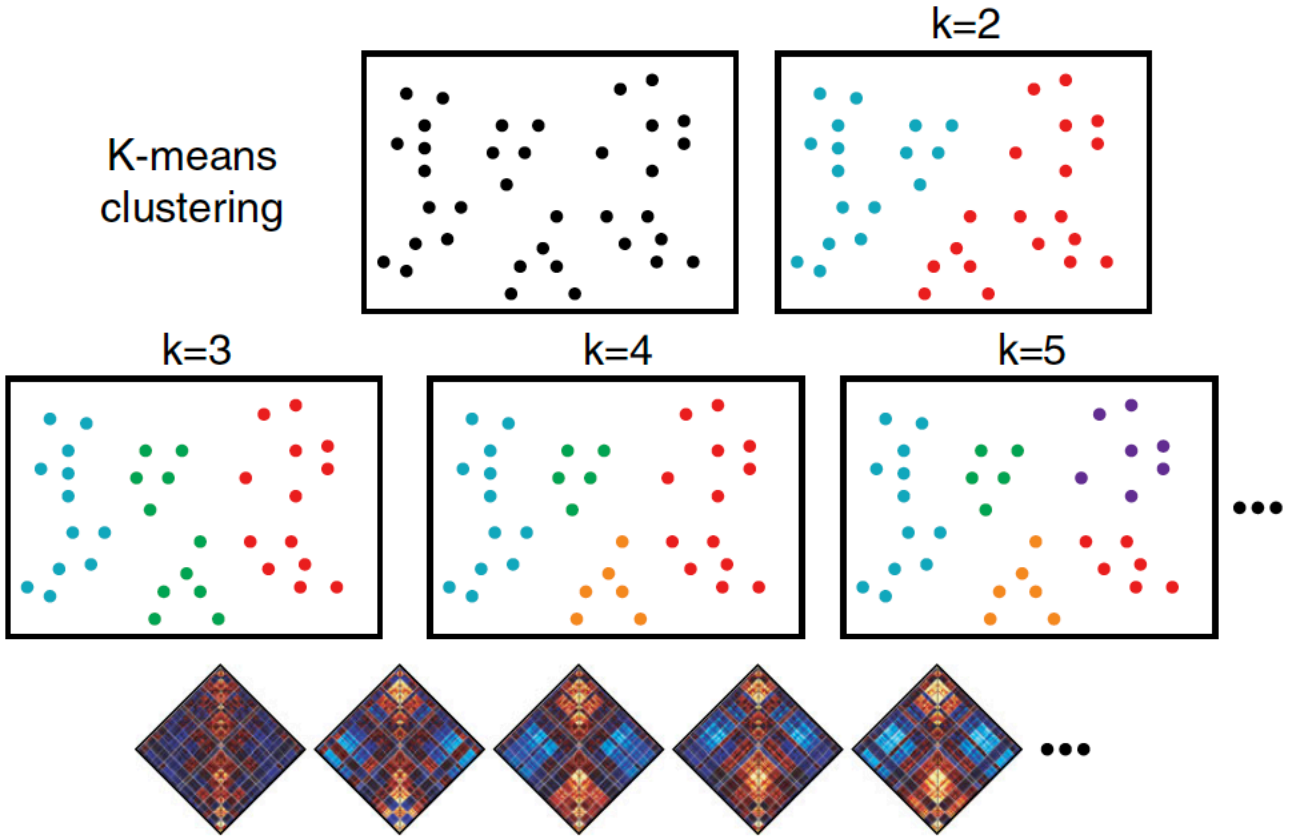
  K-means Clustering is used for

  - Identifying clusters in the multi-dimensional data, by identifying the prototypic cluster centers.

  Example:

  Finding clusters of resting-state functional connectivity (fMRI) patterns in the brain.

K-means Clustering (contd.)



Reinen et al **2018**, *Nat. Comm.*, 9:1157

K-means Clustering (contd.)

- **Mathematical formulation:**

Need to minimize $J = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}\|\vec{x}_n - \vec{\mu}_k\|^2$ with respect to r_{nk} and $\vec{\mu}_k$

where r_{nk} is the membership of data point $\vec{x}_n$ in cluster k, $\in \{0,1\}$;
$\vec{\mu}_k$ is the reference point for cluster k

Alternating estimation of these two sets of parameters:

1. First, assign points to the K clusters, either randomly, or efficiently to improve convergence.
2. Then calculate $\vec{\mu}_k$ = center of the points assigned to cluster k.
3. Reassign the points to clusters according to which $\vec{\mu}_k$ they are closest to.
4. Return to step 2, and repeat until results converge.

K-means Clustering (contd.)

- **Cost function/Guiding Principle:** Identify clusters by minimizing the distance between samples and defined cluster centers.

- **scikit-learn Code:**

1. Import libraries:

```
from sklearn.cluster import KMeans
```

2. Define objects and number of clusters:

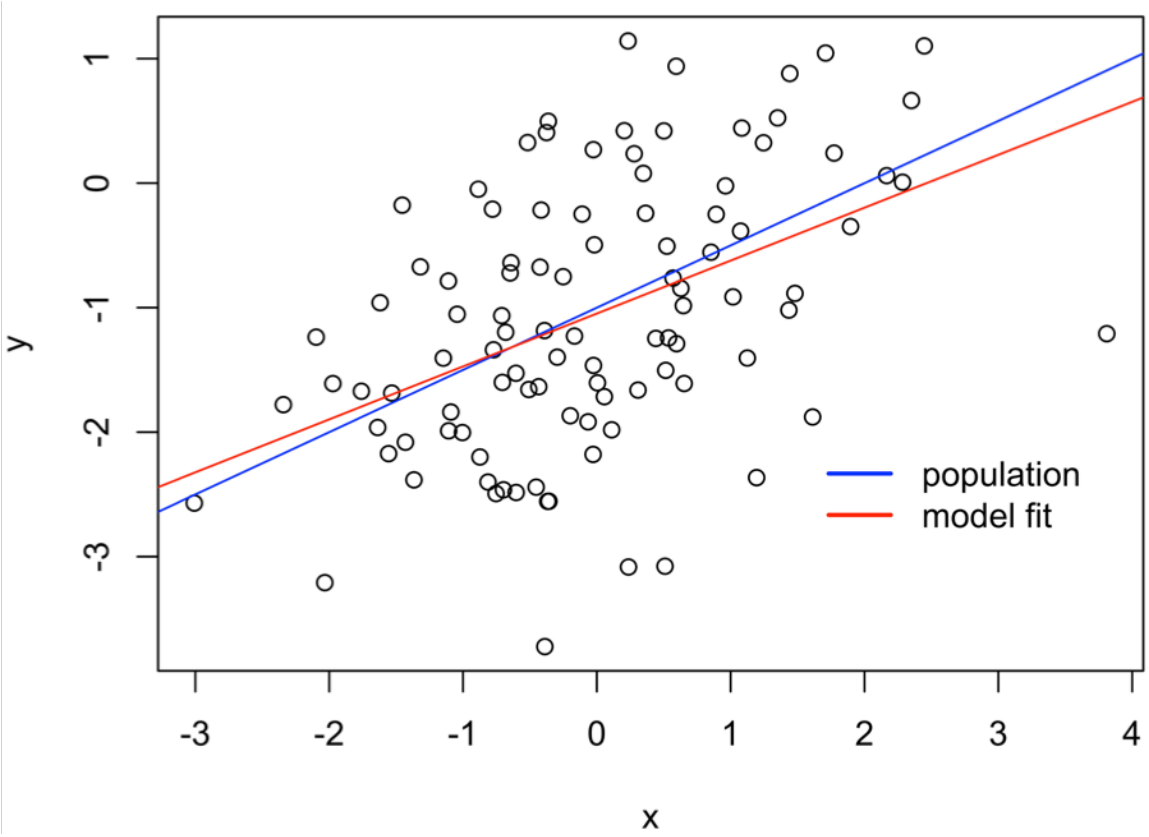
```
kmeans_object = KMeans(n_clusters=4, init='random')
```

3. Do a fit and predict cluster assignments; extract cluster labels and centers:

```
kmeans_object.fit(data_matrix)
kmeans_object.labels_
kmeans_object.cluster_centers
```

Supervised Learning Ia:
Linear Regression

Definition: A simple model in which a response variable y is linearly dependent on predictor variable(s) → Build model to estimate population regression function

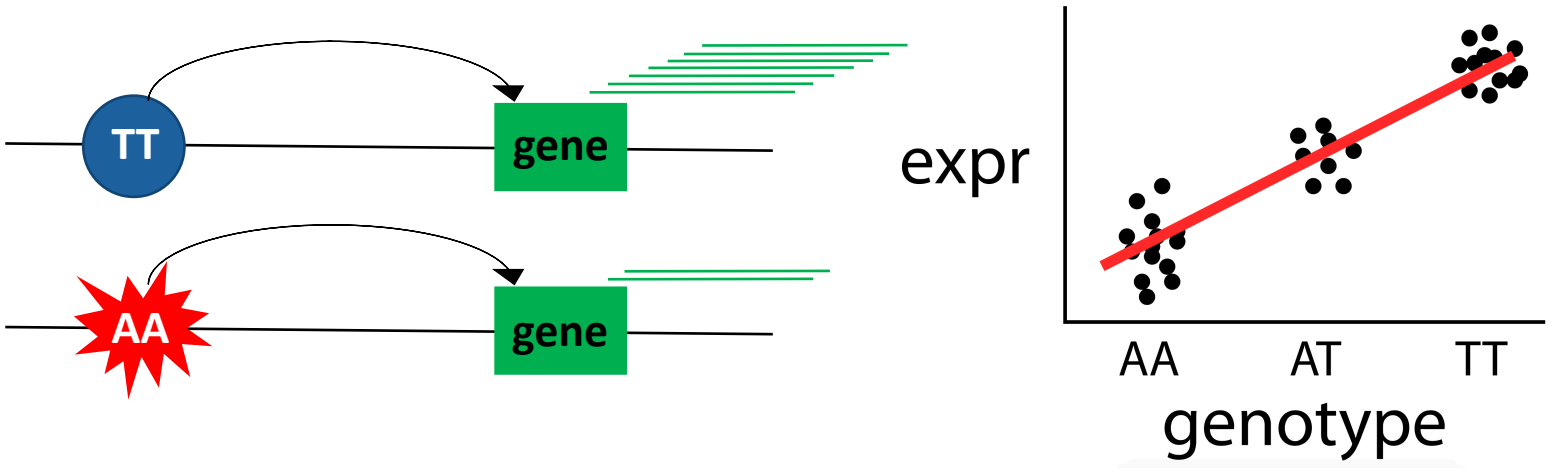


Data from James et al., 2013

Linear Regression (contd.)

Context of use with examples: LR is used if you believe that changes in the response variable are directly proportional to changes in one or more independent variables.

Example: changes in gene expression as a consequence of genomic variants (ie., eQTLs)



Linear Regression (contd.)

Mathematical formulation:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \cdots + \hat{\beta}_p x_p$$

$\boldsymbol{\beta} = [\mathbf{X}^T\mathbf{X}]^{-1}\mathbf{X}^T\mathbf{y}$

X: n x p matrix of n samples and p variables

y: the n-length vector of response values

Cost function/Guiding Principle:

Obtain the vector of β coefficients (ie, **β**) by minimizing the sum of squared residuals – ie, obtain **β** by minimizing:

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}\left(y_i - \sum_{j=0}^{p}\beta_j x_{ij}\right)^2$$

Linear Regression (contd.)

scikit-learn Code:

1. Import libraries:

```
import numpy as np
from sklearn.linear_model import LinearRegression
```

2. Import data or explore method using synthetic dataset

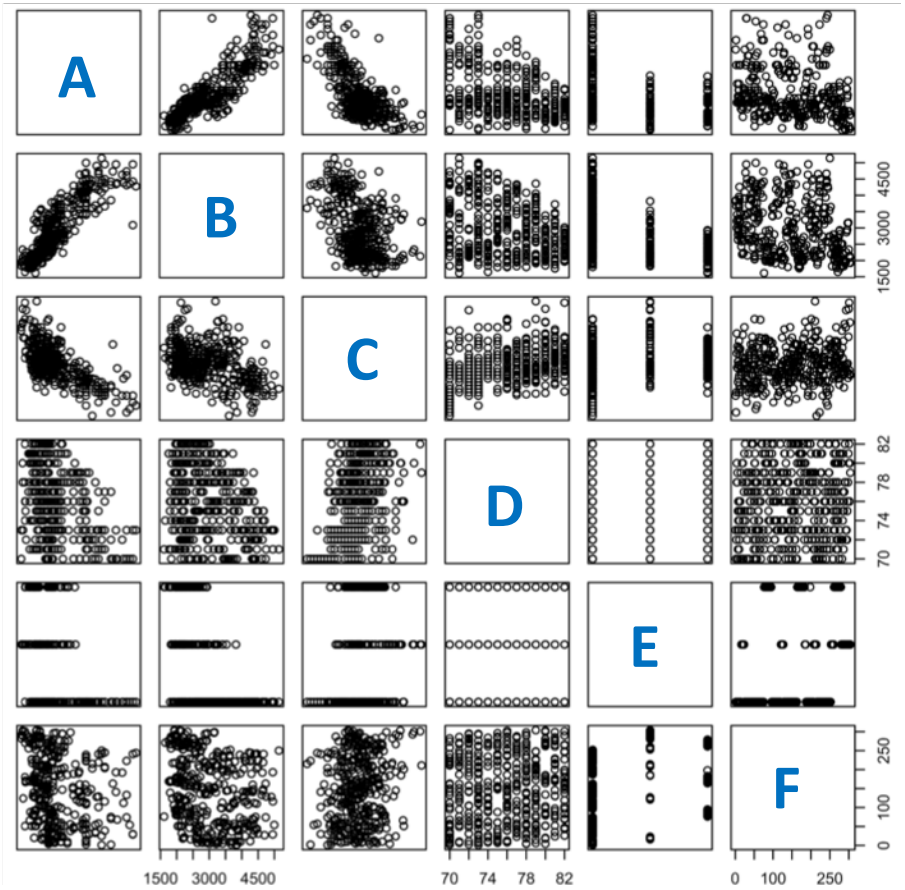
```
X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
# y = 1 * x_0 + 2 * x_1 + 3
y = np.dot(X, np.array([1, 2])) + 3
```

3. Perform regression

```
reg = LinearRegression().fit(X, y)
```

Linear Regression (contd.)

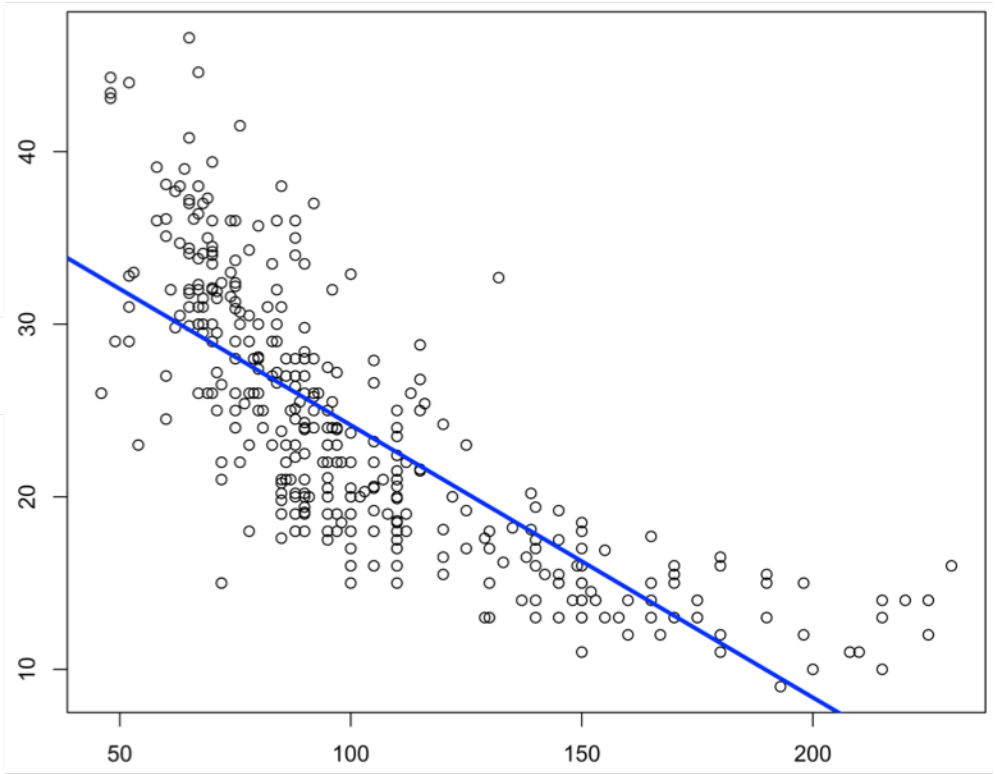
There may be many predictors available (A-F shown), in which pairwise relationships can easily be displayed graphically



Data from James et al., 2013

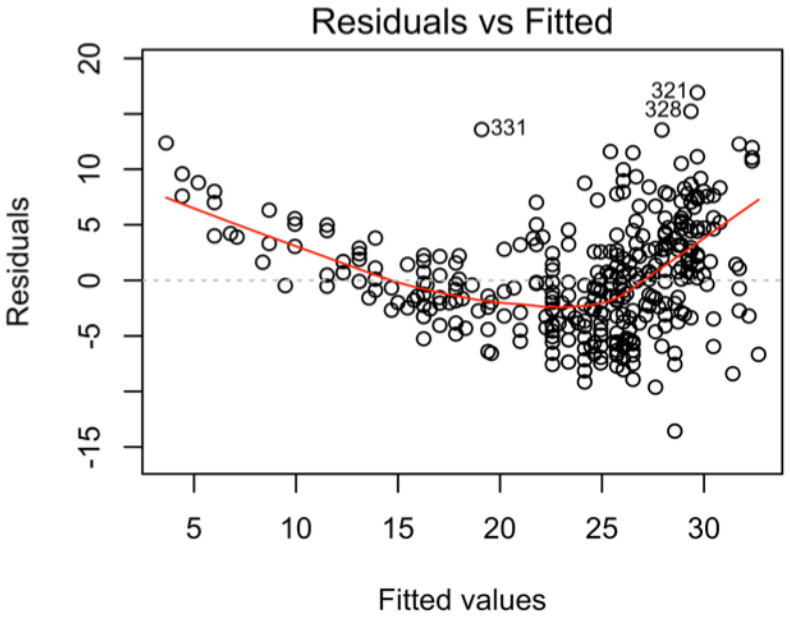
Linear Regression: Diagnostics

The relationship may not in fact be linear, in which case it may be necessary to also use transformations of the original variables (ex: polynomials). Check for non-linearity by plotting residuals.



Data from James et al., 2013

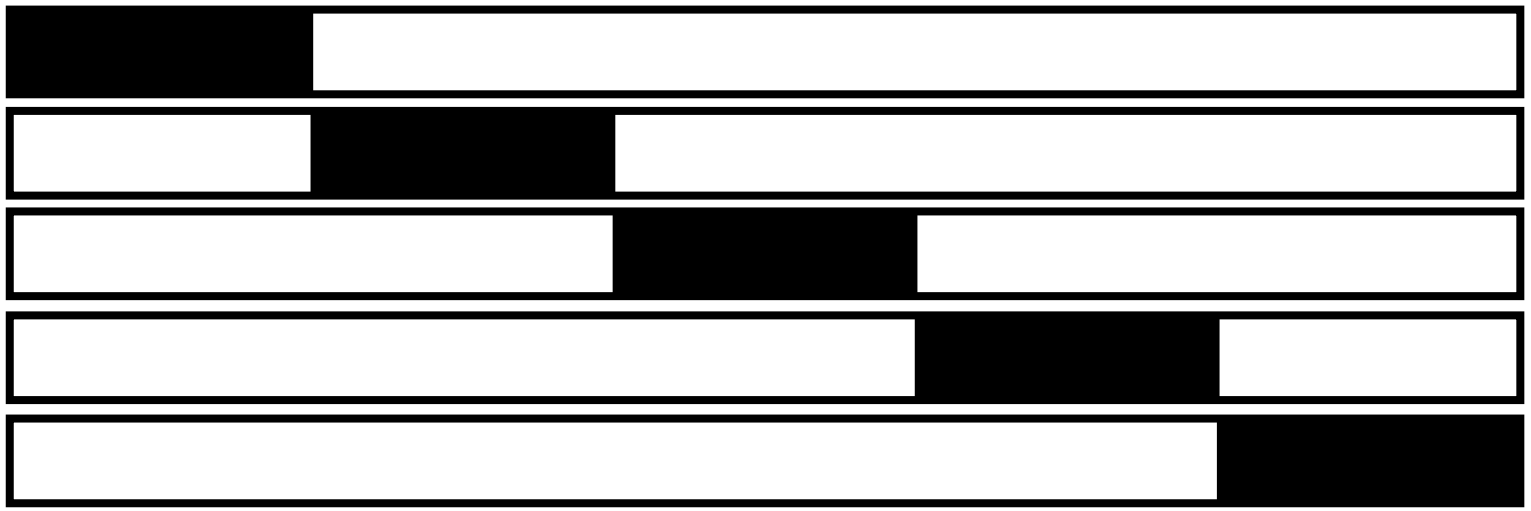
*Note: a log transformation will be applied in your lab on Warfarin dosing.



Linear Regression: Application of Cross-Validation

Sub-sample the data using k-fold cross-validation in order to perform parameter selection and/or to set parameter values

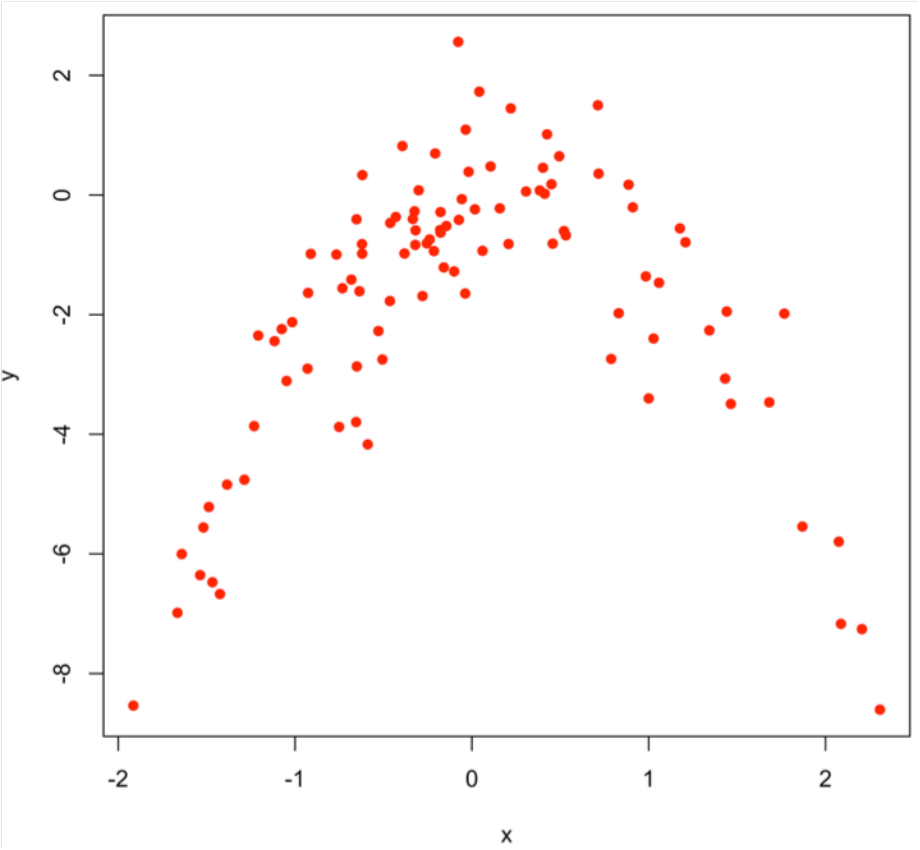
- 5-fold cross-validation schematized here



$$\mathrm{CV}_{(k)} = \frac{1}{k}\sum_{i=1}^{k}\mathrm{MSE}_i$$

Linear Regression: Application of Cross-Validation

Sub-sample the data using k-fold cross-validation in order to perform parameter selection and/or to set parameter values



i. $Y = \beta_0 + \beta_1 X + \epsilon$

ii. $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon$

iii. $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \epsilon$

iv. $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \beta_4 X^4 + \epsilon$

```
cv.error_i:   5.890979

cv.error_ii:  1.086595

cv.error_iii: 1.102585

cv.error_iv:  1.114772
```

Supervised Learning Ib:
Lasso for Regularization

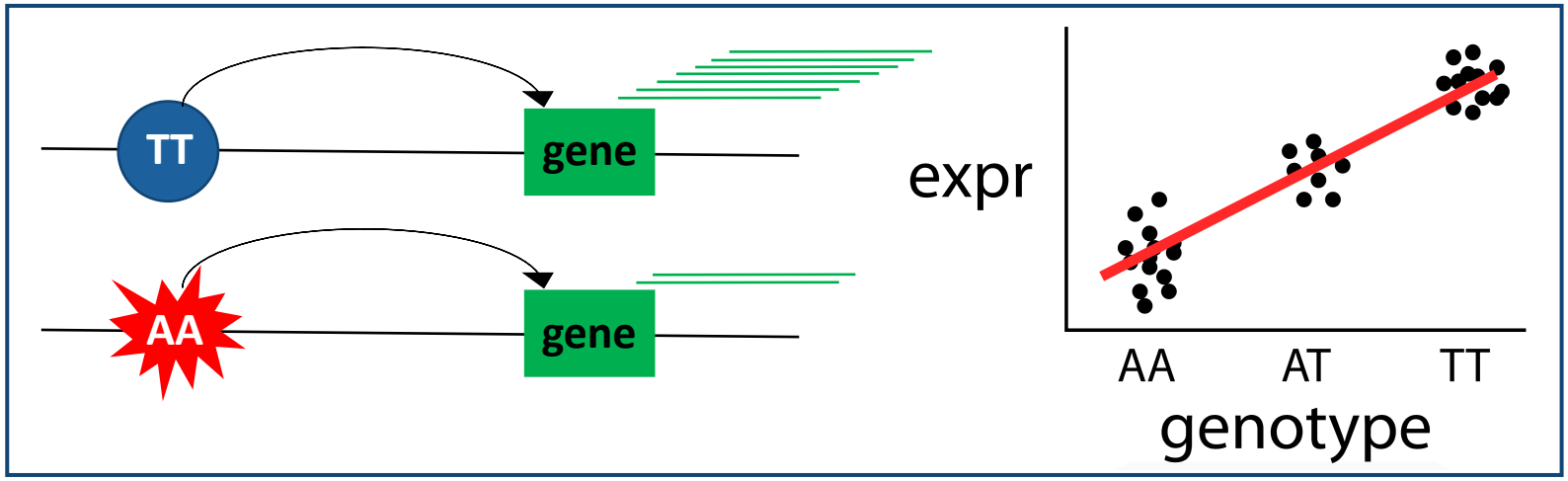
Definition: Similar to linear regression, but instead of just trying to minimize the squared errors in our training data, we simultaneously aim to:

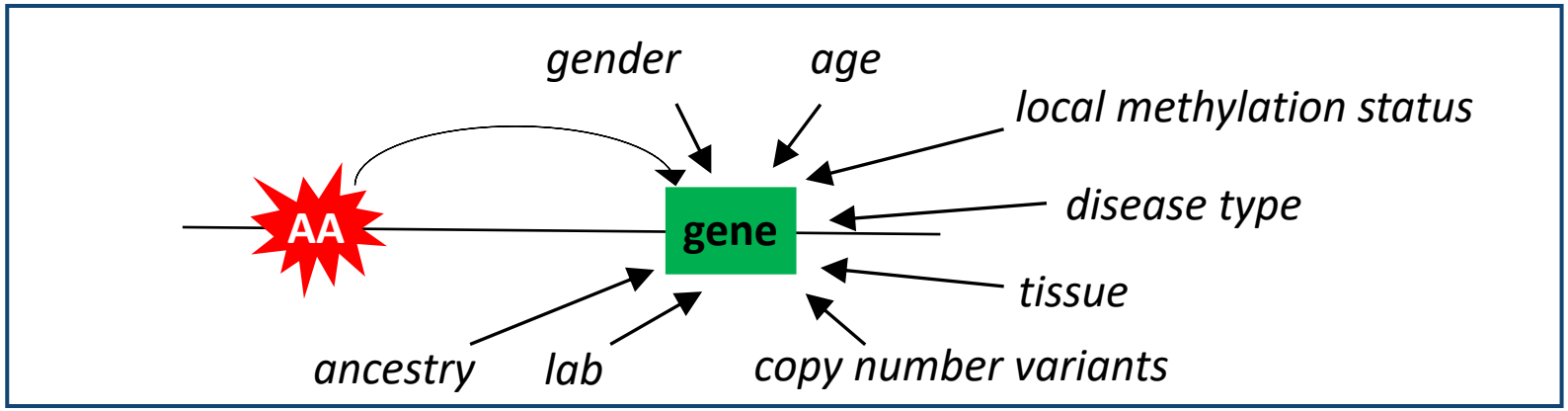
a) eliminate irrelevant variables by setting their associated coefficient values to 0 → *This is good! It accomplishes variable selection (thereby making models simpler, and thus more interpretable).*

b) reduce the magnitudes of the coefficients (i.e, β_j) associated with the relevant variables → *This is also good! It can often make the model more "stable" by reducing the variance of the response value being estimated. This reduction in variance often leads to improved predictive performance*

Lasso for Regularization

Context of use with examples:



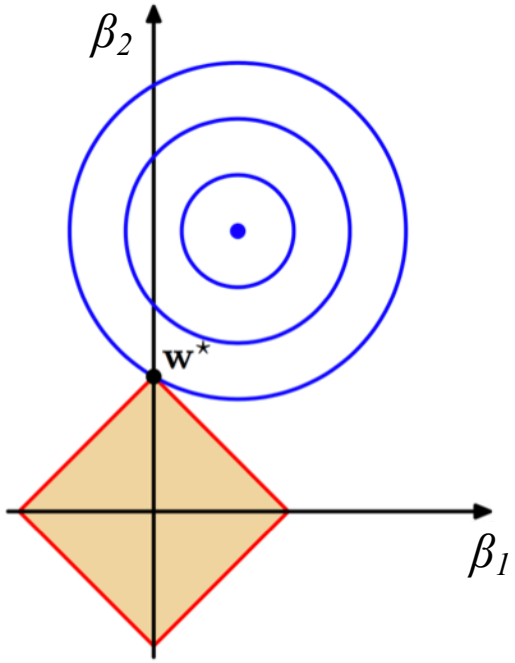


Lasso for Regularization

Mathematical formulation:

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}|\beta_j|$$

$$\underset{\beta}{\text{minimize}}\left\{\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2\right\} \quad \text{subject to} \quad \sum_{j=1}^{p}|\beta_j|\le s$$



Adapted from C. Bishop, 2006

Cost function/guiding principle:

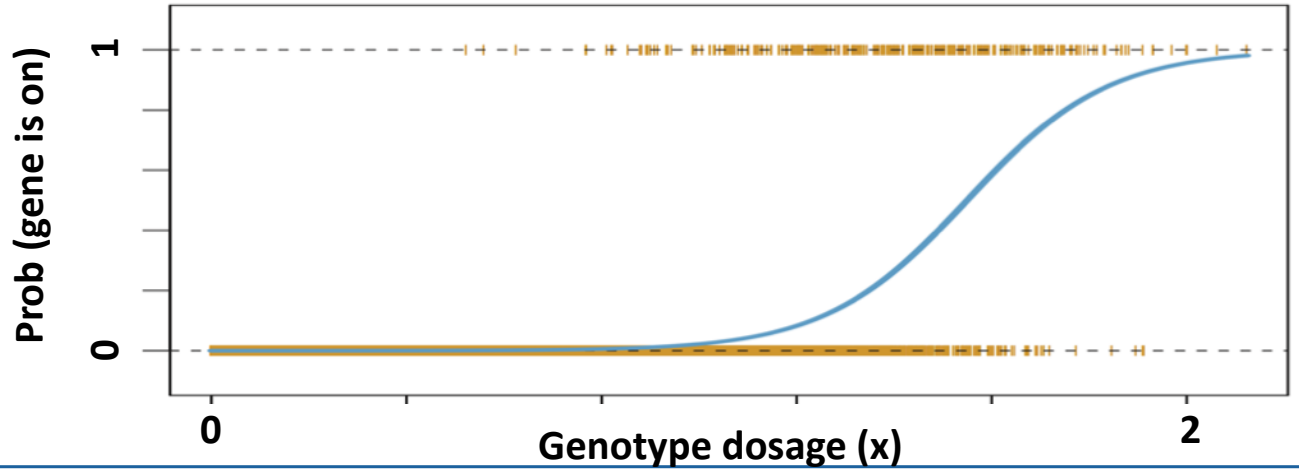
Reduce the magnitude of the regression coefficients while fitting the data

Supervised Learning II:
Logistic Regression (Predicting binary outcomes)

Definition/Description: The variable you're trying to predict might be a *binary variable*. Logistic regression allows you to use a training dataset to estimate the probability p that the response y will lie in one or the other of 2 different classes.

Example context: Predicting whether a gene is **on** or **off**, given the genotype dosage x (a continuous value between 0 and 2):

$$probability(gene\ is\ \boldsymbol{on}) = \frac{e^{\alpha+\beta x}}{1 + e^{\alpha+\beta x}}$$



Adapted from James et al., 2013

Logistic Regression (contd.)

scikit-learn Code:

1. Import library:

```
from sklearn.linear_model import LogisticRegression
```

2. Using previously imported data, fit model to estimate logistic regression parameters:

```
clf = LogisticRegression(C=(10**(1-i)),penalty='l2',solver='liblinear')
```

Supervised Learning II:
Tree-based methods (such as Random Forests)

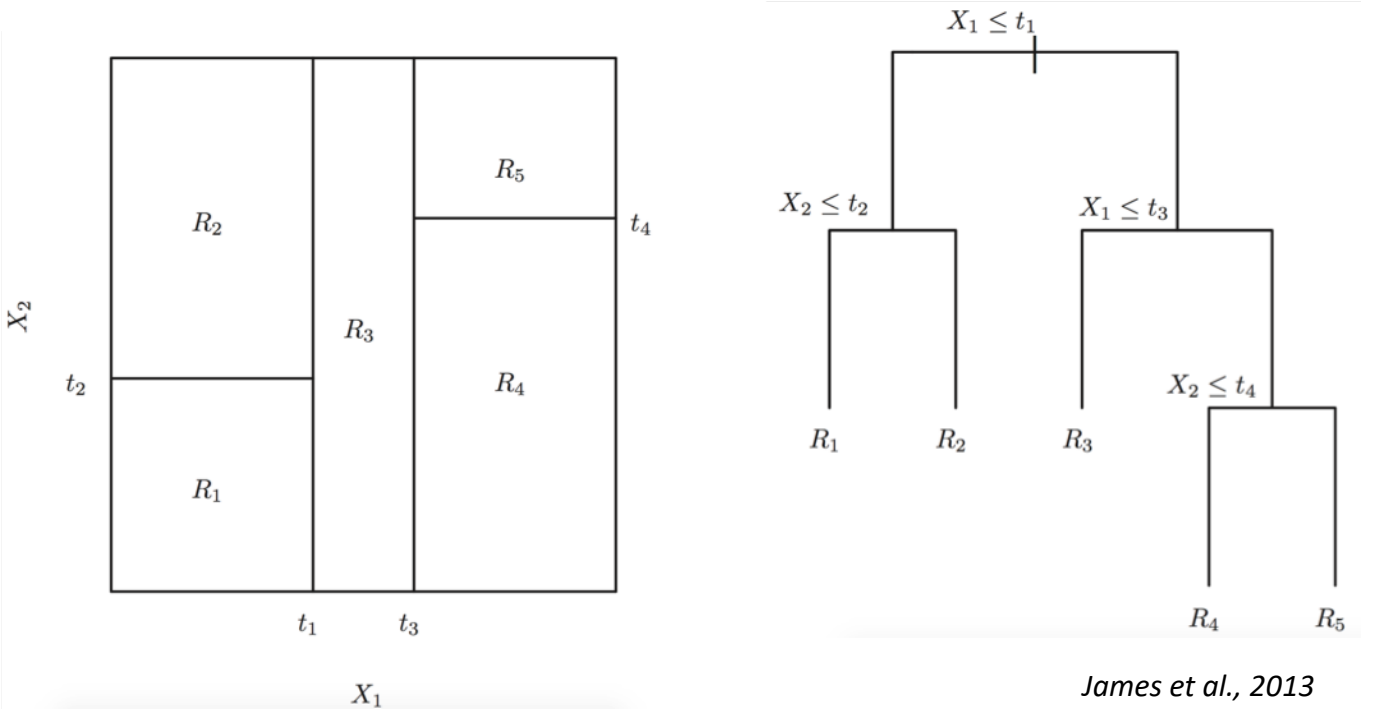
Definition: The idea is to partition the parameter space in such a way that each split in the space results in the best prediction within each subspace

Left: A 2-dimensional parameter space is partitioned into 5 subspaces

Right: Tree representation of the resultant partition

Let X_1 = gene dosage

Let X_2 = age



James et al., 2013

Tree-based methods (contd.)

scikit-learn Code:

1. Import library:

```
from sklearn.ensemble import RandomForestClassifier
```

2. Using previously imported data (X and y), build a random forest classifier:

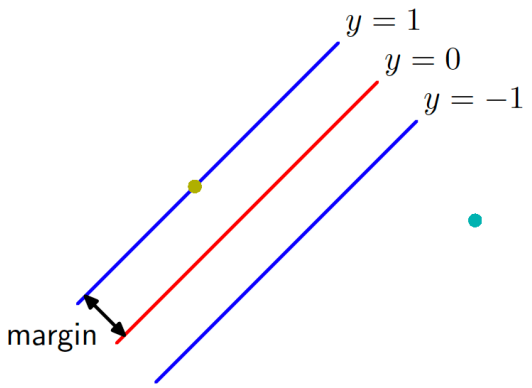
```
clf = RandomForestClassifier(n_estimators=20, max_depth=4, random_state=0)
clf.fit(X, y)
```

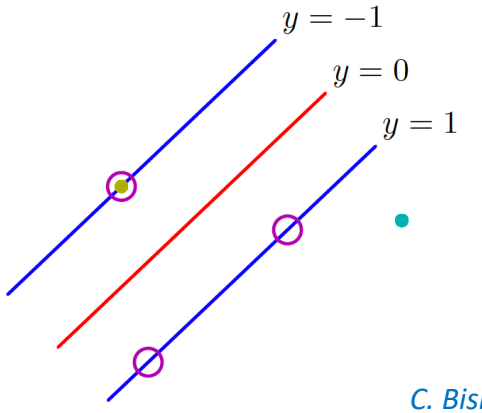
Supervised Learning III:
Non-linear Supervised Methods: SVMs

- Linear SVMs learn a classifier in the original feature space
- Non-linear SVMs learn a classifier in a transformed feature space; this may be **high-dimensional**

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}) + b$$

- SVMs maximize the **margin** in the high-d feature space

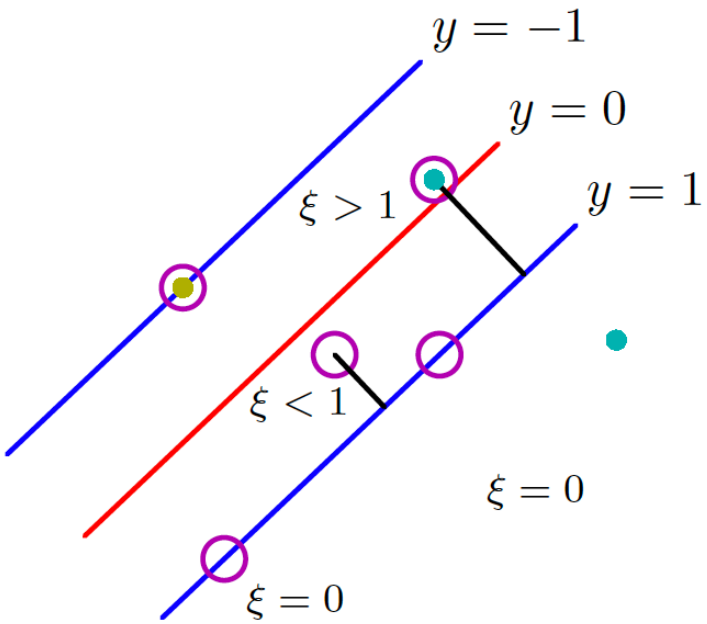




C. Bishop, 2006

Non-linear Supervised Methods: SVMs

- If data is not separable, may introduce **slack-variables**



C. Bishop, 2006

Non-linear Supervised Methods: SVMs

- Can be formulated as a quadratic (convex) optimization problem

Separable:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$$
$$\text{subject to: } y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1,\ \forall i \in [m]\,.$$

Non-separable:

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\xi_i^p$$
$$\text{subject to } y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i \ \wedge\ \xi_i \geq 0, i \in [m]\,,$$

Mohri et. al., 2018

Non-linear Supervised Methods: SVMs

- SVMs can use different **kernels** to increase complexity of feature space

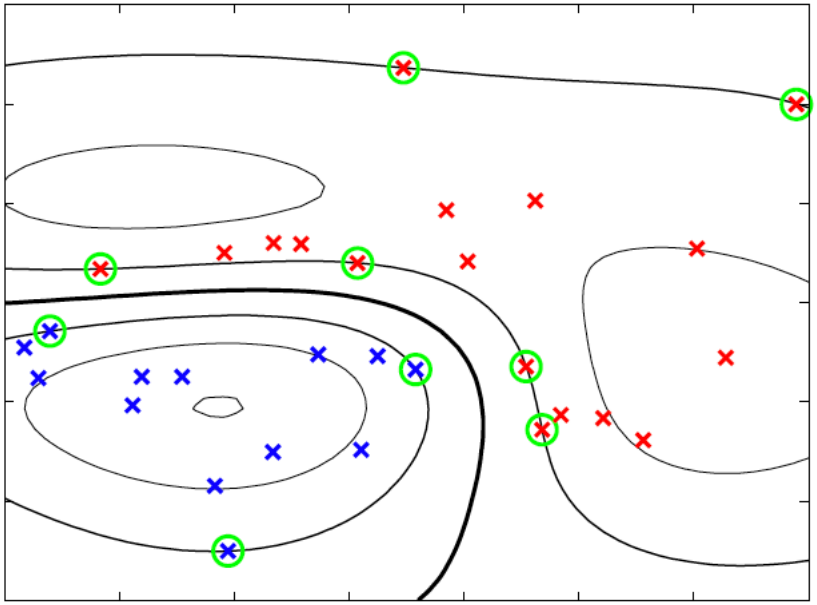
Linear: $K(x,x') = x^T x' + c$

Polynomial: $\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^N, \quad K(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + c)^d.$

Gaussian (RBF): $\forall\, \mathbf{x}, \mathbf{x}' \in \mathbb{R}^N, \quad K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}' - \mathbf{x}\|^2}{2\sigma^2}\right).$

Non-linear Supervised Methods: SVMs

- Gaussian (RBF) kernel example:



C. Bishop, 2006

Non-linear Supervised Methods: SVMs

scikit-learn code:

1. Training and testing:

```
from sklearn import svm
```

```
clf_svm = svm.SVC(kernel='linear', C=10)
clf_svm.fit(X, y)
```

```
train_acc = clf_svm.score(X, y)
print("training accuracy: " + str(train_acc))
```

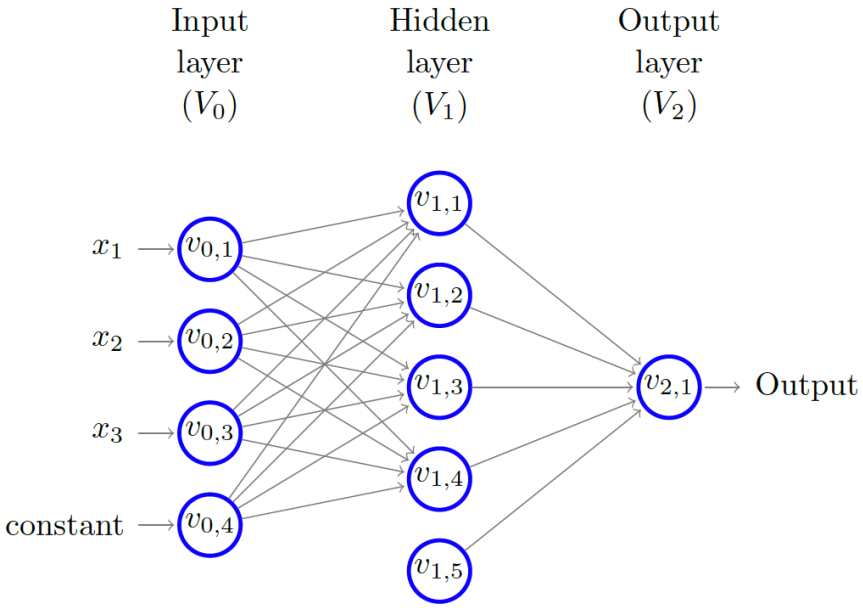
2. Useful code: cross validation

```
from sklearn.model_selection import cross_val_score
```

```
sc = cross_val_score(clf, X, y, cv=5)
```

Supervised Learning IV:
Non-linear Supervised Methods: Neural Networks

- Neural networks include a series of hidden layers between input and output



Recursive definition: $f(\boldsymbol{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x})))$.

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right)$$

Compare to logistic regression:

$$P(y) = \sigma(\mathbf{w}^{\mathbf{T}}\mathbf{x} + b)$$

C. Bishop, 2006; Goodfellow et. al., 2016

Non-linear Supervised Methods: Neural Networks

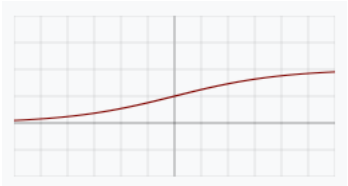
- Neural networks are typically trained using **back-propagation**

Gradient descent: $\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$

  - Back-propagation used to evaluate $\nabla E(\mathbf{w}^{(\tau)})$
  - May also use **stochastic gradient descent** (SGD) for robustness

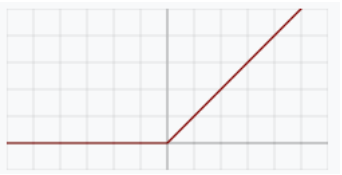
- Can use many different activation functions
  - Sigmoid
  - Tanh
  - RELU
  - Soft-max …

Sigmoid



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

RELU

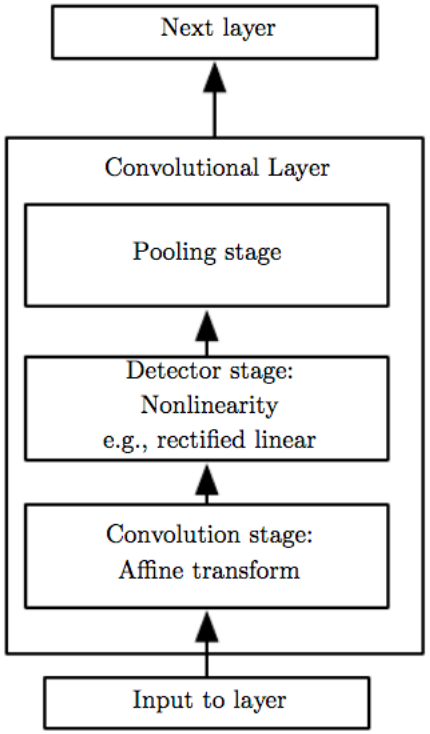


$$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$$

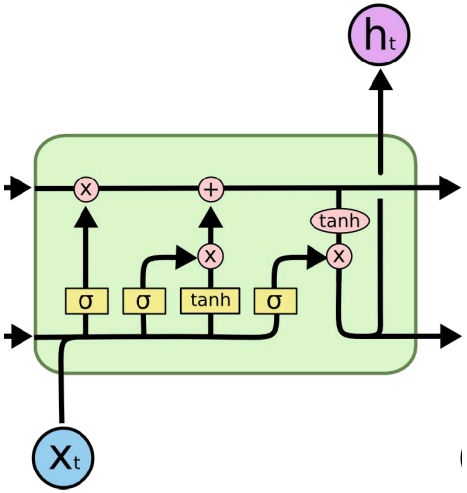
Non-linear Supervised Methods: Neural Networks

- Different kinds of NN used in deep-learning:
  - Fully connected
  - Convolutional (CNN)
  - Recurrent (RNN, LSTM)
  - Generative (VAE, GAN, DBM)

CNN:



RNN:



Goodfellow et. al., 2016

Non-linear Supervised Methods: Neural Networks

- Examples of uses in Bioinformatics and generally
  - Natural language processing (NLP)
  - Epigenomics (binding site prediction etc.)
  - Disease prediction
  - Image-based diagnosis

scikit code (MLP = multilayer perceptron = fully connected NN):

```
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(5, 2), random_state=1)

clf.fit(X, y)
```

Model training and evaluation:
Cross Validation

- Schematic of cross-validation based training:

Training set (75%), with cross validation partitions: Testing set (25%):

  - Use training set partitions to set hyper-parameters: e.g. for each hyper-parameter setting, train on 4 partitions and test on 1, and calculate mean score over 5 divisions
  - Choose hyper-parameters with best score, and retrain on whole training set
  - Assess performance on hold out testing set data

scikit code:

```
from sklearn.model_selection import cross_val_score
sc = cross_val_score(clf, X, y, cv=5)
```

Model training and evaluation:
ROC curves

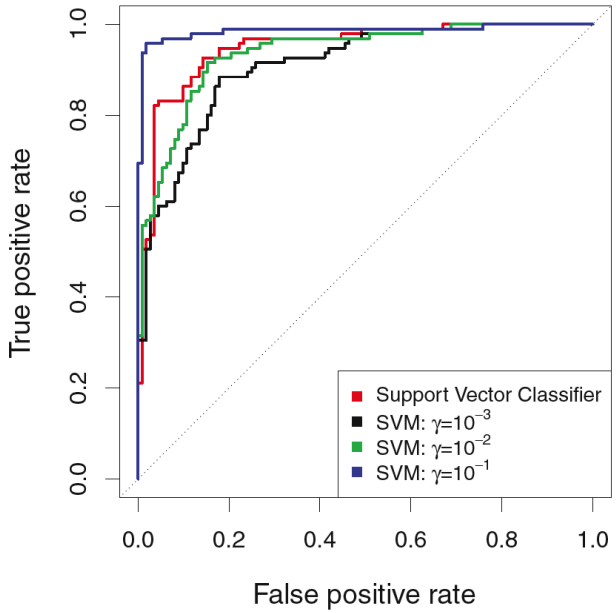
- Post training, any classifier outputting continuous scores can be assessed across a range of **operating points**, which trade off the cost of making type I and type II errors.
- Trade off represented by an **ROC curve**, and general performance assessed using **area under the curve** (AUC)

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$$

scikit code:

```
from sklearn.metrics import roc_curve, auc
y_score_svm = clf_svm.decision_function(X2)
fpr, tpr, _ = roc_curve(y2, y_score_svm)
auc = auc(fpr, tpr)
```



James et. al., 2013

Model training and evaluation:
Model interpretation

- Different models require different kinds of interpretation
  - **Logistic regression** and **linear SVMs** are directly interpretable: the most important features are those whose coefficients have the highest absolute values (+ve and –ve)
  - **Non-linear SVMs** can be interpreted by considering the examples selected as support vectors
  - **Random forests** can be interpreted by looking at the features most used to split the data
  - **Neural Networks** are harder to interpret: a simple approach is to evaluate the gradient of the output with respect to each feature

References

- Machine Learning References:

  - Bishop, Christopher M. *Pattern recognition and machine learning*. Springer, 2006.
  - *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*, Aurélien Géron, O'Reilly Media 2017.
  - James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An Introduction to Statistical Learning with applications in R. Springer.
  - Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
  - Mohri, Mehryar, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

References (contd.)

- Silver et al. 2016, "Mastering the game of Go with deep neural networks and tree search", *Nature* 529, Pgs. 484-489.