ML & Advanced Analytics For Biomedicine

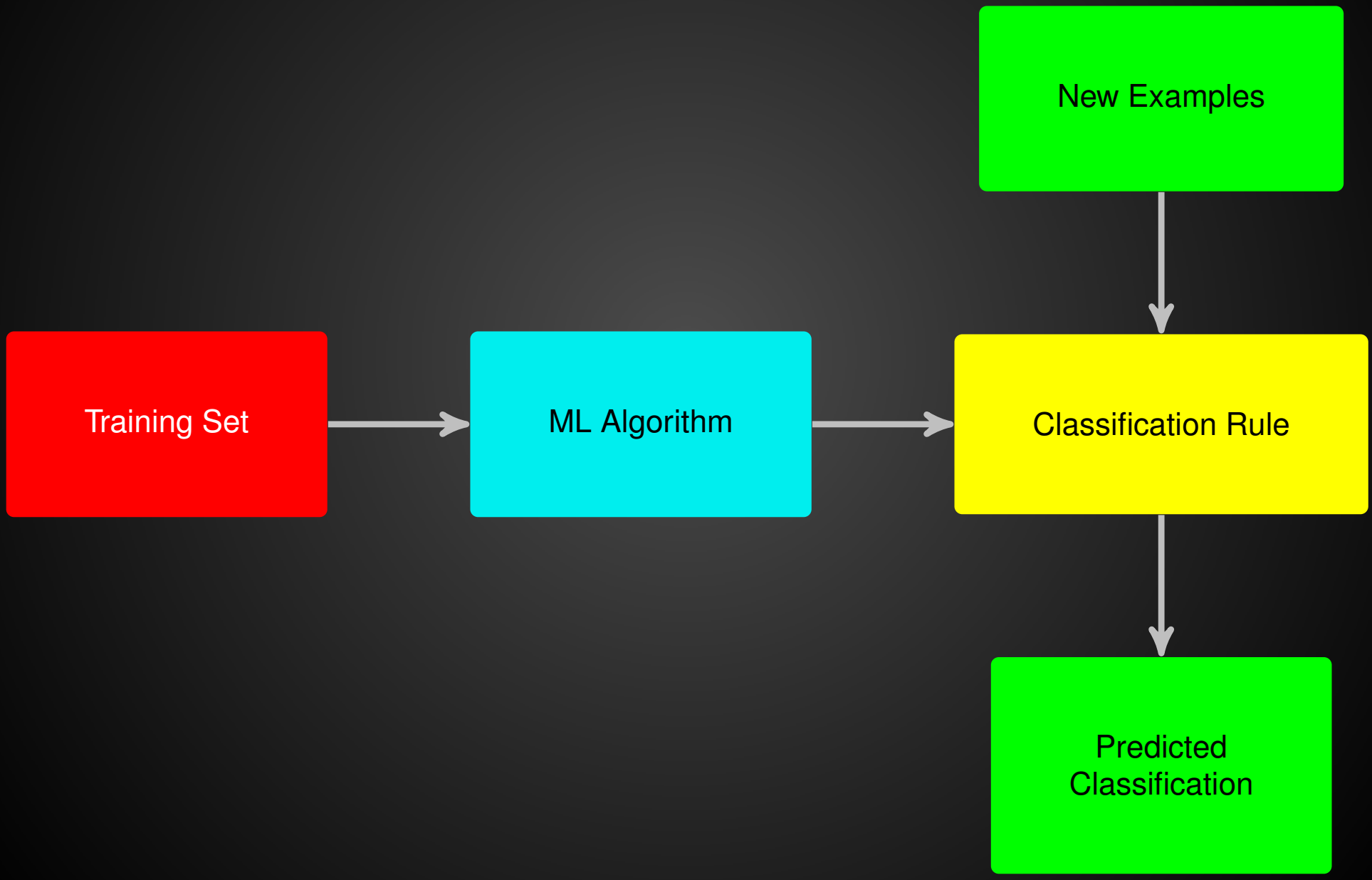
Ishanu Chattopadhyay
Department of Medicine
University of Chicago

CCTS 40500 / CCTS 20500 / BIOS 29208

Winter 2019

Classification

Classification

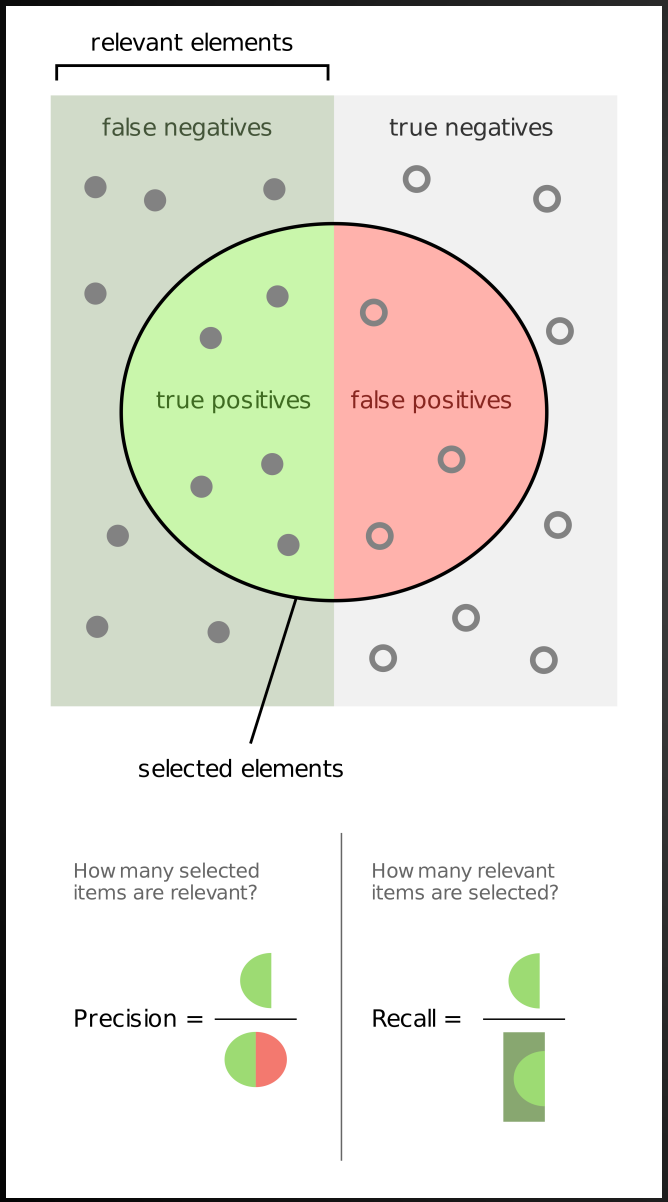


The Confusion Matrix

Binary Classification Problem

	True	False
True Pred.	TP	FP
False Pred.	FN	TN

Performance Metrics



- Sensitivity, recall, hit rate, or true positive rate (TPR)
  $$\mathrm{TPR} = \frac{\mathrm{TP}}{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$
- Specificity or true negative rate (TNR)
  $$\mathrm{TNR} = \frac{\mathrm{TN}}{N} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}}$$
- Precision or positive predictive value (PPV) $\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$
- Negative predictive value (NPV) $\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FN}}$
- Miss rate or false negative rate (FNR)
  $$\mathrm{FNR} = \frac{\mathrm{FN}}{P} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TP}} = 1 - \mathrm{TPR}$$
- Fall-out or false positive rate (FPR)
  $$\mathrm{FPR} = \frac{\mathrm{FP}}{N} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} = 1 - \mathrm{TNR}$$
- False discovery rate (FDR) $\mathrm{FDR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TP}} = 1 - \mathrm{PPV}$
- False omission rate (FOR) $\mathrm{FOR} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TN}} = 1 - \mathrm{NPV}$
- Accuracy (ACC) $\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{P + N}$
- F1 score is the harmonic mean of precision and sensitivity
  $$F_1 = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

Performance Metrics

Insufficiency of Single Metrics

- Task: Predict earthquakes in California from weather data
- Build classification rule that categorizes each day into a positive or negative class: 1) If it is sunny, we have earthquakes 2) if it snows we do not have earthquakes
- Test for 100 days

	true	false
true pred.	3	97
false pred.	0	0

	true	false
true pred.	0	0
false pred.	3	97

Performance Metrics

Insufficiency of Single Metrics

- Task: Predict earthquakes in California from weather data
- Build classification rule that categorizes each day into a positive or negative class: 1) If it is sunny, we have earthquakes 2) if it snows we do not have earthquakes
- Test for 100 days

	true	false
true pred.	3	97
false pred.	0	0

ACC:	3/100	0.03
FPR:	97/97	1.0
TPR:	3/3	1.0
PRC:	3/100	0.03

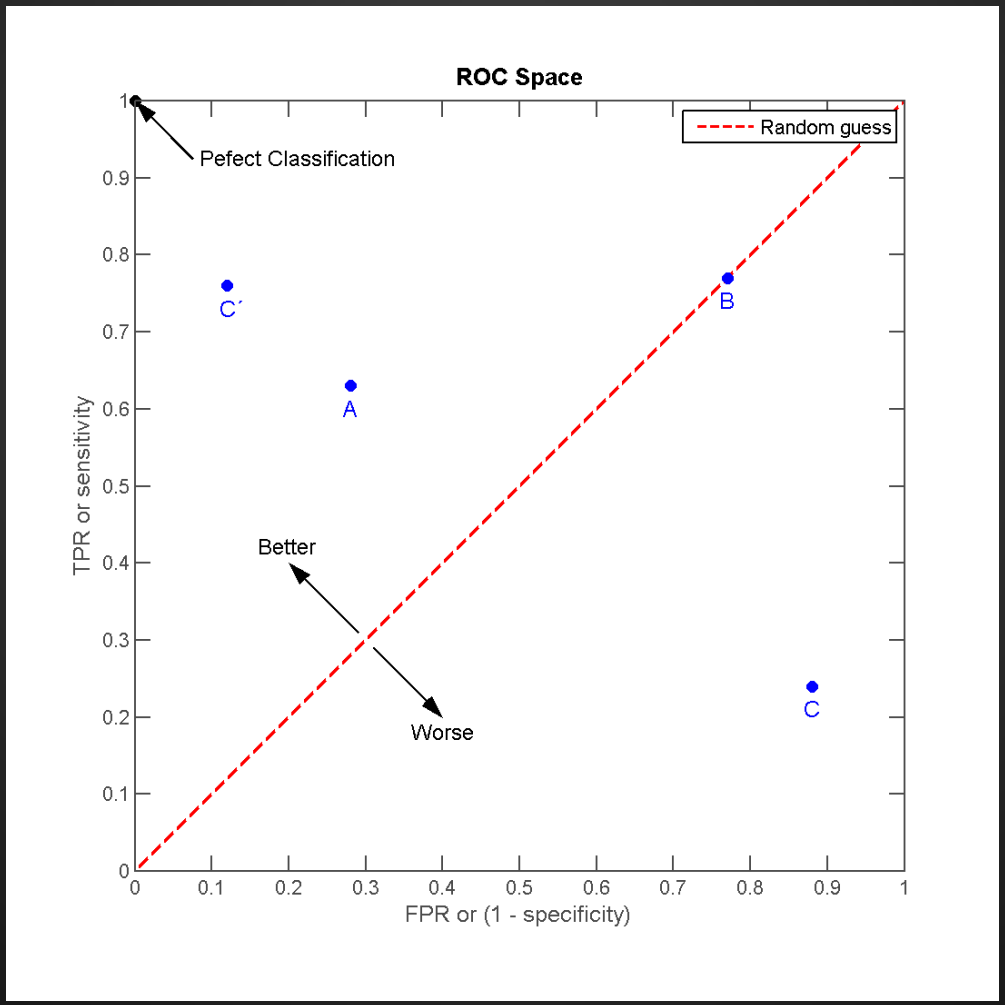
	true	false
true pred.	0	0
false pred.	3	97

ACC:	97/100	0.97
FPR:	0/97	0
TPR:	0/3	0
PRC:	0/0	0

Performance Metrics

ROC Curve

- For a family of binary classifers
  plot FPR vs TPR as some classifier parameter is varied



Goal of Classification Algorithms

- NOT uncover underlying “truth”
- Goal is statistical discrimination of data
- Automatic performance evaluation is crucial

Goal of Classification Algorithms

- NOT uncover underlying “truth”
- Goal is statistical discrimination of data
- Automatic performance evaluation is crucial

- Algorithmic classification rules are more accurate than hand crafted rules
- Humans often cannot express what they “know” or how they “know” it

Input Data Structure

Training, Validation, & Cross-Validation

Training Data:

- samples s_i
- features f_j
- classification class $\mathscr{C}_i$

	f_1	f_2	$\cdots$	f_m	$\mathscr{C}$
s_1					C_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
s_n					C_n

Test Data:

- samples s_i
- features f_j
- Find classification class $\mathscr{C}_i$

	f_1	f_2	$\cdots$	f_m	$\mathscr{C}$
s_1					?
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
s_n					?

Naive Bayes Classifier

The Simplest Classifier

- Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features.
- Given a class variable y and a dependent feature vector x_1 through x_n, Bayes' theorem states the following relationship:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots x_n \mid y)}{P(x_1, \ldots, x_n)}$$

- Using the naive independence assumption that

$$P(x_i|y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|y)$$

for all i, this relationship is simplified to

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \ldots, x_n)}$$

Naive Bayes Classifier

- Since $P(x_1, \ldots, x_n)$ is constant given the input, we can use the following classification rule:

$$P(y \mid x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y) \Longrightarrow \hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

- Use Maximum A Posteriori (MAP) estimation to estimate $P(y)$ and $P(x_i \mid y)$
- $P(y)$ is then the relative frequency of class y in the training set.

Naive Bayes Classifier

Modifications

- The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $P(x_i \mid y)$.
- In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering.
- They require a small amount of training data to estimate the necessary parameters.

Naive Bayes Classifier

Modifications

Experiment with python notebook

Naive Bayes Classifier

Gaussian Naive Bayes

- `GaussianNB` implements the Gaussian Naive Bayes algorithm for classification.
- The likelihood of the features is assumed to be Gaussian:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Naive Bayes Classifier

Multinomial Naive Bayes

- `MultinomialNB` implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification (where the data are typically represented as word vector counts.
- The distribution is parametrized by vectors $\theta_y = (\theta_{y1}, \ldots, \theta_{yn})$ for each class y, where n is the number of features (in text classification, the size of the vocabulary) and θ_{yi} is the probability $P(x_i \mid y)$ of feature i appearing in a sample belonging to class y.
- The parameters θ_y is estimated by a smoothed version of maximum likelihood, i.e. relative frequency counting:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

  where $N_{yi} = \sum_{x \in T} x_i$ is the number of times feature i appears in a sample of class y in the training set T, and $N_y = \sum_{i=1}^{|T|} N_{yi}$ is the total count of all features for class y.
- The smoothing priors $\alpha \geq 0$ accounts for features not present in the learning samples and prevents zero probabilities in further computations.
- Setting $\alpha = 1$ is called Laplace smoothing, while $\alpha < 1$ is called Lidstone smoothing.

Bernoulli Naive Bayes

- `BernoulliNB` implements the naive Bayes training and classification algorithms for data that is distributed according to multivariate Bernoulli distributions; i.e., there may be multiple features but each one is assumed to be a binary-valued (Bernoulli, boolean) variable.
- Requires samples to be represented as binary-valued feature vectors; if handed any other kind of data
- The decision rule for Bernoulli naive Bayes is based on

$$P(x_i \mid y) = P(i \mid y)x_i + (1 - P(i \mid y))(1 - x_i)$$

- differs from multinomialNB's rule in that it explicitly penalizes the non-occurrence of a feature i that is an indicator for class y, where the multinomial variant would simply ignore a non-occurring feature.

Naive Bayes Classifier

Advantages & Disadvantages

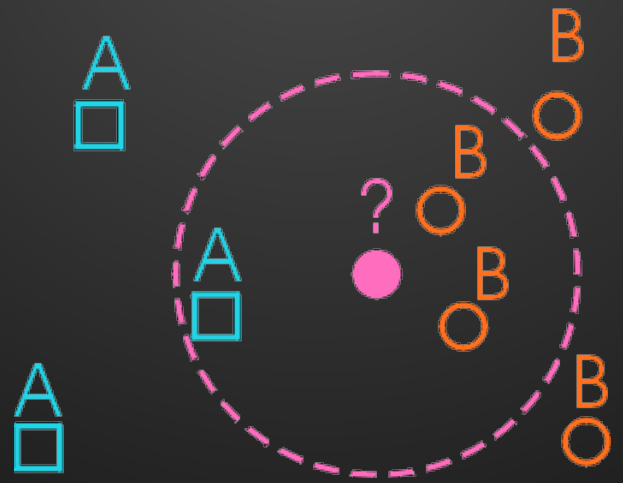
- Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods.
- The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution.
- This in turn helps to alleviate problems stemming from the curse of dimensionality.
- On the flip side, although naive Bayes is known as a decent classifier, it is known to be a bad estimator.

Naive Bayes Classifier

Experiment with python notebook

Nearest Neighbor Classifier

- A new sample is classified by calculating the distance to the nearest training case.
- More generally, we may consider majority voting by the k closest neighbors from the training data (KNN Classifier)
- We can attempt to directly subsume naive Bayes by choosing a subset of the training data such that 1-NN rule (using the subset) approximates the Bayes classifier.



Nearest Neighbor Classifier

Distance Metric

Generalized Notion of Distance

$d : X \times X \to \mathbb{R}$ **is a distance metric if:**

$$\forall x, y \in X, d(x, y) = d(y, x) \quad \text{(SYMMETRIC)}$$

$$d(x, x) \geq 0 \quad \text{(NON-NEGATIVE)}$$

$$d(x, y) + d(y, z) \geqq d(x, z) \quad \text{(IRANGULAR INEQUALITY)}$$

Any function that satifies the above relations is a "distance" in a rigorous sense.

"Neighbors" require the notion of a distance.

Nearest Neighbor Classifier

- Nearest neighbors is the foundation of many other learning methods, notably manifold learning and spectral clustering.
- Supervised neighbors-based learning comes in two flavors: classification for data with discrete labels, and regression for data with continuous labels.

Nearest Neighbor Classifier

- Nearest neighbors is the foundation of many other learning methods, notably manifold learning and spectral clustering.
- Supervised neighbors-based learning comes in two flavors: classification for data with discrete labels, and regression for data with continuous labels.
- The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these. The number of samples can be a user-defined constant (k-nearest neighbor learning), or vary based on the local density of points (radius-based neighbor learning).

Nearest Neighbor Classifier

- Nearest neighbors is the foundation of many other learning methods, notably manifold learning and spectral clustering.
- Supervised neighbors-based learning comes in two flavors: classification for data with discrete labels, and regression for data with continuous labels.
- The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these. The number of samples can be a user-defined constant (k-nearest neighbor learning), or vary based on the local density of points (radius-based neighbor learning).
- The distance can, in general, be any metric measure: standard Euclidean distance is a common choice.
- Neighbors-based methods are known as non-generalizing machine learning methods, since they simply remember all of its training data (possibly transformed into a fast indexing structure such as a Ball Tree or KD Tree.).

Nearest Neighbor Classifier

- Nearest neighbors is the foundation of many other learning methods, notably manifold learning and spectral clustering.
- Supervised neighbors-based learning comes in two flavors: classification for data with discrete labels, and regression for data with continuous labels.
- The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these. The number of samples can be a user-defined constant (k-nearest neighbor learning), or vary based on the local density of points (radius-based neighbor learning).
- The distance can, in general, be any metric measure: standard Euclidean distance is a common choice.
- Neighbors-based methods are known as non-generalizing machine learning methods, since they simply remember all of its training data (possibly transformed into a fast indexing structure such as a Ball Tree or KD Tree.).

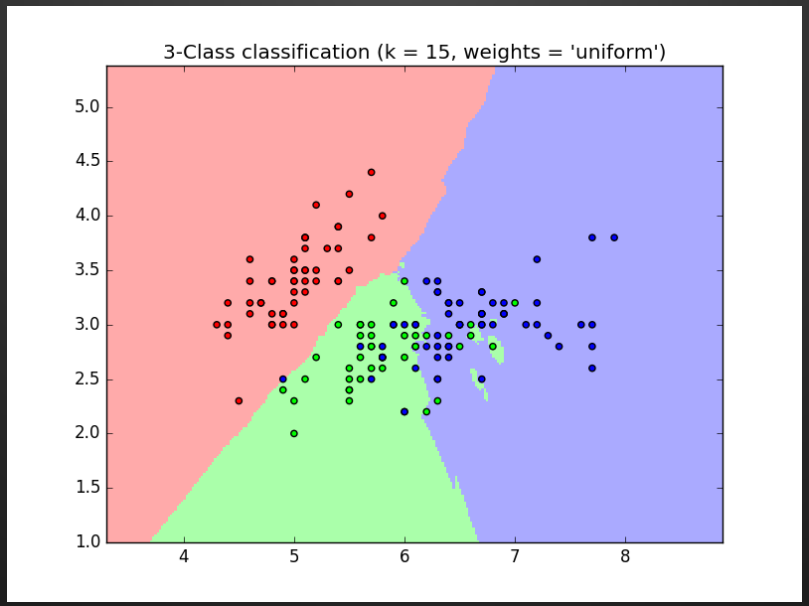
- Despite its simplicity, nearest neighbors has been successful in a large number of classification and regression problems, including handwritten digits or satellite image scenes.
- Being a non-parametric method, it is often successful in classification situations where the decision boundary is very irregular.

Nearest Neighbor Classifier

`sklearn.neighbors` Notes

- `KNeighborsClassifier` implements learning based on the k nearest neighbors of each query point, where k is an integer value specified by the user.
- `RadiusNeighborsClassifier` implements learning based on the number of neighbors within a fixed radius r of each training point, where r is a floating-point value specified by the user.

The k-neighbors classification in KNeighborsClassifier is the more commonly used of the two techniques.



Nearest Neighbor Classifier

`sklearn.neighbors` Notes

- The optimal choice of the value k is highly data-dependent: in general a larger k suppresses the effects of noise, but makes the classification boundaries less distinct.
- In cases where the data is not uniformly sampled, radius-based neighbors classification in RadiusNeighborsClassifier can be a better choice.
- For high-dimensional parameter spaces, this method becomes less effective due to the “curse of dimensionality”.
- The basic nearest neighbors classification uses uniform weights: that is, the value assigned to a query point is computed from a simple majority vote of the nearest neighbors.
- Under some circumstances, it is better to weight the neighbors such that nearer neighbors contribute more to the fit.

  This can be accomplished through the weights keyword. The default value, weights = ’uniform’, assigns uniform weights to each neighbor. weights = ’distance’ assigns weights proportional to the inverse of the distance from the query point. Alternatively, a user-defined function of the distance can be supplied which is used to compute the weights.

Nearest Neighbor Classifier

Data Structures: Brute, KD-tree & Ball-tree

Brute Force:

- Compute distances between all pairs of points in the dataset: for N samples in D dimensions, this approach scales as $O[DN^2]$. Efficient brute-force neighbors searches can be very competitive for small data samples. However, as the number of samples N grows, the brute-force approach quickly becomes infeasible.

  In the classes within sklearn.neighbors, brute-force neighbors searches are specified using the keyword algorithm = 'brute'.

K-D Tree:

- Attempt to reduce the required number of distance calculations by efficiently encoding aggregate distance information for the sample. The basic idea is that if point A is very distant from point B, and point B is very close to point C, then we know that points A and C are very distant, without having to explicitly calculate their distance. In this way, the computational cost of a nearest neighbors search can be reduced to $O[DN\,\mathbf{log}(N)]$ or better.

  In scikit-learn, KD tree neighbors searches are specified using the keyword algorithm = 'kd_tree'.

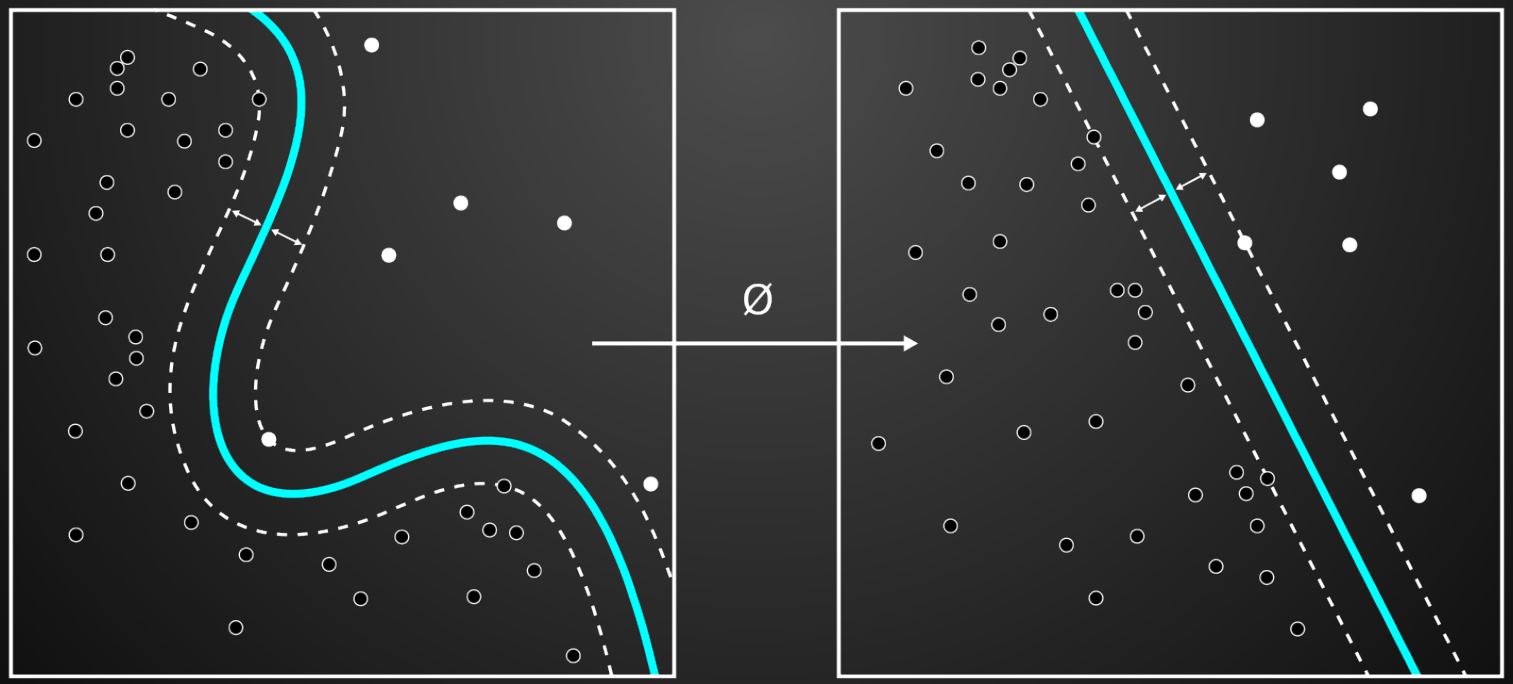
Nearest Neighbor Classifier

Choice of Nearest Neighbor Algorithms

Experiment with python notebook

Support Vector Machines

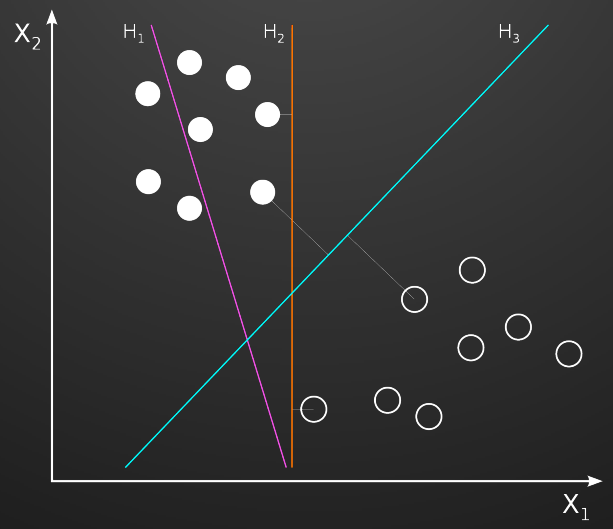
- Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.
- An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible.



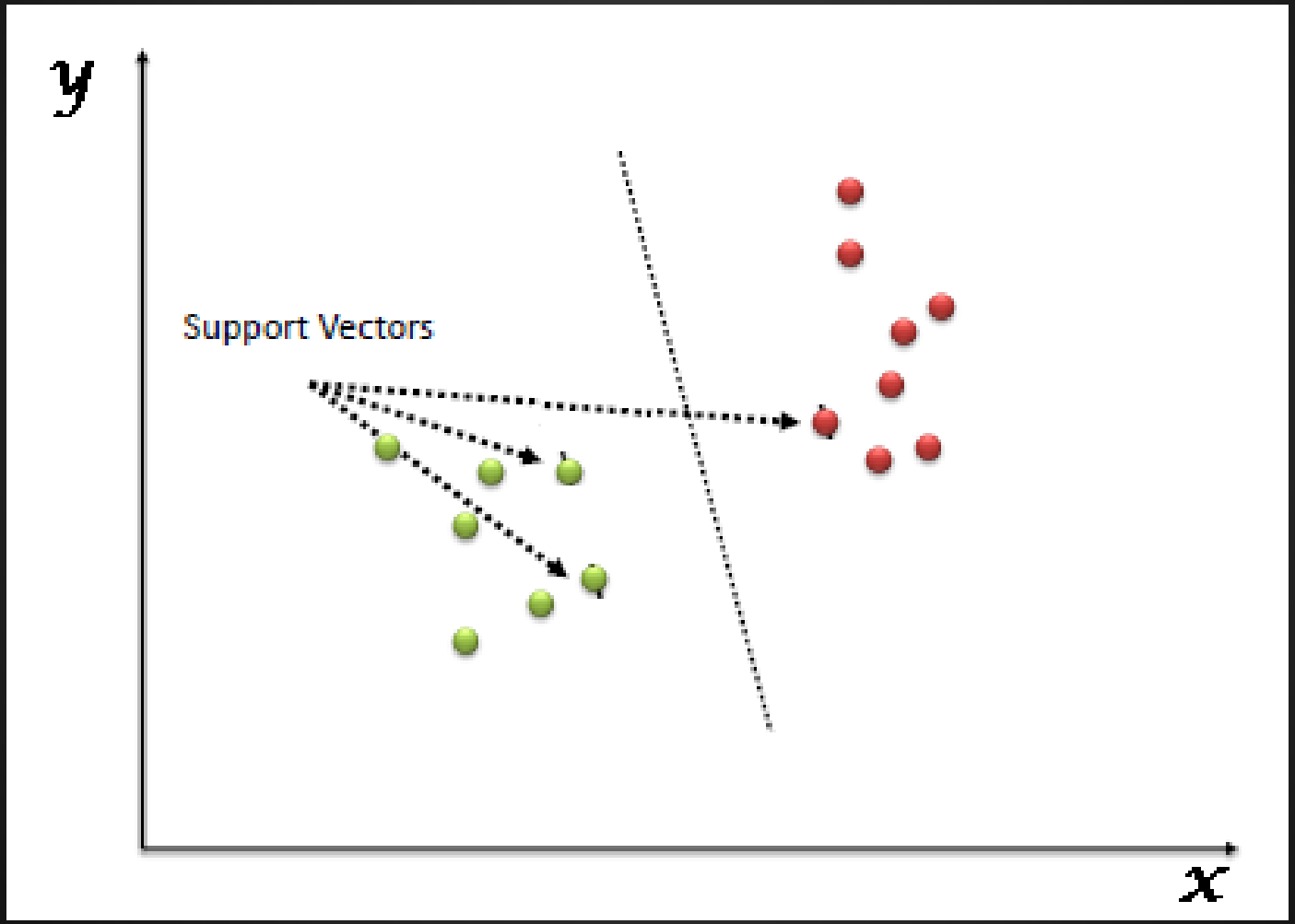
Support Vector Machines

- Suppose some given data points each belong to one of two classes, and the goal is to decide which class a new data point will be in. In the case of support vector machines, a data point is viewed as a p-dimensional vector (a list of p numbers), and we want to know whether we can separate such points with a $(p-1)$-dimensional hyperplane. This is called a linear classifier.

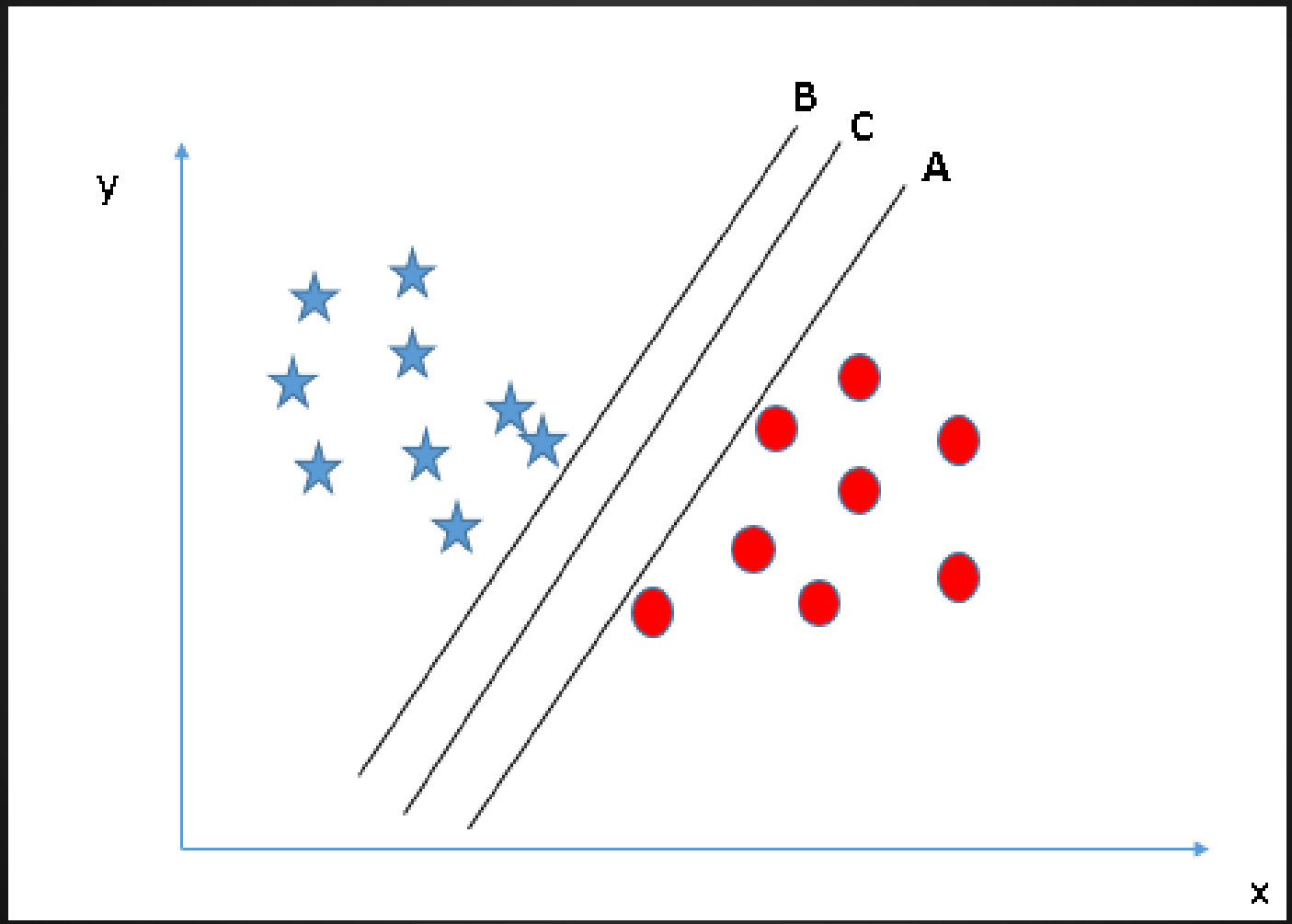
- There are many hyperplanes that might classify the data. One reasonable choice as the best hyperplane is the one that represents the largest separation, or margin, between the two classes. So we choose the hyperplane so that the distance from it to the nearest data point on each side is maximized. If such a hyperplane exists, it is known as the maximum-margin hyperplane and the linear classifier it defines is known as a maximum margin classifier.



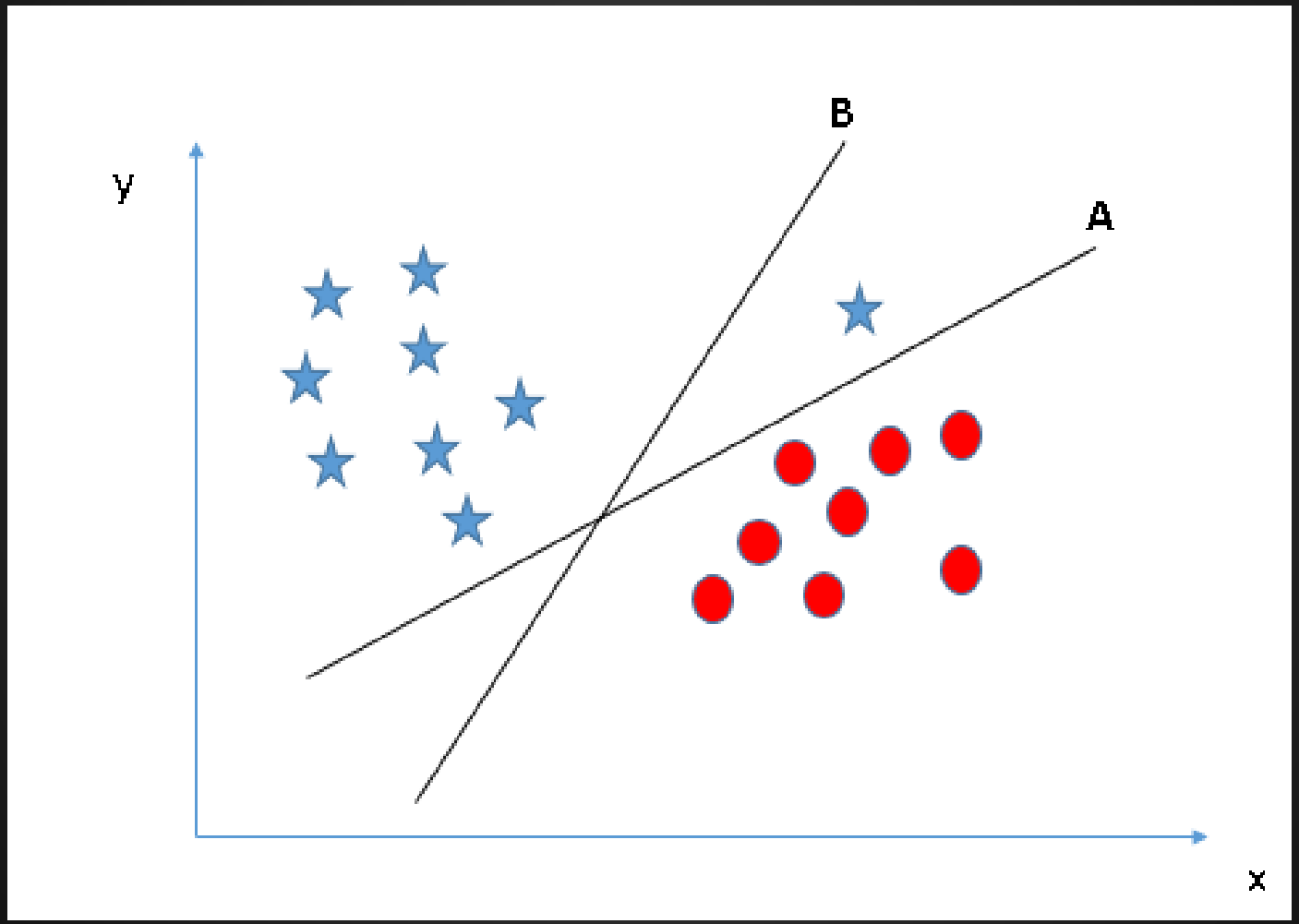
Support Vector Machines



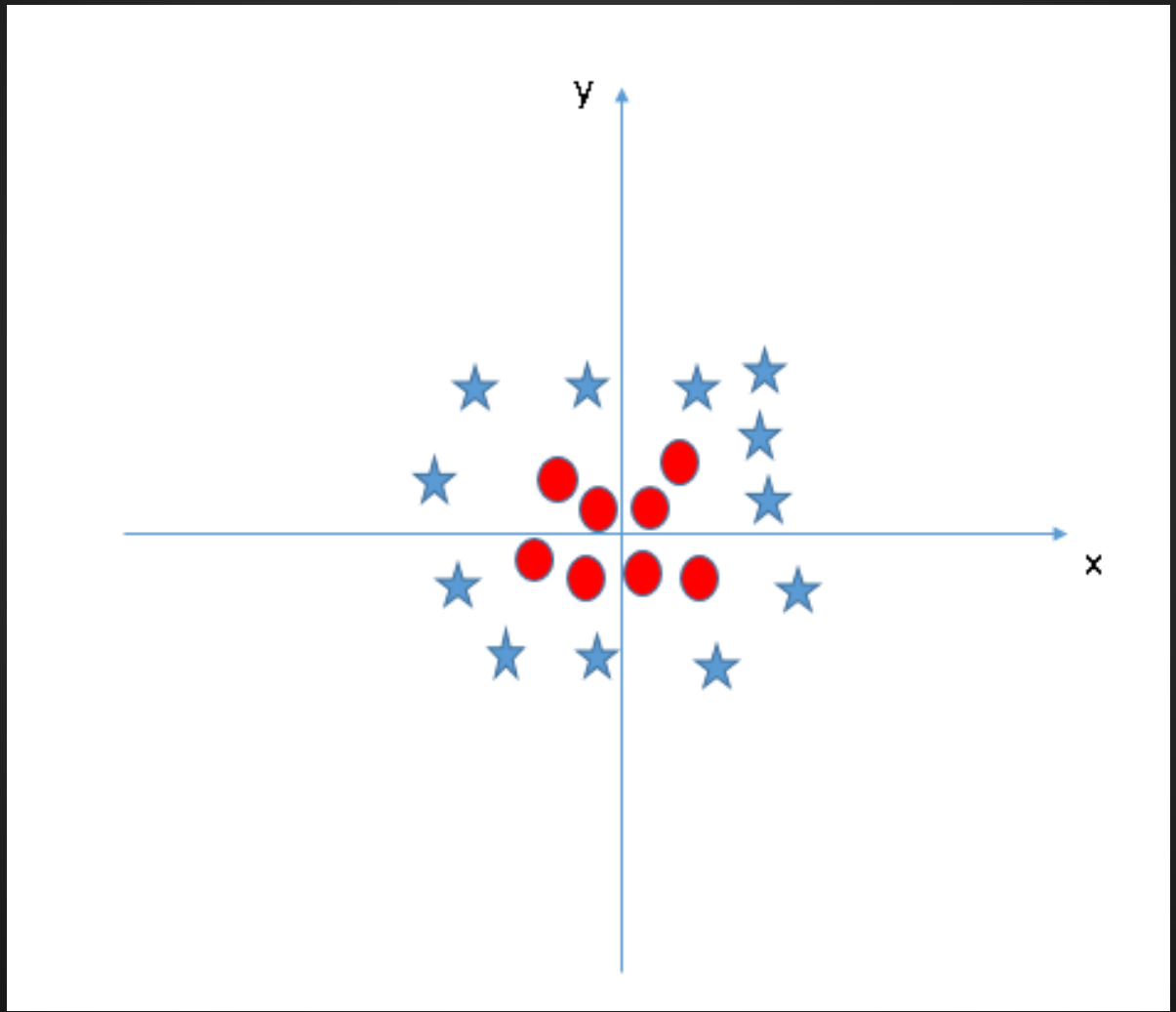
Support Vector Machines



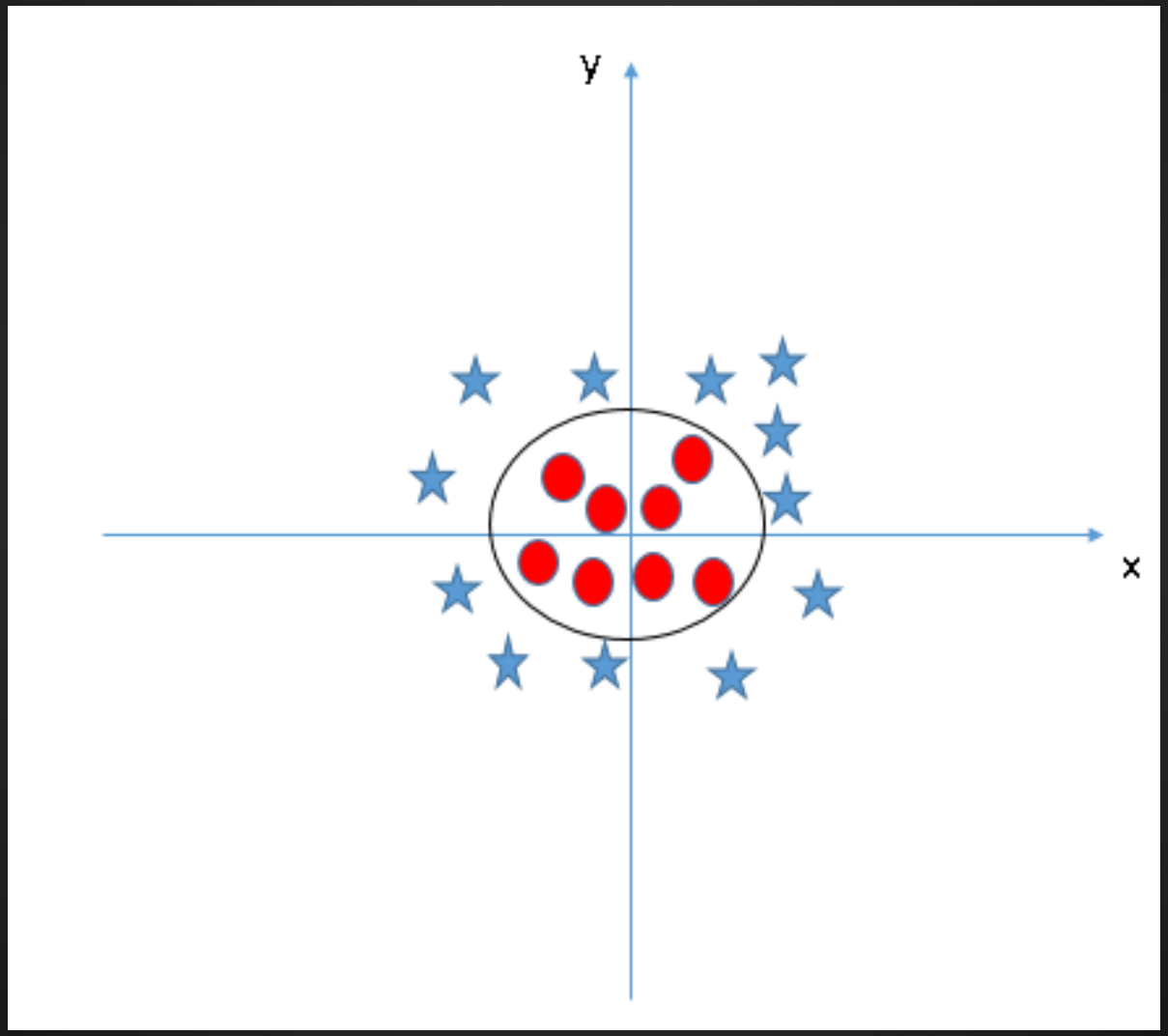
Support Vector Machines



Support Vector Machines



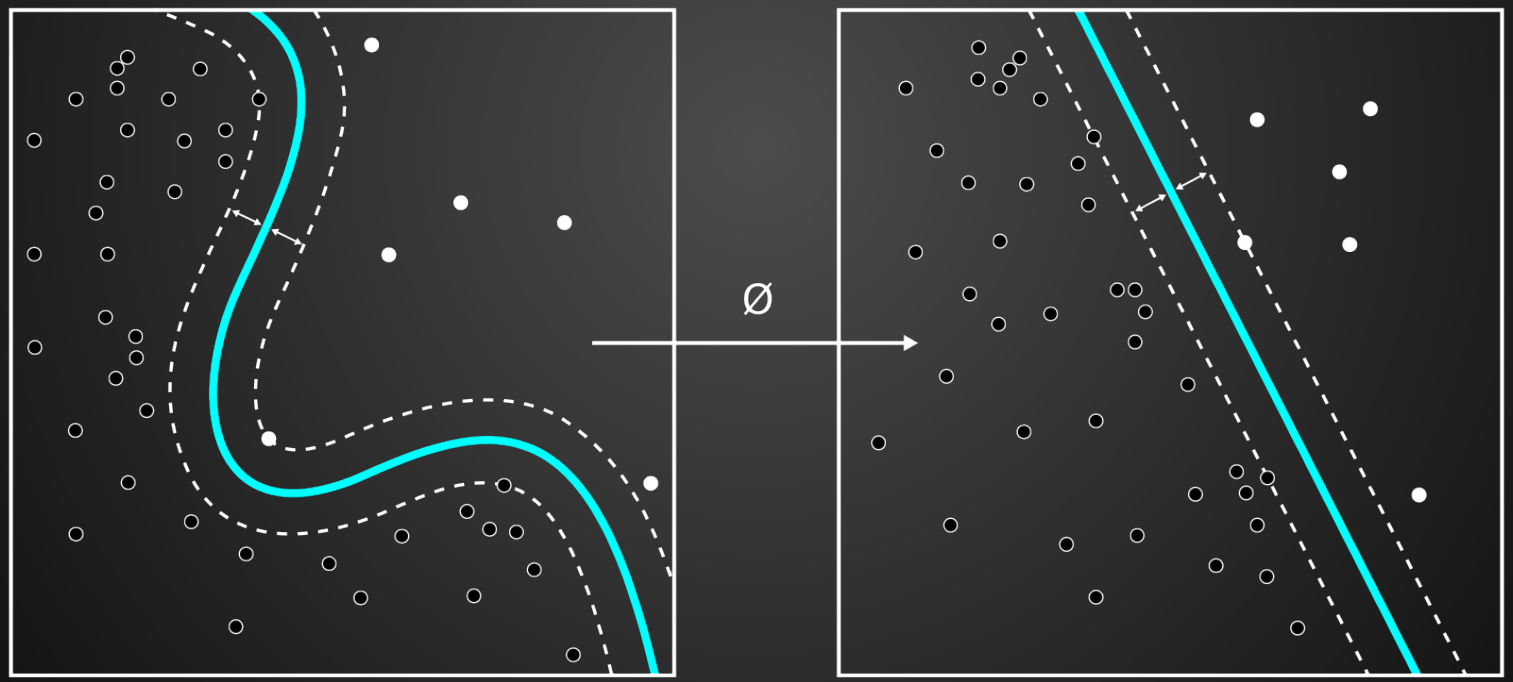
Support Vector Machines



Support Vector Machines

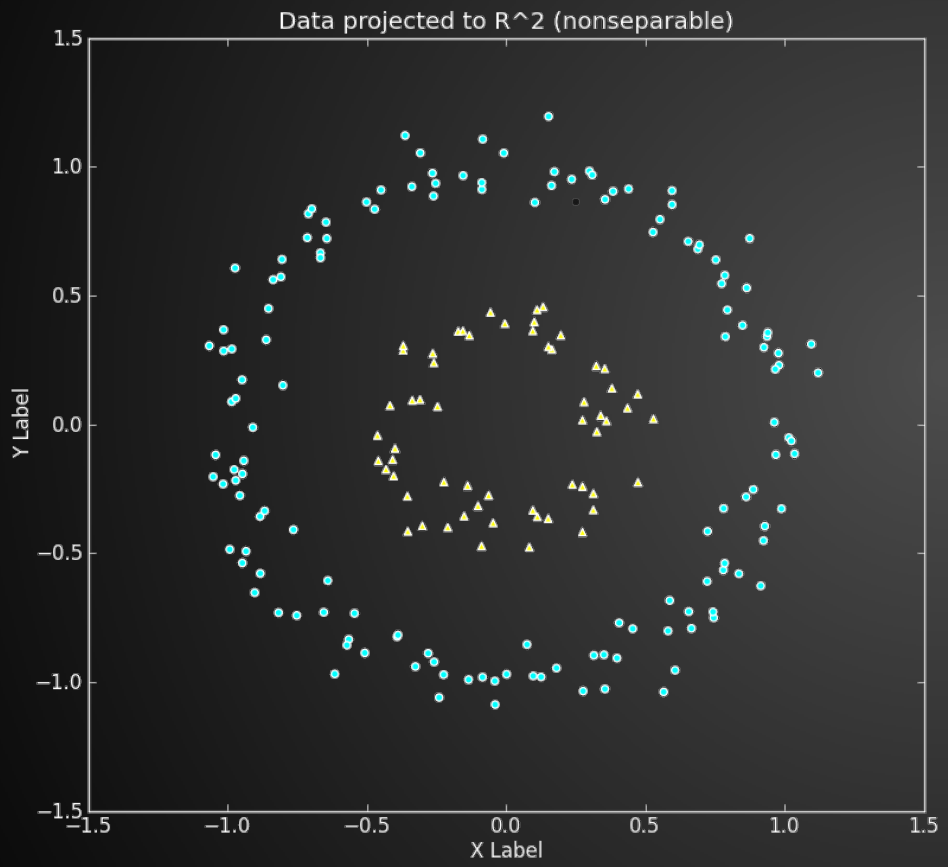
The Kernel Trick

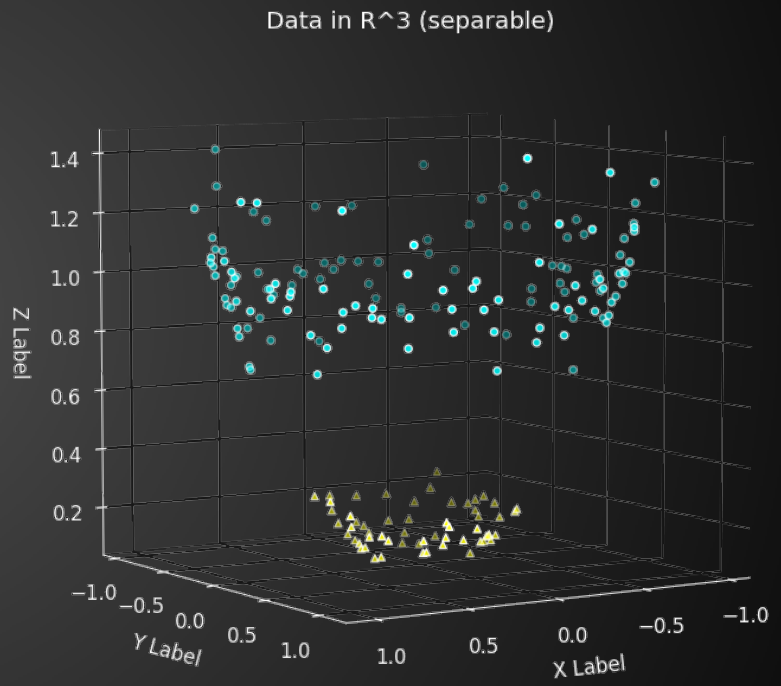
- Map data to a high dimensional space in which a hyper plane separation is possible. Use Kernels
- Map back to original space



Support Vector Machines

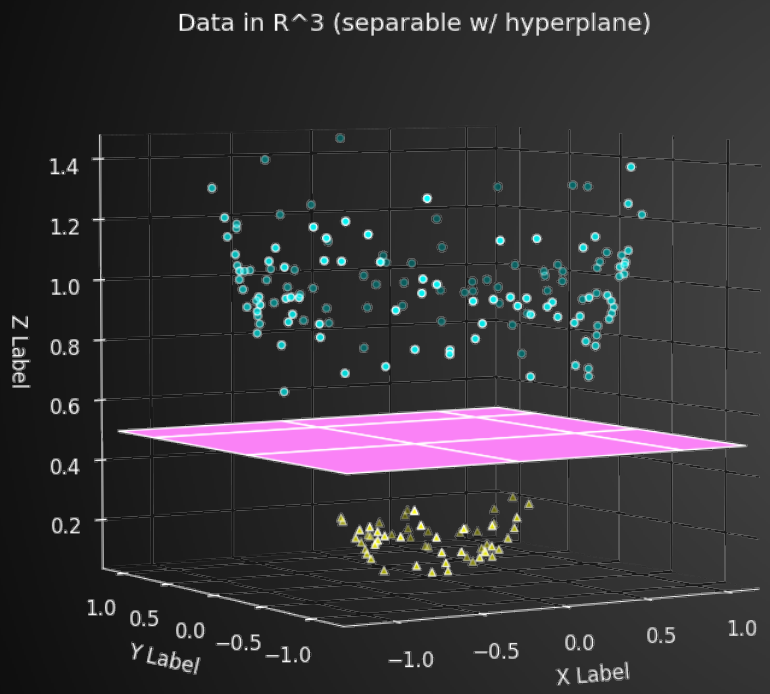
The Kernel Trick

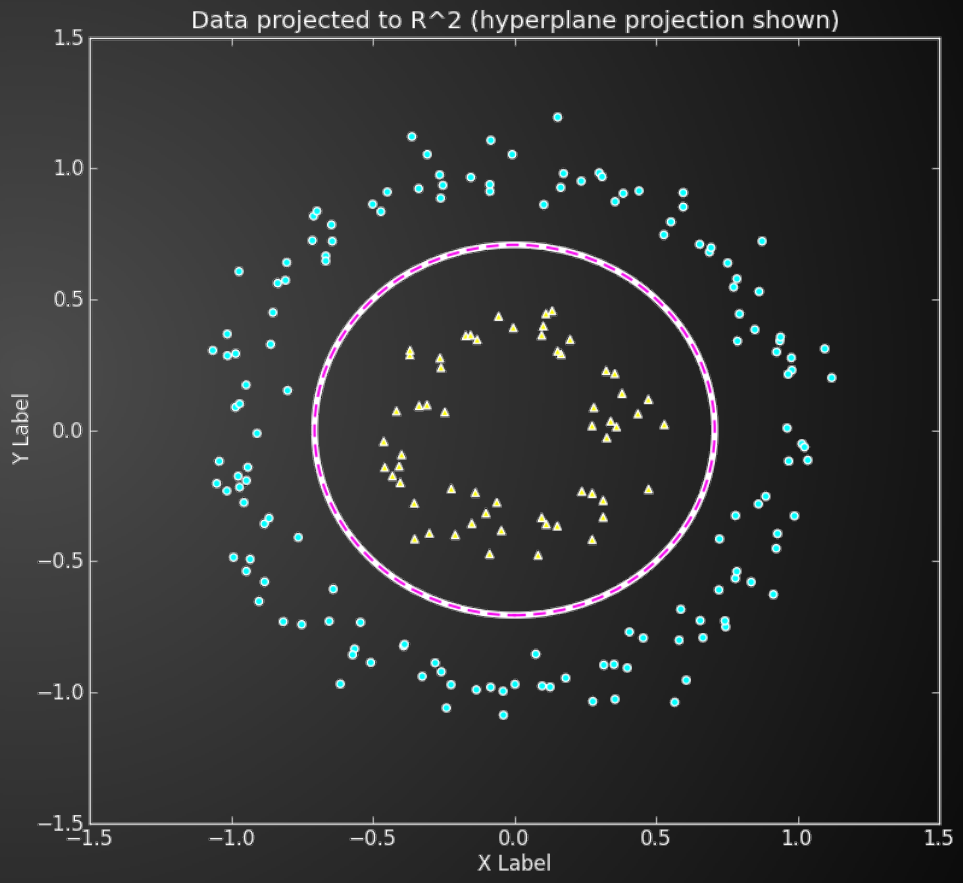




Support Vector Machines

The Kernel Trick





Support Vector Machines

The Kernel Trick

- It turns out that the SVM has no need to explicitly work in the higher-dimensional space at training or testing time.
- One can show that during training, the optimization problem only uses the training examples to compute pair-wise dot products.
- Why is this significant? It turns out that there exist functions that, given two vectors v and w in $\mathbb{R}^n$, implicitly computes the dot product between v and w in a higher-dimensional $\mathbb{R}^m$ without explicitly transforming v and w to $\mathbb{R}^m$. Such functions are called kernel functions.
- By using a kernel , we can implicitly transform datasets to a higher-dimensional using no extra memory, and with a minimal effect on computation time.
- Applicable to “Big Data Problems”.

SVM

Experiment with python notebook

Decision Trees

- Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression.
- The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

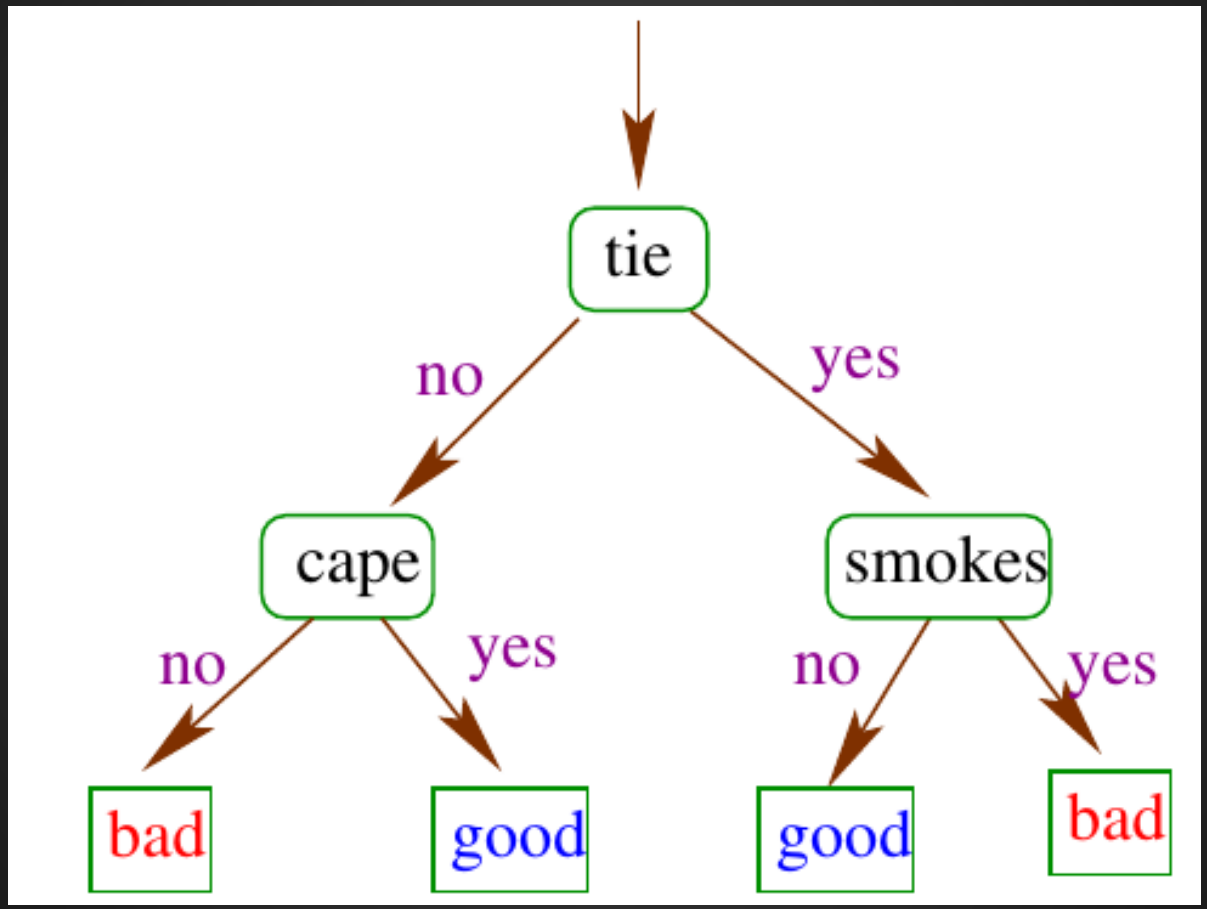
Example: Good versus Evil

- problem: identify people as good or bad from their appearance

	sex	mask	cape	tie	ears	smokes	class
			training data				
batman	male	yes	yes	no	yes	no	Good
robin	male	yes	yes	no	no	no	Good
alfred	male	no	no	yes	no	no	Good
penguin	male	no	no	yes	no	yes	Bad
catwoman	female	yes	no	no	yes	no	Bad
joker	male	no	no	no	no	no	Bad
			test data				
batgirl	female	yes	yes	no	yes	no	??
riddler	male	yes	no	no	no	no	??

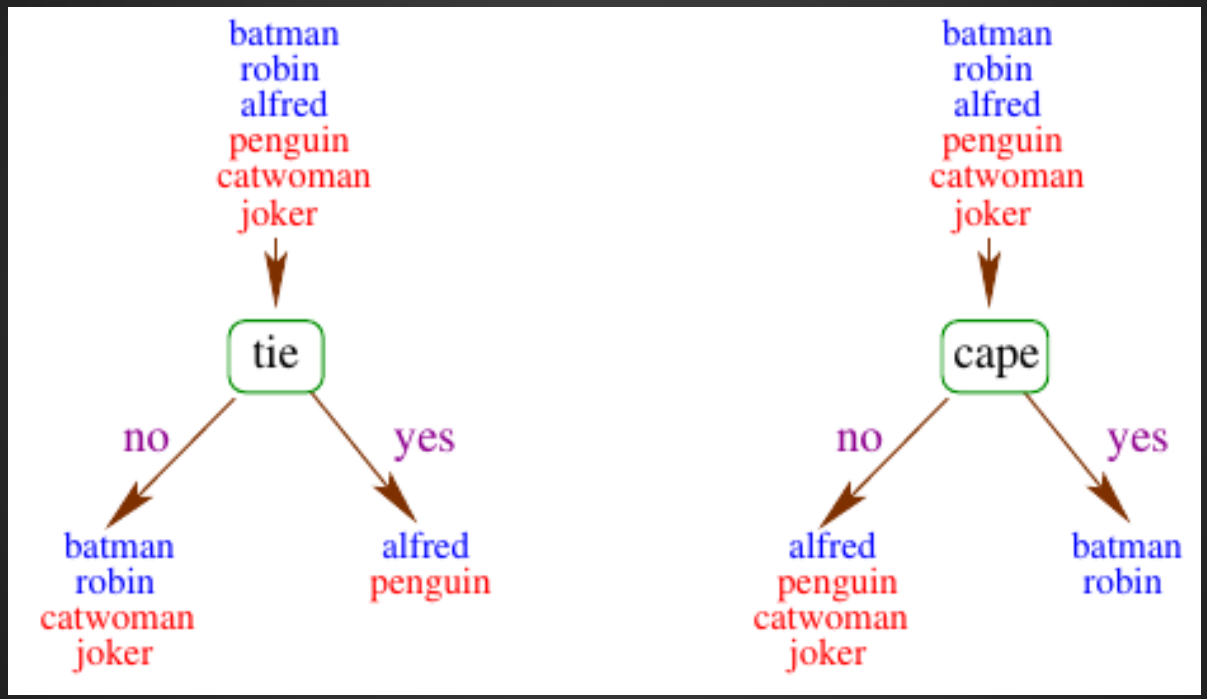
Decision Trees

Example



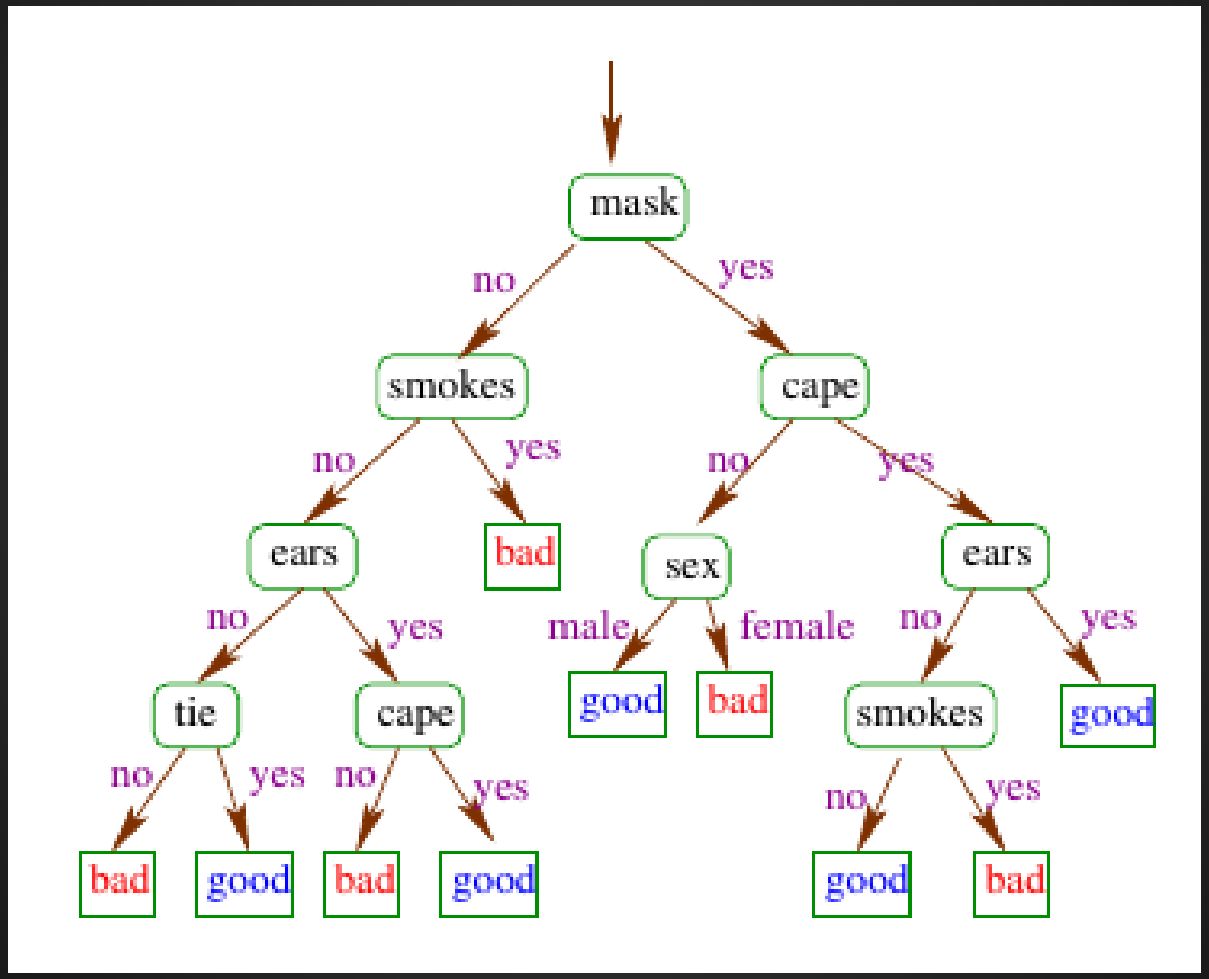
Decision Trees

How to Split



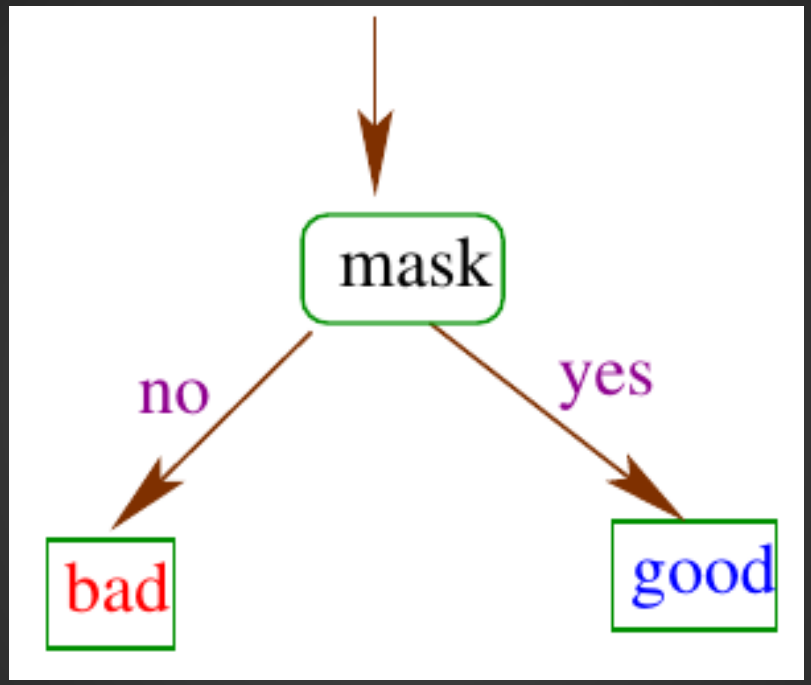
Decision Trees

Exhaustive Tree



Decision Trees

Simple Tree: which one is better?



- Training Error: no. of training samples misclassified
- Test Error: no. of test samples misclassified
- Generalization Error: Probability of misclassification of new samples

Decision Trees

Advantages

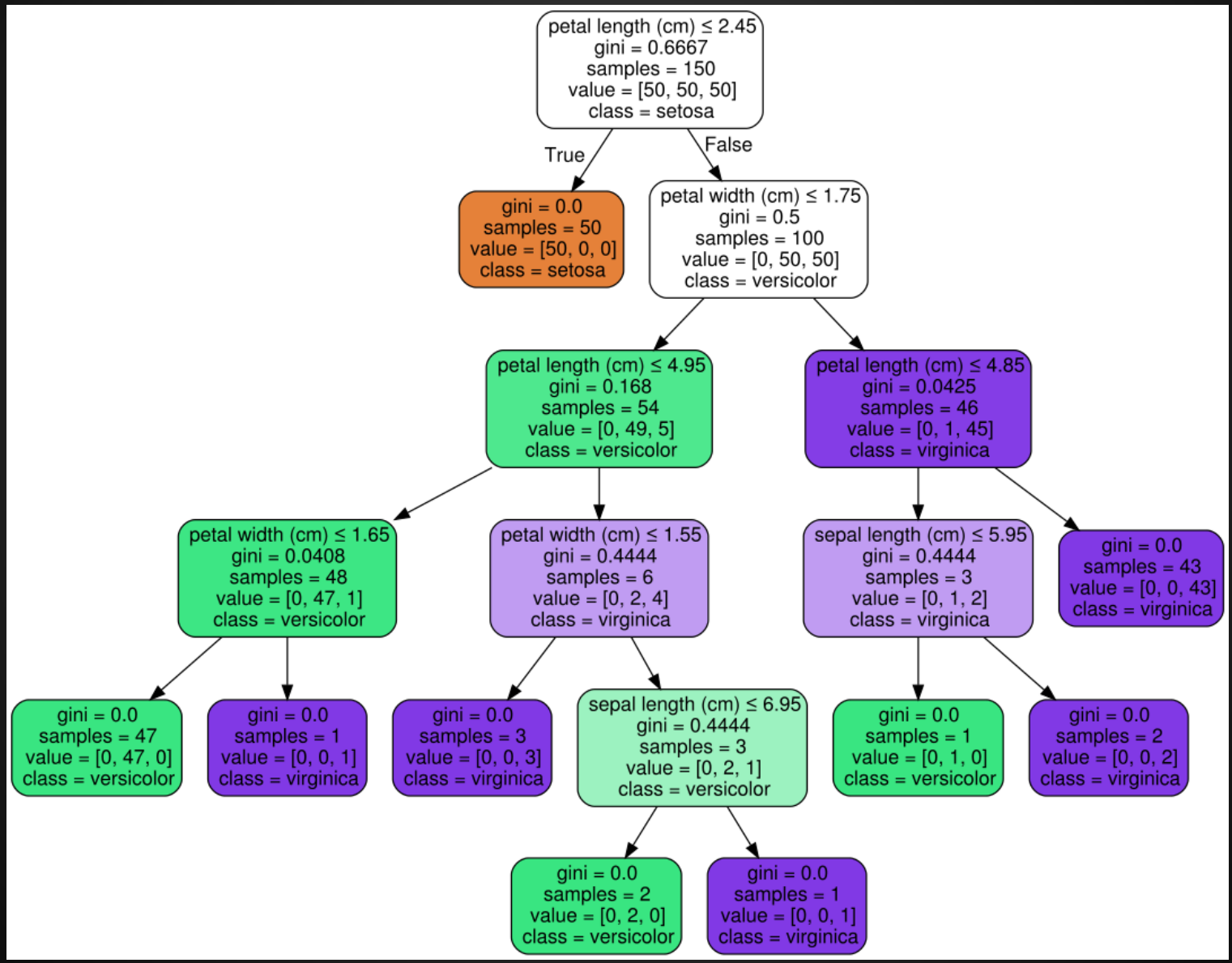
Some advantages of decision trees are:

- Simple to understand and to interpret. Trees can be visualised.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data.
- Able to handle multi-output problems.
- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic.

  By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.

Decision Trees

iris dataset



Decision Trees

Construction Approach

How do we split the nodes?

- Gini Impurity can be computed by summing the probability f_i of an item with label i being chosen times the probability $1 - f_i$ of a mistake in categorizing that item.

  It reaches its minimum (zero) when all cases in the node fall into a single target category.

  To compute Gini impurity for a set of items with J classes, suppose $i \in \{1, 2, ..., J\}$ $i \in \{1, 2, ..., J\}$, and let f_i be the fraction of items labeled with class i in the set.

$$I_G(f) = \sum_{i=1}^{J} f_i(1 - f_i) = \sum_{i=1}^{J} (f_i - {f_i}^2) = \sum_{i=1}^{J} f_i - \sum_{i=1}^{J} {f_i}^2 = 1 - \sum_{i=1}^{J} {f_i}^2 = \sum_{i \neq k} f_i f_k$$

- Information gain is used to decide which feature to split on at each step in building the tree. Simplicity is best, so we want to keep our tree small. To do so, at each step we should choose the split that results in the purest daughter nodes.

Decision Trees

The disadvantages of decision trees include:

- Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble (Random Forests).
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- There are concepts that are hard to learn because decision trees do not express them easily.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

Decision Trees

Experiment with python notebook

Random Forests

- In random forests each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set.
- In addition, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features.
- Instead, the split that is picked is the best split among a random subset of the features.
- As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model.

Random Forests

Experiment with python notebook