Overview of ML models

Parameter

- The machine learns from the training data to map the target function, but the configuration of the function is unknown.
- Different algorithms make various conclusions or biases about the function‘s structure, so our task as machine learning practitioners is to test various machine learning algorithms to see which one is effective at modeling the underlying function.
- Thus machine learning models are parameterized so that their behavior can be tuned for a given problem. These models can have many parameters and finding the best combination of parameters can be treated as a search problem.

What is a parameter in a machine learning model?

- A model parameter is a configuration variable that is internal to the model and whose value can be estimated from the given data.
    - They are required by the model when making predictions.
    - Their values define the skill of the model on your problem.
    - They are estimated or learned from historical training data.
    - They are often not set manually by the practitioner.
    - They are often saved as part of the learned model.
- The examples of model parameters include:
    - The weights in an artificial neural network.
    - The support vectors in a support vector machine.
    - The coefficients in linear regression or logistic regression.

What is the parametric model?

- A learning model that summarizes data with a set of fixed-size parameters (independent on the number of instances of training).Parametric machine learning algorithms are which optimizes the function to a known form.
- In a parametric model, you know exactly which model you are going to fit in with the data, for example, linear regression line.
- Following the functional form of a linear line clarifies the learning process greatly. Now we'll have to do is estimate the line equation coefficients and we have a predictive model for the problem. With the intercept and the coefficient, one can predict any value along with the regression.
- Some more examples of parametric machine learning algorithms include:
    - Logistic Regression
    - Linear Discriminant Analysis
    - Perceptron
    - Naive Bayes
    - Simple Neural Networks

nonparametric model?

- Nonparametric machine learning algorithms are those which do not make specific assumptions about the type of the mapping function.
- They are prepared to choose any functional form from the training data, by not making assumptions.
  The word nonparametric does not mean that the value lacks parameters existing in it, but rather that the parameters are adjustable and can change.
- A simple to understand the nonparametric model is the k-nearest neighbors' algorithm, making predictions for a new data instance based on the most similar training patterns k. The only assumption it makes about the data set is that the training patterns that are the most similar are most likely to have a similar result.
- Some more examples of popular nonparametric machine learning algorithms are:
  - k-Nearest Neighbors
  - Decision Trees like CART and C4.5
  - Support Vector Machines

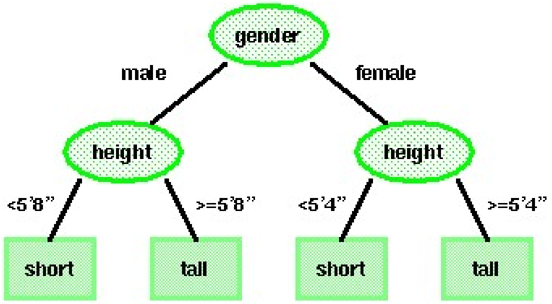
Example ML models for classification

Decision Trees

- A decision tree is a flowchart-like tree structure where an internal node represents a feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome.
- The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in a recursive manner called recursive partitioning. This flowchart-like structure helps you in decision-making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.
- A decision tree is a white box type of ML algorithm. It shares internal decision-making logic, which is not available in the black box type of algorithms such as with a neural network. Its training time is faster compared to the neural network algorithm.
- The time complexity of decision trees is a function of the number of records and attributes in the given data. The decision tree is a distribution-free or non-parametric method which does not depend upon probability distribution assumptions. Decision trees can handle high-dimensional data with good accuracy.

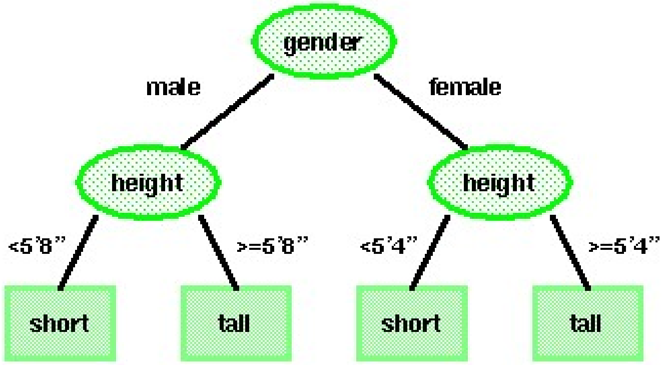
Overview

- Consider a classification problem that involves nominal data – data described by a list of attributes (e.g., categorizing people as short or tall using gender, height, age, and ethnicity).
- How can we use such nominal data for classification? How can we learn the categories of such data? *Nonmetric* methods such as decision trees provide a way to deal with such data.
- Decision trees attempt to classify a pattern through a sequence of questions. For example, attributes such as gender and height can be used to classify people as short or tall. But the best threshold for height is gender dependent.



- A decision tree consists of nodes and leaves, with each
- Classes (tall or short) are the outputs of the tree.
- Attributes (gender and height) are a set of features that describe the data.
- The input data consists of values of the different attributes. Using these attribute values, the decision tree generates a class as the output for each input data.

Basic Principles



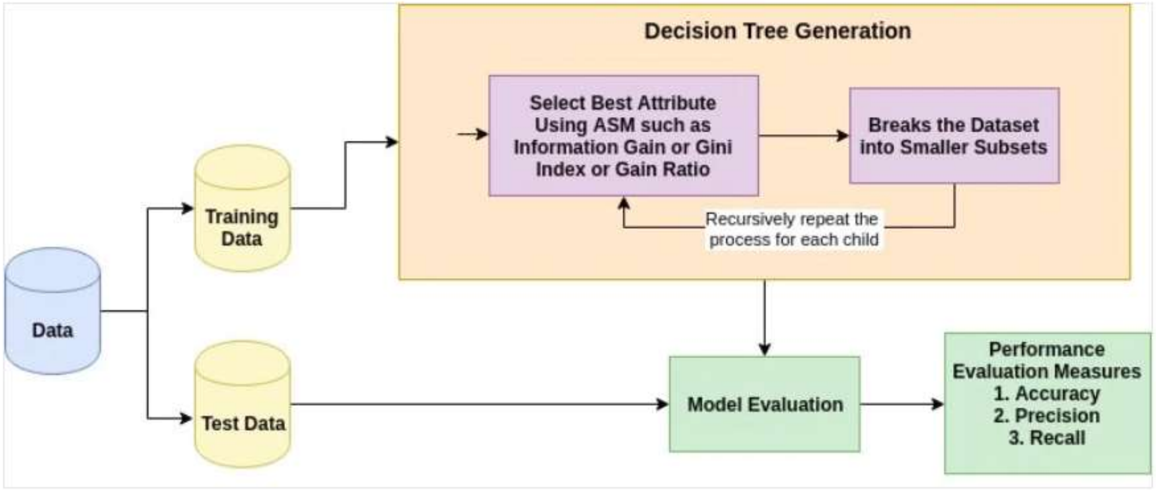
- The top, or first node, is called the root node.
- The last level of nodes are the leaf nodes and contain the final classification.
- The intermediate nodes are the descendant or "hidden" layers.
- Binary trees, like the one shown to the right, are the most popular type of tree. However, M-ary trees (M branches at each node) are possible.
- Nodes can contain one more questions. In a binary tree, by convention if the answer to a question is "yes", the left branch is selected. Note that the same question can appear in multiple places in the network.
- Decision trees have several benefits over neural network-type approaches, including interpretability and data-driven learning.
- Key questions include how to grow the tree, how to stop growing, and how to prune the tree to increase generalization.
- Decision trees are very powerful and can give excellent performance on closed-set testing. Generalization is a challenge.

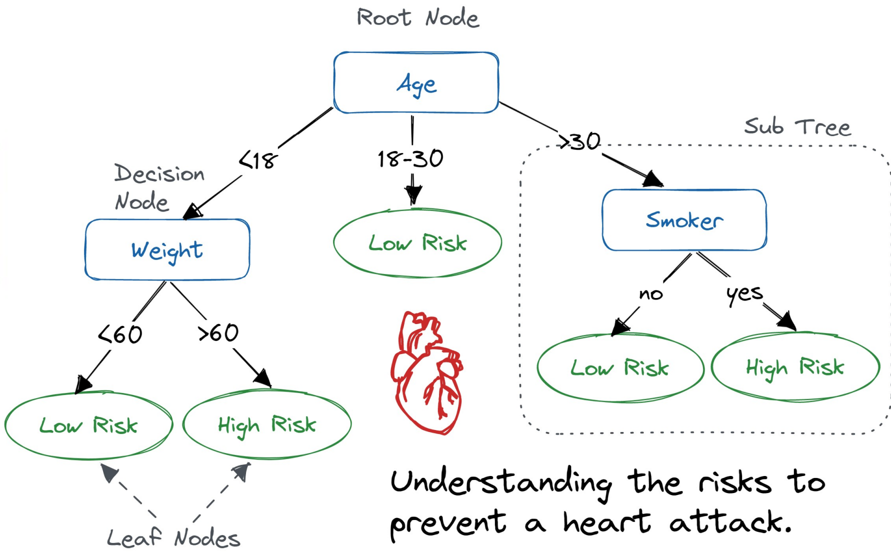
How Does the Decision Tree Algorithm Work?

- The basic idea behind any decision tree algorithm is as follows:
- Select the best attribute using Attribute Selection Measures (ASM) to split the records.
- Make that attribute a decision node and breaks the dataset into smaller subsets.
- Start tree building by repeating this process recursively for each child until one of the conditions will match:
    - All the tuples belong to the same attribute value.
    - There are no more remaining attributes.
    - There are no more instances.
- Stopping: if each leaf node contains data samples from the same class, or some pre-set threshold is not satisfied, stop. Otherwise, continue splitting.
- Pruning: use an independent test set or cross-validation to prune the tree.

Alternate Splitting Criteria/Attribute selection criteria

- Attribute selection measure is a heuristic for selecting the splitting criterion that partitions data in the best possible manner. It is also known as splitting rules because it helps us to determine breakpoints for tuples on a given node. ASM provides a rank to each feature (or attribute) by explaining the given dataset. The best score attribute will be selected as a splitting attribute . In the case of a continuous-valued attribute, split points for branches also need to define.
- The most popular selection measures are Information Gain, Gain Ratio, and Gini Index.
- **In practice, simple entropy splitting (choosing the question that splits the data into two classes of equal size) is very effective.**


Decision Tree Generation
Select Best Attribute Using ASM such as Information Gain or Gini Index or Gain Ratio
Breaks the Dataset into Smaller Subsets
Recursively repeat the process for each child
Data
Training Data
Test Data
Model Evaluation
Performance Evaluation Measures
1. Accuracy
2. Precision
3. Recall



Understanding the risks to prevent a heart attack.

When To Stop Splitting

- **If we continue to grow the tree until each leaf node has the lowest impurity, then the data will be overfit.**
- **Two strategies: (1) stop tree from growing or (2) grow and then prune the tree.**
- **A traditional approach to stopping splitting relies on *cross-validation*:**
  - ***Validation*: train a tree on 90% of the data and test on 10% of the data (referred to as the held-out set).**
  - ***Cross-validation*: repeat for several independently chosen partitions.**
  - ***Stopping Criterion*: Continue splitting until the error on the held-out data is minimized.**
- ***Reduction In Impurity*: stop if the candidate split leads to a marginal reduction of the impurity (drawback: leads to an unbalanced tree).**
- ***Cost-Complexity*: use a global criterion function that combines size and impurity:**

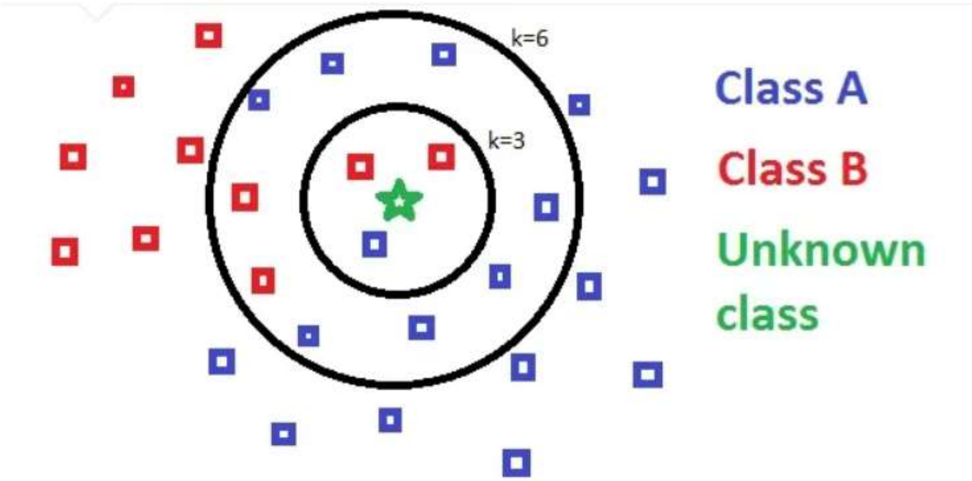
  . This approach is related to *minimum description length* when the impurity is based on entropy.

  $$\alpha \cdot size + \sum_{leaf\ nodes} i(N)$$
- **Other approaches based on *statistical significance* and *hypothesis testing* attempt to assess the quality of the proposed split.**

K Nearest Neighbour

KNN

- KNN (K-Nearest Neighbor) is a simple supervised classification algorithm we can use to assign a class to new data point. It can be used for regression as well, KNN does not make any assumptions on the data distribution, hence it is non-parametric. It keeps all the training data to make future predictions by computing the similarity between an input sample and each training instance.
- 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 the most 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 and satellite image scenes. Being a non-parametric method, it is often successful in classification situations where the decision boundary is very irregular.



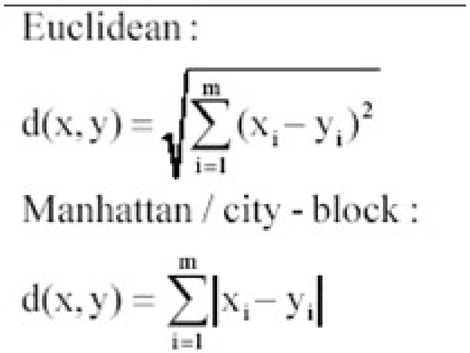
With K=3, Class B will be assigned, with K=6 Class A will be assigned

- KNN algorithm is one of the simplest classification algorithm
- non-parametric
  - it does not make any assumptions on the underlying data distribution
- lazy learning algorithm.
  - there is *no explicit training phase* or it is very minimal.
  - also means that the training phase is pretty fast .
  - Lack of generalization means that KNN keeps all the training data.
- Its purpose is to use a database in which the data points are separated into several classes to predict the classification of a new sample point.

K-Nearest Neighbor

- Features
  - All instances correspond to points in an n-dimensional Euclidean space
  - Classification is delayed till a new instance arrives
  - Classification done by comparing feature vectors of the different points
  - Target function may be discrete or real-valued

- K-nearest neighbours uses the local neighborhood to obtain a prediction
- The K memorized examples more similar to the one that is being classified are retrieved
- A distance function is needed to compare the examples similarity



Euclidean :

$$d(x, y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Manhattan / city - block :

$$d(x, y) = \sum_{i=1}^{m} \left| x_i - y_i \right|$$

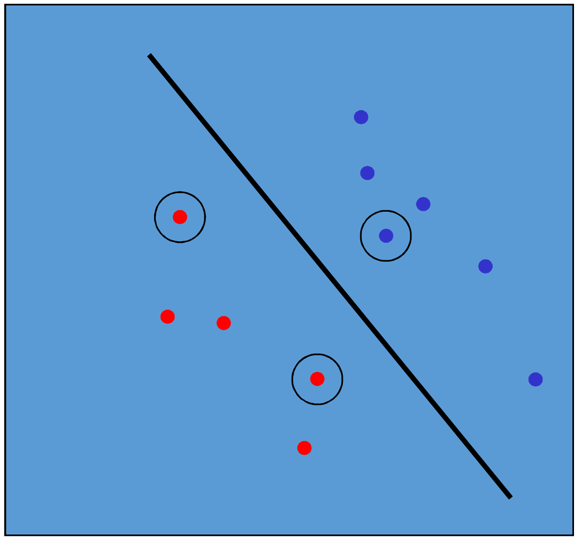
- This means that if we change the distance function, we change how examples are classified

- **KNN can be summarized as below:**
- Computes the distance between the new data point with every training example.
- For computing the distance measures such as Euclidean distance, Hamming distance or Manhattan distance will be used.
- Model picks K entries in the database which are closest to the new data point.
- Then it does the majority vote i.e the most common class/label among those K entries will be the class of the new data point.

Support Vector Machines

Support Vector Machines

- The line that maximizes the minimum margin is a good bet.
  - The model class of "hyper-planes with a margin of m" has a low VC dimension if m is big.
- This maximum-margin separator is determined by a subset of the datapoints.
  - Datapoints in this subset are called "support vectors".
  - It will be useful computationally if only a small fraction of the datapoints are support vectors, because we use the support vectors to decide which side of the separator a test case is on.



The support vectors are indicated by the circles around them.

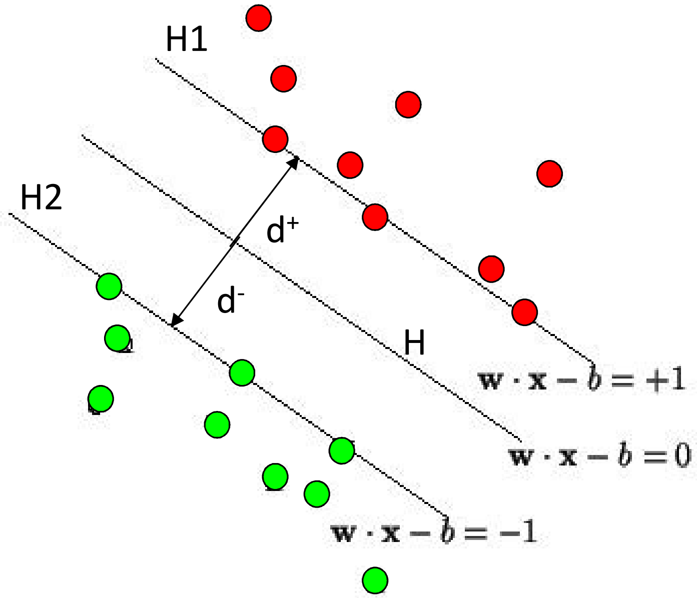
Definitions

Define the hyperplane H such that:

$\mathbf{x}_i \bullet \mathbf{w} + b \geq +1$ when $y_i = +1$

$\mathbf{x}_i \bullet \mathbf{w} + b \leq -1$ when $y_i = -1$

H1 and H2 are the planes:
H1: $\mathbf{x}_i \bullet \mathbf{w} + b = +1$
H2: $\mathbf{x}_i \bullet \mathbf{w} + b = -1$
The points on the planes H1 and H2 are the Support Vectors


H1
H2
d+
d-
H
$\mathbf{w} \cdot \mathbf{x} - b = +1$
$\mathbf{w} \cdot \mathbf{x} - b = 0$
$\mathbf{w} \cdot \mathbf{x} - b = -1$

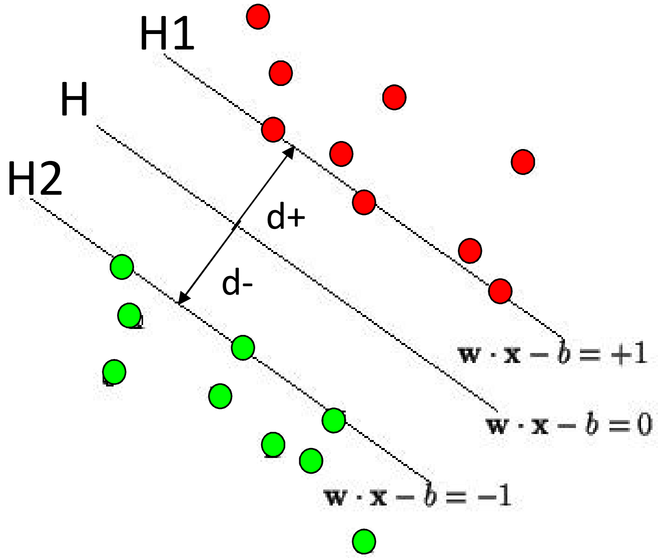
d+ = the shortest distance to the closest positive point

d- = the shortest distance to the closest negative point

The margin of a separating hyperplane is $d^+ + d^-$.

Maximizing the margin

We want a classifier with as big margin as possible.


H1
H
H2
d+
d-
$\mathbf{w} \cdot \mathbf{x} - b = +1$
$\mathbf{w} \cdot \mathbf{x} - b = 0$
$\mathbf{w} \cdot \mathbf{x} - b = -1$

Recall the distance from a point(x_0,y_0) to a line:
Ax+By+c = 0 is|A x_0 +B y_0 +c|/sqrt(A^2+B^2)

The distance between H and H1 is:
|**w•x**+b|/||w||=1/||w||

The distance between H1 and H2 is: 2/||w||

In order to maximize the margin, we need to minimize ||w||. With the condition that there are no datapoints between H1 and H2:

$\mathbf{x}_i \bullet \mathbf{w} + b \geq +1$ when y_i =+1
$\mathbf{x}_i \bullet \mathbf{w} + b \leq -1$ when y_i =-1 } **Can be combined into yi(x_i•w) ≥ 1**

Training a linear SVM

- To find the maximum margin separator, we have to solve the following optimization problem:

$$\mathbf{w}.\mathbf{x}^c + b > +1 \quad \textit{for positive cases}$$

$$\mathbf{w}.\mathbf{x}^c + b < -1 \quad \textit{for negative cases}$$

$$\textit{and} \quad \|\mathbf{w}\|^2 \ \textit{is as small as possible}$$

- This is tricky but it’s a convex problem. There is only one optimum and we can find it without fiddling with learning rates or weight decay or early stopping.
  - Don’t worry about the optimization problem. It has been solved. Its called quadratic programming.
  - It takes time proportional to N^2 which is really bad for very big datasets
    - so for big datasets we end up doing approximate optimization!

Testing a linear SVM

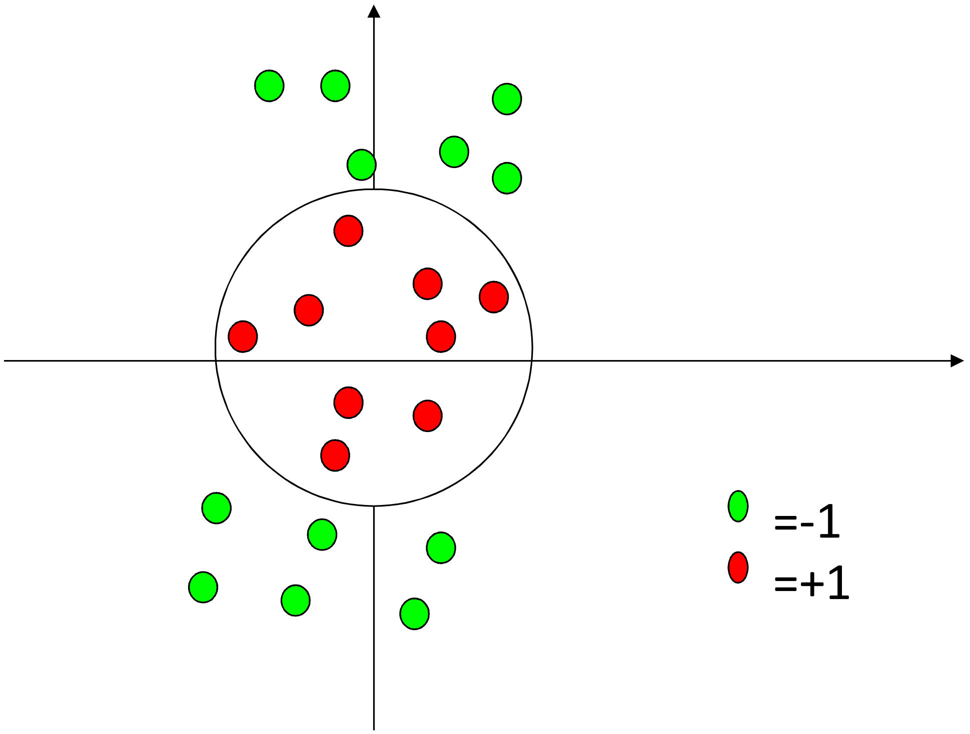
- The separator is defined as the set of points for which:

$$\mathbf{w}.\mathbf{x} + b = 0$$

$$so\ if \quad \mathbf{w}.\mathbf{x}^c + b > 0 \quad say\ its\ a\ positive\ case$$

$$and\ if \quad \mathbf{w}.\mathbf{x}^c + b < 0 \quad say\ its\ a\ negative\ case$$

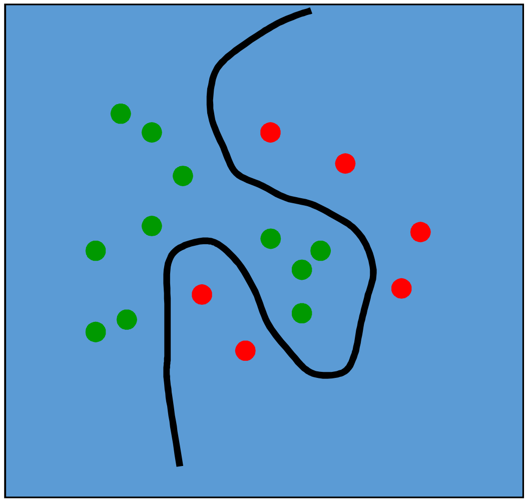
Problems with linear SVM



What if the decision function is not a linear?

How to make a plane curved

- Fitting hyperplanes as separators is mathematically easy.
  - The mathematics is linear.
- By replacing the raw input variables with a much larger set of features we get a nice property:
  - A planar separator in the high-dimensional space of feature vectors is a curved separator in the low dimensional space of the raw input variables.

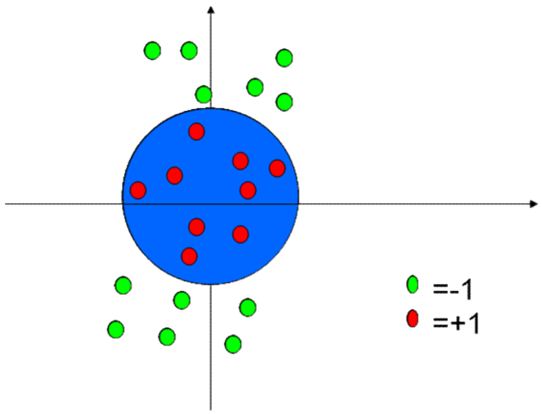


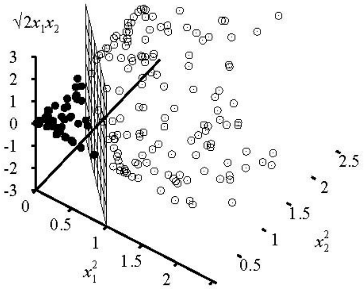
A planar separator in a 20-D feature space projected back to the original 2-D space

A potential problem and a magic solution

- If we map the input vectors into a very high-dimensional feature space, surely the task of finding the maximum-margin separator becomes computationally intractable?
  - The mathematics is all linear, which is good, but the vectors have a huge number of components.
  - So taking the scalar product of two vectors is very expensive.
- The way to keep things tractable is to use “the kernel trick”
- The kernel trick makes your brain hurt when you first learn about it, but its actually very simple.

Kernel Trick





Data points are linearly separable in the space $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$

What the kernel trick achieves

- All of the computations that we need to do to find the maximum-margin separator can be expressed in terms of scalar products between pairs of datapoints (in the high-dimensional feature space).
- These scalar products are the only part of the computation that depends on the dimensionality of the high-dimensional space.
  - So if we had a fast way to do the scalar products we would not have to pay a price for solving the learning problem in the high-D space.
- The kernel trick is just a magic way of doing scalar products a whole lot faster than is usually possible.
  - It relies on choosing a way of mapping to the high-dimensional feature space that allows fast scalar products.

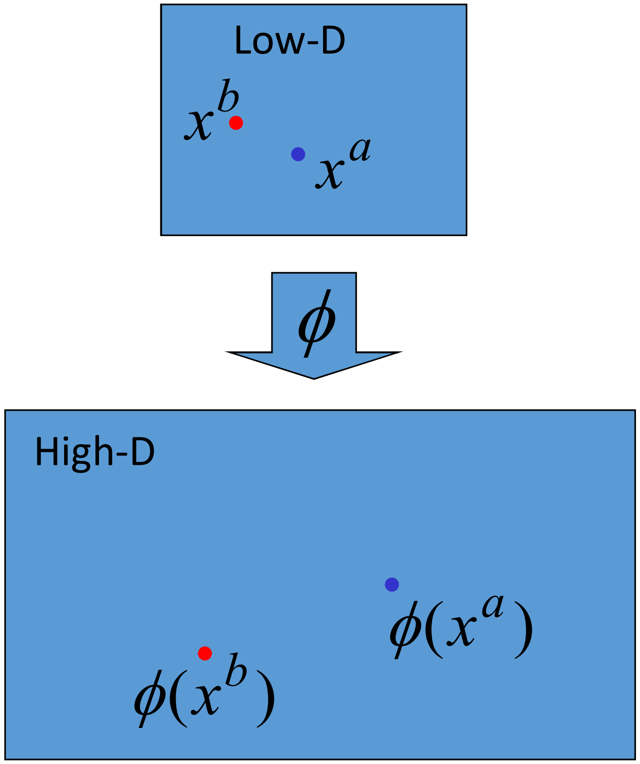
The kernel trick

- For many mappings from a low-D space to a high-D space, there is a simple operation on two vectors in the low-D space that can be used to compute the scalar product of their two images in the high-D space.

$$K(x^a, x^b) = \phi(x^a).\phi(x^b)$$

Letting the kernel do the work

doing the scalar product in the obvious way



Dealing with the test data

- If we choose a mapping to a high-D space for which the kernel trick works, we do not have to pay a computational price for the high-dimensionality when we find the best hyper-plane.
    - We cannot express the hyperplane by using its normal vector in the high-dimensional space because this vector would have a huge number of components.
    - Luckily, we can express it in terms of the support vectors.
- But what about the test data. We cannot compute the scalar product because its in the high-D space.

$$\mathbf{w} \,.\, \phi(\mathbf{x})$$

Dealing with the test data

- We need to decide which side of the separating hyperplane a test point lies on and this requires us to compute a scalar product.
- We can express this scalar product as a weighted average of scalar products with the stored support vectors
  - This could still be slow if there are a lot of support vectors .

The classification rule

- The final classification rule is quite simple:

$$bias + \sum_{s \, \varepsilon \, SV} w_s K(x^{test}, x^s) \quad > \; 0$$

The set of support vectors

- All the cleverness goes into selecting the support vectors that maximize the margin and computing the weight to use on each support vector.
- We also need to choose a good kernel function and we may need to choose a lambda for dealing with non-separable cases.

Some commonly used kernels

Polynomial: $K(\mathbf{x},\mathbf{y}) = (\mathbf{x}.\mathbf{y} + 1)^p$

Gaussian radial basis function $K(\mathbf{x},\mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|^2/2\sigma^2}$

Parameters that the user must choose

Neural net: $K(\mathbf{x},\mathbf{y}) = \tanh(k\,\mathbf{x}.\mathbf{y} - \delta)$

For the neural network kernel, there is one "hidden unit" per support vector, so the process of fitting the maximum margin hyperplane decides how many hidden units to use. Also, it may violate Mercer's condition.

Performance

- Support Vector Machines work very well in practice.
  - The user must choose the kernel function and its parameters, but the rest is automatic.
  - The test performance is very good.
- They can be expensive in time and space for big datasets
  - The computation of the maximum-margin hyper-plane depends on the square of the number of training cases.
  - We need to store all the support vectors.
- SVM's are very good if you have no idea about what structure to impose on the task.
- The kernel trick can also be used to do PCA in a much higher-dimensional space, thus giving a non-linear version of PCA in the original space.

Naive Bayes Classifier

- Naive Bayes is a statistical classification technique based on Bayes Theorem. It is one of the simplest supervised learning algorithms. Naive Bayes classifier is the fast, accurate and reliable algorithm. Naive Bayes classifiers have high accuracy and speed on large datasets.
- Naive Bayes classifier assumes that the effect of a particular feature in a class is independent of other features. For example, a loan applicant is desirable or not depending on his/her income, previous loan and transaction history, age, and location. Even if these features are interdependent, these features are still considered independently. This assumption simplifies computation, and that's why it is considered as naive. This assumption is called class conditional independence.

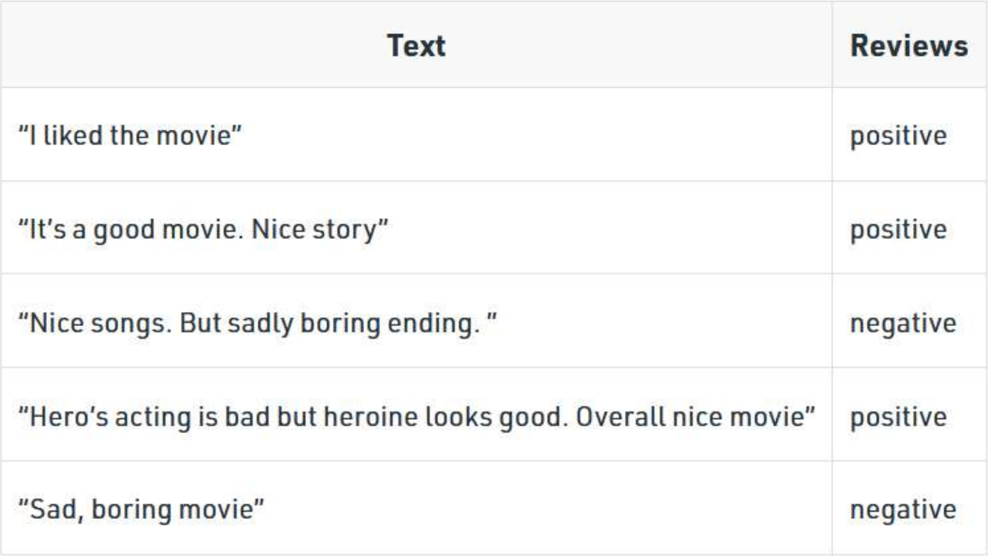
$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

- P(h): the probability of hypothesis h being true (regardless of the data). This is known as the prior probability of h.
- P(D): the probability of the data (regardless of the hypothesis). This is known as the prior probability.
- P(h|D): the probability of hypothesis h given the data D. This is known as posterior probability.
- P(D|h): the probability of data d given that the hypothesis h was true. This is the likelihood which is the probability of predictor given class.

- **First Approach (In case of a single feature)**
- Naive Bayes classifier calculates the probability of an event in the following steps:
- Step 1: Calculate the prior probability for given class labels
- Step 2: Find Likelihood probability with each attribute for each class
- Step 3: Put these value in Bayes Formula and calculate posterior probability.
- Step 4: See which class has a higher probability, given the input belongs to the higher probability class.

Example

- Training Dataset



Text	Reviews
“I liked the movie”	positive
“It’s a good movie. Nice story”	positive
“Nice songs. But sadly boring ending. ”	negative
“Hero’s acting is bad but heroine looks good. Overall nice movie”	positive
“Sad, boring movie”	negative

Test data: whether the text “overall liked the movie” has a positive review or a negative review.

Example

- We have to calculate,
  P(positive | overall liked the movie) — the probability that the tag of a sentence is positive given that the sentence is "overall liked the movie".
  P(negative | overall liked the movie) — the probability that the tag of a sentence is negative given that the sentence is "overall liked the movie".
- Our features will be the counts of each of these words.
  In our case, we have *P(positive | overall liked the movie)*, by using this theorem:
  - **P(positive | overall liked the movie)** = P(overall liked the movie | positive) * P(positive) / P(overall liked the movie)
  - **P(negative| overall liked the movie)** = P(overall liked the movie | negative) * P(negative) / P(overall liked the movie)
- Since for our classifier we have to find out which tag has a bigger probability, we can discard the divisor which is the same for both tags,
  - **P(positive | overall liked the movie) =** P(overall liked the movie | positive)* P(positive)
  - **P(negative| overall liked the movie)** = P(overall liked the movie | negative) * P(negative)
- Therefore
  - **P(overall liked the movie| positive)** = P(overall | positive) * P(liked | positive) * P(the | positive) * P(movie | positive)
- To get probability of a class, First, we calculate the a priori probability of each tag: for a given sentence in our training data, the probability that it is positive
  - P(positive) is 3/5. Then,
  - P(negative) is 2/5

The Scikit-learn provides different naïve Bayes classifiers models namely Gaussian, Multinomial, Complement and Bernoulli. All of them differ mainly by the assumption they make regarding the distribution of $\boldsymbol{P}$: $P(features|Y)$ i.e. the probability of predictor given class.

1 Gaussian Naïve Bayes Gaussian Naïve Bayes classifier assumes that the data from each label is drawn from a simple Gaussian distribution.

2 Multinomial Naïve Bayes It assumes that the features are drawn from a simple Multinomial distribution.

3 Bernoulli Naïve Bayes The assumption in this model is that the features binary (0s and 1s) in nature. An application of Bernoulli Naïve Bayes classification is Text classification with 'bag of words' model

4 Complement Naïve Bayes It was designed to correct the severe assumptions made by Multinomial Bayes classifier. This kind of NB classifier is suitable for imbalanced data sets

Machine Learning Algorithm/model

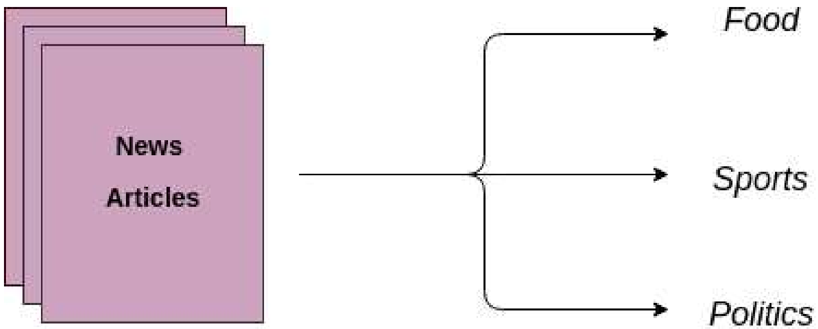
- **We have looked at several Ml algorithms for classification:**
  - Naïve bayes classifier
  - Decision Trees
  - K Nearest neibours
  - Logistic regression
  - Support vector machine
  - Artificial Neural Networks
  - …

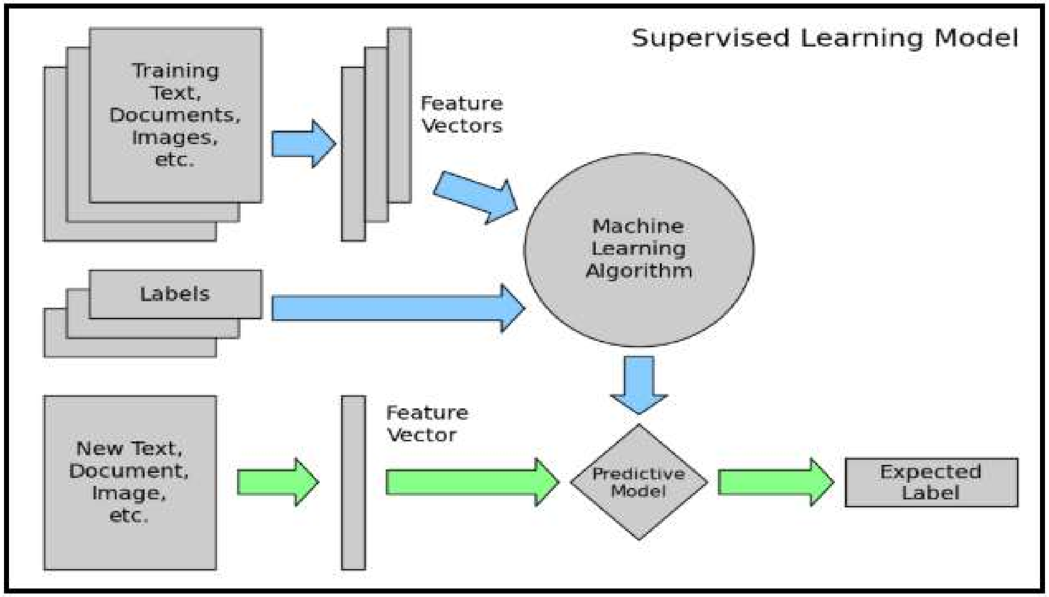
- But, at the end, we need to find the effectiveness of an algorithm.
- Next we look at model evaluation

Model Evaluation

Example Text Classification

- Text Classification is an example of supervised machine learning task since a labelled dataset containing text documents and their labels is used for train a classifier




Supervised Learning Model
Training Text, Documents, Images, etc.
Feature Vectors
Machine Learning Algorithm
Labels
New Text, Document, Image, etc.
Feature Vector
Predictive Model
Expected Label

Evaluation

- The training dataset trains the model to predict the unknown labels of population data.
- There are multiple algorithms, namely, Logistic regression, K-nearest neighbor, Decision tree, Naive Bayes etc. All these algorithms have their own style of execution and different techniques of prediction.
- But, at the end, we need to find the effectiveness of an algorithm.
- To find the most suitable algorithm for a particular business problem, there are few model evaluation techniques.

Evaluation

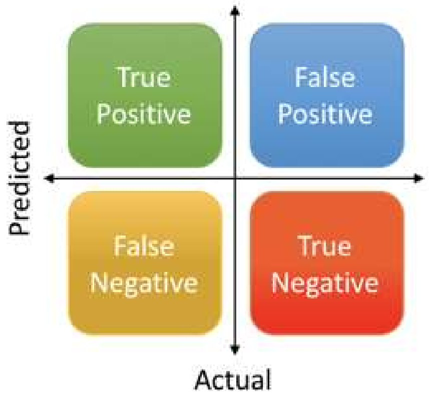
- For a classification task, *positive* means that an instance is labeled as belonging to the class of interest: we may want to automatically gather all news articles about Microsoft out of a news feed, or identify fraudulent credit card transactions, classify emails as spam or not e.t.c.
- A *false positive* is concluding that something is positive when it is not. False positives are sometimes called *Type I errors*.
- A *false negative* is concluding that something is negative when it is not. False negatives are sometimes called *Type II errors*.
- True negative is concluding that something is negative when it is actually negative
- True positive is concluding that something is positive when it is actually positive

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

$$\textbf{Precision} = \frac{\text{True Positive}}{\text{Actual Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Positive}}$$

$$\textbf{Recall} = \frac{\text{True Positive}}{\text{Predicted Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

$$\textbf{Accuracy} = \frac{\text{True Positive + True Negative}}{\text{Total}}$$



Predicted

True Positive

False Positive

False Negative

True Negative

Actual

Evaluation

- precision, recall
  - Precision means the percentage of your results which are relevant.
  - recall refers to the percentage of total relevant results correctly classified by your algorithm
- f-score
  - the harmonic mean of precision and recall:
  - near one when both precision and recall are high, and near zero when they are both low.
  - It is a convenient single score to characterize overall accuracy, especially for comparing the performance of different classifiers.

$$F_1 = \frac{2}{(1/\mathrm{precision} + 1/\mathrm{recall})} = 2 \times \frac{\mathrm{precision} \times \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}.$$

Example

- Classify email as either spam or not spam

Classify email as either spam or not spam

		Actual	
		Spam	Not spam
System (Predicted)	spam	12	8
	Not spam	3	77

- True Positive: **12 (You have predicted the positive case correctly!), system predicted spam and the are truly spam.**
- True Negative: **77 (You have predicted negative case correctly!), system predicted not spam and the email are truly not spam.**
- False Positive: **8 (Oh! You have predicted these emails are spam, but in actual they are not spam. This is type-II error in this case.)**
- False Negative: **3 (Oh ho! You have predicted that these three emails are not spam. But actually are spam. This is dangerous! Be careful! This is type-I error in this case.)**

Classify email as either spam or not spam

- Accuracy the ratio of the accurately predicted number and the total number of people which is *(12+77)/100 = 0.89*.
- Precision - the ratio, ***12/(12+8) = 0.6*** is the measure of the accuracy of your model in detecting a person to have the disease.
- Recall - the ratio, 12/(12+3) = 0.8 is the measure of the accuracy of your model to detect a person having disease out of all the people who are having the disease in actual.

Actual

		Spam	Not spam
System (Predicted)	spam	12	8
	Not spam	3	77

Cross-Validation

- Cross-validation involves **partitioning** your data into distinct **training** and **test** subsets.

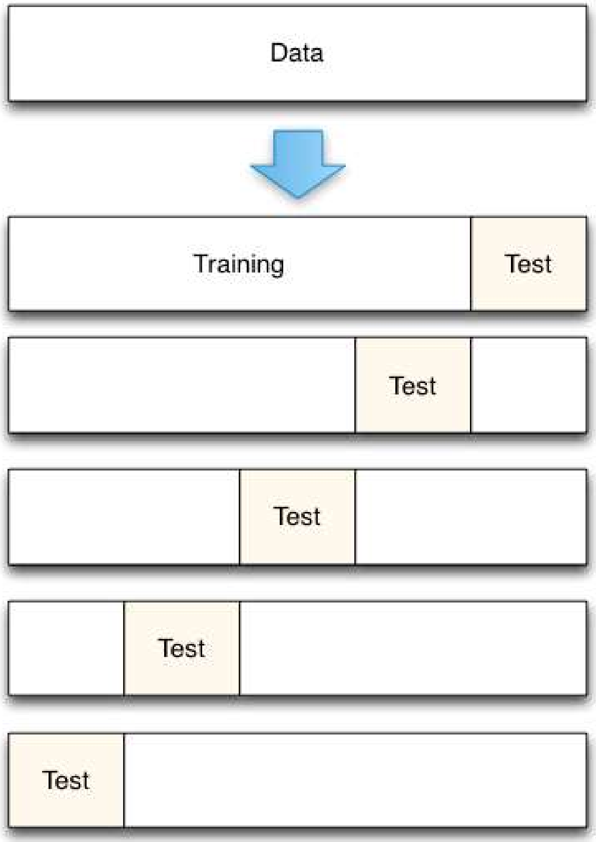
- The test set **should never** be used to **train** the model.

- The test set is then used to **evaluate** the model after training.

K-fold Cross-Validation

- To get more accurate estimates of performance you can do this k times.
- Break the data into k equal-sized subsets A_i
- For each i in 1,…,k do:
  - Train a model on all the other folds A_1,…, A_{i-1}, A_{i+1},…, A_k
  - Test the model on A_i
- Compute the **average performance** of the k runs

5-fold Cross-Validation



Model Selection

Model selection and generalization

- Machine learning problems (classification, regression and others) are typically ill-posed : the observed data is finite and does not uniquely determine the classification or regression function.
- In order to find a unique solution, and learn something useful, we must make assumptions (= inductive bias of the learning algorithm).

•Ex: the use of a hypothesis class H; the use of the largest margin; the use of the least-squares objective function.

- How to choose the right inductive bias, in particular the right hypothesis class H? This is the model selection problem.

Generalization

- Generalization: not memorization.

–minimizing error on the training set in general does not guarantee good generalization.

–too complex hypotheses could overfit training sample.

–how much complexity vs. training sample size?

Model selection and generalization

- The goal of ML is not to replicate the training data, but to predict unseen data well, i.e., to generalize well.
- For best generalization, we should match the complexity of the hypothesis class H with the complexity of the function underlying the data:

•If H is less complex: underfitting. Ex: fitting a line to data generated from a cubic polynomial.

•the excessively simple model fails to “Learn” the intricate patterns and underlying trends of the given dataset.

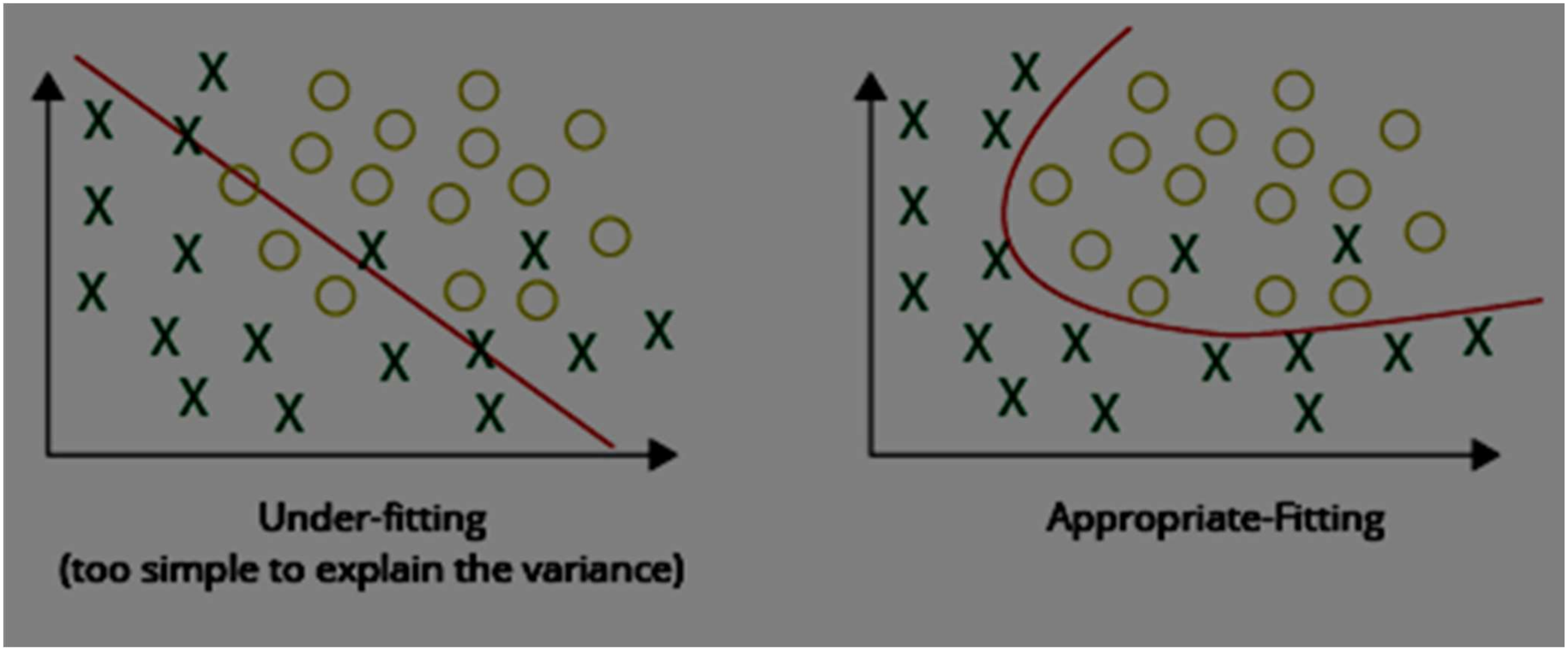
•If H is more complex: overfitting. Ex: fitting a cubic polynomial to data generated from a line.

- increasing model complexity, the model tends to fit the Noise present in data

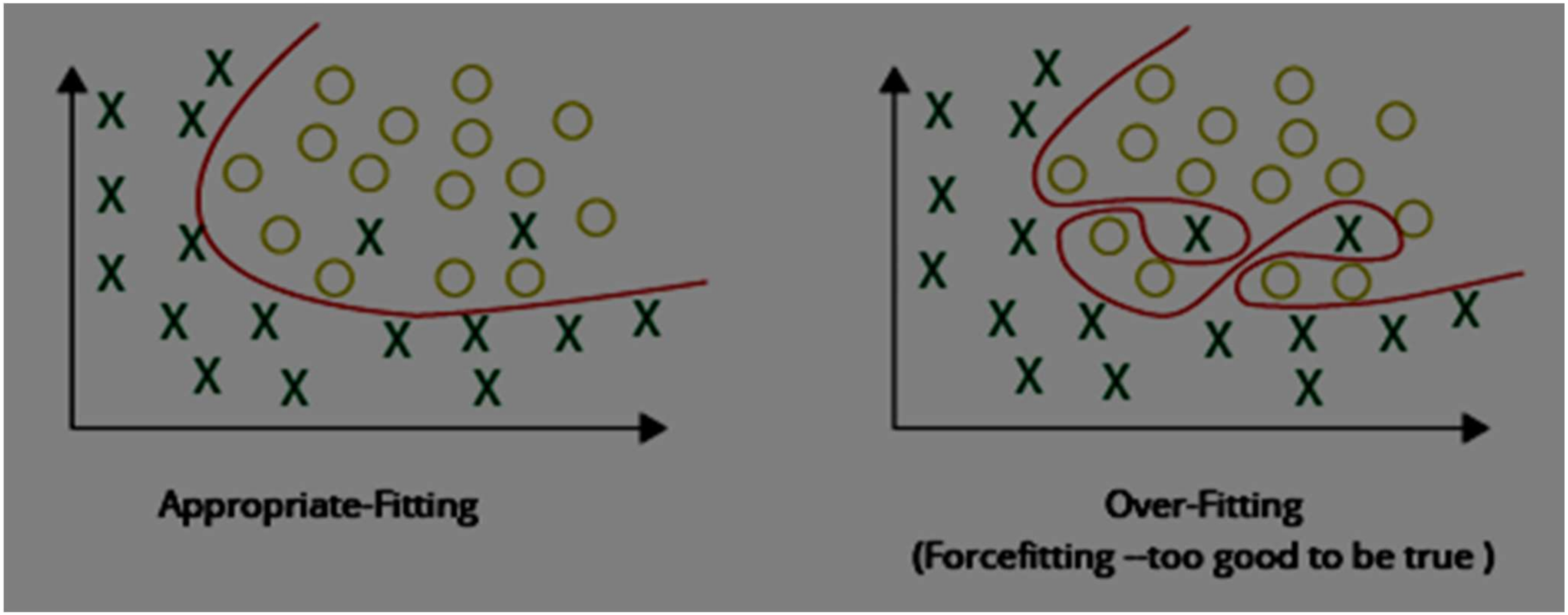
Over-fitting

- Your model should ideally fit an **infinite sample** of the type of data you're interested in.
- In reality, you only have a **finite set** to train on. A good model for this subset is a good model for the infinite set, up to a point.
- Beyond that point, the model quality (measured on new data) starts to **decrease**.
- Beyond that point, the model is **over-fitting** the data.

Model selection and generalization



Model selection and generalization



Appropriate-Fitting

Over-Fitting
(Forcefitting –too good to be true)

Bias-Variance Tradeoff

- Bias is the amount of error introduced by approximating real-world phenomena with a simplified model.
- Variance is how much your model's test error changes based on variation in the training data. It reflects the model's sensitivity to the idiosyncrasies of the data set it was trained on.
- As a model increases in complexity and it becomes more wiggly (flexible), its bias decreases (it does a good job of explaining the training data), but variance increases (it doesn't generalize as well). Ultimately, in order to have a good model, you need one with low bias and low variance.

Bias-Variance Tradeoff

- There are two major sources of error in machine learning: bias and variance. Understanding them will help you decide whether adding data, as well as other tactics to improve performance, are a good use of time.
- Suppose you hope to build a cat recognizer that has 5% error. Right now, your training set has an error rate of 15%, and your dev set has an error rate of 16%. In this case, adding training data probably won't help much.
- First, the algorithm's error rate on the training set. In this example, it is 15%. We think of this informally as the algorithm's bias.
- • Second, how much worse the algorithm does on the dev (or test) set than the training set. We think of this informally as the algorithm's variance.

Bias-Variance Tradeoff

- Consider our cat classification task. An “ideal” classifier (such as a human) might achieve nearly perfect performance in this task.
- Suppose your algorithm performs as follows:
- • Training error = 1%
- • Dev error = 11%
- What problem does it have? Applying the definitions from the previous chapter, we estimate the bias as 1%, and the variance as 10% (=11%-1%). Thus, it has high variance. The classifier has very low training error, but it is failing to generalize to the dev set. This is also called **overfitting**.

Bias-Variance Tradeoff

- Now consider this:
- • Training error = 15%
- • Dev error = 16%
- We estimate the bias as 15%, and variance as 1%. This classifier is fitting the training set poorly with 15% error, but its error on the dev set is barely higher than the training error.
- This classifier therefore has high bias, but low variance. We say that this algorithm is **underfitting**.

Bias-Variance Tradeoff

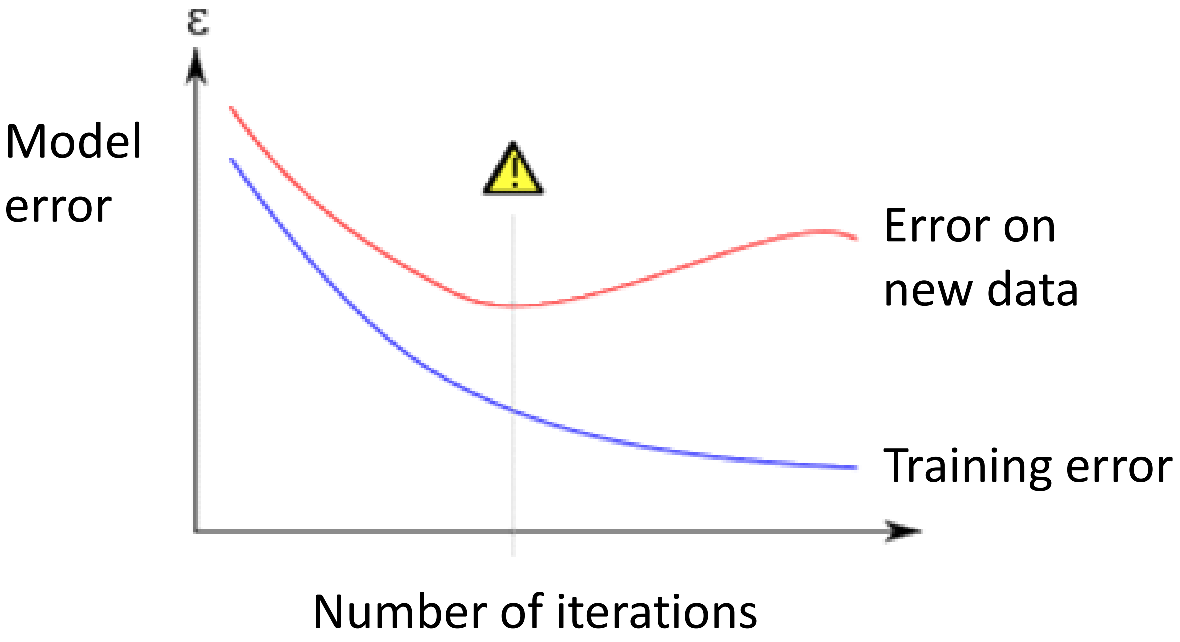
- Now, consider this:
- • Training error = 15%
- • Dev error = 30%
- We estimate the bias as 15%, and variance as 15%. This classifier has high bias and high variance: It is doing poorly on the training set, and therefore has high bias, and its performance on the dev set is even worse, so it also has high variance.
- The overfitting/underfitting terminology is hard to apply here since the classifier is simultaneously overfitting and underfitting.

Bias-Variance Tradeoff

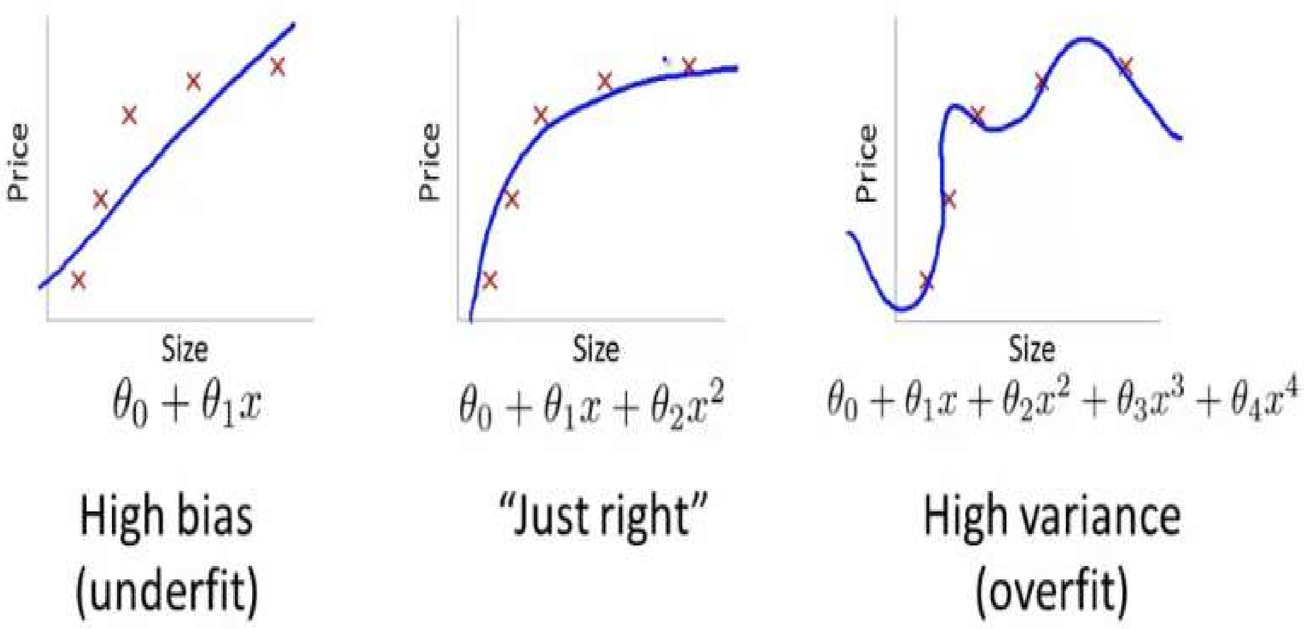
- Finally, consider this:
- • Training error = 0.5%
- • Dev error = 1%
- This classifier is doing well, as it has low bias and low variance. **Congratulations on achieving this great performance!**

Over-fitting

Over-fitting during training



Bias-Variance Tradeoff



Remember that the only thing we care about is how the model performs on test data.

Classification example - Complex model

- Should we keep the hypothesis class simple rather than complex?
- Easier to use and to train (fewer parameters, faster).
- Easier to explain or interpret.
- Less variance in the learned model than for a complex model (less affected by single instances), but also higher bias.
- Given comparable empirical error, a simple model will generalize better than a complex one. (**Occam's razor** : simpler explanations are more plausible; eliminate unnecessary complexity.)

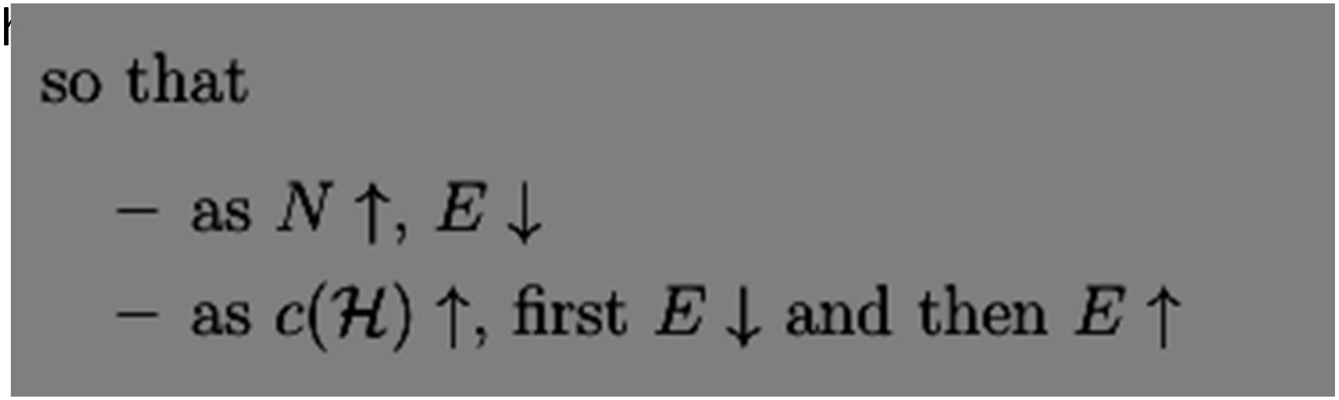
Model selection and generalization

- In summary, in ML algorithms there is a tradeoff between 3 factors:

•the complexity c(H) of the hypothesis class

•the amount of training data N

•th



so that

- as $N\uparrow$, $E\downarrow$
- as $c(\mathcal{H})\uparrow$, first $E\downarrow$ and then $E\uparrow$

Noise

- Noise is any unwanted anomaly in the data. It can be due to:

•Imprecision in recording the input attributes: xn .

•Errors in labeling the input vectors: yn .

•Attributes not considered that affect the label (hidden or latent attributes, may be unobservable).

- Noise makes learning harder.