DS
IA

2019

Data Science and AI

Module 5

Supervised ML: Classification

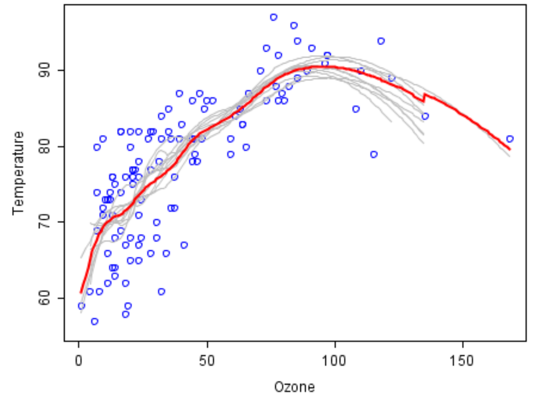
Agenda: Module 5

- Introduction to **Classification**
- **Logistic Regression**
- **Evaluating** Classification Results
- **Neural Networks**
- **Support Vector Machines**
- **Bayesian Inference**
- **Applications**

Classification

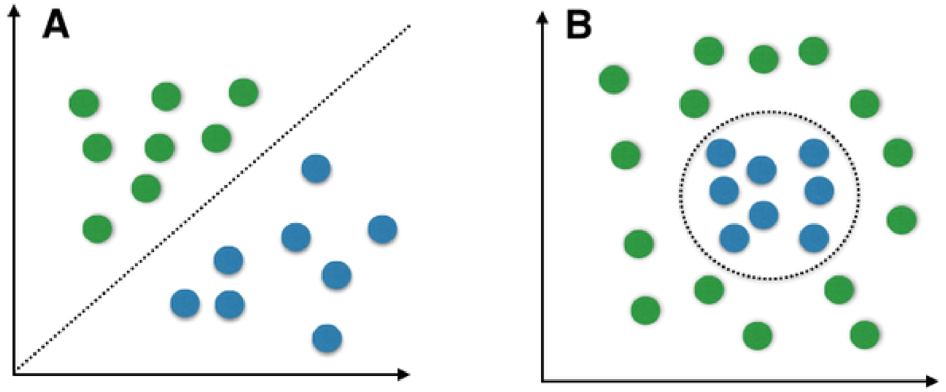
Regression

- train a model by fitting data to a **continuous** response
- *predict **continuous** numbers*



Classification

- train a model by fitting data to a **discrete** response
- predict **class membership**



Logistic Regression

- Introduction to **classification**
- Logistic regression **algorithm**
- **Evaluating** classification results
- Measuring the **quality** of classification models
- **Dummy variables**

Classification

examples

- fraud detection
  - True / False

- customer segmentation:
  - frequent, high-value / regular, medium-value / occasional / transient

- credit risk
  - low / medium / high

- disease status
  - NYHA Class I / II / III / IV

Predicting Class Membership by Supervised Machine Learning

training data

- **features**
    - for now, assume these are continuous
- **response**
    - for now, assume this is binary (True/False)
    - Could be multiple classes in some case

goal

- predict $p(y = 1 \mid X)$

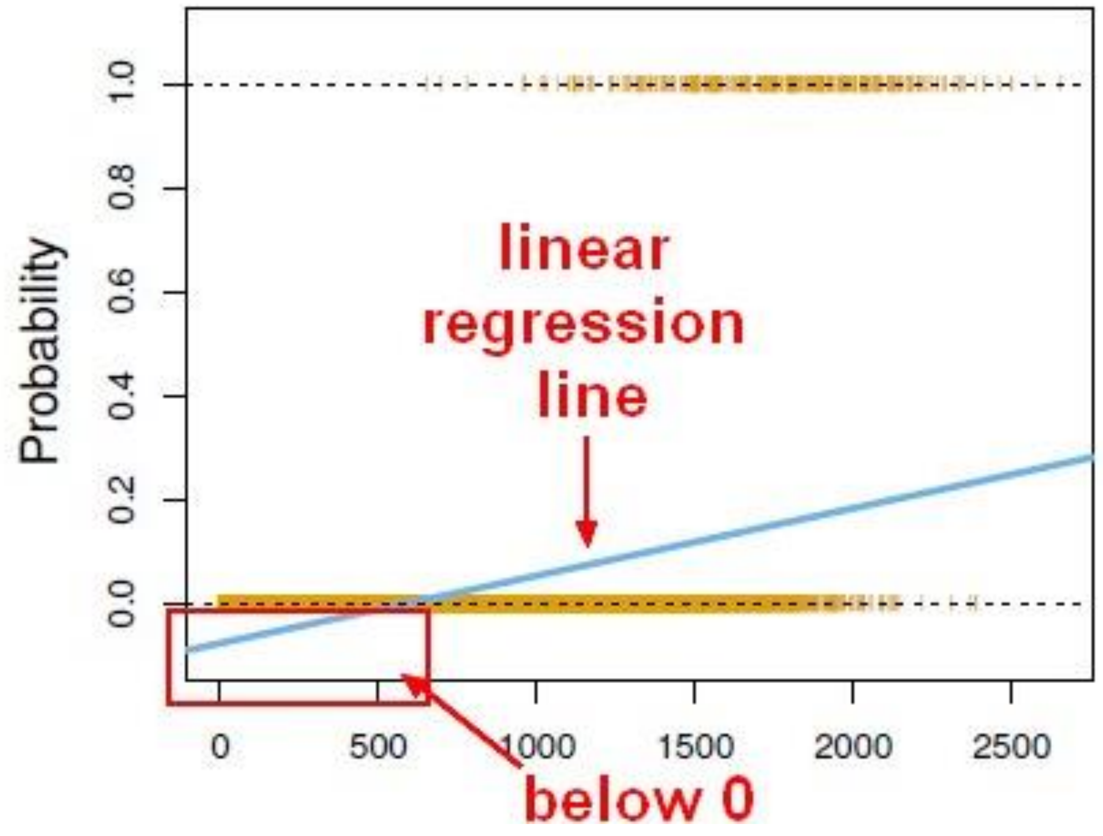
  the probability of ***y*** being True given the predictor(s) ***X***

Binary Class Prediction

- **can we use linear regression?**

$\boldsymbol{y} \in$ {False, True} ➡ $\boldsymbol{y} = \beta \boldsymbol{X} + \varepsilon$

- binary response variable results in large residuals
- predictions can be outside [0, 1]



Binary Class Prediction – cont’d

- need to transform the response so that ***y*** becomes discrete
  - model the ***probability*** of class membership!

- how about this:

$$\boldsymbol{p}\ (\boldsymbol{y} = 1 \mid \boldsymbol{X}) = \beta_0 + \beta_1 \boldsymbol{X}$$

  - still gives $\boldsymbol{y} < 0$, $\boldsymbol{y} > 1$

  - need an approximating function that ensures $\boldsymbol{y} \in [0,1\,]$

Binary Class Prediction – cont’d

- $\frac{p}{1-p}$ = odds ratio

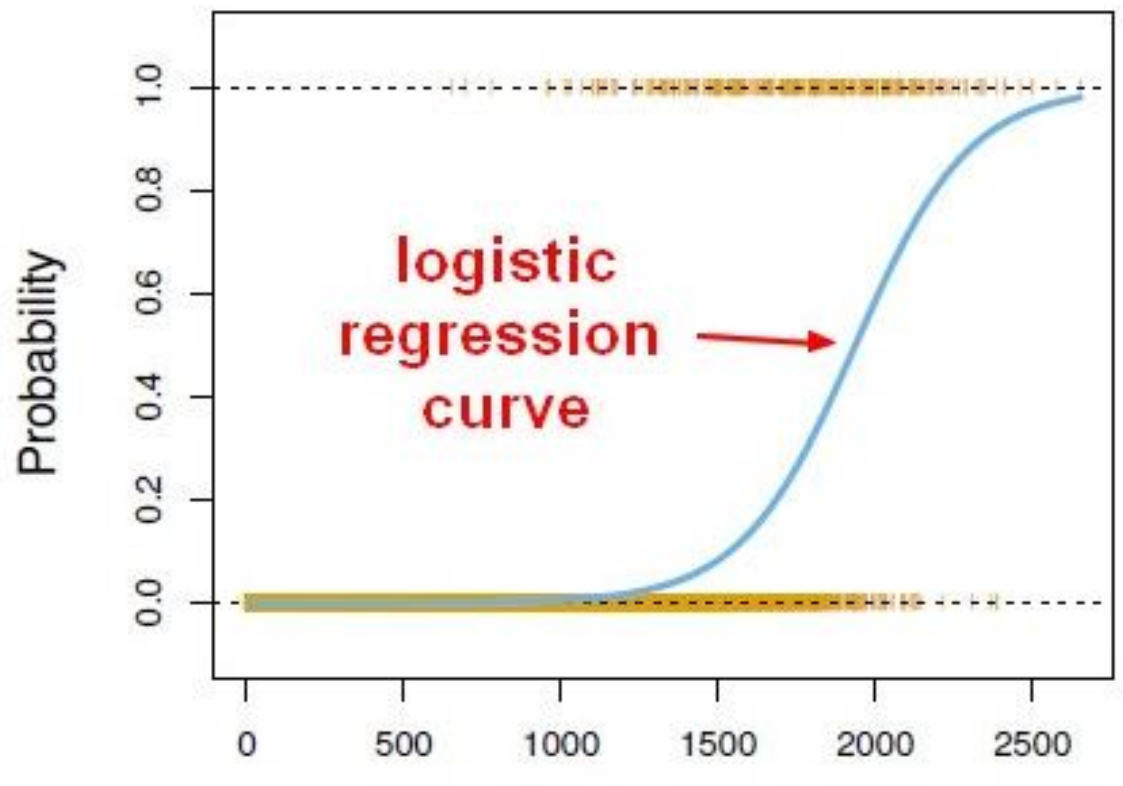
  (p = probability of True)

$\log\left(\frac{p}{1-p}\right)$ = logit (*aka* log odds)

solve:

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 \boldsymbol{X}$$

logit function



Logistic Regression

- β_0, β_1 are known from regression results:

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 \boldsymbol{X}$$

let $\boldsymbol{X}$ be a new data point for prediction,

calculate $\hat{y}$:

$$\hat{y} = \beta_0 + \beta_1 \boldsymbol{X}$$

then calculate p:

$$p = \frac{e^{\hat{y}}}{e^{\hat{y}} + 1} = \frac{1}{1 + e^{-\hat{y}}}$$

Conditions:

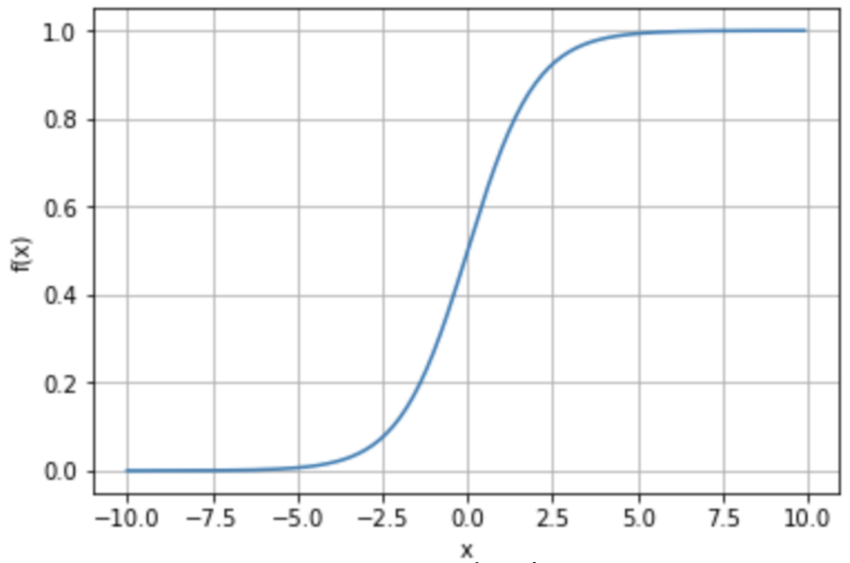
$$p > 0$$
$$p < 1$$

Logistic Regression

Appropriate when predicting a **binary categorical** outcome variable from a set of predictor variables (features) that may be **continuous and/or categorical**

Features should be independent with no missing data.

Logistic Regression typically requires **a relatively large sample size.** A general guideline is that you need at minimum of 10 cases with the least frequent outcome for each independent variable in your model.



Sigmoid or logistic curve

Logistic Regression – SciKit Learn

from **sklearn.linear_model** import LogisticRegression

- Can deal with any number of features
- **Features must be numeric**
    - Categorical features should be converted to dummy features
- Can perform **multi-class classification**

Evaluating Classification Results

There are a number of metrics that can be used to evaluate a classification model. Many of these metrics revolve around values drawn from the Confusion Matrix.

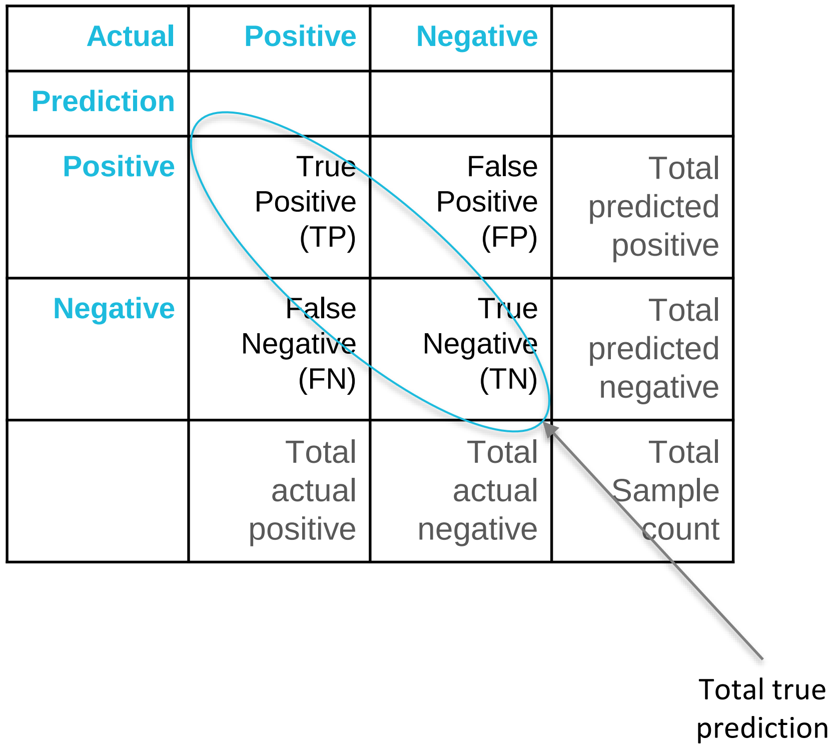
The **Confusion Matrix** is a table that contains counts of the predictions of the model versus the actuals.

Note that if we are interested in predicting the opposite class, the entries would be reversed (the positive becomes negative and vice versa).

Actual	**Positive**	**Negative**
Prediction		
Positive	True Positive (TP)	False Positive (FP)
Negative	False Negative (FN)	True Negative (TN)

Confusion Matrix

It is useful to develop an intuition of the meaning of each row and column and key combinations of the counts in the Confusion Matrix

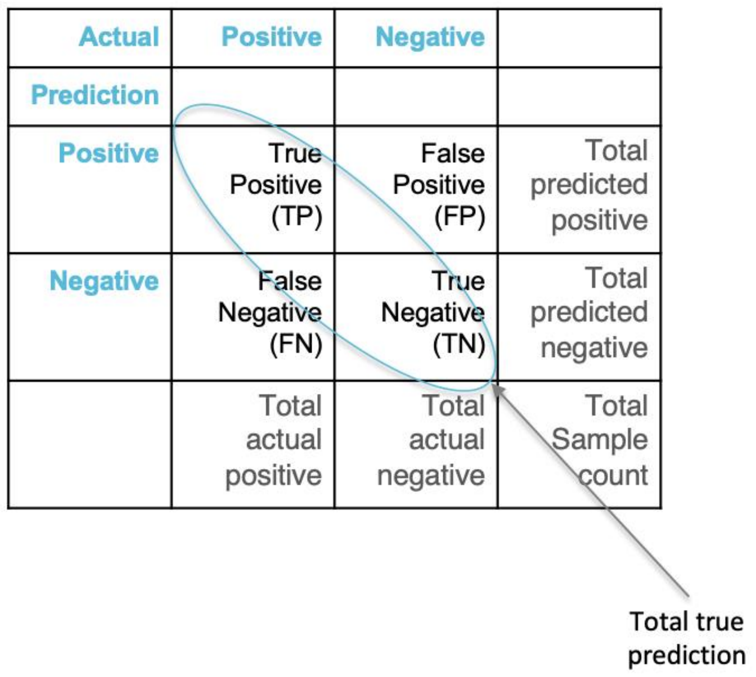


Actual	Positive	Negative	
Prediction			
Positive	True Positive (TP)	False Positive (FP)	Total predicted positive
Negative	False Negative (FN)	True Negative (TN)	Total predicted negative
	Total actual positive	Total actual negative	Total Sample count

Total true prediction

Evaluating Classification Results – cont’d

- **Accuracy**: the fraction of predictions that the model got right. i.e. Number of correct prediction / total prediction
  - Accuracy = Total true prediction / Total sample count
- **Precision**: how many of the prediction were correct. It is a measure of **exactness**.
  - Precision = TP / Total positive prediction (TP + FP)
- **Recall**: how many of the actual positive did the model predict. It is a measure of **completeness**.
  - Recall = TP / Total actual positive (TP + FN)

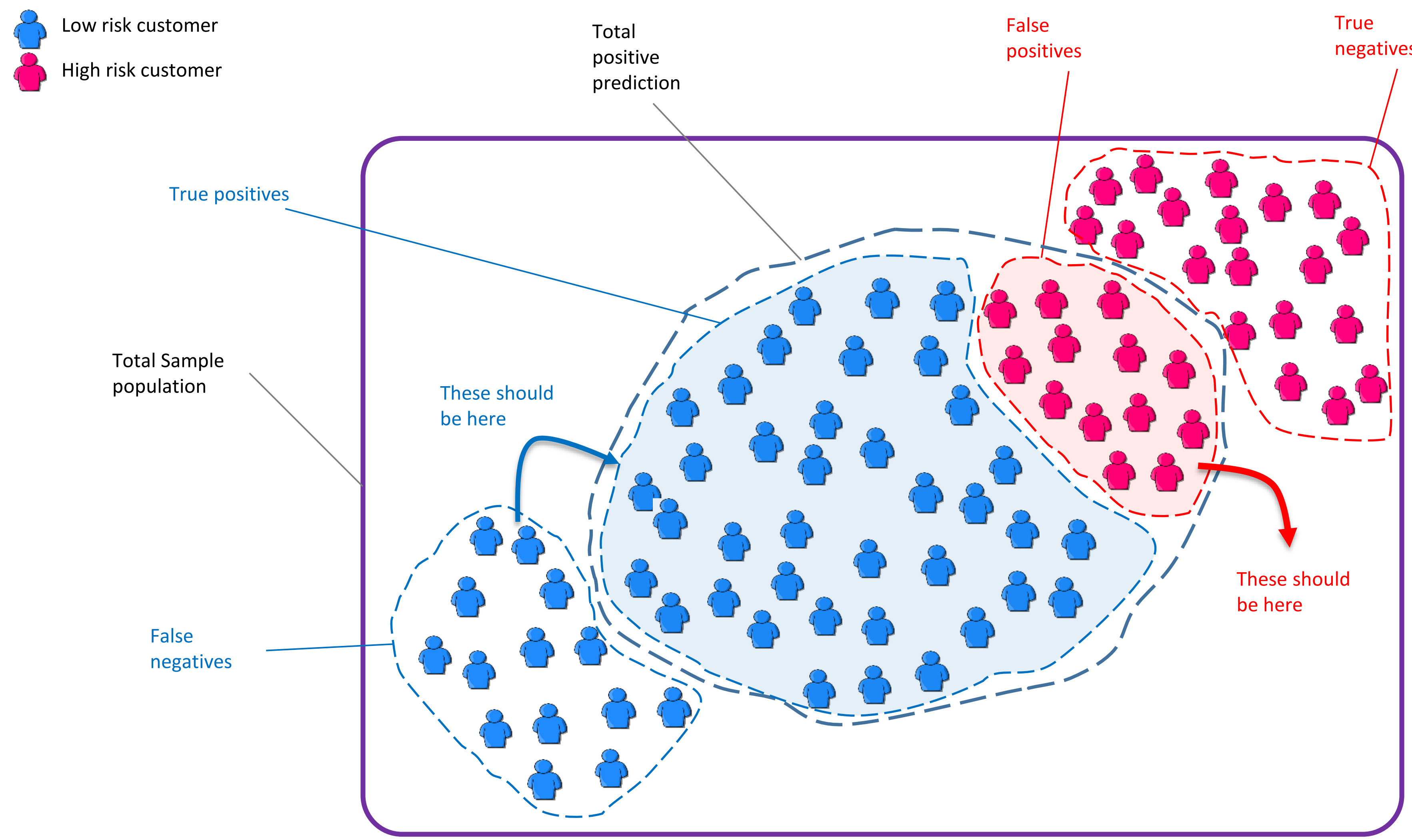


Actual	Positive	Negative	
Prediction			
Positive	True Positive (TP)	False Positive (FP)	Total predicted positive
Negative	False Negative (FN)	True Negative (TN)	Total predicted negative
	Total actual positive	Total actual negative	Total Sample count

Total true prediction

Evaluating Classification Results – cont'd

- **F1-score**: is another measure of a model's accuracy that considers both the precision and the recall.
  - F1-score = 2 (Precision * Recall)/ (Precision + Recall)



Actual	Positive	Negative	
Prediction			
Positive	True Positive (TP)	False Positive (FP)	Total predicted positive
Negative	False Negative (FN)	True Negative (TN)	Total predicted negative
	Total actual positive	Total actual negative	Total Sample count

Total true prediction

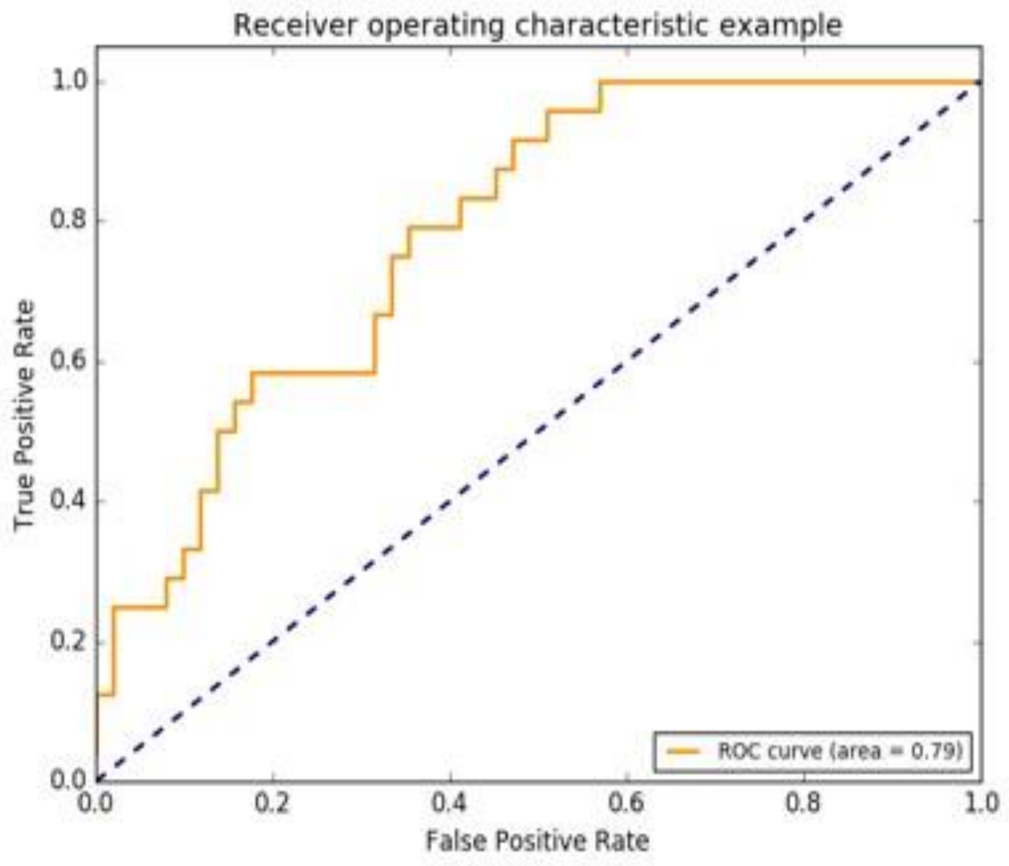
Note: icons are positioned where they are for illustration purpose only.

Accuracy = Total true prediction / Total sample count
Precision = TP / Total positive prediction (TP + FP)
Recall = TP / Total actual positive (TP + FN)

Evaluating Classification Results – cont'd

Receiver operating characteristics (ROC) curve

- Compares True Positive Rate and False Positive Rate
- **True Positive Rate (TPR)** = Recall
- **False Positive Rate (FPR)** is FP/ Total negative count (FP+ TN)
- ROC plots **TPR** vs **FPR** by varying threshold over the entire range of threshold settings. It depicts relative **trade-offs between true positive (benefits) and false positive (costs)**
- **Area Under Curve (AUC)** is equal to the probability that the model will rank a randomly chosen positive instance higher than a randomly chosen negative one.



Evaluating Classification Results – cont’d

- There are many other metrics and many other names for the same metrics
- It is better to stick with a small number of metrics that make sense in your domain before using other metrics
- **Accuracy, precision, recall** and **AUC** are the most common metrics

Dummy Variables

How can we use categorical variables in an algorithm that requires numerical predictors?

- *ordinal categoricals*
  - can be converted to a sequence of integers, if it makes sense to do so

cold	cool	moderate	warm	hot
1	2	3	4	5

✓

  - the above implies hot < warm < moderate < cool < cold
    … which makes sense

Dummy Variables

How can we use categorical variables in an algorithm that requires numerical predictors?

- ***cardinal categoricals***

apples	bananas	peaches	oranges	pears
1	2	3	4	5

> *this implies pears > oranges > peaches > bananas > apples ... does not make sense!*

> must convert to dummy variables instead

Cardinal Dummy Variables

- full definition (number of variables = number of categories):
  - fruit_apples: 1 = apples, 0 = no apples
  - fruit_bananas: 1 = bananas, 0 = no bananas
  - fruit_peaches: 1 = peaches, 0 = no peaches
  - fruit_oranges; 1 = oranges, 0 = no oranges
  - fruit_pears; 1 = pears, 0 = no pears

- compact definition (number of variables is one less than number of categories):
  - fruit_bananas: 1 = bananas, 0 = apples
  - fruit_peaches: 1 = peaches, 0 = no peaches
  - fruit_oranges; 1 = oranges, 0 = no oranges
  - fruit_pears; 1 = pears, 0 = no pears

Python:

pandas.get_dummies

Lab 5.1: Logistic Regression

- Purpose:
  - To predict survival amongst Titanic passengers using the LogisticRegression() method of Scikit-Learn

- Materials:
  - ‘Lab 5.1.ipynb’

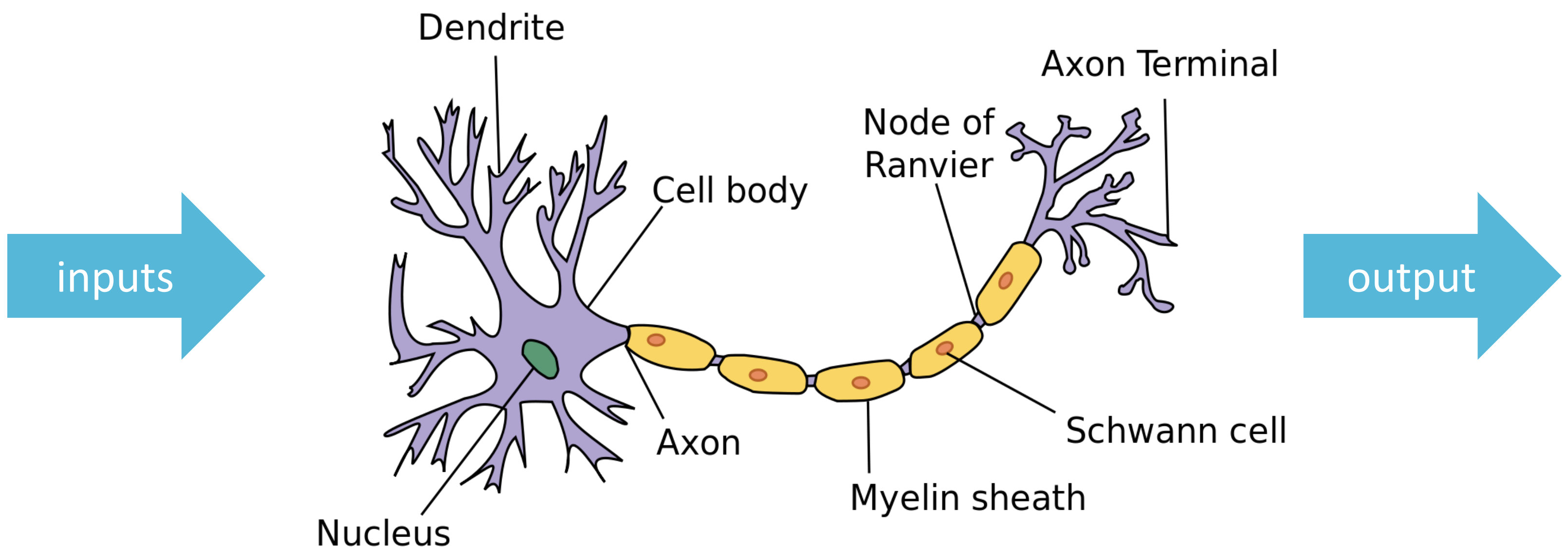
Perceptron (Neural Networks)

- **Biological** and **artificial** neurons
- **Activation** functions
- **Classification** with a perceptron
- **Back propagation**
- **Gradient descent**

How does a nerve cell make a decision?

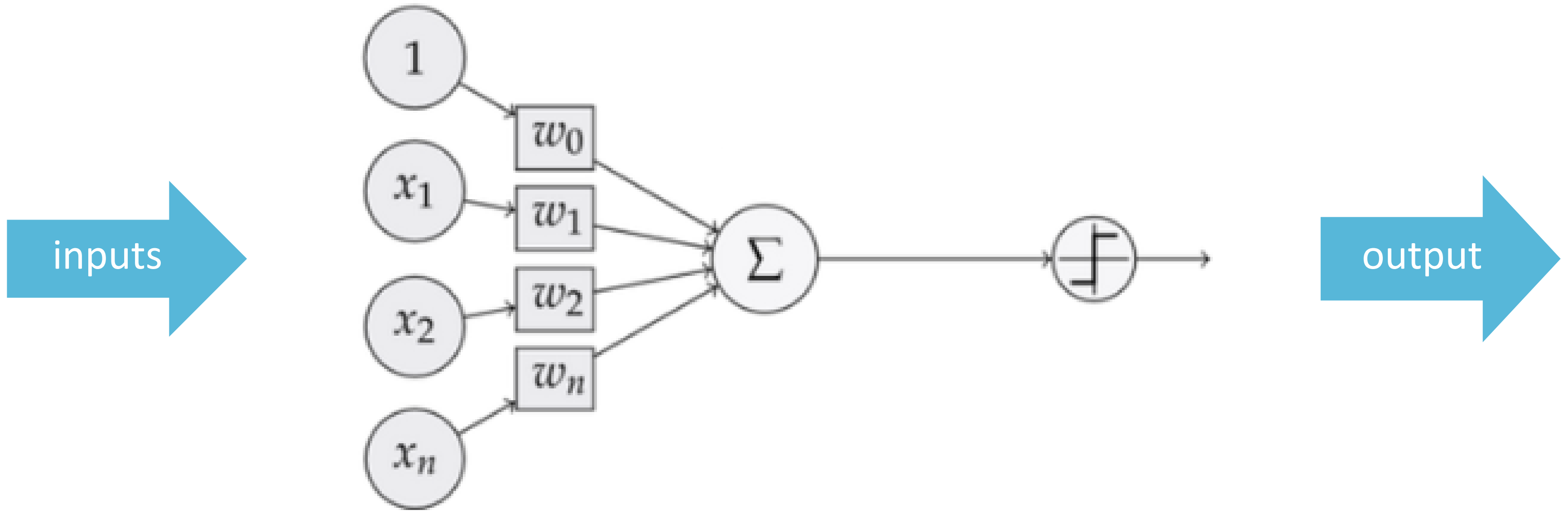
- Neuron receives inputs at antennae-like structures (‘**dendrites**’)
- Each incoming connection is **dynamically strengthened or weakened** by:
  - frequency of use (‘weighting’)
  - neurotransmitters

- **Weighted inputs** are summed in the cell body (transformed into a new signal)
- New signal is **propagated** along the cell’s axon to be detected by other neurons



The Perceptron

- A **basic imitation** of the natural neuron
- Inputs are given **weights**
- **Weighted** signals are summed
- summed signal is transformed by an **activation function** to produce an output



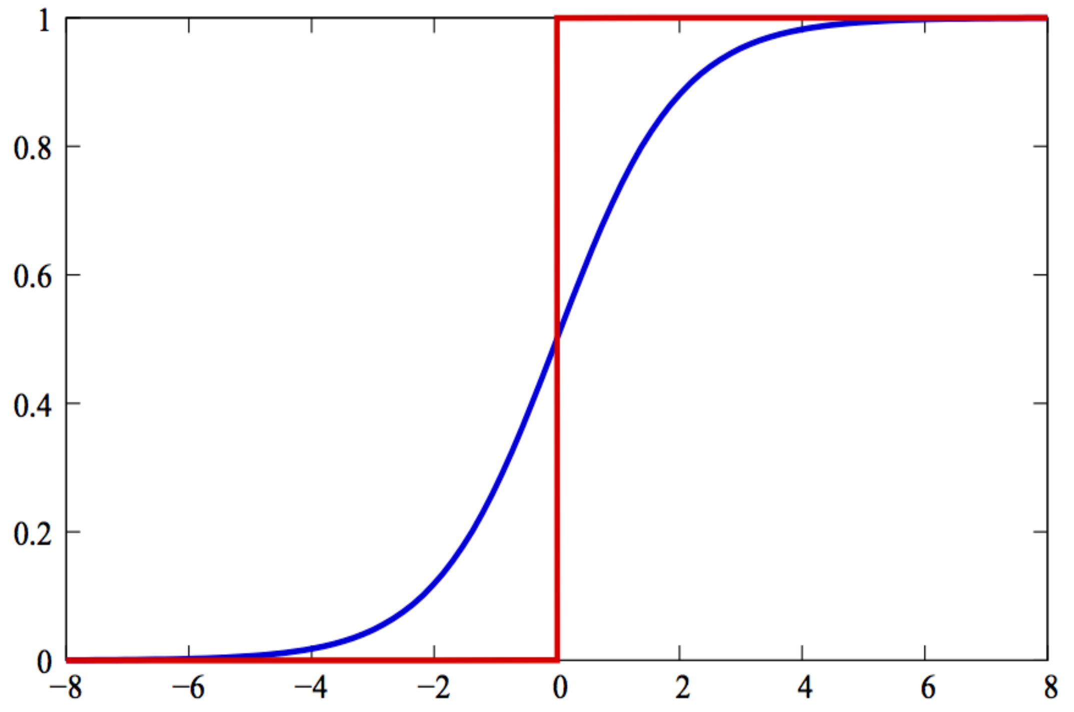
Perceptron Activation Function

- z is the **weighted sum of inputs** (similar to Logistic Regression):

$$z = \sum_{j=0}^{n} w_j \, x_j$$

- A transfer function f(z) converts z to the output of the node
- f(z) is called the **activation function**

example: $$f_{log}(z) = \frac{1}{1 + e^{-z}}$$



Classification Perceptron

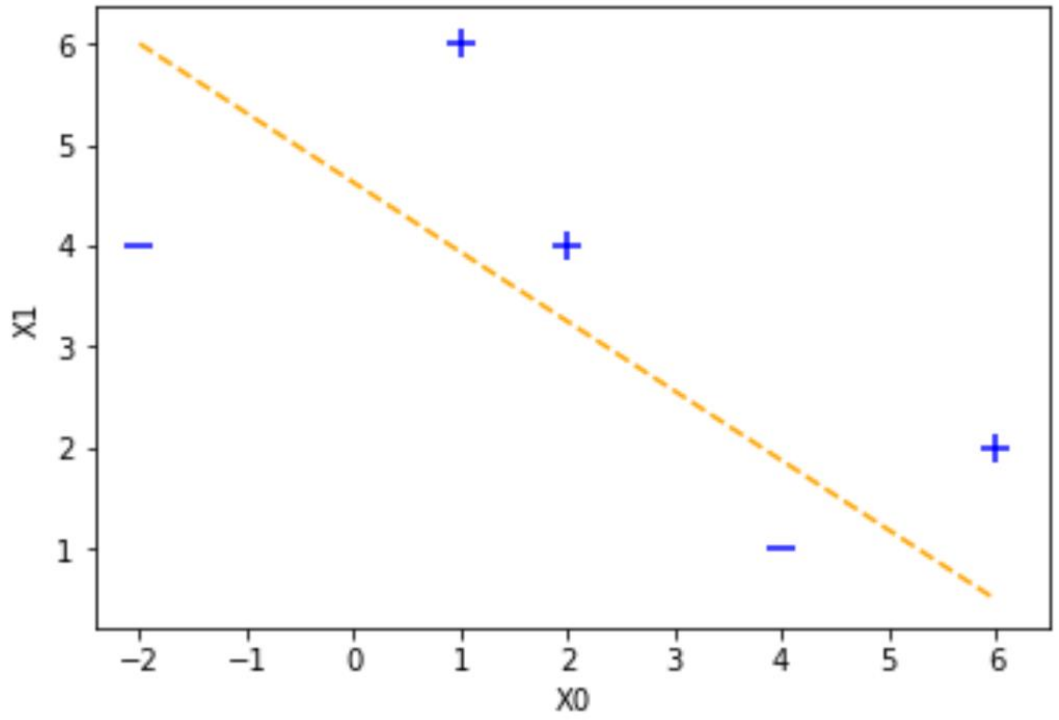
- Given
  - a set of *n*-D **features** $\boldsymbol{X}$
  - a set of **labels** (response var) $\boldsymbol{y}$

- Objective:

  compute the weights $w_1, w_2, \dots w_n$

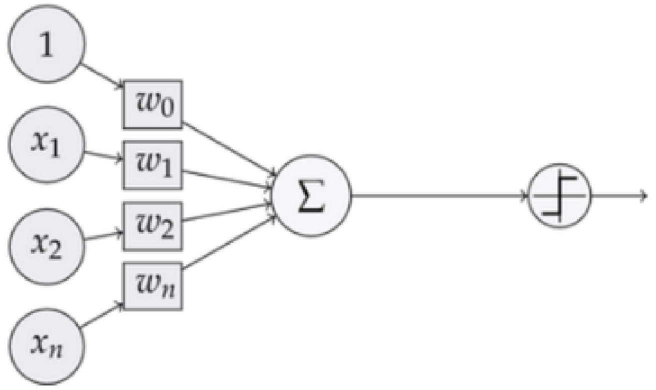
  that describe the ***hyperplane*** that separates points $\boldsymbol{X}$ by class $\boldsymbol{y}$

  example: labels $\boldsymbol{y} \in \{-1, 1\}$



Back Propagation with gradient descent

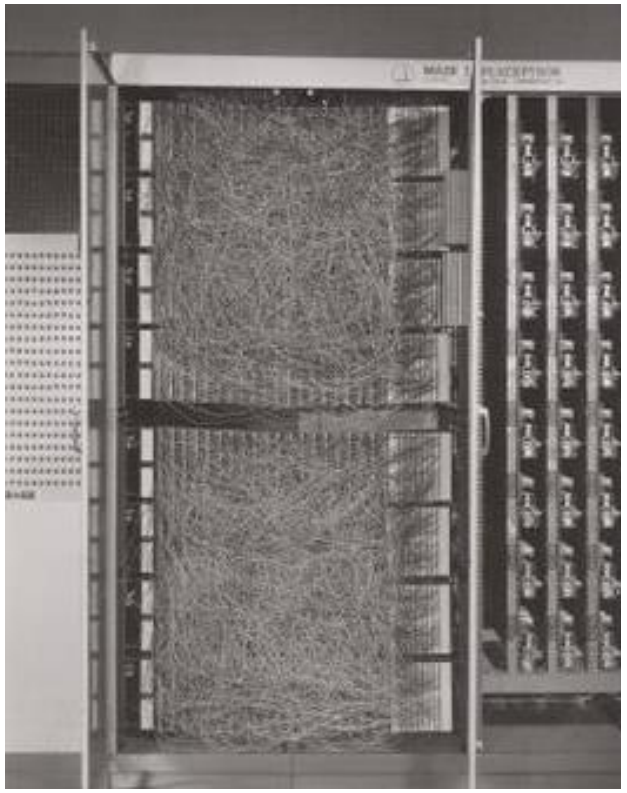
- **Error** at a node is a function of the weights on the inputs flowing into the node
- Use **gradient of error function** to step in direction of decreasing error until minimum is found
- **Update weight** with a Learning Rate times the error and direction of the error (which is the derivative of the cost function)
- w = w + Learning Rate * (expected - predicted) * x



Lab 5.2: Classification with a Perceptron

- Purpose:
  - To evaluate a simple perceptron for predicting classes from numeric features.

- Materials:
  - ‘Lab 5.2.ipynb’



Mark 1 Perceptron

By Source (WP:NFCC#4), Fair use, https://en.wikipedia.org/w/index.php?curid=47541432

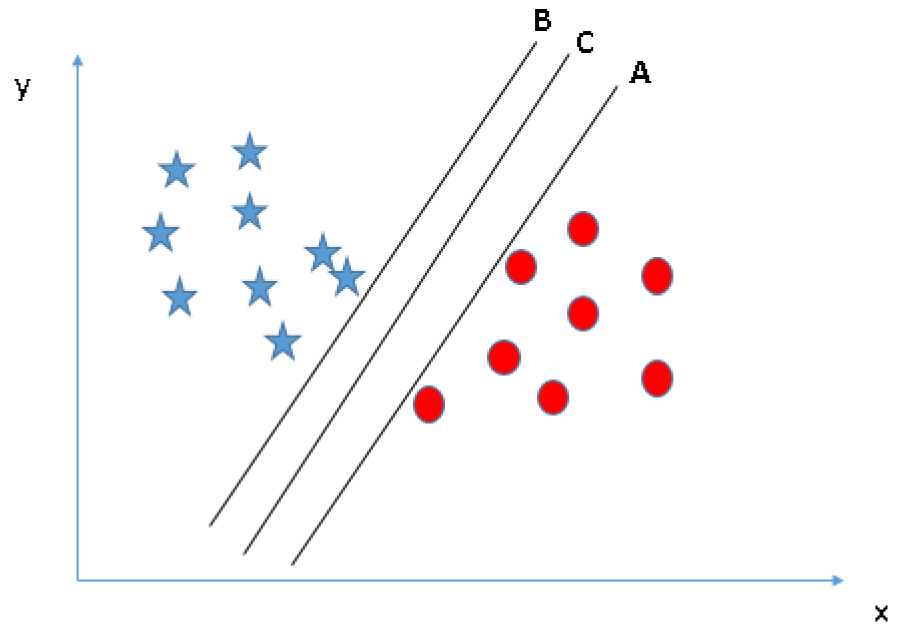
Support Vector Machines

- Concepts
- Linear SVMs
- Nonlinear SVMs
- Limitations
- Applications

Support Vector Machines

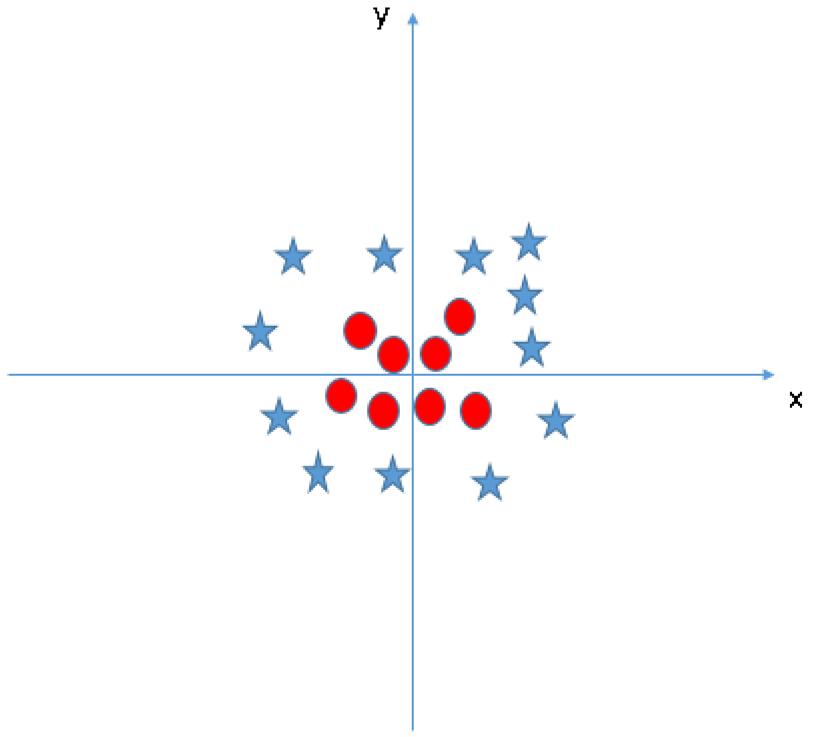
- A **linear** algebraic method for separating ***n*-dimensional** data into classes
  - Data points are separated by a ***hyperplane*** (i.e. a boundary that has dimensionality *n*−1)

example: 2D



- lines A, B, C are **hyperplanes** in a 2D space
- each line correctly separates the two classes
- line C is preferred, because it **maximises** the average squared distance (***margin***) between the boundary and the points

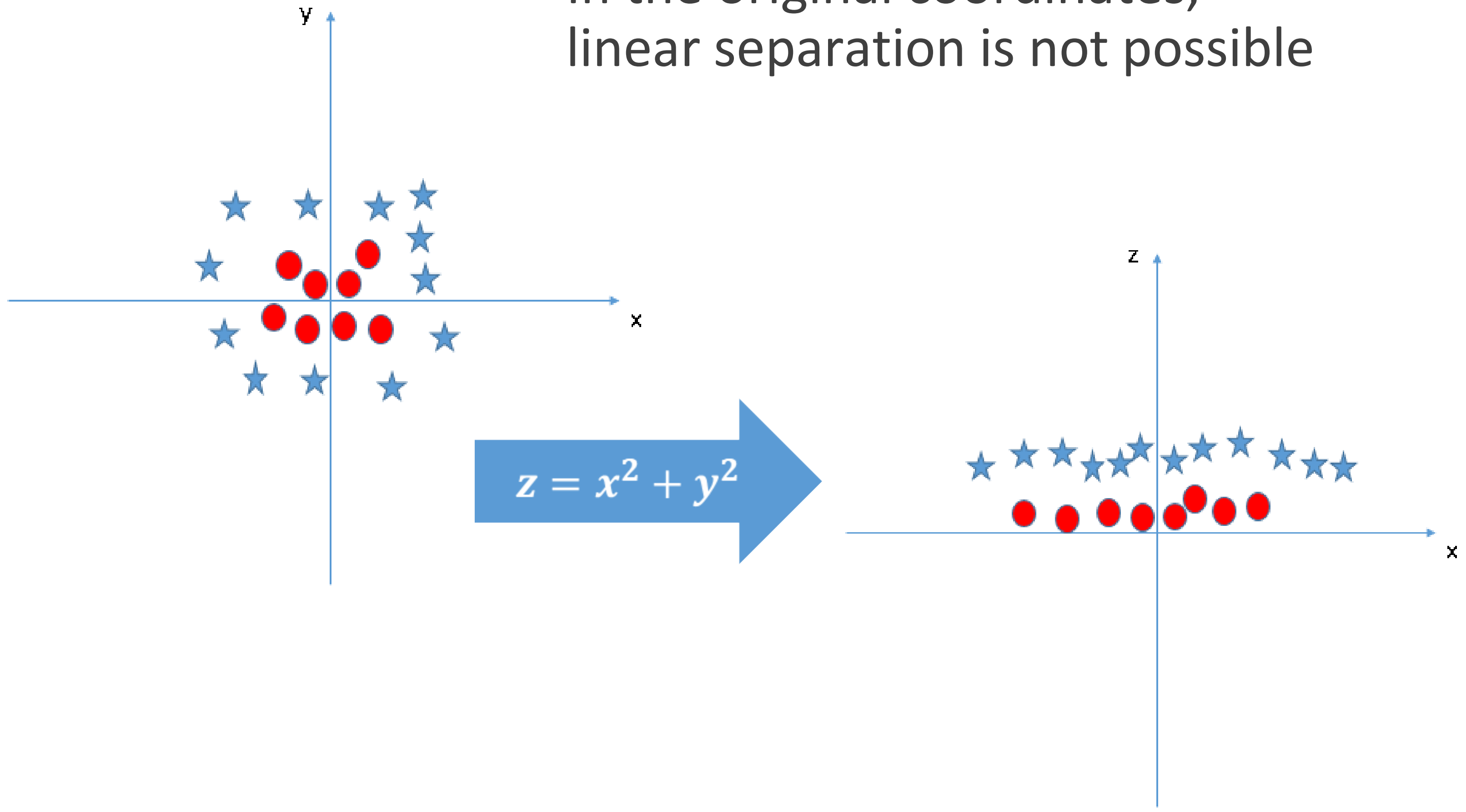
Support Vector Machines – cont’d



- there is no line that can separate these classes
  - will an SVM fail, here?
  - *Hint: we could* ***transform*** *the coordinates or create a new feature*

Support Vector Machines – cont'd

- In the original coordinates, linear separation is not possible



- in these **coordinate**, there *is* a line that can separate the classes

*> the **kernel trick**:*

- an inseparable problem can be transformed into a separable problem in a higher dimension
- the transform function is called the ***kernel***

Applications of SVMs

- Text categorisation
- Image classification
- Handwriting interpretation
- Protein classification
- Customer segmentation

SVMs in Python

- Predictors **X**
- Response **y**
- Test data **Xtest**

- c = **regularization** parameter: controls sensitivity to outliers
- gamma = kernel coefficient (‘rbf’, ‘poly’ and ‘sigmoid’ kernels): controls influence of nearby points

```
from sklearn import svm

# Create SVM classification object:
model = svm.svc(kernel = 'linear', c = 1, gamma = 1)

# Train model:
model.fit(X, y)

# Evaluate quality of fit:
model.score(X, y)

#Predict output for test data:
predicted = model.predict(Xtest)
```

Lab 5.3: Support Vector Machines

- Purpose:
  - To apply the SVM method to linear and nonlinear classification problems.
- Materials:
  - ‘Lab 5.3.ipynb’

Bayesian Inference

- **Frequentist** vs **Bayesian** probability
- **Bayes' theorem**
- Example: Disease detection
- **Bayesian modelling**
- **Naïve Bayes Classification**

Frequentist Probability

- Frequentist statistics
  - a.k.a. “orthodox statistics”
  - Probability = frequency of occurrences in infinite # of trials
  - Arose from sciences with populations
  - p-values, t-tests, etc.
- Bayesian vs. frequentist debates have been long and acrimonious

Frequentist Probability

- Probability:
  - ***a proportion of outcomes***

- In the frequentist approach, we have some **sample results** from which we calculate the **frequency** of the *positive* result
  - *we estimate the future probability of a positive result based entirely on this sample*

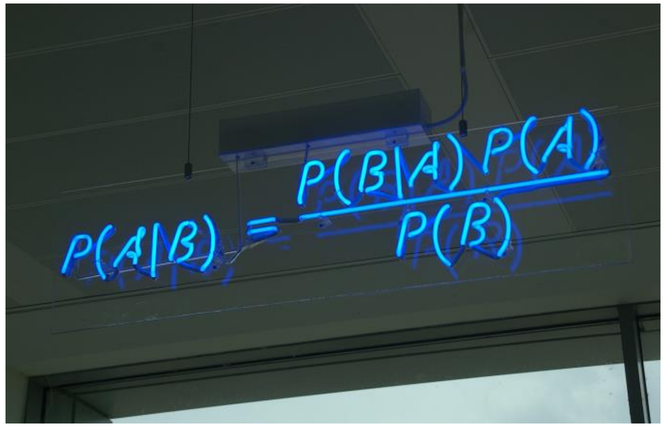
Bayesian Probability

- Probability:
  - ***a degree of belief***

- In the Bayesian approach, we also have pre-existing information (a sample or distribution) about something that has a causal connection to the thing we are sampling
  - *we use this **prior** distribution in addition to the current sample to estimate the future probability of a positive result*

Bayes’ inference theorem

- Bayes’ inference theorem is used to update the probability for a hypothesis as more **evidence** or information becomes available.
- Theorem:
  - P(H|E) = P(E|H). P(H) /P(E)
- Definition:
  - P(A|B): The probability of a event A given B



A blue neon sign showing the simple statement of Bayes' theorem at the offices of HP Autonomy

Example: Disease Detection

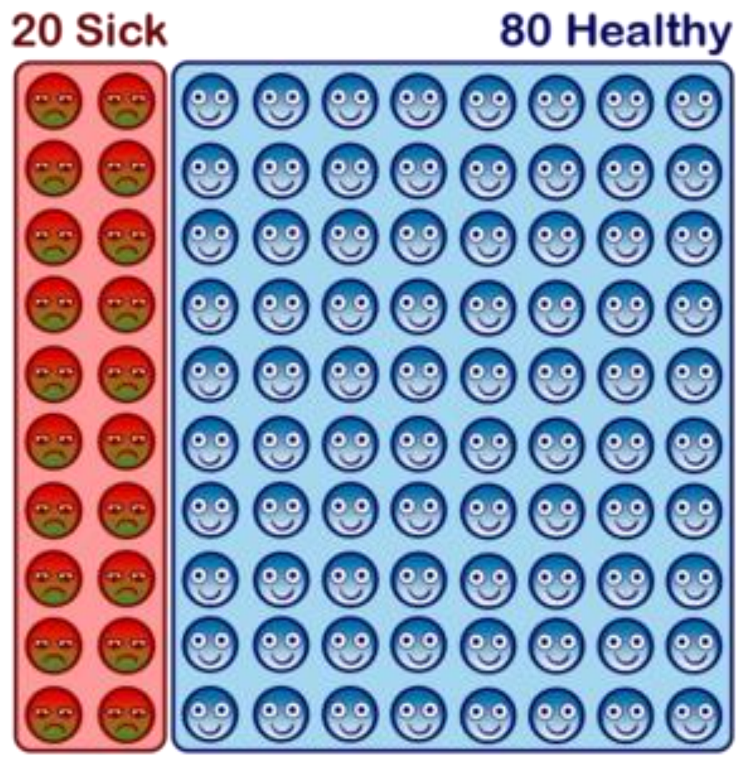
- 90% of subjects with disease are correctly detected
- True Positive (TP) = 0.90

- 30% of disease-free subjects are erroneously detected
- False Positive (FP) = 0.30

> *if a person tests positive, what is the probability that they are sick?*

Example: Disease Detection

- prior knowledge:
  - 20% of population has the disease
    P / (P + N) = 0.20
  - in a random sample of 100 people,
    20 will have the disease (on average)



https://arbital.com/p/bayes_frequency_diagram/

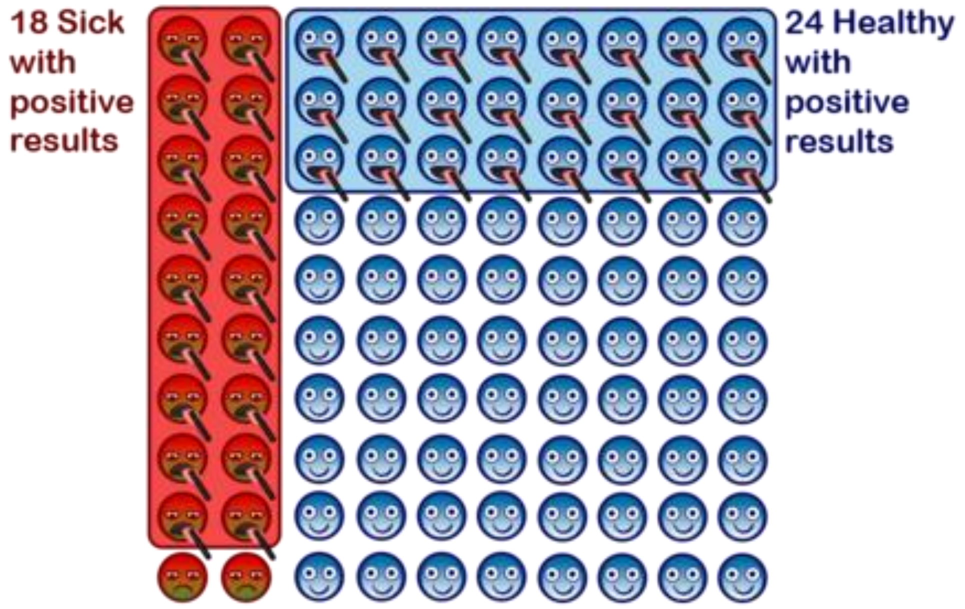
Example: Disease Detection

- Recall of test:
  - 90% of subjects with disease are correctly detected

  P(test|sick) = 0.90 * 20 = 18

  - 30% of disease-free subjects are erroneously detected

  FP = 0.30 * 80 = 24


18 Sick with positive results
24 Healthy with positive results

> *if a person tests positive, what is the probability that they are sick?*

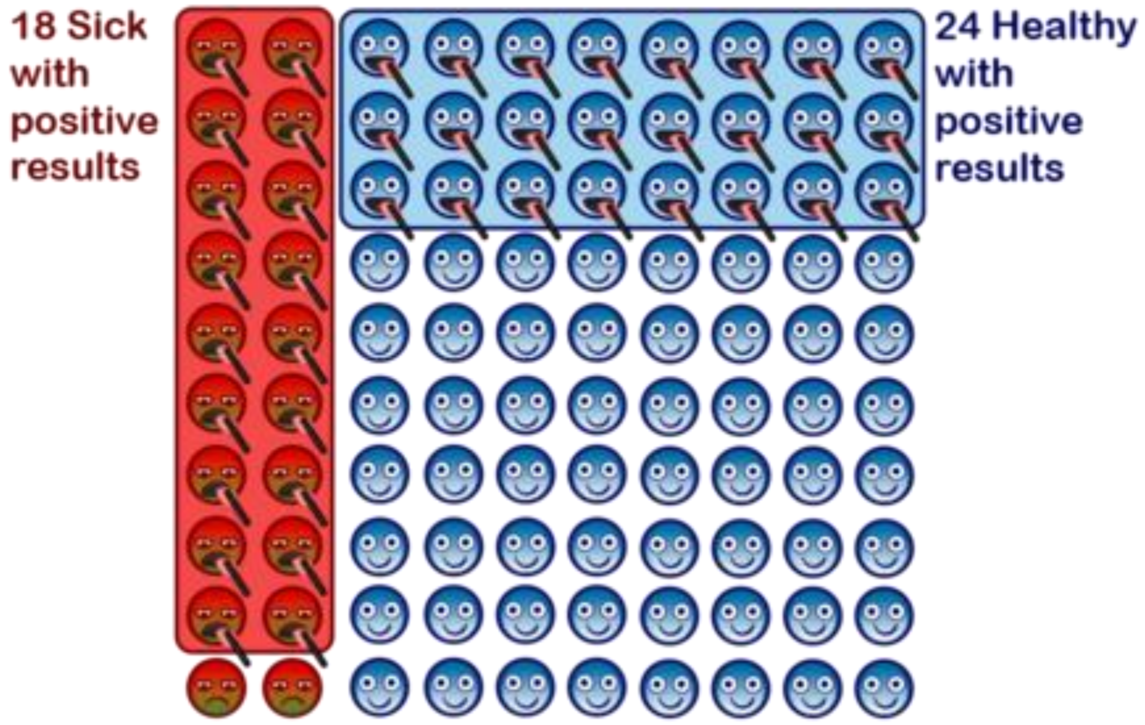
Example: Disease Detection

- probability that subject with a positive test result is actually sick:

```
P(sick+|test+)
= P(test+| sick+)* P(sick+)/ P(test+)
= P(test+| sick+)*          P(sick+)/
    P(test+ |sick-)+P(test⁺|sick+)
= (0.9*(0.2)/((0.9*0.2)+(0.3*0.8)))
≈ 0.43
```

- This matches with the intuitive conclusion from the frequency diagram.

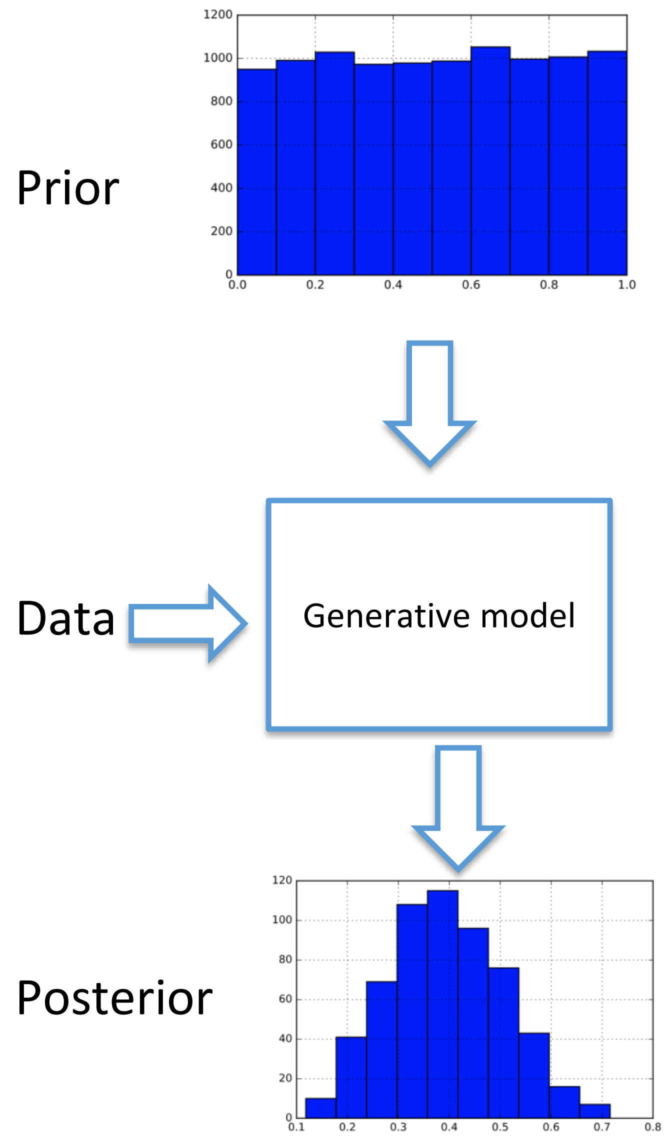
```
P(sick+|test+)
= TP / (TP + FP)
= 18 / (18 + 24)
= 18 / 42
≈ 0.43
```


18 Sick with positive results
24 Healthy with positive results

https://arbital.com/p/bayes_frequency_diagram/

Bayesian data analysis

- Bayesian data analysis uses Bayes' inference theorem to infer target variable from observed data
- A simple way to implement Bayesian data analysis is compute the posterior probability distribution for the target variable using:
  - **Training data**
  - **A generative model**
  - **Prior**
- The generative model produces **simulated data** from the prior probability distribution, matched with the training data and produces the posterior probability distribution.
- We can use the posterior distribution to predict the target variable

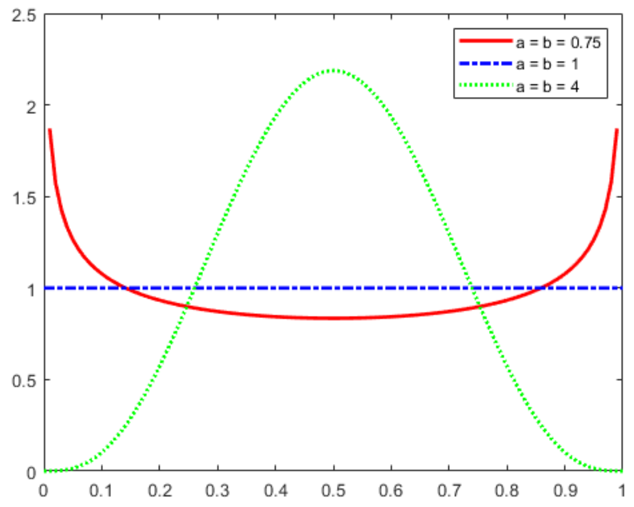


Bayesian Modelling

- **Model-based** approach
  - results not dependent on arbitrary ***p*-value, confidence interval**
- **Naïve Bayes** assumption
  - **Predictors are independent**
    - variables come from distributions that do not interact
- prior distribution is not usually known explicitly
  - type of distribution must be assumed:
    - **from domain knowledge**
    - by choosing a mathematically sound distribution
  - parameters of distribution are fitted to the prior data

How to Choose the Prior Distribution?

- Informative, **empirical**:
  - Data from **related experiments** informs our prior beliefs
  - Prior beliefs will influence final predictions
- Informative, **non-empirical**:
  - Prefer certain values over others (e.g. need to **regularise coefficients**)
  - Prior beliefs will influence final predictions
- Informative, **domain-knowledge**:
  - Using a flexible probability distribution such as beta distribution
  - No supporting data, but certain facts are known to be more true than others
  - Prior beliefs will influence final predictions



Examples of beta distributions

Naïve Bayes Classification

- Probabilistic classification methods

- Assumptions
  - predictors are independent (hence ‘naïve’)
  - predictors are normally distributed

- Applications
  - text classification (spam, topic)
  - medical diagnosis
  - fraud detection
  - insurance risk category

Naïve Bayes Classification – cont’d

- Advantages
  - Algorithms **scale linearly** with number of variables
  - Uses marginal distributions of variables

- Disadvantges
  - **Correlated features** bias the model
  - Absolute probabilities cannot be relied upon
    - only the **probability rankings** should be used
  - Assigns zero probability if a new category appears in test data
    - training data must span all possible levels of categorical features

      or
    - apply a smoothing technique (Sklearn uses Laplace estimation)

Naïve Bayes in sklearn

- **Gaussian Naive Bayes (GaussianNB)**
  - for data that its variables follow normal (Gaussian) distribution.
- **Bernoulli Naive Bayes (BernoulliNB)**
  - for data that is distributed according to Bernoulli distribution.
- **Multinomial Naive Bayes (MultinomialNB)**
  - for data that is distributed according to Multinomial distribution.
- **Complement Naive Bayes (ComplementNB)**
  - for data that has unbalanced Multinomial distribution.

Discussion

- more on Bayesian inference:
  - https://www.analyticsvidhya.com/blog/2016/06/bayesian-statistics-beginners-simple-english/
  - https://towardsdatascience.com/bayesian-statistics-for-data-science-45397ec79c94

- to be covered in a future module:
  - decision trees (nonlinear classification methods)

Hyperparameters

- What are **hyperparameters**
- Key hyperparameters **across models**
- Key hyperparameters for **specific models:**
  - **Logistic Regression**
  - **Support Vector Machine**
  - **Naïve Bayes**
- **Approach** for setting hyperparameters

Hyperparameters

- Hyperparameters are parameters that their values are set **before** the learning process begins. By contrast, the values of other parameters are derived via training.
- Different models require different hyperparameters but the are common ones.
- Hyperparameters can optimise the **performance** or **effectiveness** of its learning process.
- For instance, Ridge Regression adds a regularisation hyperparameter to ordinary linear regression, which has to be set before estimating the parameters through the training algorithm.

Common type hyperparameters

- Regularisation
- Kernel function
- Learning rate
- Batching
- Number of iterations
- Error tolerance level
- Random seed
- Loss function

Logistic Regression hyperparameters

- **penalty**
  - Used to specify the norm used in the penalty portion of the cost function. Options are l1 which uses abstract form or l2 which uses square form.
- **tol**
  - Tolerance for stopping criteria. May be used to reduce time for learning or improve accuracy.
- **C**
  - Inverse of regularisation strength. Smaller values specify stronger regularization. Could be useful when data has large number of features.
- **Solver**
  - Algorithm to use in the optimisation. The default (’liblinear’) is adequate for most cases. Use Stochastic Average Gradient (SAG) for large datasets, when both the number of samples and the number of features are large.

Support Vector Machine hyperparameters

- **kernel**
  - Specifies the kernel type to be used in the algorithm. Use ‘linear’ before experimenting with other options. Should be selected using cross validation techniques.

- **Gamma**
  - Gamma is used to tune the Radial Basis Function (RBF) kernel. It defines how far **the influence of a single training example reaches**, with low values meaning ‘far’ and high values meaning ‘close’. Same as the kernel, it should be selected using cross validation techniques

- **Probability**
  - Whether to enable probability estimates. This must be enabled prior to calling fit, and **will slow down** that method.

- **nu (nuSVC)**
  - An upper bound on the fraction of training errors and a lower bound of the fraction of support vectors. Should be in the interval (0, 1]. For example, if you set it to 0.05 you are guaranteed to find at most 5% of your training examples being **misclassified** (at the cost of a small margin, though) and at least 5% of your training examples being support vectors.

Naïve Bayes hyperparameters

- **fit_prior (Multinominal)**
  - Whether to learn class prior probabilities or not. If false, a uniform prior will be used.

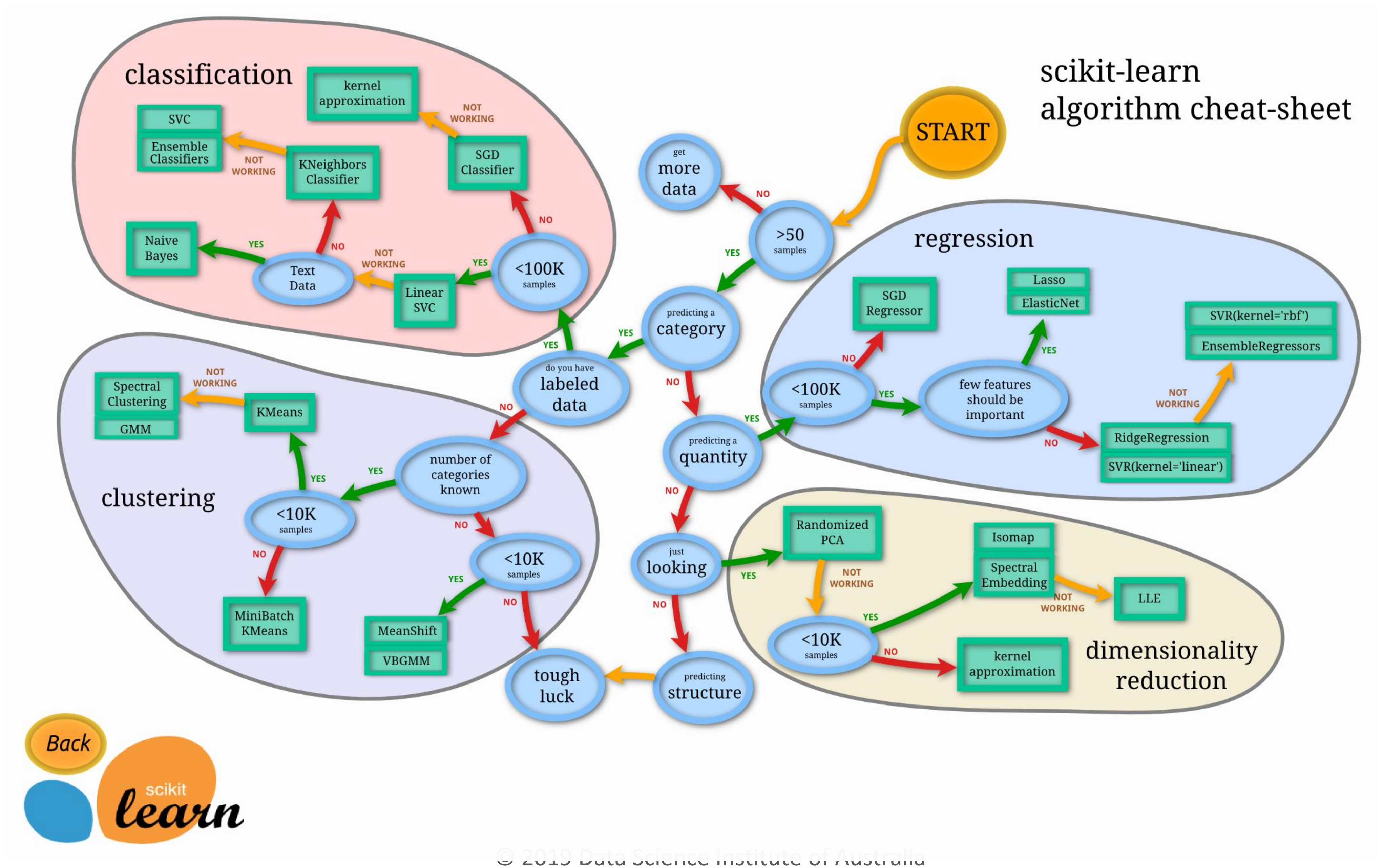
- **var_smoothing**
  - The smoothing priors accounts for features not present in the learning samples and prevents zero probabilities in further computations.

Approaches for setting hyperparameters

- **Use default!**
  - Most of hyperparameters have default values which are adequate for most of cases. So make sure to **work first on the data** (by cleaning it, remove irreverent features, etc) before tuning the hyperparameters.
- If model **evaluation results or performance** is not satisfactory, experiment with the relevant hyperparameters **one parameter at a time**.
- For advanced fine tuning use sklearn functions for selecting optimal values:
  - **Model specific cross-validation**
    - For example, Logistic Regression CV classifier.
  - **GridSearchCV**
    - GridSearchCV exhaustively considers all parameter combinations
  - **RandomizedSearchCV**
    - RandomizedSearchCV samples a given number of candidates from a parameter space with a specified distribution. **a computation budget**, can be set using the n_iter parameter.

Questions?

Appendices



scikit-learn
algorithm cheat-sheet
START
>50 samples
get more data
predicting a category
do you have labeled data
<100K samples
predicting a quantity
just looking
predicting structure
tough luck
classification
Linear SVC
Text Data
Naive Bayes
KNeighbors Classifier
SVC
Ensemble Classifiers
SGD Classifier
kernel approximation
clustering
number of categories known
<10K samples
KMeans
Spectral Clustering
GMM
MiniBatch KMeans
MeanShift
VBGMM
regression
SGD Regressor
few features should be important
Lasso
ElasticNet
RidgeRegression
SVR(kernel='linear')
SVR(kernel='rbf')
EnsembleRegressors
dimensionality reduction
Randomized PCA
Isomap
Spectral Embedding
LLE
kernel approximation
YES
NO
NOT WORKING
Back
scikit learn

End of presentation