CS 644: Introduction to Big Data

Daqing Yun (daqing.yun@njit.edu)
New Jersey Institute of Technology

Big Data Analytics Algorithms

- Similarity (Distance) Measurements
- Clustering
  - K-Means
- Classification
  - k Nearest Neighbor
  - Logistics Regression (via Gradient Descent)
  - Decision Tree Induction
  - Naïve Bayes
  - Support Vector Machine
- Regression
  - Linear Regression (Single and Multiple) and Non-linear Regression
- Ensemble Methods
  - Random Forest
  - AdaBoost
- Principle Component Analysis
- *Neural Networks*

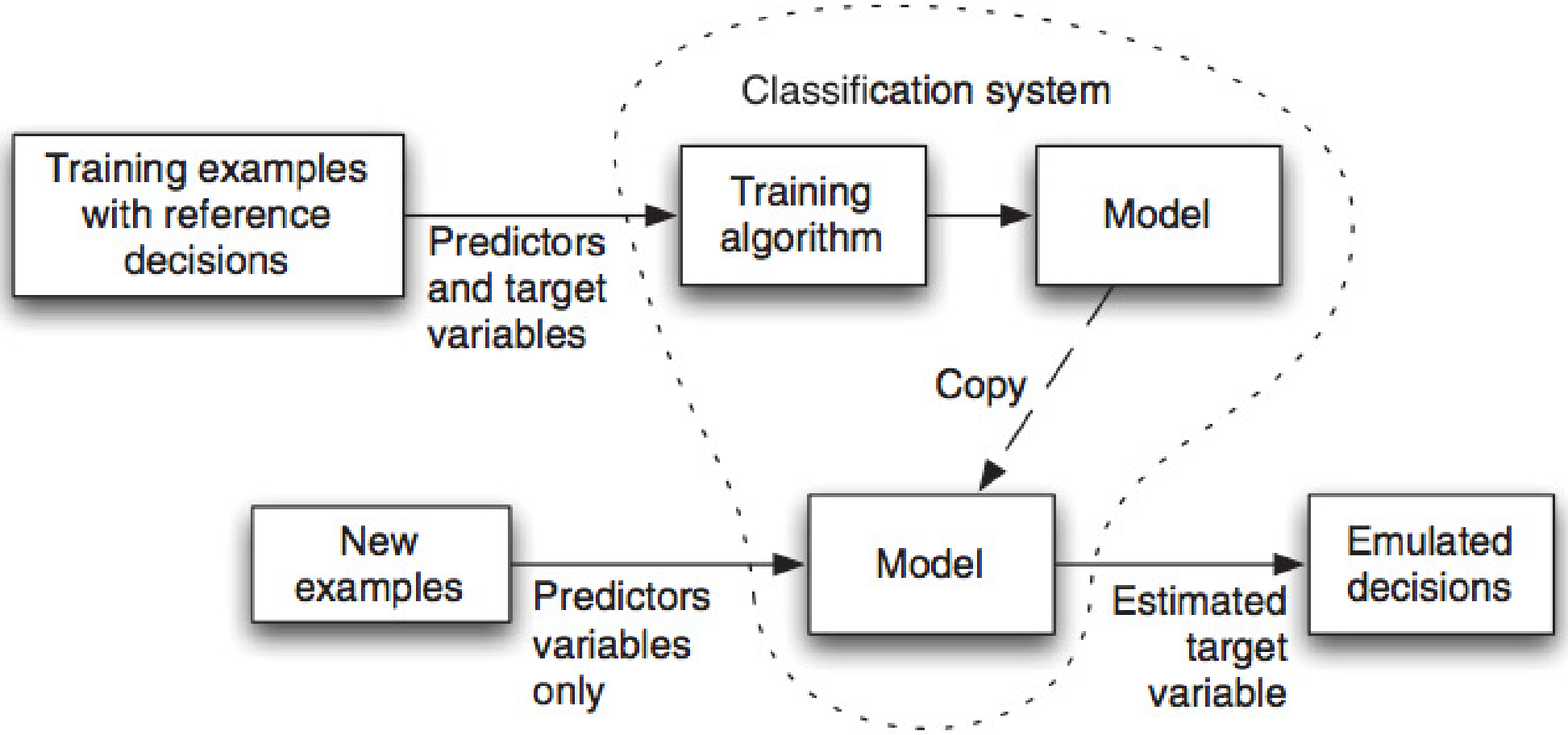
Hadoop-related Apache Projects

- **Ambari™**: A web based tool for provisioning, managing, and monitoring Hadoop clusters. It also provides a dashboard for viewing cluster health and ability to view MapReduce, Pig, and Hive applications visually.
- **Avro™**: A data serialization system.
- **Cassandra™**: A scalable multi-master database with no single points of failure.
- **Chukwa™**: A data collection system for managing large distributed systems.
- **HBase™**: A scalable, distributed database that supports structured data storage for large tables.
- **Hive™**: A data warehouse infrastructure that provides data summarization and ad hoc querying
- **Mahout™**: A scalable machine learning and data mining library.
- **Pig™**: A high-level data-flow language and execution framework for parallel computation
- **Spark™**: A fast and general compute engine for Hadoop data. Spark provides a simple and expressive programming model that supports a wide range of applications, including ETL, machine learning, stream processing, and graph computation.
- **Tez™**: A generalized data-flow programming framework, built on Hadoop YARN, which provides a powerful and flexible engine to execute an arbitrary DAG of tasks to process data for both batch and interactive use-cases.
- **ZooKeeper™**: A high-performance coordination service for distributed applications.

Classification - Definition

- Given a collection of records (*training set*)
  - Each record contains a set of *attributes*, one of the attributes is the (label) *class*
- Find a *model* for class attribute as a function of the values of other attributes
- Goal: *previously unseen* records should be assigned a class as accurately as possible
  - A *test set* is used to determine the accuracy of the model. Usually, the given data set is divided into training and test sets, with training set used to build the model and test set used to validate it
- Computer classification systems are a form of machine learning that use learning algorithms to provide a way for computers to make decision based on experience and, in the process, emulate certain forms of human decision making

How does a classification system work?



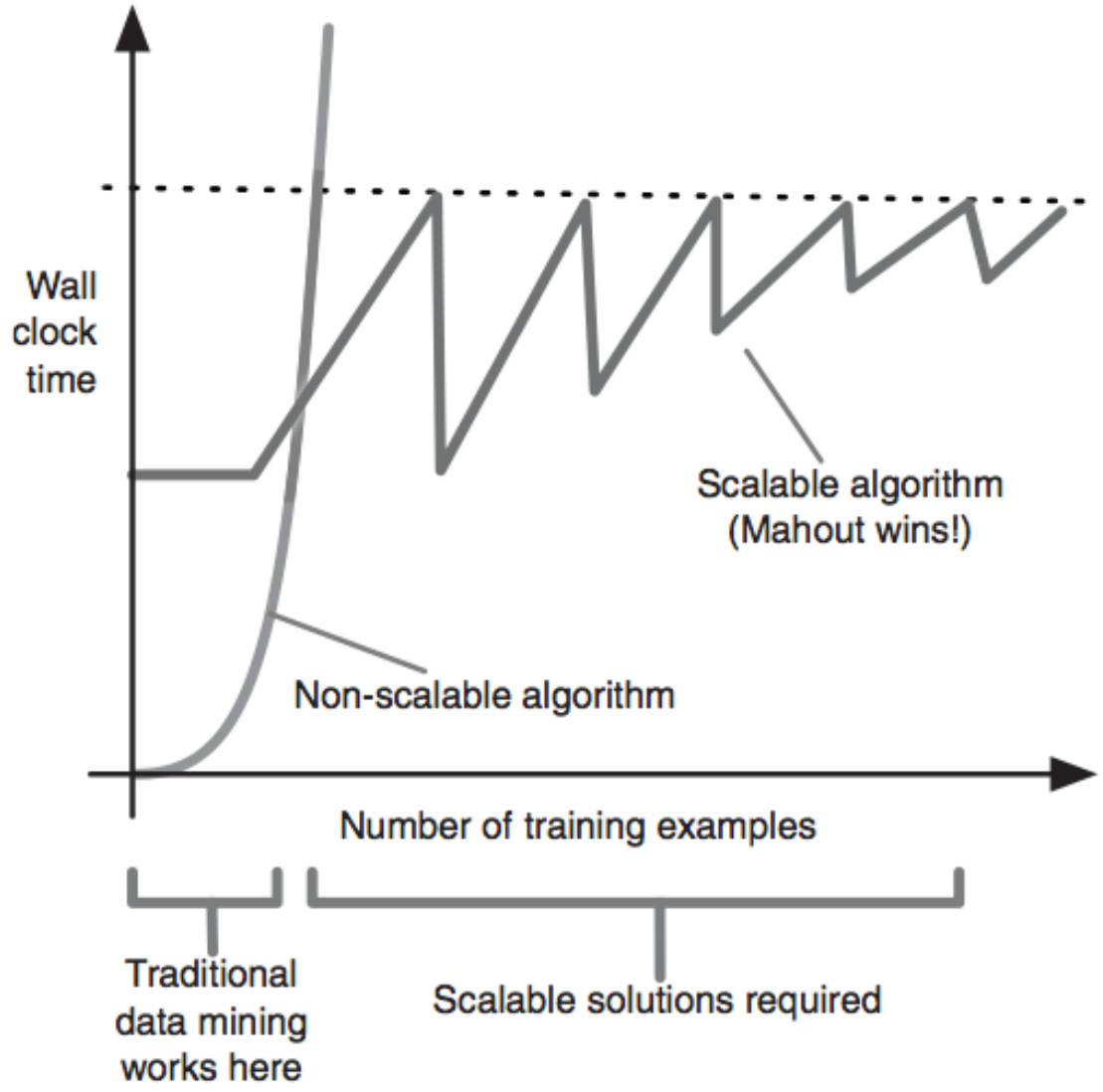
Recognizing the difference in cost of classification errors

- False alarm:
  - The cost of a false alarm may be much less than the cost of a false negative, e.g., Cancer, Missile
- Missed detection:
  - The cost of a false negative may be less than the cost of a false alarm, e.g., Spam

System size in number of examples	Choice of classification approach
< 100,000	Traditional, non-Mahout approaches should work very well. Mahout may even be slower for training.
100,000 to 1 million	Mahout begins to be a good choice. The flexible API may make Mahout a preferred choice, even though there is no performance advantage.
1 million to 10 million	Mahout is an excellent choice in this range.
> 10 million	Mahout excels where others fail.

When to use Mahout for classification?

The advantage of using Mahout for classification



Four types of values for predictor variables

Type of value	Description
Continuous	This is a floating-point value. This type of value might be a price, a weight, a time, or anything else that has a numerical magnitude and where this magnitude is the key property of the value.
Categorical	A categorical value can have one of a set of prespecified values. Typically the set of categorical values is relatively small and may be as small as two, although the set can be quite large. Boolean values are generally treated as categorical values. Another example might be a vendor ID.
Word-like	A word-like value is like a categorical value, but it has an open-ended set of possible values.
Text-like	A text-like value is a sequence of word-like values, all of the same kind. Text is the classic example of a text-like value, but a list of email addresses or URLs is also text-like.

Sample data that illustrates all four value types

Name	Type	Value
from-address	Word-like	`George <george@fumble-tech.com>`
in-address-book?	Categorical (`TRUE`, `FALSE`)	`TRUE`
non-spam-words	Text-like	`Ted, Mahout, User, lunch`
spam-words	Text-like	`available`
unknown-words	Continuous	`0`
message-length	Continuous	`31`

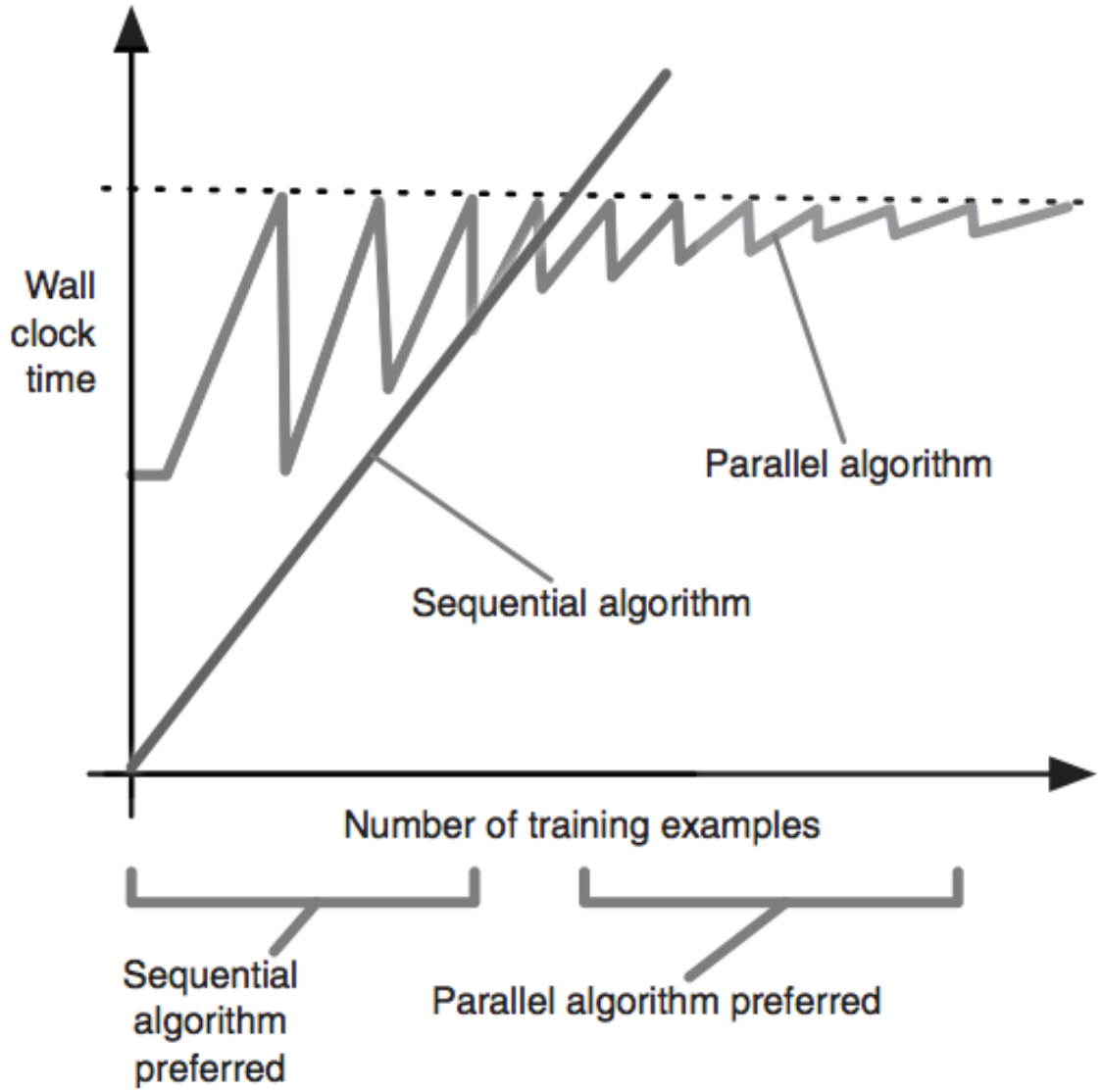
Supervised vs. Unsupervised Learning

- Classification algorithms are related to, but still quite different from, clustering algorithms such as the k-means algorithm described in previous notes.
  - Classification algorithms are a form of supervised learning, as opposed to unsupervised learning, which happens with clustering algorithm.
  - A supervised learning algorithm is one that's given examples that contain the desired value of a target variable.
  - Unsupervised algorithms aren't given the desired answer, but instead must find something plausible on their own.
- Supervised and unsupervised learning algorithms can often be usefully combined.
  - A clustering algorithm can be used to create feature that can then be used by a learning algorithm, or the output of several classifiers can be used as features by a clustering algorithm.
  - Clustering systems often build a model that can be used to categorize new data. This clustering system model works much like the model produced by a classification system.
  - The difference lies in what data was used to produce the model.
  - For classification, the training data set includes the target variables; for clustering, the training data set doesn't include target variables.

Workflow in a typical classification project

Stage	Step
1. Training the model	Define target variable. Collect historical data. Define predictor variables. Select a learning algorithm. Use the learning algorithm to train the model.
2. Evaluating the model	Run test data. Adjust the input (use different predictor variables, different algorithms, or both).
3. Using the model in production	Input new examples to estimate unknown target values. Retrain the model as needed.

Comparing two types of Mahout Scalable algorithms



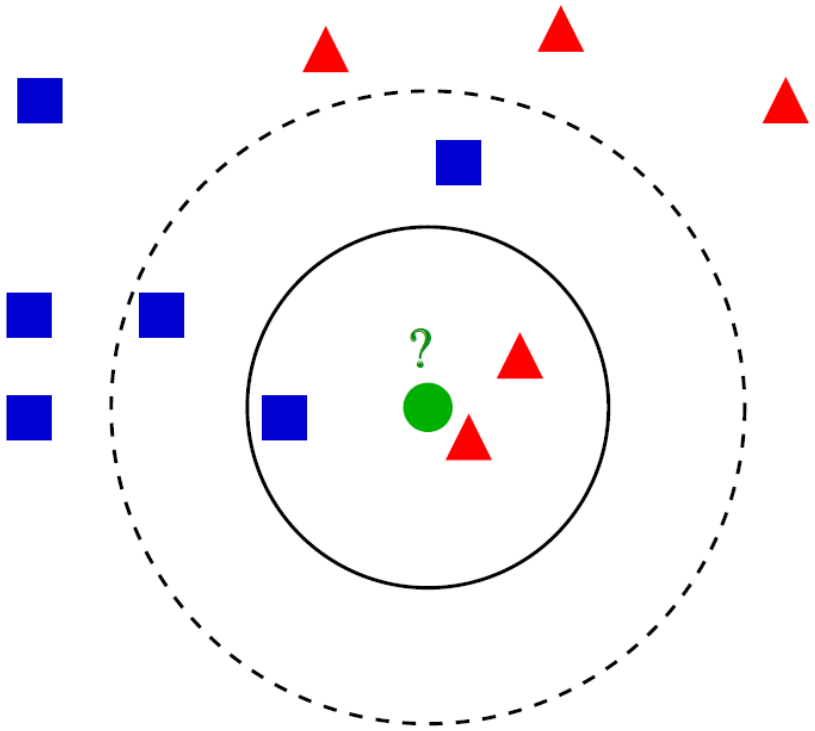
Choose algorithm via Mahout

Size of data set	Mahout algorithm	Execution model	Characteristics
Small to medium (less than tens of millions of training examples)	Stochastic gradient descent (SGD) family: `OnlineLogisticRegression`, `CrossFoldLearner`, `AdaptiveLogisticRegression`	Sequential, online, incremental	Uses all types of predictor variables; sleek and efficient over the appropriate data range (up to millions of training examples)
	Support vector machine (SVM)	Sequential	Experimental still; sleek and efficient over the appropriate data range
Medium to large (millions to hundreds of millions of training examples)	Naive Bayes	Parallel	Strongly prefers text-like data; medium to high overhead for training; effective and useful for data sets too large for SGD or SVM
	Complementary naive Bayes	Parallel	Somewhat more expensive to train than naive Bayes; effective and useful for data sets too large for SGD, but has similar limitations to naive Bayes
Small to medium (less than tens of millions of training examples)	Random forests	Parallel	Uses all types of predictor variables; high overhead for training; not widely used (yet); costly but offers complex and interesting classifications, handles nonlinear and conditional relationships in data better than other techniques

Classification
(k Nearest Neighbor)

k Nearest Neighbor

- kNN categorizes objects based on the classes of their nearest neighbors in the dataset
- kNN predictions assume that objects near each other are similar.
- **Distance metrics**, such as Euclidean, city block, cosine, and Chebychev, are used to find the nearest neighbor.



Example of k-NN classification

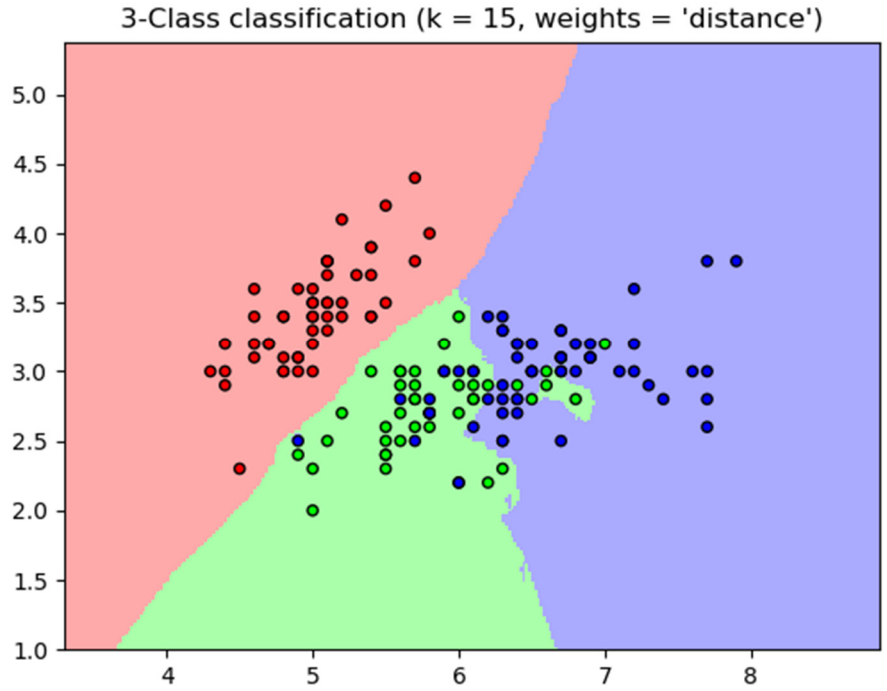
The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles

- If k = 3 (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle
- If k = 5 (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle)

k Nearest Neighbor

An object is classified by a majority vote of its neighbors

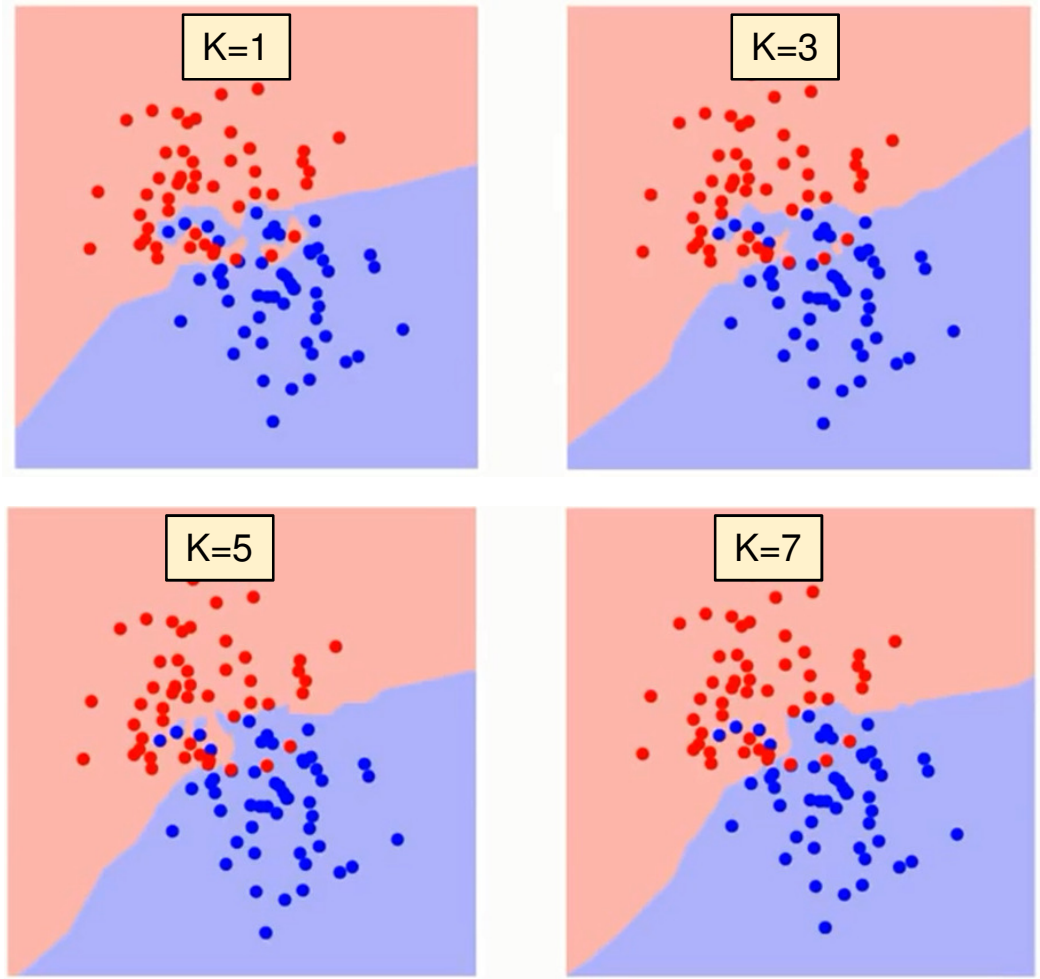
- **KNN algorithm is one of the simplest classification algorithm**
  Despite its simplicity, nearest neighbors has been successful in a large number of classification and regression problems, including handwritten digits or satellite image scenes
- **Often used in classification**
  Being a non-parametric method, it is often successful in classification situations where the decision boundary is very irregular
- **Classification is computed from a simple majority vote of the nearest neighbors of each point**
  In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors
- **K is constant specified by user**
  In the classification phase, k is a user-defined constant. The best choice of k depends upon the data; generally, larger values of k reduce the effect of noise on the classification, but make boundaries between classes less distinct
- **KNN is computationally expensive**



https://goo.gl/bUpyQv

k Nearest Neighbor

- How do we choose the factor K?



- If you watch carefully, you can see that the boundary becomes smoother with increasing value of K. With K increasing to infinity it finally becomes all blue or all red depending on the total majority

- At times, choosing K turns out to be a challenge while performing KNN modeling

https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/

Classification
(Logistic Regression)

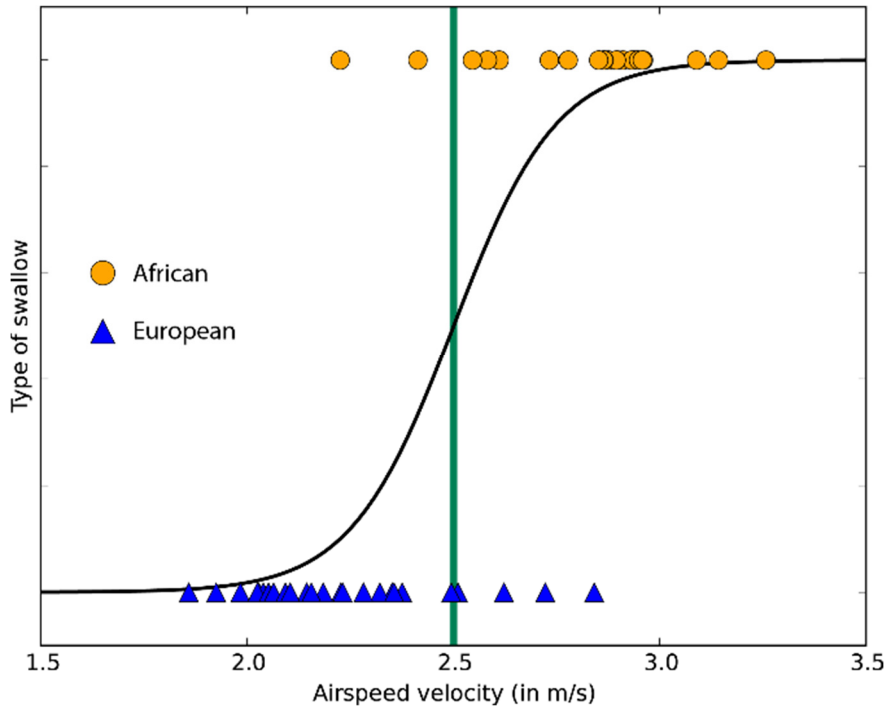
Logistic Regression

What is Logistic Regression?

- Logistic regression is a simple classification algorithm that can predict the probability of a binary response belonging to one class or the other

- Logistic Regression is commonly used to estimate the probability that an instance belongs to a particular class. (e.g., what is the probability that this email is spam?). This makes it a binary classifier
  - If the estimated probability is greater than 50%, then the model predicts that the instance belongs to that class (called the positive class, labeled "1"),
  - or else it predicts that it does not (i.e., it belongs to the negative class, labeled "0")

- Because of its simplicity, logistic regression is commonly used as a starting point for binary classification problems. Logistic regression can be used as a baseline for evaluating more complex classification methods

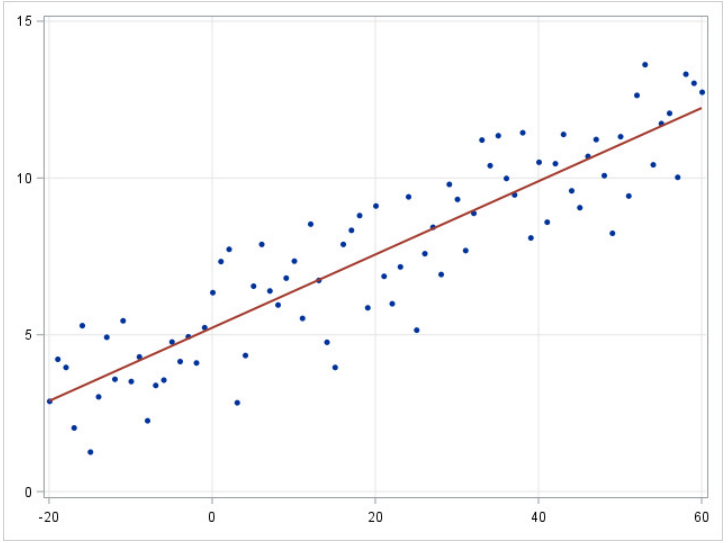


A logistic regression to two-class data with just one feature -- the class boundary is the point at which the logistic curve is just as close to both classes

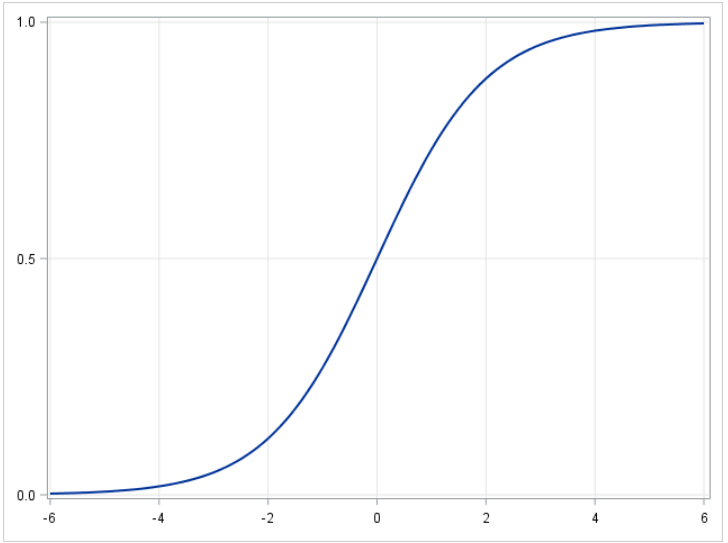
https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-choice

Logistic Regression

- Logistic regression is a classification rather than regression algorithm
  - Don't be confused by its name
  - A powerful tool for two-class and multiclass classification
- It is fast and simple
  - The fact that it uses an "S-shape" curve instead of a straight line makes it a natural fit for dividing data into groups
- The probabilities describing the possible outcomes of a single trial are modeled using a logistic function
- It predicts the probability whose output values lies between 0 and 1 (as expected)



Linear Regression



Logistic Regression

https://www.youtube.com/watch?v=gNhogKJ_q7U

https://blogs.sas.com/content/subconsciousmusings/2017/04/12/machine-learning-algorithm-use/

Logistic Regression

Let's recall the Linear regression hypothesis: $h_\theta(x)$

- Univariate linear regression (simple linear regression)

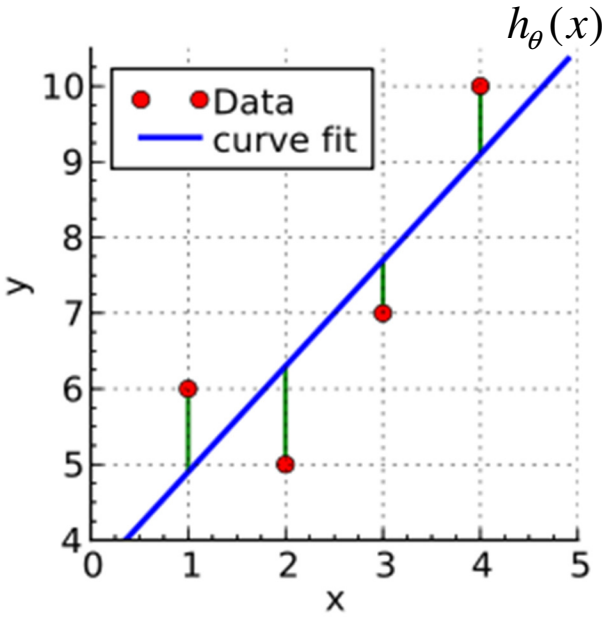
$$h_\theta(x) = \theta_0 + \theta_1 x$$

- Multivariate linear regression

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_0 x_0 + \theta_1 x_1 + \ldots \theta_n x_n = \sum_{i=0}^{n} \theta_i x_i$$

Vectorized ⇩

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^{\mathrm{T}} \mathbf{x}$$



- Linear regression is not a good approach for classification problem
  - Linear regression could be used to classify the dataset, but often not a good idea to do that
  - Linear regression's output value is in a continues range. So linear regression will predict values outside the acceptable range (e.g., predicting probabilities outside the range 0 to 1)

https://www.youtube.com/watch?v=-la3q9d7AKQ http://www.saedsayad.com/logistic_regression.htm

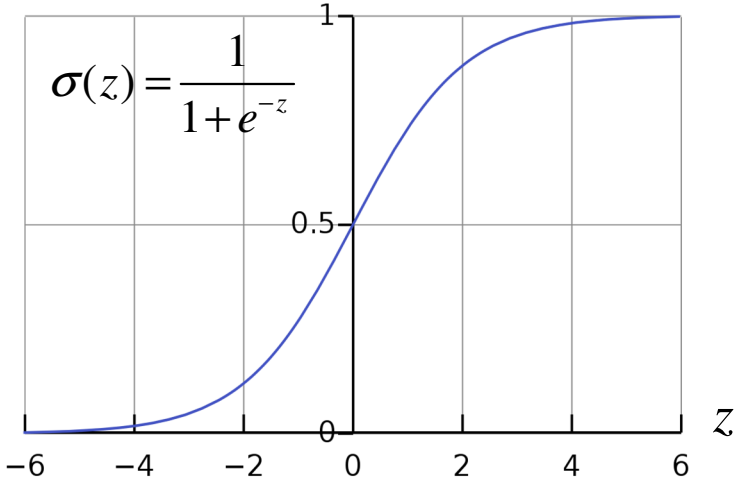
Logistics Regression

- Use a different hypothesis to predict the probability that a given example belongs to the "1" class vs. the probability that it belongs to the "0" class -- binary classification
  - 1 (positive): e.g., spam email
  - 0 (negative): e.g., regular (non-spam) email
- Use sigmoid function for our hypothesis so that the output is bounded (0, 1)

$$0 \le h_\theta(x) \le 1$$

- Use sigmoid function to "transform" the linear regression equation

$$\left.\begin{aligned} h_{\boldsymbol{\theta}}(\mathbf{x}) &= \boldsymbol{\theta}^{\mathrm{T}}\mathbf{x} \\ \sigma(z) &= \frac{1}{1+e^{-z}} \end{aligned}\right\} \Rightarrow h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma\left(h_{\boldsymbol{\theta}}(\mathbf{x})\right) = \frac{1}{1+e^{-h_{\boldsymbol{\theta}}(\mathbf{x})}} = \frac{1}{1+e^{-\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}}}$$



- A logistic function or logistic curve is a common "S" shape (sigmoid curve)
- The benefit of this curve is that the input values can range from -∞ to ∞ whilst the output ranges from 0 to 1, exactly the range for probability values
- Its common use is as a transfer function

http://www.saedsayad.com/logistic_regression.htm

Logistics Regression

Logistics regression hypothesis $h_{\theta}(\mathbf{x})$

- Treat output of $h_{\theta}(\mathbf{x})$ as **estimated probability** that (y = 1 (i.e., positive) on input **x**
  - e.g., if we use feature x of an email to predict if the email is spam and suppose the hypothesis outputs 0.7, then we might predict that this email, has a 70% chance, or a 0.7 chance of being spam

- Write this out in math, interpret the hypothesis output as $h_{\theta}(\mathbf{x}) = P(y = 1 \mid \mathbf{x})$

$$P(y = 1 \mid \mathbf{x}) = h_{\theta}(\mathbf{x}) = \sigma\left(\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}}}$$

e.g., $P(y$ = spam $\mid \mathbf{x})$ is the probability of "incoming email (represented by) **x** is spam"

$$P(y = 0 \mid \mathbf{x}) = 1 - P(y = 1 \mid \mathbf{x}) = 1 - h_{\theta}(\mathbf{x}) = 1 - \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}}}$$

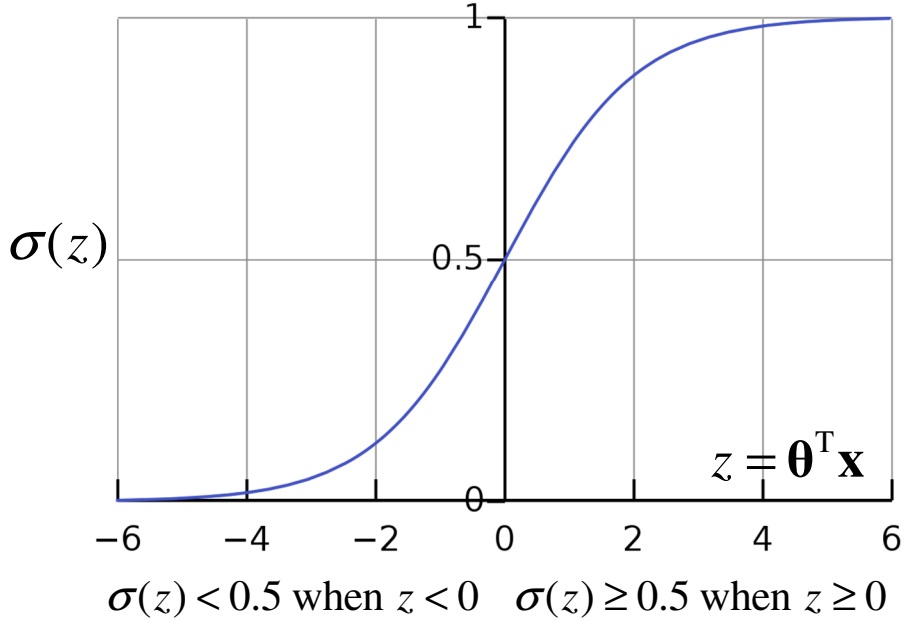
e.g., $P(y$ = normal email $\mid \mathbf{x})$ is the probability of "incoming email (represented by) **x** is normal (not spam)"

https://www.youtube.com/watch?v=t1IT5hZfS48

Logistics Regression

Use the logistics regression model to predict

- Once the Logistic Regression model has estimated the **probability** $\hat{P} = h_{\boldsymbol{\theta}}(\mathbf{x})$ that an instance **x** belongs to the positive class, it can easily make its prediction $\hat{y}$



$\sigma(z) < 0.5$ when $z < 0$ $\sigma(z) \geq 0.5$ when $z \geq 0$

so a logistic regression model predicts 1 if $\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}$ is positive, and 0 if it is negative

$$\hat{P} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}}}, \text{ let } z = \boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}$$

$$\hat{y} = \begin{cases} 0, \text{ if } \hat{P} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \dfrac{1}{1+e^{-\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}}} < 0.5 \\ 1, \text{ if } \hat{P} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \dfrac{1}{1+e^{-\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}}} \geq 0.5 \end{cases}$$

e.g., classify the email as a normal one if $\hat{P} < 0.5$

e.g., classify the email as a spam one if $\hat{P} \geq 0.5$

https://www.youtube.com/watch?v=F_VG4LNjZZw

Logistics Regression

$$\hat{p}^{(i)} = h_\theta\left(x^{(i)}\right) = \frac{1}{1+e^{-\theta^{\mathrm{T}} x^{(i)}}}$$

How to find the value of theta for fitting the model?

- Our goal is to search for a value of **θ** so that model estimates high probabilities for positive instances (i.e., instances where y = 1) and low probabilities for negative instances (i.e., instances where y = 0).
- Instead of finding the best fitting line by minimizing the squared residuals as in ordinary least squares, logistic regression model uses a different approach with logistic -- Maximum Likelihood Estimation
- **Logistic regression cost function (log loss)**

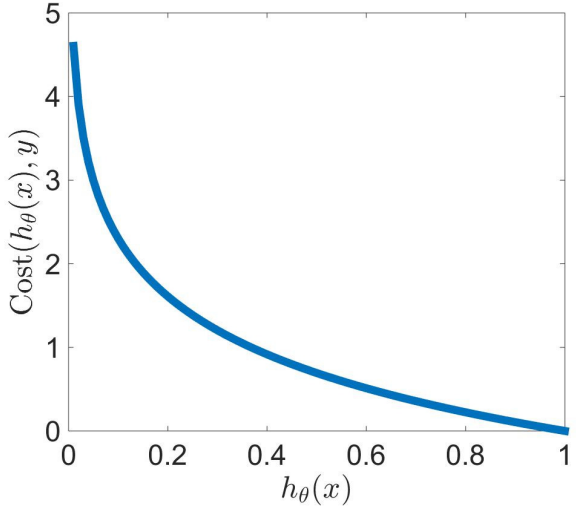
$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right)+\left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right)$$

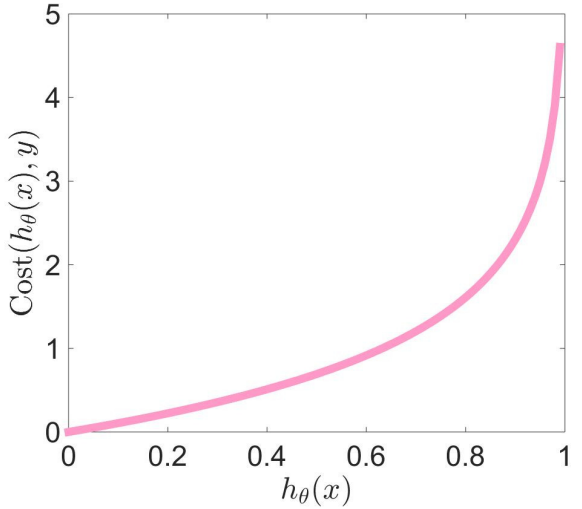
Convex function

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\mathrm{Cost}\left(h_\theta(x^{(i)}), y^{(i)}\right)$$

$$\mathrm{Cost}\left(h_\theta(x), y\right) = \begin{cases} -\log\left(h_\theta(x)\right), & \text{if } y = 1 \\ -\log\left(1-h_\theta(x)\right), & \text{if } y = 0 \end{cases}$$

$$\mathrm{Cost}\left(h_\theta(x), y\right) = -y\log\left(h_\theta(x)\right) - (1-y)\log\left(1-h_\theta(x)\right)$$





https://www.youtube.com/watch?v=TTdcc21Ko9A

Logistics Regression

$$\hat{p}^{(i)} = h_\theta\left(x^{(i)}\right) = \frac{1}{1+e^{-\theta^{\mathrm{T}} x^{(i)}}}$$

How to find the value of theta for fitting the model?

- We now have a cost function that measures how well a given hypothesis $h_{\boldsymbol{\theta}}(\mathbf{x})$ fits out training data
- Learn to classify training data by minimizing $J(\boldsymbol{\theta})$ to find the best choice of **θ**
- As the cost function is convex, Gradient Descent can be used to find the global minimum -- if the learning rate is not too large and you wait long enough

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right)+\left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right)$$

Minimize the cost function $J(\boldsymbol{\theta})$ to find the best choice of **θ**

Partial derivative of the cost function $J(\boldsymbol{\theta})$ with respect to θ_j is:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)x_j^{(i)}\right)$$

Gradient Descent for logistic regression

To minimize the cost function $J(\boldsymbol{\theta})$

Repeat until convergence {

$$\theta_{j+1} := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)x_j^{(i)}\right)$$

}

Simultaneously update θ_k for every $k = 0,1,\ldots,n$ at j^{th} iteration for $j = 1, 2, \ldots$

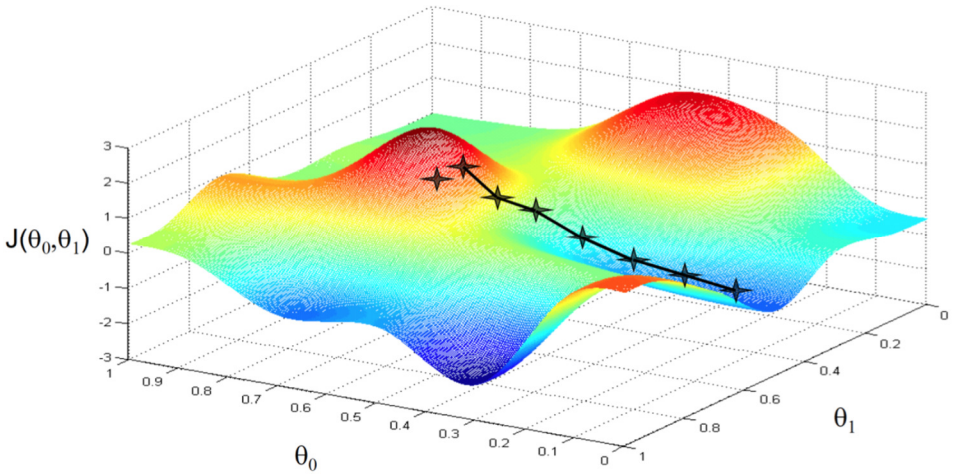
https://www.youtube.com/watch?v=TTdcc21Ko9A

Gradient Descent

Gradient Descent

An iterative algorithm to find a minimum of a cost function

- A very generic optimization algorithm capable of finding optimal solutions to a wide range of problems
- The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function



The goal of gradient descent is to start on a random point on this error surface, and find out the best parameters (θ_0, θ_1) which yields the minimum value of the cost function. In other words, the best parameters (θ_0, θ_1) corresponds to the lowest point on the error surface

Gradient Descent Algorithm

To minimize the cost function

1. Initialize the parameters with some random values
2. Keep changing these parameters iteratively in such a way it minimizes the cost function $J(\boldsymbol{\theta})$

Repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

}

Simultaneously update θ_0 and θ_1

Old position

New position

Learning rate

Gradient of function J for θ_1

Gradient Descent

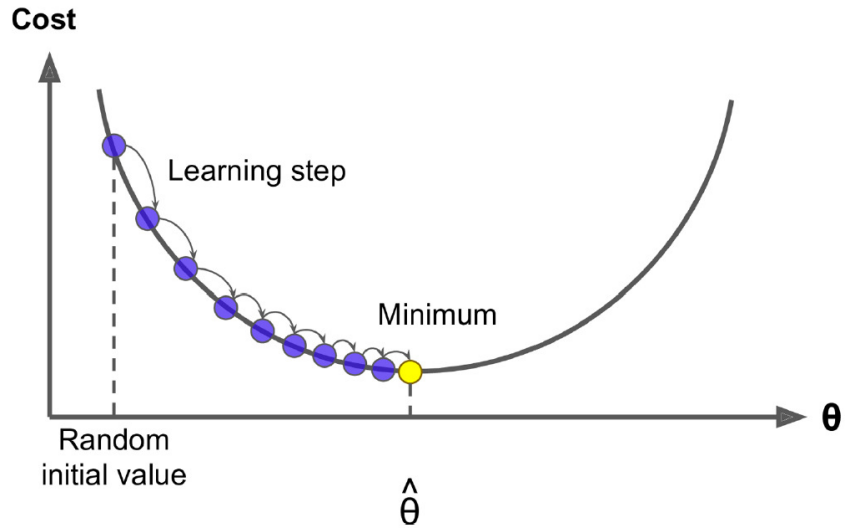
- There are three variants of gradient descent, which differ in how much data we use to compute the gradient of the objective function

- Depending on the amount of data, we make a trade-off between the accuracy of the parameter update and the time it takes to perform an update

- **Batch gradient descent**
  - Use all examples in each iteration

- **Stochastic gradient descent**
  - Use 1 example in each iteration

- **Mini-batch gradient descent**
  - Use b examples in each iteration, (b = mini-batch size, e.g., b = 10)



http://ruder.io/optimizing-gradient-descent/

Gradient Descent -- Batch Gradient Descent

- The traditional Batch Gradient Descent computes the gradient of the function w.r.t. to the parameters θ for the entire training dataset

Use all examples in each iteration
Batch Gradient Descent uses the whole batch of training data at every step

$$\theta := \theta - \alpha \cdot \nabla_\theta J(\theta)$$

Gradient of the cost function

- **Batch Gradient Descent is NOT suitable for huge datasets**
  - Need to calculate the gradients for the whole dataset to perform just one update, batch gradient descent can be very slow and is intractable for datasets that do not fit in memory
  - Batch gradient descent does not allow us to update our model online, i.e., with new examples on-the-fly
- **How large of an update determined by the learning rate**
- **Batch gradient descent is guaranteed**
  - To converge to the global minimum for convex error surfaces
  - To a local minimum for non-convex surfaces

http://ruder.io/optimizing-gradient-descent/

Gradient Descent -- Stochastic Gradient Descent

- Instead of going through all examples, Stochastic Gradient Descent (SGD) performs the parameters update on each example $(x^{(i)}, y^{(i)})$

Use 1 example in each iteration
It uses only one training example in every iteration to compute the gradient of cost function

$$\theta := \theta - \alpha \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

Gradient of the cost function

- **Stochastic Gradient Descent is much faster and suitable for huge datasets**
  - Stochastic Gradient Descent just picks a random instance in the training set at every step and computes the gradient is based only on that single instance. It is therefore usually much faster and can also be used to learn online
- **The randomness is good to escape from local optima**
  - Now due to these frequent updates ,parameters updates have high variance and causes the Loss function to fluctuate to different intensities. This is actually a good thing because it helps us discover new and possibly better local minima , whereas Standard Gradient Descent will only converge to the minimum of the basin
- **Stochastic Gradient Descent can never settle at the minimum**
  - But the problem with SGD is that due to the frequent updates and fluctuations it ultimately complicates the convergence to the exact minimum and will keep overshooting due to the frequent fluctuations. It means that the algorithm can never settle at the minimum. One solution to this dilemma is to gradually reduce the learning rate

Gradient Descent -- Mini-batch Gradient Descent

- Mini-batch gradient descent performs an update for every mini-batch of b training examples

Use b examples in each iteration, i.e., b = mini-batch size

Common mini-batch sizes range between 50 and 256, but can vary for different applications

$$\theta := \theta - \alpha \cdot \nabla_\theta J(\theta; x^{(i:i+b)}; y^{(i:i+b)})$$

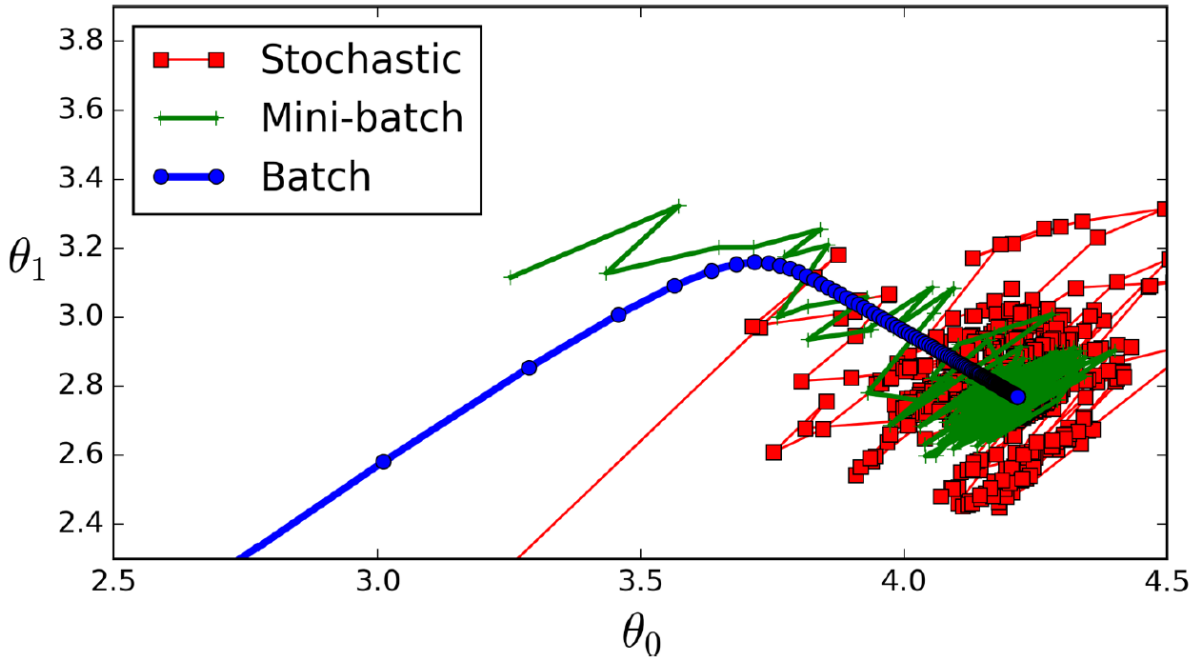
Gradient of the cost function

- **Mini-batch gradient descent finally takes the best of both Batch GD and Stochastic GD**
  - At each step, instead of computing the gradients based on the full training set (as in Batch GD) or based on just one instance (as in Stochastic GD), Mini-Gradient Descent computes the gradients on small random sets of instances called mini-batches. Mini-batch gradient descent is a trade-off between stochastic gradient descent and batch gradient descent
  - It reduces the variance of the parameter updates, which can lead to more stable convergence (than Stochastic GD)
  - It can make use of highly optimized matrix optimizations common to state-of-the-art deep learning libraries that make computing the gradient w.r.t. a mini-batch very efficient
  - Mini-batch gradient descent is faster than Batch Gradient Descent because it goes through less examples
  - If you choose reasonable value of mini-batch size and if you use a good vectorized implementation, sometimes it can be faster than Stochastic gradient descent

http://ruder.io/optimizing-gradient-descent/

Gradient Descent -- Variants Comparison

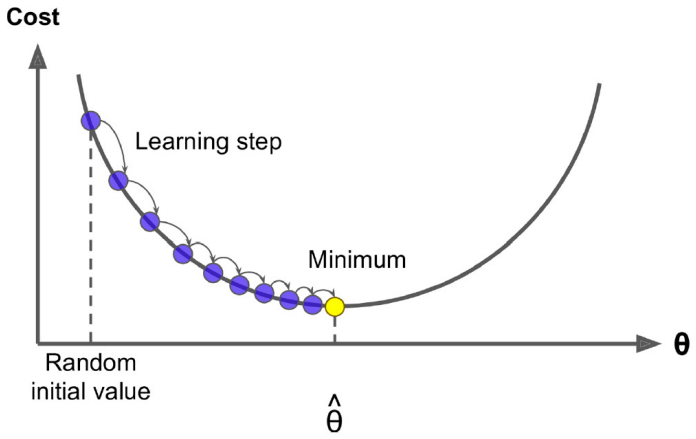
- **Comparison of the 3 Gradient Descent variants getting to the local minima**
  - They all end up near the minimum, but Batch GD's path actually stops at the minimum, while both Stochastic GD and Mini-batch GD continue to walk around
  - However, don't forget that Batch GD takes a lot of time to take each step, and Stochastic GD and Mini-batch GD would also reach the minimum if you used a good learning schedule



A. Géron, Hands-On Machine Learning with Scikit-Learn & TensorFlow, O'Reilly, 2017

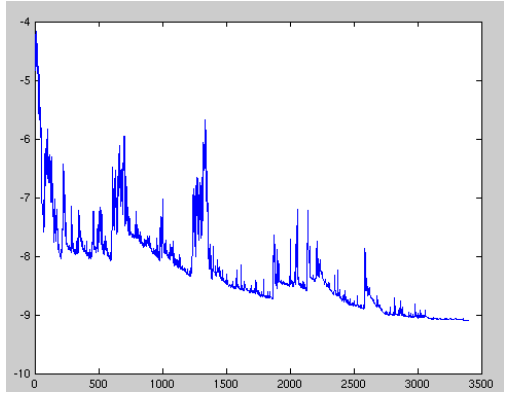
Gradient Descent

- Batch Gradient Descent
  - Use all examples in each iteration
- Stochastic Gradient Descent (SGD)
  - Use one example in each iteration
- Mini-batch Gradient Descent
  - Use b examples in each iteration (i.e., b = mini-batch size, e.g., b = 10)



Stochastic gradient descent (SGD)

- Its direction towards the minimum is very noisy compared to mini-batch
- Due to its stochastic (i.e., random) nature, this algorithm is much less regular than Batch Gradient Descent: instead of gently decreasing until it reaches the minimum, the cost function will bounce up and down, decreasing only on average
- Over time it will end up very close to the minimum, but once it gets there it will continue to bounce around, never settling down. So once the algorithm stops, the final parameter values are good, but not optimal

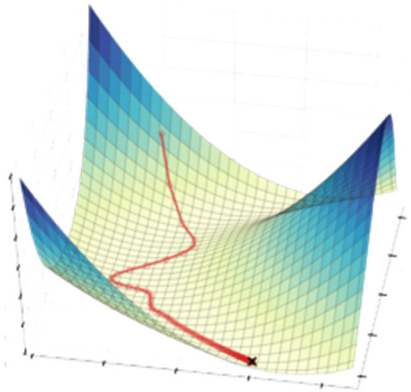


http://ruder.io/optimizing-gradient-descent/

Gradient Descent – Challenges

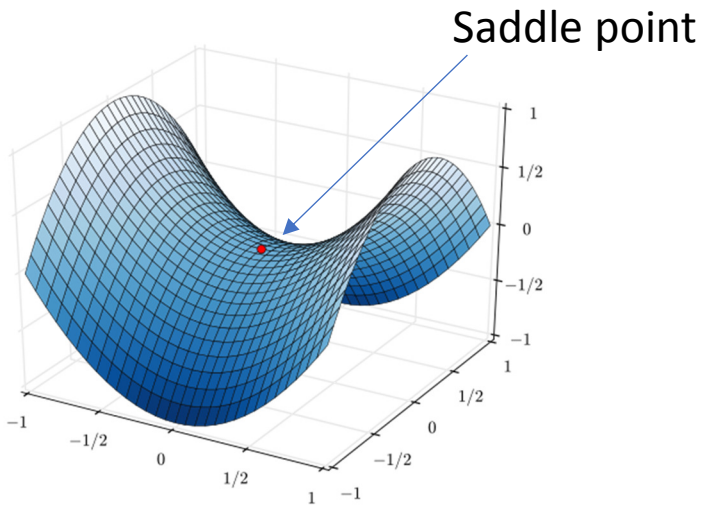
Choosing a proper learning rate can be difficult

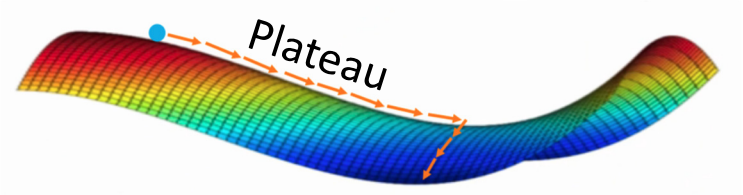
- A learning rate that is too small leads to painfully slow convergence, while a learning rate that is too large can hinder convergence and cause the loss function to fluctuate around the minimum or even to diverge
- Learning rate schedules try to adjust the learning rate during training by e.g. annealing, i.e. reducing the learning rate according to a pre-defined schedule or when the change in objective between epochs falls below a threshold. These schedules and thresholds, however, have to be defined in advance and are thus unable to adapt to a dataset‘s characteristics
- Additionally, the same learning rate applies to all parameter updates. If our data is sparse and our features have very different frequencies, we might not want to update all of them to the same extent, but perform a larger update for rarely occurring features



Gradient Descent might get trapped in local minima or saddle points

- Another key challenge of minimizing highly non-convex error functions common for neural networks is avoiding getting trapped in their numerous suboptimal local minima. Actually, the difficulty arises in fact not from local minima but from saddle points, i.e. points where one dimension slopes up and another slopes down. These saddle points are usually surrounded by a plateau of the same error, which makes it notoriously hard for SGD to escape, as the gradient is close to zero in all dimensions



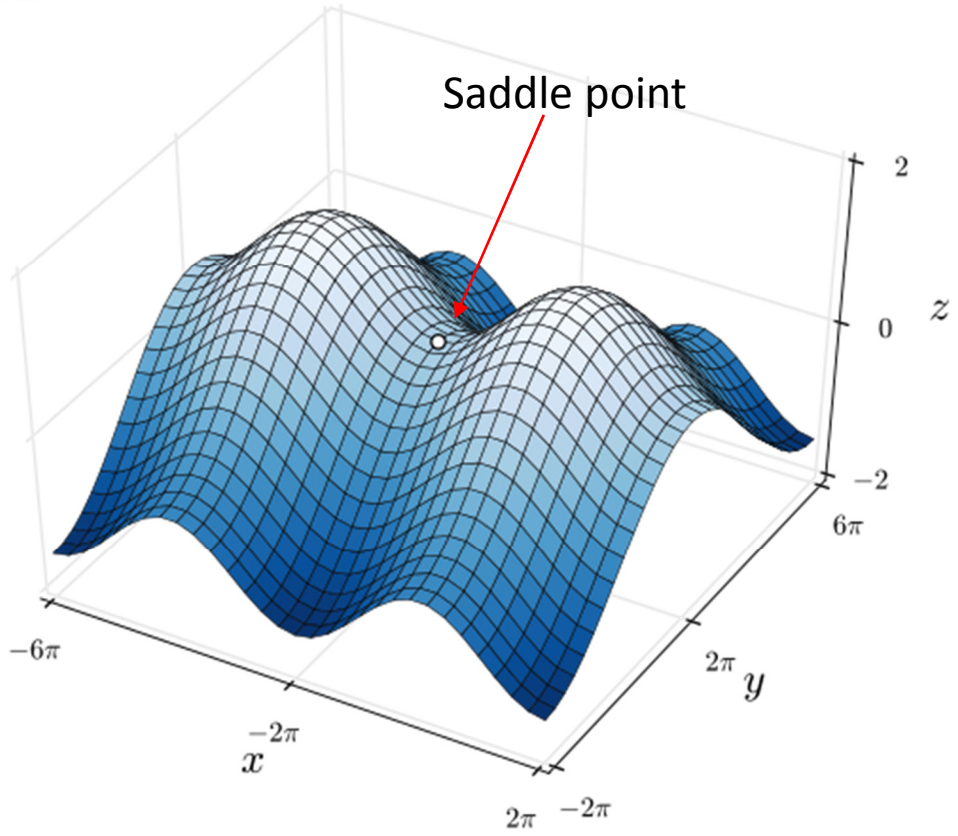


http://dsdeepdive.blogspot.com/2016/03/optimizations-of-gradient-descent.html

Gradient Descent – Saddle Point

- A point where one dimension slopes up while another slopes down, usually surrounded by a plateau of about equal error
- Regardless of the direction gradient descent goes, it is difficult to escape because the surrounds gradients are usually around zero

Gradient Descent may get stuck in saddle points rather than local minima



Gradient Descent

Finding a good Learning Rate can be tricky

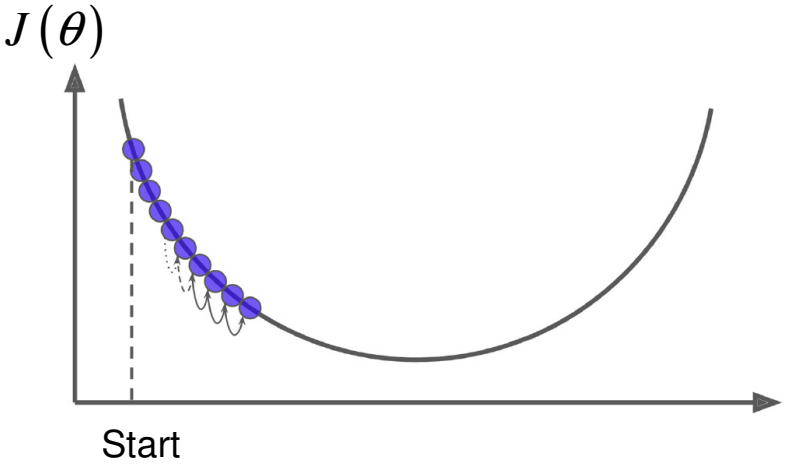
- An important parameter in Gradient Descent is the size of the steps, determined by the learning rate hyper-parameter

$$\theta := \theta - \alpha \frac{\partial}{\partial \theta} J(\theta)$$

Learning rate (step size)

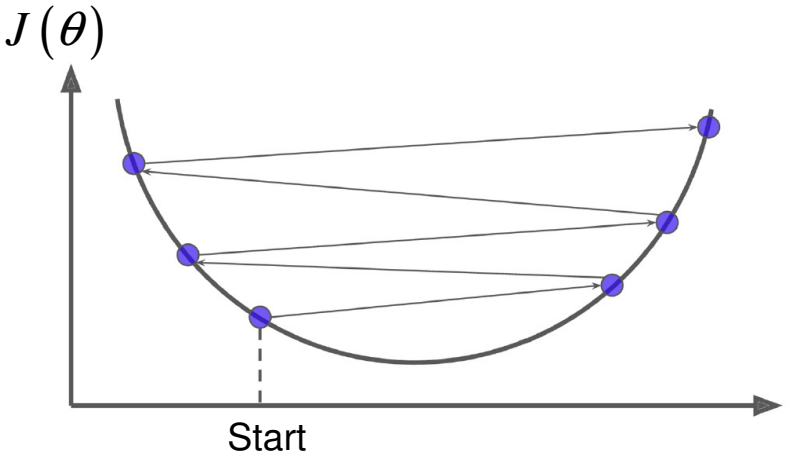
If learning rate is too small: slow convergence

gradient descent would take long time to converge and can be very slow



If learning rate is too large: gradient may not decrease on every iteration; may not converge

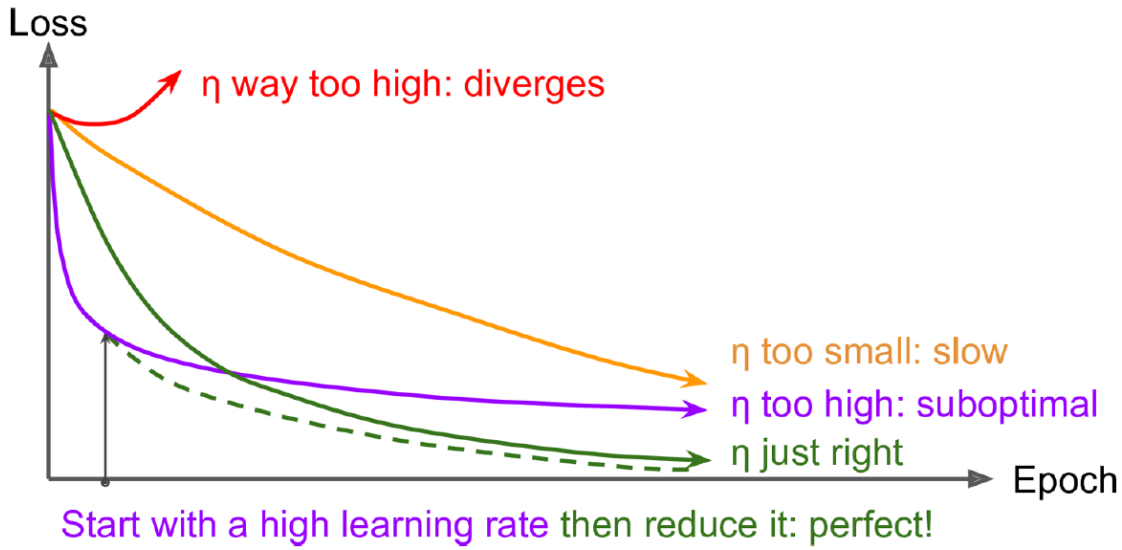
gradient descent can overshoot the minimum. You might jump across the valley and end up on the other side, possibly even higher up than you were before. So the algorithm may fail to converge, or even diverge



Gradient Descent

- Learning Rate Scheduling (Learning Rate Annealing)
  - Slowly reducing the learning rate over time might speed up your learning gradient descent optimization

The cost function changes over time with different learning rates



- With low learning rates the improvements will be linear
- With high learning rates they will start to look more exponential
- Higher learning rates will decay the loss faster, but they get stuck at worse values of loss (purple line). This is because there is too much "energy" in the optimization and the parameters are bouncing around chaotically, unable to settle in a nice spot in the optimization landscape

- Reduce the learning rate over time
  - Usually using constant learning rate to train the model. However, you can do better than a constant learning rate: if you start with a high learning rate and then reduce it once it stops making fast progress, you can reach a good solution faster than with the optimal constant learning rate
  - Since AdaGrad, RMSProp, and Adam optimization automatically reduce the learning rate during training, it is not necessary to add an extra learning schedule. For other optimization algorithms, using exponential decay or performance scheduling can considerably speed up convergence

A. Géron, Hands-On Machine Learning with Scikit-Learn & TensorFlow, O'Reilly, 2017

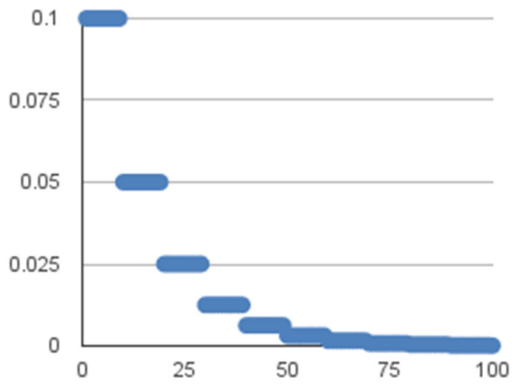
http://cs231n.github.io/neural-networks-3/

Gradient Descent

- Learning Rate Scheduling (Learning rate annealing)
  - Common strategies to decay the learning rate during training
  - Knowing when to decay the learning rate can be tricky
    - Decay it slowly and you'll be wasting computation bouncing around chaotically with little improvement for a long time
    - But decay it too aggressively and the system will cool too quickly, unable to reach the best position it can
- There are three common types of implementing the learning rate decay

Step decay

Reduce the learning rate by some factor every few epochs. Typical values might be reducing the learning rate by a half every 5 epochs, or by 0.1 every 20 epochs. These numbers depend heavily on the type of problem and the model. One heuristic you may see in practice is to watch the validation error while training with a fixed learning rate, and reduce the learning rate by a constant (e.g. 0.5) whenever the validation error stops improving

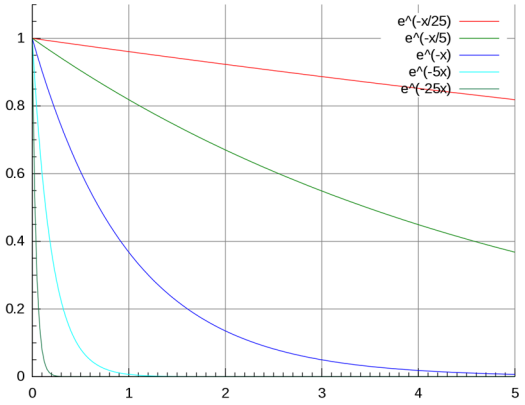


Exponential decay

It has the mathematical form

$$\alpha = \alpha_0 e^{-kt}$$

where xx (initial leaning rate) , *k* (decay rate) are hyper-parameters and *t* is the iteration number (but you can also use units of epochs)

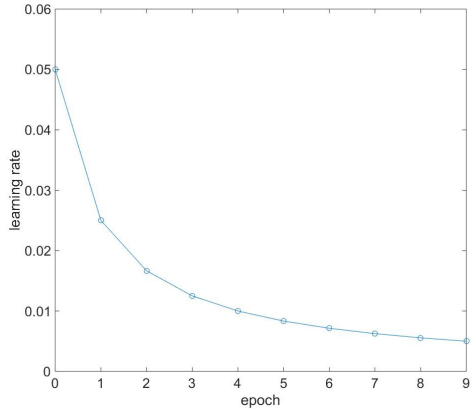


1/t decay

It has the mathematical form

$$\alpha = \frac{\alpha_0}{1+kt}$$

where xx (initial leaning rate) , *k* (decay rate) are hyper-parameters and *t* is the iteration number



http://cs231n.github.io/neural-networks-3/
https://www.youtube.com/watch?v=QzulmoOg2JE

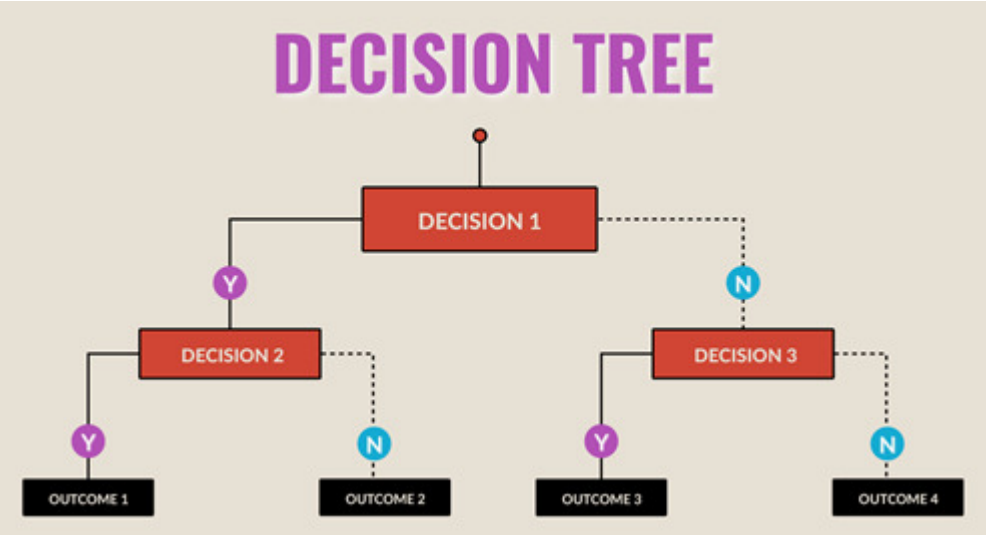
Reading

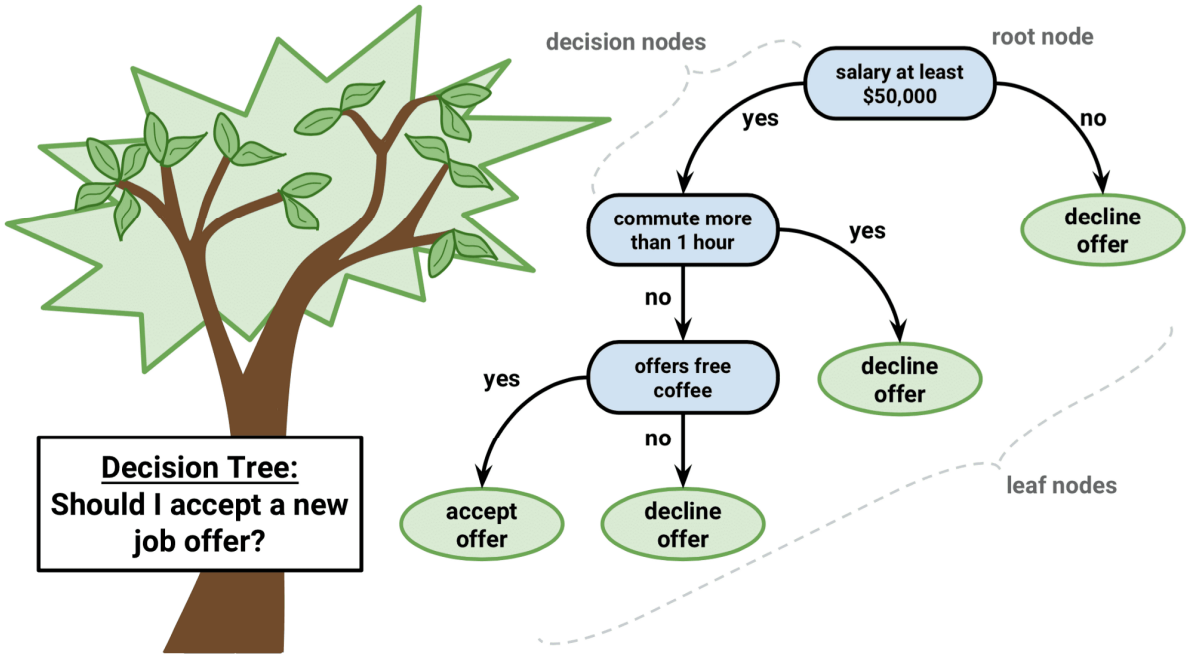
- A good video about Adam Optimization Algorithm
  - https://www.youtube.com/watch?v=g25U4HSZKmQ
- ADAM: a method for Stochastic Optimization
  - https://arxiv.org/pdf/1412.6980.pdf

Classification
(Decision Tree Induction)

Outline

- Classification
  - Decision Tree Induction





Decision Tree

Example: using decision tree to decide whether or not to offer someone a loan

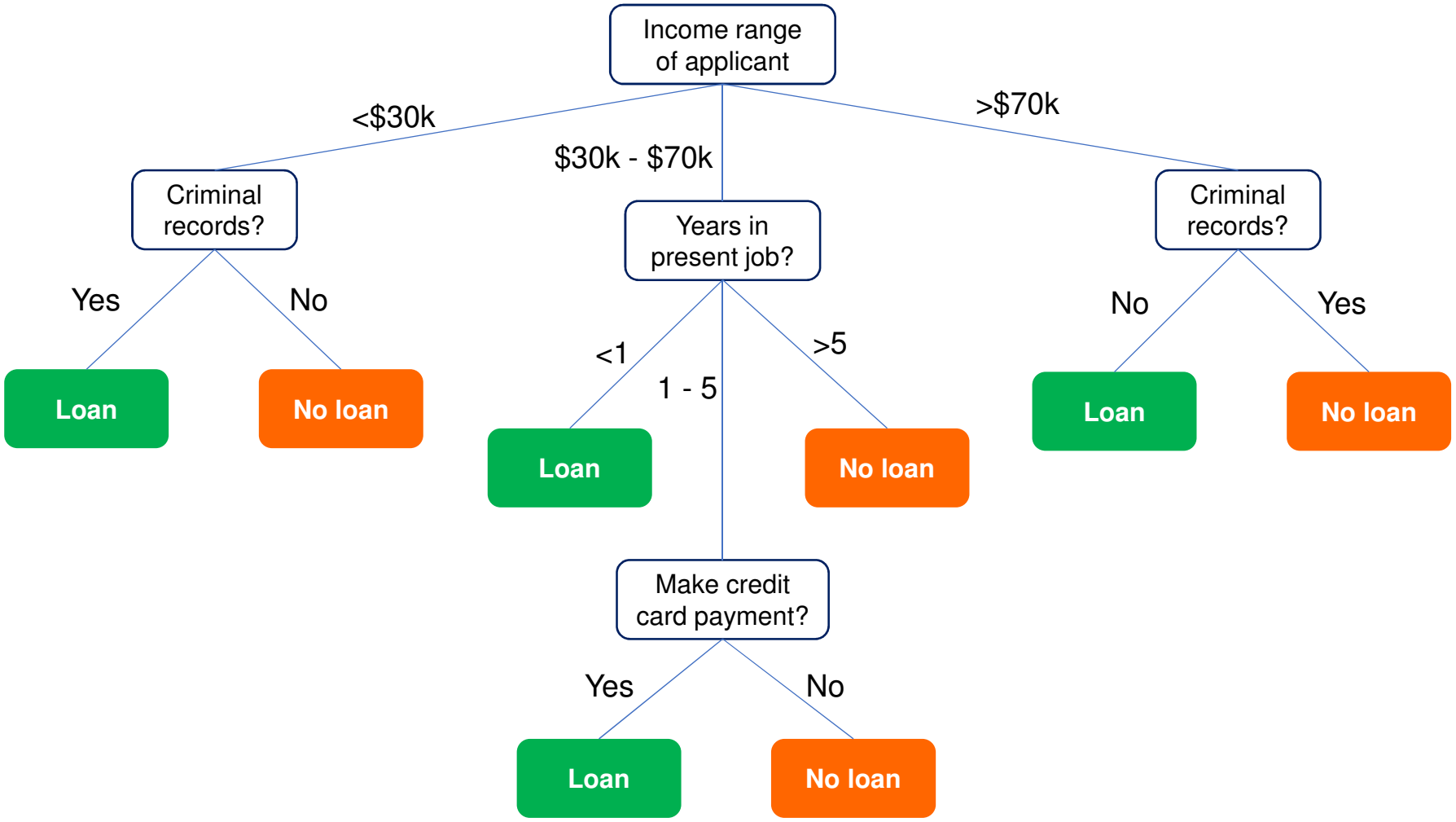


Image credit: https://towardsdatascience.com/what-is-a-decision-tree-22975f00f3e1

Decision Tree

Decision Tree construction

Strategy: top-down, recursive, and divide and conquer

1. Select attribute for root node
2. Split instances into subsets
3. Repeat recursively for each branch, using only instances that reach the branch
4. Stop if all instances have the same class

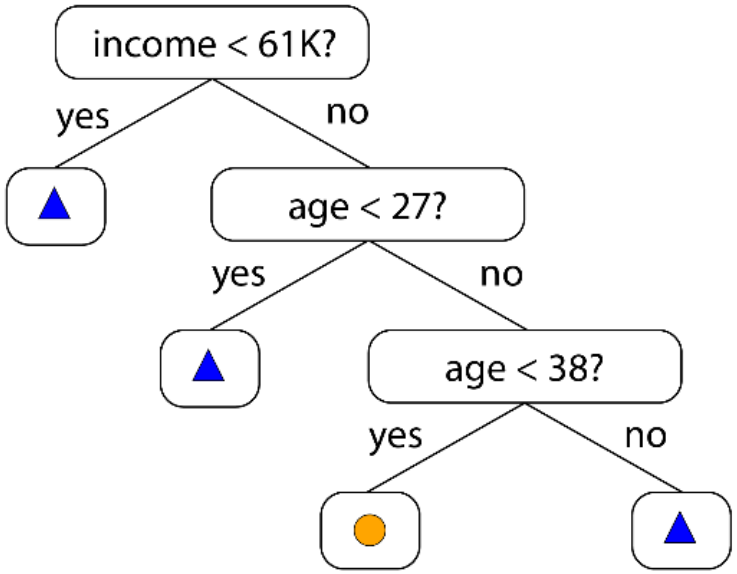
Recursively partitioning the data into subsets that contain instances with similar values (homogenous)

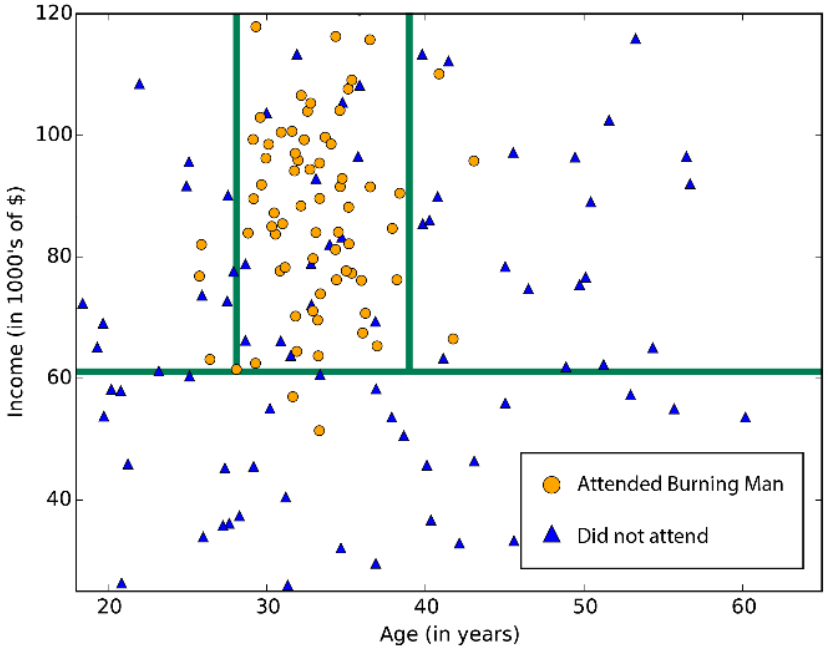
- Recursively partitions the training set until each partition consists entirely or dominantly of examples from one class
- Each non-leaf node of the tree contains a split point that is a test on one or more attributes and determines how the data is partitioned
- The tree is built by recursively partitioning the data and partitioning continues until each partition is either “pure” (all members belong to the same class) or sufficiently small (a parameter set by the user)

Decision Tree

Want subsets to be as *homogeneous* as possible

- Growing a tree involves deciding on **which features to choose** and **what conditions to use** for splitting, along with knowing when to stop
- All the features are considered and different split points are tried and tested using a cost function -- he split with the best cost (or lowest cost) is selected





https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-choice

Example: Predict if John will play tennis

- Training example: **9 yes** / **5 no**

Day	Outlook	Temp	Humidity	Windy	Play?
1	sunny	hot	high	false	No
2	sunny	hot	high	true	No
3	overcast	hot	high	false	Yes
4	rain	mild	high	false	Yes
5	rain	cool	normal	false	Yes
6	rain	cool	normal	true	No
7	overcast	cool	normal	true	Yes
8	sunny	mild	high	false	No
9	sunny	cool	normal	false	Yes
10	rain	mild	normal	false	Yes
11	sunny	mild	normal	true	Yes
12	overcast	mild	high	true	Yes
13	overcast	hot	normal	false	Yes
14	rain	mild	high	true	No
15	**rain**	**mild**	**high**	**false**	**?**

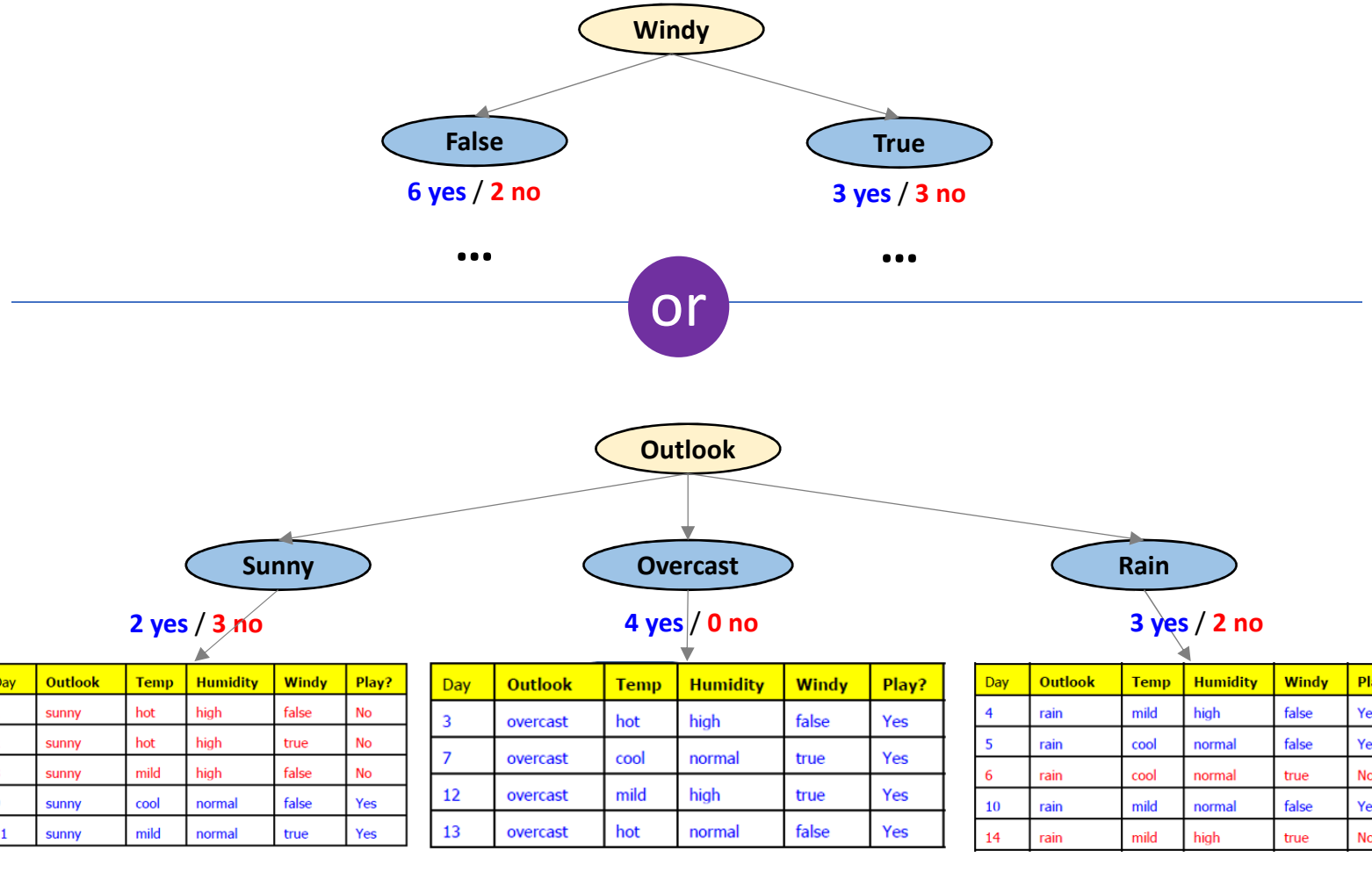
https://www.youtube.com/watch?v=eKD5gxPPeY0&t=48s

Decision Tree

- The key is to decide which attribute to split based on
- Need to pick the best attribute for splitting the training examples

- We seek smallest trees
- We'd like it happen ASAP
- Choose the attribute that produces the purest children nodes

Day	Outlook	Temp	Humidity	Windy	Play?
1	sunny	hot	high	false	No
2	sunny	hot	high	true	No
3	overcast	hot	high	false	Yes
4	rain	mild	high	false	Yes
5	rain	cool	normal	false	Yes
6	rain	cool	normal	true	No
7	overcast	cool	normal	true	Yes
8	sunny	mild	high	false	No
9	sunny	cool	normal	false	Yes
10	rain	mild	normal	false	Yes
11	sunny	mild	normal	true	Yes
12	overcast	mild	high	true	Yes
13	overcast	hot	normal	false	Yes
14	rain	mild	high	true	No
15	**rain**	**mild**	**high**	**false**	**?**



Day	Outlook	Temp	Humidity	Windy	Play?
1	sunny	hot	high	false	No
2	sunny	hot	high	true	No
8	sunny	mild	high	false	No
9	sunny	cool	normal	false	Yes
11	sunny	mild	normal	true	Yes

Day	Outlook	Temp	Humidity	Windy	Play?
3	overcast	hot	high	false	Yes
7	overcast	cool	normal	true	Yes
12	overcast	mild	high	true	Yes
13	overcast	hot	normal	false	Yes

Day	Outlook	Temp	Humidity	Windy	Play?
4	rain	mild	high	false	Yes
5	rain	cool	normal	false	Yes
6	rain	cool	normal	true	No
10	rain	mild	normal	false	Yes
14	rain	mild	high	true	No

http://www.saedsayad.com/decision_tree.htm

Decision Tree

To compare the different ways to split data in a node

- Need to measure "purity" of the split
  - Pure subset (**4 yes** / **0 no**) → completely certain
  - Impure subset (**3 yes** / **3 no**) → completely uncertain
- Split on all attributes are tested
  - Constructing a decision is all about finding the attribute that produces the **highest Information Gain** or **lowest Gini Index** (i.e., the most homogeneous branches (subtrees))
- Measures that can be used to capture the purity of split
  - Information Gain
  - Information Gain Ratio
  - Gini Index

Decision Tree

- More uncertainty, more entropy.

High Entropy
- High disorder
- Messy
- High randomness
- High uncertainty
- Low certainty

entropy is a measure of
disorder or **uncertainty**

Entropy increases

Low Entropy
- Well-ordered
- Neat
- Low randomness
- Low uncertainty
- High certainty

Decision Tree

Mathematical definition of entropy

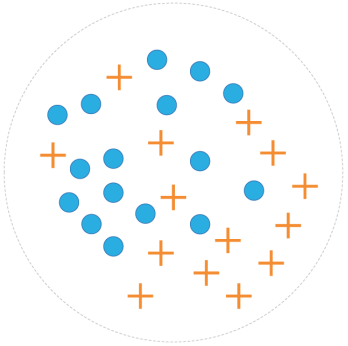
- The measure of information entropy associated with each possible data value is the negative logarithm of the probability mass function for the value

$$\text{Entropy} = -\sum_{i=1}^{k} p_i \log_2 p_i$$

the **probability** of getting the ith value when randomly electing one from the set containing k possible values, i.e., there are k classes, and pi is the probability an object from the ith class appears

fair coin toss

$$\text{Entropy} = -(0.5 \cdot \log_2 0.5 + 0.5 \cdot \log_2 0.5) = 1$$



entire population: 30

16/30 blue circles:

$$\log_2\left(\frac{16}{30}\right) = -0.9$$

14/30 orange crosses:

$$\log_2\left(\frac{14}{30}\right) = -1.1$$

$$\text{Entropy} = -\sum_{i=1}^{k} p_i \log_2 p_i = -\left(\frac{16}{30} \cdot (-0.9) + \frac{14}{30} \cdot (-1.1)\right) = 0.99$$

https://youtu.be/9r7FIXEAGvs

Decision Tree

- **Another example of calculating entropy**

$$\text{Entropy} = -\sum_{i=1}^{k} p_i \log_2 p_i$$

- Use entropy to measure the "purity" of the split

The entropy is 1 if the split is equally divided

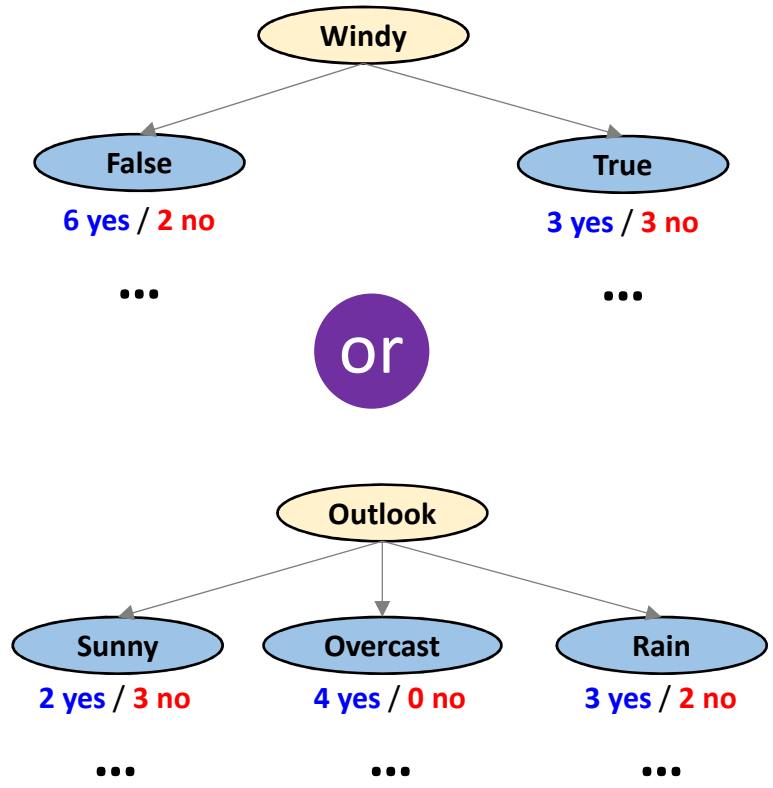
- Pure subset (**3 yes** / **3 no**) → completely uncertain (50%)

$$\text{Entropy} = -\left(\frac{3}{6}\log_2\frac{3}{6} + \frac{3}{6}\log_2\frac{3}{6}\right) = 1$$

The entropy is 0 if the split is completely homogeneous

- Pure subset (**4 yes** / **0 no**) → completely certain (100%)

$$\text{Entropy} = -\left(\frac{4}{4}\log_2\frac{4}{4} + \frac{0}{4}\log_2\frac{0}{4}\right) = 0$$



Decision Tree -- Information Gain

Measures that can be used to capture the purity of split -- **Information Gain**

Uses entropy to calculate the homogeneity of a sample, and a reduction of entropy is often called an information gain, i.e., **Information Gain = $\text{Entropy}_{\text{before}}$ - $\text{Entropy}_{\text{after}}$**

Constructing a decision tree is all about finding attribute that returns the highest information gain (i.e., the most homogeneous branches)

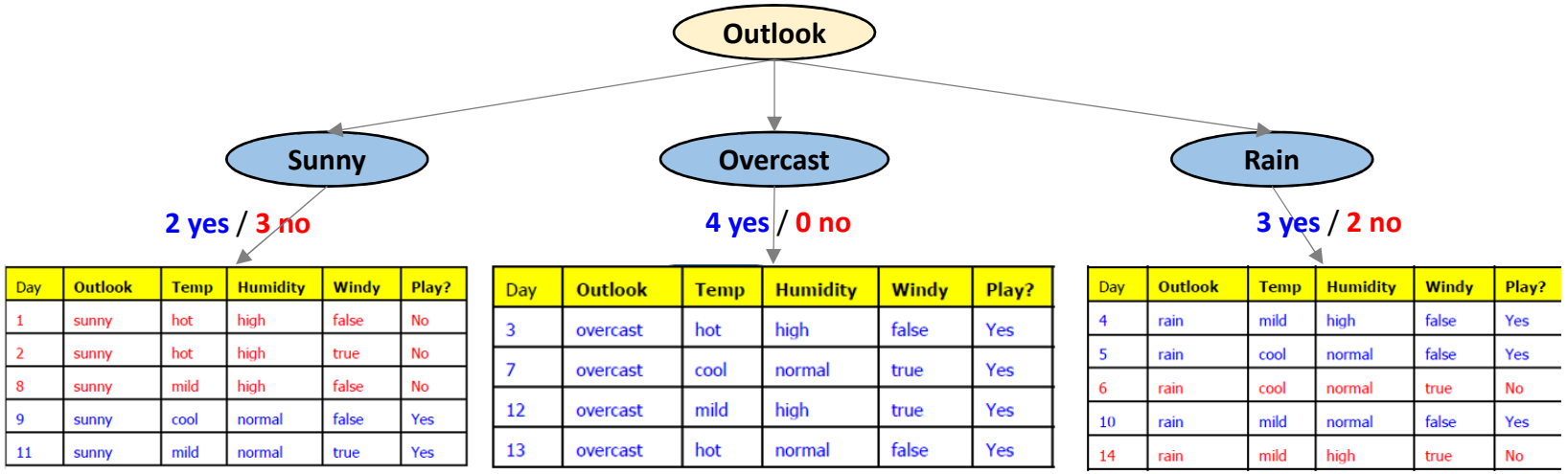
A decision tree is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values (homogenous)

The information gain is based on the decrease in entropy after a dataset is split on an attribute

http://www.saedsayad.com/decision_tree.htm

Decision Tree -- Information Gain

The goal is to decrease in entropy (uncertainty) after splitting and also to want more items in pure sets



Day	Outlook	Temp	Humidity	Windy	Play?
1	sunny	hot	high	false	No
2	sunny	hot	high	true	No
8	sunny	mild	high	false	No
9	sunny	cool	normal	false	Yes
11	sunny	mild	normal	true	Yes

Day	Outlook	Temp	Humidity	Windy	Play?
3	overcast	hot	high	false	Yes
7	overcast	cool	normal	true	Yes
12	overcast	mild	high	true	Yes
13	overcast	hot	normal	false	Yes

Day	Outlook	Temp	Humidity	Windy	Play?
4	rain	mild	high	false	Yes
5	rain	cool	normal	false	Yes
6	rain	cool	normal	true	No
10	rain	mild	normal	false	Yes
14	rain	mild	high	true	No

Before split: 14 records, 9 yes and 5 no: $\text{Entropy}_{\text{before}} = -\left(\frac{9}{14}\log_2\frac{9}{14} + \frac{5}{14}\log_2\frac{5}{14}\right) = 0.94$

Outlook = "Sunny": 5 records, 2 yes and 3 no: $\text{Entropy}_{\text{sunny}} = -\left(\frac{2}{5}\log_2\frac{2}{5} + \frac{3}{5}\log_2\frac{3}{5}\right) = 0.97$

Outlook = "Overcast": 4 records, 4 yes and 0 no: $\text{Entropy}_{\text{overcast}} = -\left(\frac{4}{4}\log_2\frac{4}{4} + 0\log_2\frac{0}{4}\right) = 0$

Outlook = "Rain": 5 records, 3 yes and 0 no: $\text{Entropy}_{\text{rain}} = -\left(\frac{3}{5}\log_2\frac{3}{5} + \frac{2}{5}\log_2\frac{2}{5}\right) = 0.97$

Expected new Entropy after split: $\text{Entropy}_{\text{after}} = \frac{4}{14}\cdot 0 + \frac{5}{14}\cdot 0.97 + \frac{5}{14}\cdot 0.97 = 0.69$

Split based on attribute "Outlook"

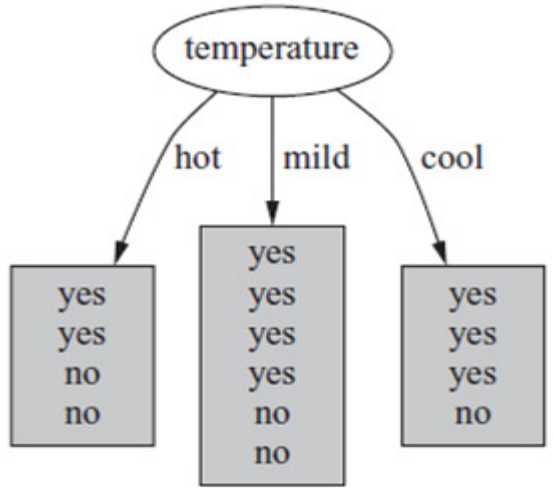
Information Gain

$$= \text{Entropy}_{\text{before}} - \text{Entropy}_{\text{after}}$$

$$= 0.94 - 0.69 = 0.25$$

Decision Tree -- Information Gain

The goal is to decrease in entropy (uncertainty) after splitting and also to want more items in pure sets



Before split: 14 records, 9 yes and 5 no: $\text{Entropy}_{\text{before}} = -\left(\frac{9}{14}\log_2\frac{9}{14}+\frac{5}{14}\log_2\frac{5}{14}\right)=0.94$

Temperature = "cool": 4 records, 3 yes and 1 no: $\text{Entropy}_{\text{cool}} = -\left(\frac{3}{4}\log_2\frac{3}{4}+\frac{1}{4}\log_2\frac{1}{4}\right)=0.81$

Temperature = "hot": 4 records, 2 yes and 2 no: $\text{Entropy}_{\text{hot}} = -\left(\frac{2}{4}\log_2\frac{2}{4}+\frac{2}{4}\log_2\frac{2}{4}\right)=1.0$

Temperature = "mild": 6 records, 4 yes and 2 no: $\text{Entropy}_{\text{mild}} = -\left(\frac{4}{6}\log_2\frac{4}{6}+\frac{2}{6}\log_2\frac{2}{6}\right)=0.92$

Expected new Entropy after split: $\text{Entropy}_{\text{after}} = \frac{4}{14}\cdot 0.81+\frac{4}{14}\cdot 1.0+\frac{6}{14}\cdot 0.92=0.91$

Split based on attribute "temperature"

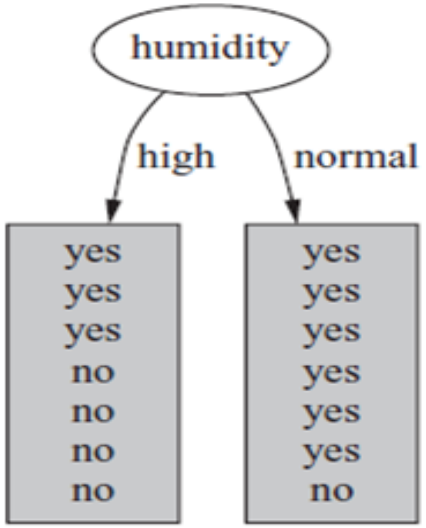
Information Gain

$$= \text{Entropy}_{\text{before}} - \text{Entropy}_{\text{after}}$$

$$= 0.94 - 0.69 = 0.03$$

Decision Tree -- Information Gain

The goal is to decrease in entropy (uncertainty) after splitting and also to want more items in pure sets



Before split: 14 records, 9 yes and 5 no: $\text{Entropy}_{\text{before}} = -\left(\frac{9}{14}\log_2\frac{9}{14} + \frac{5}{14}\log_2\frac{5}{14}\right) = 0.94$

Humidity = "normal": 7 records, 6 yes and 1 no: $\text{Entropy}_{\text{normal}} = -\left(\frac{6}{7}\log_2\frac{6}{7} + \frac{1}{7}\log_2\frac{1}{7}\right) = 0.59$

Humidity = "high": 7 records, 3 yes and 4 no: $\text{Entropy}_{\text{high}} = -\left(\frac{3}{7}\log_2\frac{3}{7} + \frac{4}{7}\log_2\frac{4}{7}\right) = 0.99$

Expected new Entropy after split: $\text{Entropy}_{\text{after}} = \frac{7}{14}\cdot 0.59 + \frac{7}{14}\cdot 0.99 = 0.79$

Split based on attribute "Humidity"

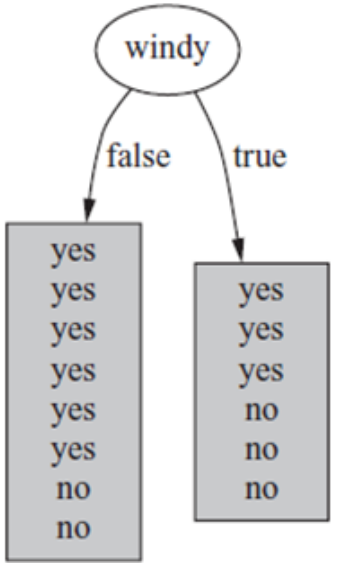
Information Gain

$$= \text{Entropy}_{\text{before}} - \text{Entropy}_{\text{after}}$$

$$= 0.94 - 0.79 = 0.15$$

Decision Tree -- Information Gain

The goal is to decrease in entropy (uncertainty) after splitting and also to want more items in pure sets



Before split: 14 records, 9 yes and 5 no: $\text{Entropy}_{\text{before}} = -\left(\frac{9}{14}\log_2\frac{9}{14} + \frac{5}{14}\log_2\frac{5}{14}\right) = 0.94$

Windy = "false": 8 records, 6 yes and 2 no: $\text{Entropy}_{\text{false}} = -\left(\frac{6}{8}\log_2\frac{6}{8} + \frac{2}{8}\log_2\frac{2}{8}\right) = 0.81$

Windy = "true": 6 records, 3 yes and 3 no: $\text{Entropy}_{\text{true}} = -\left(\frac{3}{6}\log_2\frac{3}{6} + \frac{3}{6}\log_2\frac{3}{6}\right) = 1.0$

Expected new Entropy after split: $\text{Entropy}_{\text{after}} = \frac{8}{14}\cdot 0.81 + \frac{6}{14}\cdot 1.0 = 0.89$

Split based on attribute "Windy"

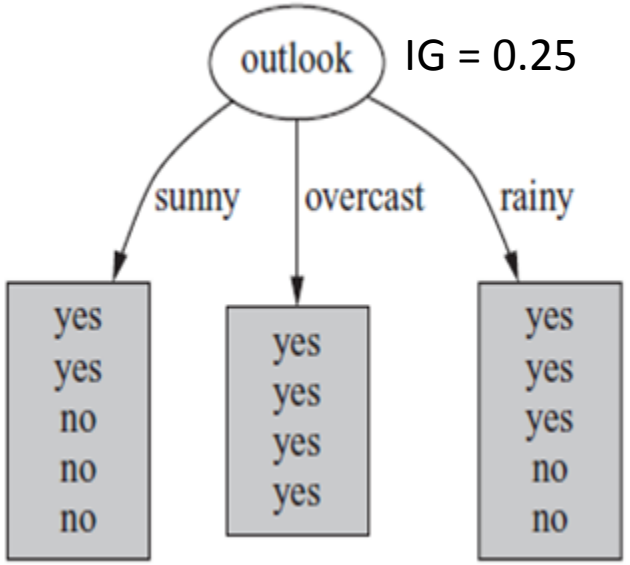
Information Gain

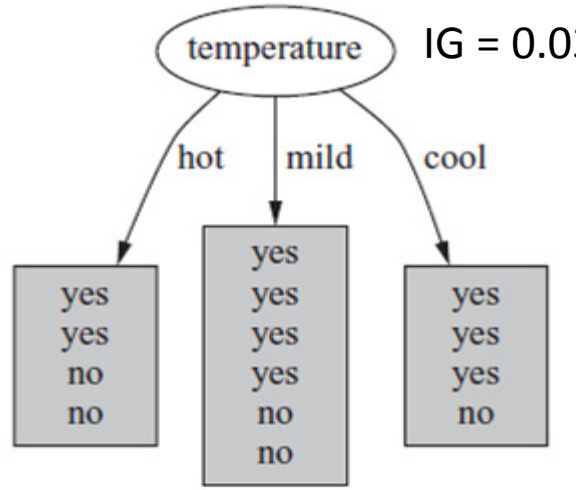
$$= \text{Entropy}_{\text{before}} - \text{Entropy}_{\text{after}}$$

$$= 0.94 - 0.89 = 0.05$$

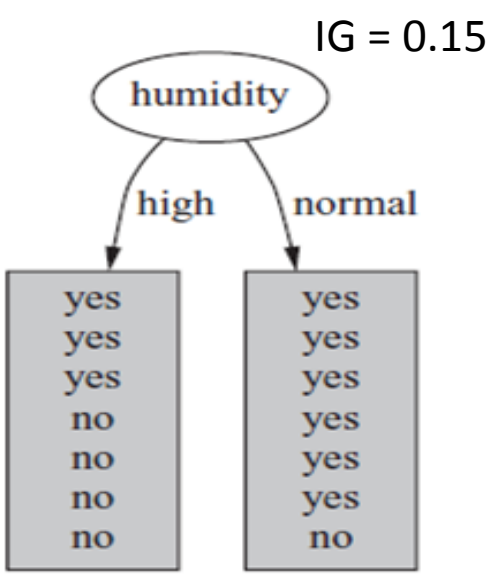
Decision Tree -- Information Gain

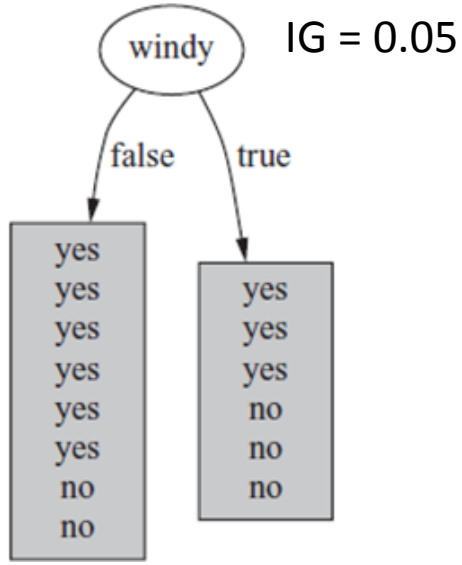
The goal is to decrease in entropy (uncertainty) after splitting and also to want more items in pure sets





IG for "Outlook": 0.25
IG for "Temperature": 0.03
IG for "Humidity": 0.15
IG for "Windy": 0.05



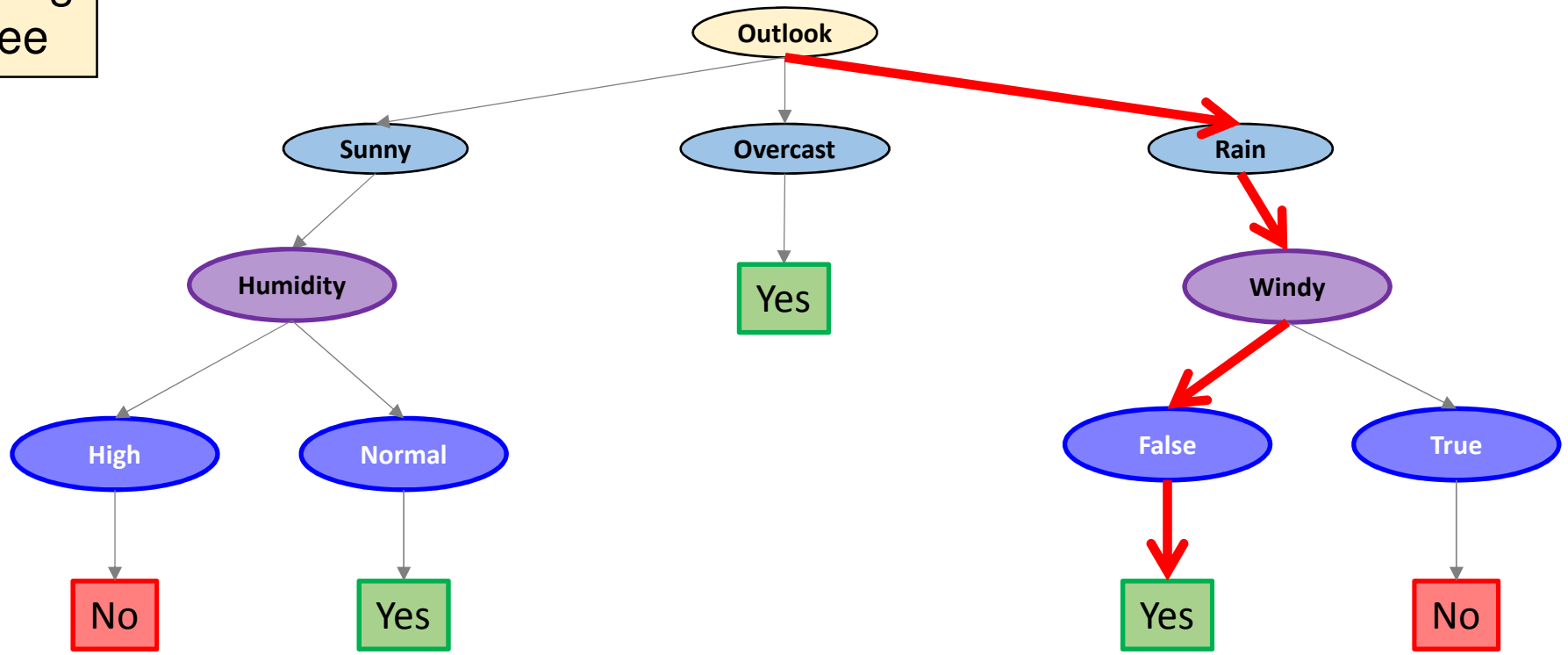


Split data using the attributes in the following order:

1. "Outlook" first, then
2. "Humidity", then
3. "Windy", then
4. "Temperature"

Decision Tree -- Information Gain

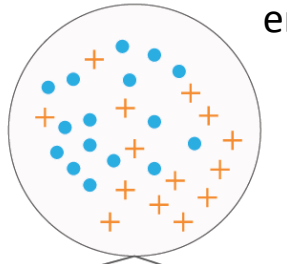
Final resulting decision tree



Day	Outlook	Temp	Humidity	Windy	Play?
Day 15	Rain	Mild	High	False	YES

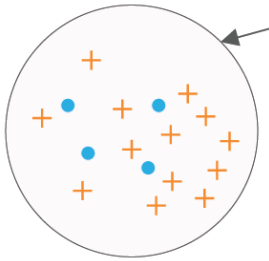
Decision Tree -- Information Gain

The goal is to decrease in entropy (uncertainty) after splitting and also to want more items in pure sets

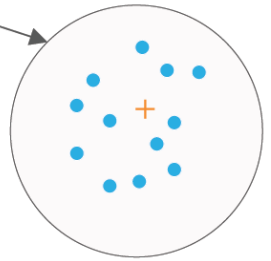


entire population: 30

$$\text{Entropy}_{\text{parent}} = -\left(\frac{14}{30}\log_2\frac{14}{30} + \frac{16}{30}\log_2\frac{16}{30}\right) = 0.996$$



17 instances
4/17 blue circles; 13/17 orange crosses



13 instances
12/13 blue circles; 1/13 orange crosses

$$\text{Entropy}_{\text{child}} = -\left(\frac{13}{17}\log_2\frac{13}{17} + \frac{4}{17}\log_2\frac{4}{17}\right) = 0.787$$

$$\text{Entropy}_{\text{child}} = -\left(\frac{1}{13}\log_2\frac{1}{13} + \frac{12}{13}\log_2\frac{12}{13}\right) = 0.391$$

$$\text{Entropy}_{\text{children}} = \frac{17}{30}\cdot 0.787 + \frac{13}{30}\cdot 0.391 = 0.615$$

For this split

$$\text{Information Gain} = \text{Entropy}_{\text{parent}} - \text{Entropy}_{\text{children}} = 0.996 - 0.615 = 0.38$$

Decision Tree -- Information Gain Ratio

Measures that can be used to capture the purity of split -- **Information Gain Ratio**

- IG has a bias towards attributes with large number of values over the ones with small number of values
- Such "super attributes" can easily be selected as the root, resulted a broad tree that classifies perfectly on the training instances but performs poorly on unseen (new) instances
- Gain Ratio: penalize attributes with large number of values

$$\text{GainRatio(A)} = \frac{\text{Gain(A)}}{\text{Split-Info(A)}} = \frac{\text{Gain(A)}}{\text{Split-Info}_{\text{A}}\text{(D)}} = \frac{\text{Gain(A)}}{-\sum_{j=1}^{V}\left(\frac{|D_j|}{|D|}\cdot\log_2\frac{|D_j|}{|D|}\right)}$$

Splitting the training data set D into V partitions, corresponding to V outcomes on attribute A

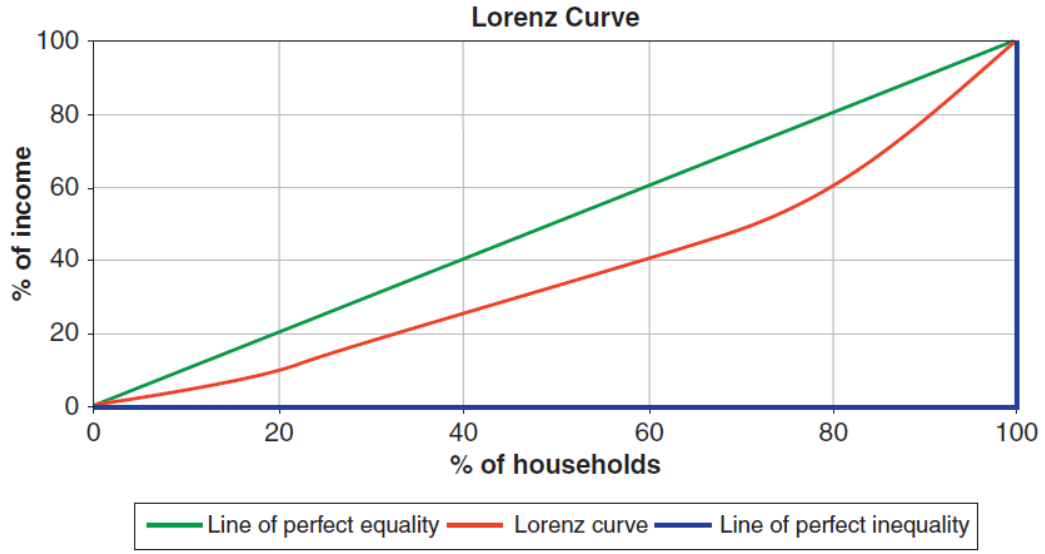
- The attribute with the **maximal gain ratio** is selected as the splitting attribute first
- Gain Ratio overcomes the bias problem of Information Gain

https://www.youtube.com/watch?v=rb1jdBPKzDk
http://www.saedsayad.com/decision_tree.htm

Decision Tree -- Gini Index

Measures that can be used to capture the purity of split -- **Gini Index**

- Gini Index quantifies the *unevenness* in variable distributions, e.g., income distributions among countries
- The theory behind the Gini Index relies on the difference between a theoretical equality of some quantity and its actual value over the range of a related variable
- Lorenz curve



- The theoretical even distribution of income among households is symbolized by the straight line through the center of the figure
- The inequality in incomes among households is shown by the red line below the line of perfect equality
- If the red line remained near the bottom of the figure until the 80th percentile, for example, it would represent a population with a few very rich people and a lot of very poor people

Decision Tree -- Gini Index

Measures that can be used to capture the purity of split -- **Gini Index**

- Gini impurity

$D = \text{buys\_computer}$

$$\text{Gini}(D) = 1 - \sum_{j=1}^{K}\left(\frac{D_j}{|D|}\right) = 1 - \left(\left(\frac{9}{14}\right)^2 + \left(\frac{5}{14}\right)^2\right) = 0.459$$

$D_1 = \text{income} \in \{\text{low, medium}\}$

$D_2 = \text{income} \in \{\text{high}\}$

$$\text{Gini}_{D_1 D_2}(D) = \text{Gini}_{\text{income}\in\{\{\text{low, medium}\},\{\text{high}\}\}}(D)$$

$$= \frac{10}{14}\text{Gini}(D_1) + \frac{4}{14}\text{Gini}(D_2)$$

$$= \frac{10}{14}\left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right)$$

$$= 0.4429$$

D_1 and D_2 | D

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

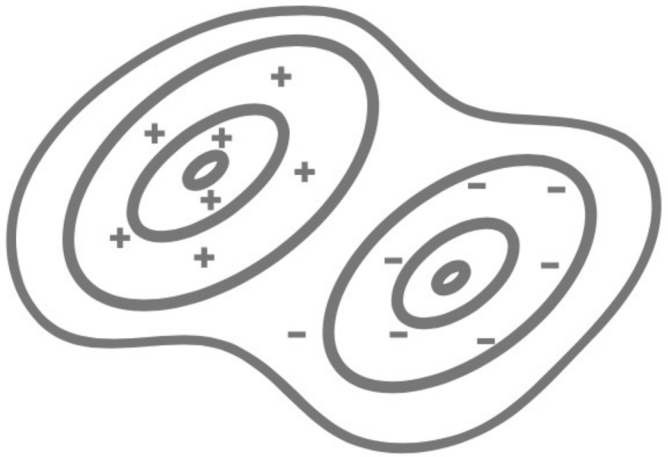
- Gini Index of the split based on attribute "income". Note that here we categorize low and medium incomes into the class
- The attribute produces the **lowest Gini Index** should be selected first to build the DT, i.e., the best binary split for attribute income is {medium, high} and {low}

$$\text{Gini}_{\text{income}\in\{\{\text{medium, high}\},\{\text{low}\}\}}(D) = 0.300 \qquad \text{Gini}_{\text{income}\in\{\{\text{low, high}\},\{\text{medium}\}\}}(D) = 0.315$$

Classification (Naïve Bayes)

Naïve Bayes

- **Naïve Bayes is a simple but important probabilistic model**
- Naive Bayes is a simple **multiclass** classification algorithm based on applying Bayes' theorem with the **"naive" assumption** of independence between the features. It assumes that the conditional probabilities of the independent variables are statistically independent.
- It computes the conditional probability distribution of each feature given label, and then it applies Bayes' theorem to compute the conditional probability distribution of label given an observation and use it for prediction.
- It classifies new data based on the **highest probability** of its belonging to a particular class.



Naive Bayes is a simple classifier model that is:

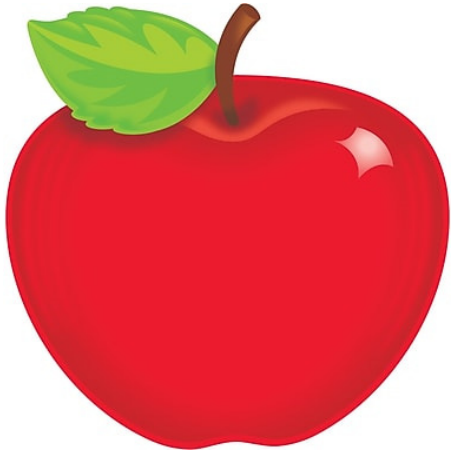
- Based on the Bayes theorem
- Supervised Learning
- Easy to build
- Faster to train, compared to other models
- Often used as a baseline classifier for benchmarking

http://spark.apache.org/docs/2.2.0/mllib-naive-bayes.html

Naïve Bayes

- **Naïve Assumption**
- It assumes that each input feature X_i is conditionally independent of every other feature X_j, given the class C.

$$P(X_1, X_2, \ldots, X_n | C) = P(X_1 | C) \cdot P(X_2 | C) \cdot \ldots \cdot P(X_n | C)$$

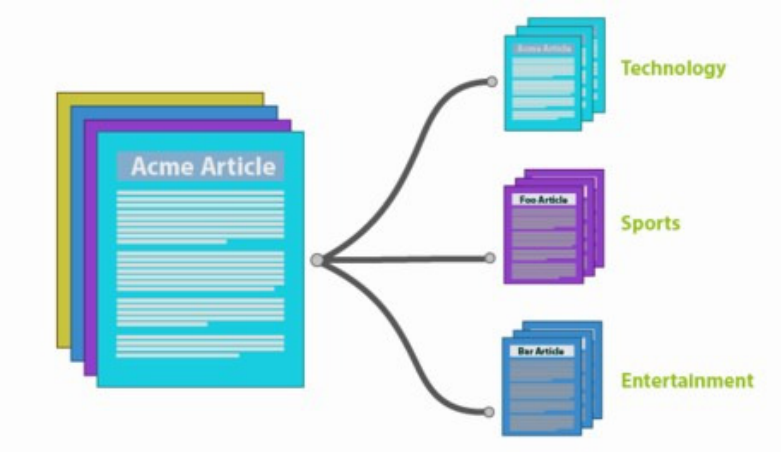


For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter

A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features

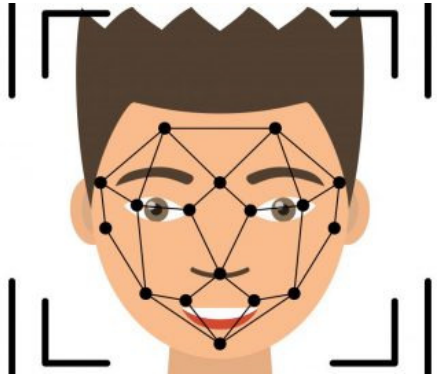
Naïve Bayes

- **Application of Naïve Bayes**
  - Although the assumption that the predictor (independent) variables are independent is not always accurate, it does simplify the classification task dramatically. 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 learners and classifiers can be extremely fast compared to more sophisticated methods



Document Classification

Spam Detection



Face Recognition

https://goo.gl/dR8ydd
http://scikit-learn.org/stable/modules/naive_bayes.html

Naïve Bayes

- **Bayes' Theorem**
  - Works on conditional probability -- the probability that something will happen, given that something else has already occurred.
  - Using the conditional probability, calculate the probability of an event using its prior knowledge.

How often A happens given that B happens

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

- P(A|B) is the probability of A given that B happens
- P(A) is the probability of A
- P(B|A) is the probability of B given that A happens
- P(B) is the probability of B

source: wikipedia

- P(Fire | Smoke): how often there is a fire accident when we see smoke
- P(Smoke | Fire): how often we see smoke when there is a fire accident

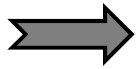
Naïve Bayes

- **Naïve Bayes Classifier**
  - To derive the maximum posteriori, i.e., the maximal $P(C_k \mid X)$
- Given training data $X = (X_1, X_2, \ldots, X_n)$, representing n features (independent variables)
  - e.g., there are 3 features, X = (pants, long hair, red t-shirt)
- Suppose C represents the class variable
  - e.g., there are 2 classes, C_1 = man, C_2 = woman
- Using Bayes' Theorem, the conditional probability can be decomposed as

$$\underbrace{P(C_k \mid X_1, X_2, \ldots, X_n)}_{\text{Posterior Probability}} = \frac{P(X_1, X_2, \ldots, X_n \mid C_k) P(C_k)}{P(X_1, X_2, \ldots, X_n)} = \frac{P(X_1 \mid C_k) P(X_2 \mid C_k) \ldots P(X_n \mid C_k) P(C_k)}{P(X_1, X_2, \ldots, X_n)}$$

For example, an observation: a person who wears pants and red t-shirt and has long hair

- P(man | pants, long hair, red t-shirt) = 0.7
- P(woman | pants, long hair, red t-shirt) = 0.3



Classify X to the class that has larger posterior, i.e., predict that X belongs to C_i if $P(C_i \mid X_1, X_2, \ldots, X_n)$ is the highest among all the $P(C_k \mid X_1, X_2, \ldots, X_n)$ for all the k classes, in this example, this person is predicted as man since its posteriori probability 0.7 > 0.3

Naïve Bayes

- Example: predict if john will go out and play tennis

Day	Outlook	Temp	Humidity	Windy	Play?
1	sunny	hot	high	false	No
2	sunny	hot	high	true	No
3	overcast	hot	high	false	Yes
4	rain	mild	high	false	Yes
5	rain	cool	normal	false	Yes
6	rain	cool	normal	true	No
7	overcast	cool	normal	true	Yes
8	sunny	mild	high	false	No
9	sunny	cool	normal	false	Yes
10	rain	mild	normal	false	Yes
11	sunny	mild	normal	true	Yes
12	overcast	mild	high	true	Yes
13	overcast	hot	normal	false	Yes
14	rain	mild	high	true	No
15	**sunny**	**cool**	**high**	**true**	**?**

5 no and **9 yes**

Given a new instance (day 15),

X = (Outlook=sunny,
Temp=cool,
Humidity=high,
Windy=true),

predict if john will go out and play or not?

Naïve Bayes

- Example: predict if john will go out and play tennis

Day	Outlook	Temp	Humidity	Windy	Play?
1	sunny	hot	high	false	No
2	sunny	hot	high	true	No
3	overcast	hot	high	false	Yes
4	rain	mild	high	false	Yes
5	rain	cool	normal	false	Yes
6	rain	cool	normal	true	No
7	overcast	cool	normal	true	Yes
8	sunny	mild	high	false	No
9	sunny	cool	normal	false	Yes
10	rain	mild	normal	false	Yes
11	sunny	mild	normal	true	Yes
12	overcast	mild	high	true	Yes
13	overcast	hot	normal	false	Yes
14	rain	mild	high	true	No
15	**sunny**	**cool**	**high**	**true**	**?**

Outlook	Play = Yes	Play = No	Total
Sunny	2/9	3/5	5/14
Overcast	4/9	0/5	4/14
Rain	3/9	2/5	5/14

Temp	Play = Yes	Play = No	Total
Hot	2/9	2/5	4/14
Mild	4/9	2/5	6/14
Cool	3/9	1/5	4/14

Humidity	Play = Yes	Play = No	Total
High	3/9	4/5	7/14
Normal	6/9	1/5	7/14

Wind	Play = Yes	Play = No	Total
True	3/9	3/5	6/14
False	6/9	2/5	8/14

Given a new instance (day 15), X = (Outlook=sunny, Temp=cool, Humidity=high, Windy=true), predict if john will go out and play or not?

- P(Outlook=sunny | Play=yes) = 2/9
- P(Temp=cool | Play=yes) = 3/9
- P(Humidity=high | Play=yes) = 3/9
- P(Windy=true | Play=yes) = 3/9
- P(Play=yes) = 9/14

- P(Outlook=sunny | Play=no) = 3/5
- P(Temp=cool | Play=no) = 1/5
- P(Humidity=high | Play=no) = 4/5
- P(Windy=true | Play=no) = 3/5
- P(Play=no) = 5/14

$$P(\text{Yes}\mid\text{sunny, cool, high, true})$$

$$= \frac{P(\text{sunny, cool, high, true}\mid\text{Yes})P(\text{Yes})}{P(\text{sunny, cool, high, true})}$$

$$= \frac{P(\text{sunny}\mid\text{Yes})P(\text{cool}\mid\text{Yes})P(\text{high}\mid\text{Yes})P(\text{true}\mid\text{Yes})P(\text{Yes})}{P(\text{sunny})P(\text{cool})P(\text{high})P(\text{true})}$$

$$= \frac{(2/9)\cdot(3/9)\cdot(3/9)\cdot(3/9)\cdot(9/14)}{(5/14)\cdot(4/14)\cdot(7/14)\cdot(6/14)} = 0.24$$

$$P(\text{No}\mid\text{sunny, cool, high, true})$$

$$= \frac{P(\text{sunny}\mid\text{No})P(\text{cool}\mid\text{No})P(\text{high}\mid\text{No})P(\text{true}\mid\text{No})P(\text{No})}{P(\text{sunny})P(\text{cool})P(\text{high})P(\text{true})}$$

$$= \frac{(3/5)\cdot(1/5)\cdot(4/5)\cdot(3/5)\cdot(5/14)}{(5/14)\cdot(4/14)\cdot(7/14)\cdot(6/14)} = 0.94$$

$$P(\text{Yes}\mid X) = 0.24$$

$$P(\text{No}\mid X) = 0.94$$

→ John will likely NOT go out and play tennis

Naïve Bayes

$$P(C_k \mid X_1, X_2, \ldots, X_n) = \frac{P(X_1, X_2, \ldots, X_n \mid C_k)\, P(C_k)}{P(X_1, X_2, \ldots, X_n)}$$

Can be ignored since it is the same for all classes (a normalization constant)

Naive assumption: attribute independence

It is difficult to learn the probability $P(X_1, X_2, ..., X_n \mid C_k)$, which requires initial knowledge of many probabilities and involves significant computational cost

However, if we assume that the feature probabilities are conditionally independent (i.e., no dependence relationship between all input features), it can be simplified as follows

$$P(X_1, X_2, \ldots, X_n \mid C) = P(X_1 \mid C) \cdot P(X_2 \mid C) \cdot \ldots \cdot P(X_n \mid C)$$

$$P(C_k \mid X_1, X_2, \ldots, X_n) = \frac{P(X_1, X_2, \ldots, X_n \mid C_k)\, P(C_k)}{P(X_1, X_2, \ldots, X_n)} = \frac{P(X_1 \mid C_k) P(X_2 \mid C_k) \ldots P(X_n \mid C_k)\, P(C_k)}{P(X_1, X_2, \ldots, X_n)}$$

Naïve Bayes Classifier

$$P(C_k \mid X_1, X_2, \ldots, X_n) = \frac{P(X_1, X_2, \ldots, X_n \mid C_k)\, P(C_k)}{P(X_1, X_2, \ldots, X_n)}$$

$$P(C_k \mid X_1, X_2, \ldots, X_n) = \frac{1}{Z} \cdot P(C_k) \prod_{i=1}^{n} P(X_i \mid C_k)$$

where $Z = P(X_1, X_2, \ldots, X_n) = \sum_k P(X_1, X_2, \ldots, X_n \mid C_k) P(C_k)$

Constructing a classifier from the probability model, e.g., use the Maximum A Posteriori (MAP) classification rule to assign a class label $\hat{y} = C_k$ to a new instance $X = (X_1, X_2, \ldots, X_n)$ for some k as follows, i.e., if $P(C_k \mid X_1, X_2, \ldots, X_n)$ is the largest, i.e.,

$$\hat{y} = \arg\max_{k \in \{1,2,\ldots K\}} P(C_k) \prod_{i=1}^{n} P(X_i \mid C_k)$$

Example -- classifying 20 newsgroups data set (~ 20K docs)

- The 20 newsgroup data set is a standard data set commonly used for machine learning research. The data is from transcripts of several months of posting made in 20 Usenet newsgroups from the early 1990s. http://qwone.com/~jason/20Newsgroups/

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

- If you examine one of the files in the training data directory, such as 20news-bydate-train/sci.crypt/15524, you will see something like this:

```
From: rdippold@qualcomm.com (Ron "Asbestos" Dippold)
Subject: Re: text of White House announcement and Q&As
Originator: rdippold@qualcom.qualcomm.com
Nntp-Posting-Host: qualcom.qualcomm.com
Organization: Qualcomm, Inc., San Diego, CA
Lines: 12

ted@nmsu.edu (Ted Dunning) writes:
>nobody seems to have noticed that the clipper chip *must* have been
>under development for considerably longer than the 3 months that
>clinton has been president. this is not something that choosing
...
```

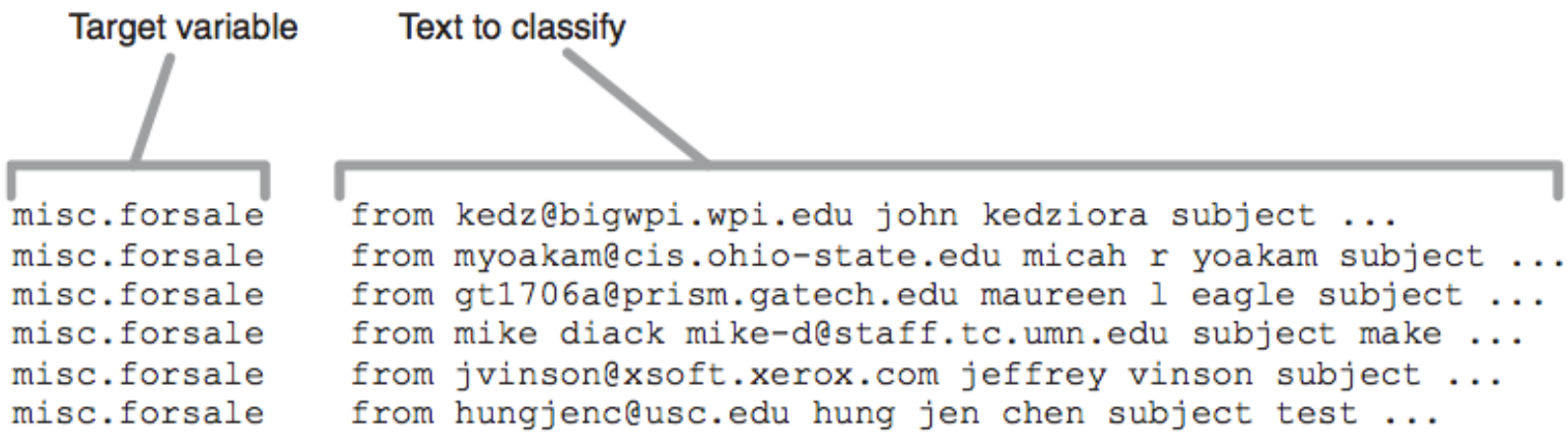
The predictor features are either headers or body

Classifying using Naive Bayes

```
$ bin/mahout prepare20newsgroups -p 20news-bydate-train/ \
       -o 20news-train/ \
       -a org.apache.lucene.analysis.standard.StandardAnalyzer \
       -c UTF-8

no HADOOP_CONF_DIR or HADOOP_HOME set, running locally
INFO: Program took 3713 ms

$ bin/mahout prepare20newsgroups -p 20news-bydate-test \
       -o 20news-test \
       -a org.apache.lucene.analysis.standard.StandardAnalyzer \
       -c UTF-8
no HADOOP_CONF_DIR or HADOOP_HOME set, running locally
INFO: Program took 2436 ms
```



Training and testing naive Bayes classifier

```
bin/mahout trainclassifier -i 20news-train \
            -o 20news-model \
            -type cbayes \
            -ng 1 \
            -source hdfs
...
INFO: Program took 250104 ms
```

The result is a model stored in the 20news-model directory, as specified by the `-o` option. The `-ng` option indicates that individual words are to be considered instead of short sequences of words. The model consists of several files that contain the components of the model. These files are in a binary format and can't be easily inspected directly, but you can use them to classify the test data with the help of the `testclassifier` program.

To run the naive Bayes model on the test data, you can use the following command:

```
bin/mahout testclassifier -d 20news-test \
            -m 20news-model \
            -type cbayes \
            -ng 1 \
            -source hdfs \
            -method sequential
```

Here the `-m` option specifies the directory that contains the model built in the previous step. The `-method` option specifies that the program should be run in a sequential mode rather than using Hadoop. For small data sets like this one, sequential operation is preferred. For larger data sets, parallel operation becomes necessary to keep the runtime reasonably small.

See training results

When given this familiar data, the model is able to get nearly 98 percent correct, which is much too good to be true on this particular problem. The best machine learning researchers only claim accuracies for their systems around 84~86 percent.

```
bin/mahout testclassifier -d 20news-train -m 20news-model\
   -type cbayes -ng 1 -source hdfs -method sequential
...
Correctly Classified Instances          :      11075    97.8876%
Incorrectly Classified Instances        :        239     2.1124%
Total Classified Instances              :      11314
```

Classification results

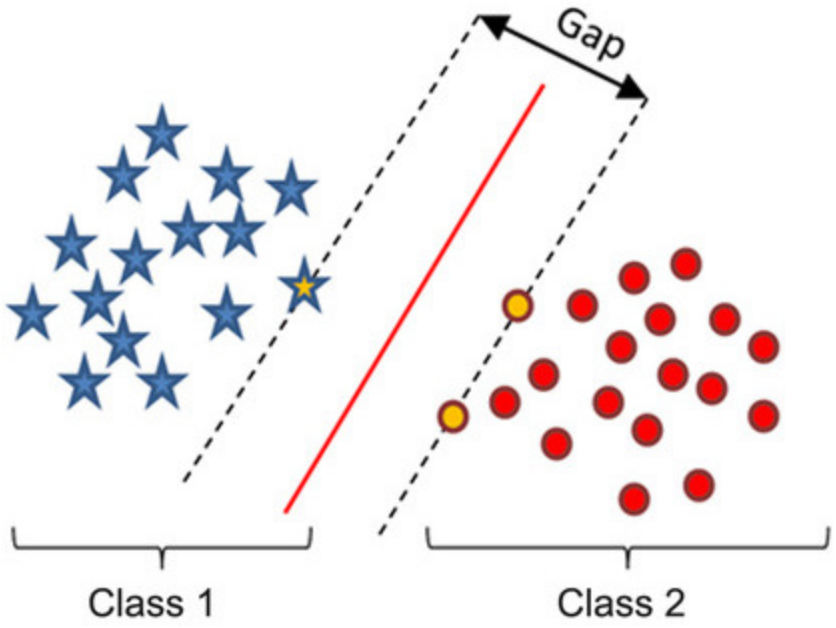
```
Confusion Matrix
-------------------------------------------------------
a b c d e f g h i j k l m n o p q r s t <--Classified as
388                                     | 397 a = rec.sport.baseball
  386                                   | 396 b = sci.crypt
    396                                 | 399 c = rec.sport.hockey
      347                               | 364 d = talk.politics.guns
        377                             | 398 e = soc.religion.christian
  12      304           18              | 393 f = sci.electronics
            281     14  43        21    | 394 g = comp.os.ms-windows.misc
              313       22 16           | 390 h = misc.forsale
     26 69      83 41                   | 251 i = talk.religion.misc
        45        225 13             11 | 319 j = alt.atheism
                    334           32    | 395 k = comp.windows.x
                      367               | 376 l = talk.politics.mideast
          19 23 15      307 13          | 392 m =comp.sys.ibm.pc.hardware
                        16 335          | 385 n = comp.sys.mac.hardware
                             371        | 394 o = sci.space
                               393      | 398 p = rec.motorcycles
                             12 364     | 396 q = rec.autos
            11      22            305   | 389 r = comp.graphics
      102                         160   | 310 s = talk.politics.misc
                                    362 | 396 t = sci.med
Default Category: unknown: 20


-------------------------------------------------------
Correctly Classified Instances          :       6398    84.9442%
Incorrectly Classified Instances        :       1134    15.0558%
Total Classified Instances              :       7532
=======================================================
```

Classification
(Support Vector Machine)

Outline

- Classification:
  - Binary Classification
    - Perceptron Classifier
  - Support Vector Machine
    - Linear SVM
    - Non-linear SVM



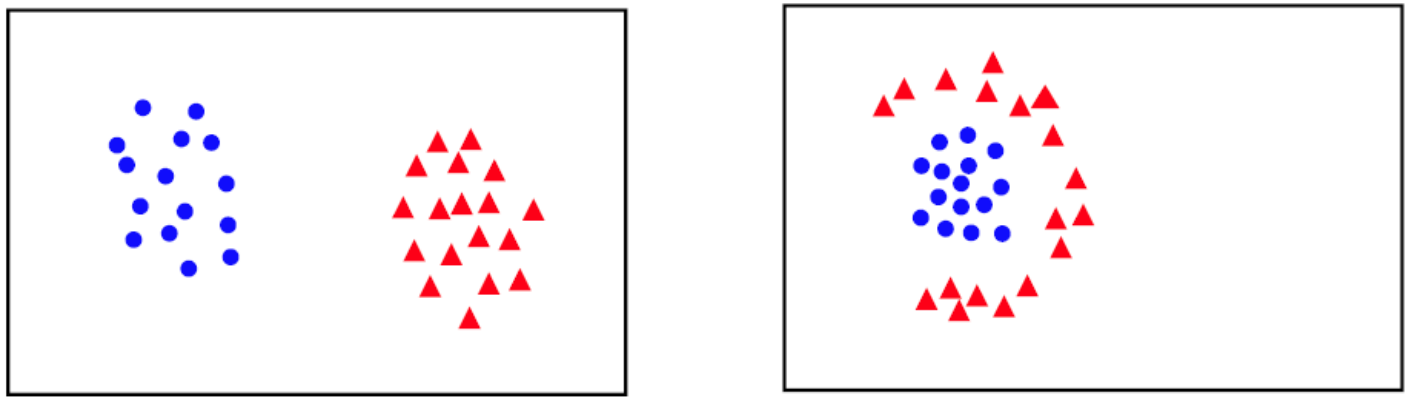
Some of these slides are based on Prof. Zisserman's lecture notes at: http://www.robots.ox.ac.uk/~az/lectures/ml/index.html

Binary Classification

- Given training data $(\mathbf{x}_i, y_i)$ for $i = 1, ..., N$, with $\mathbf{x}_i \in \mathbf{R}^d$ and $y_i = \{-1, 1\}$, learn a classifier $f(\mathbf{x})$ such that

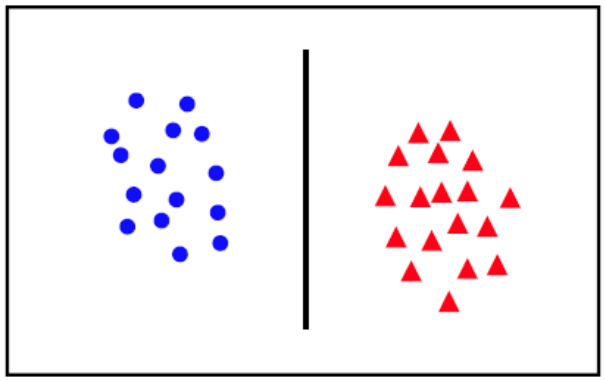
$$f(\mathbf{x}_i) = \begin{cases} \geq 0, & y_i = +1 \\ < 0, & y_i = -1 \end{cases}$$

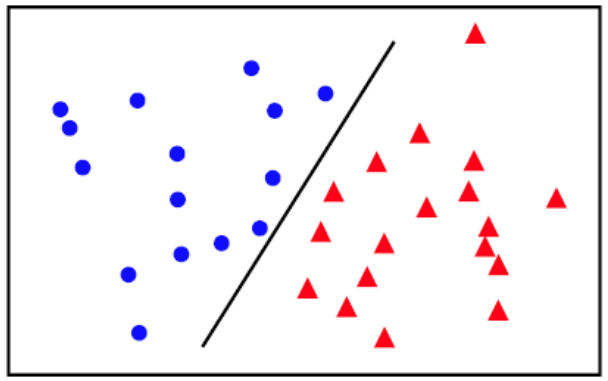
i.e., $y_i f(\mathbf{x}_i) > 0$ for a correct classification.



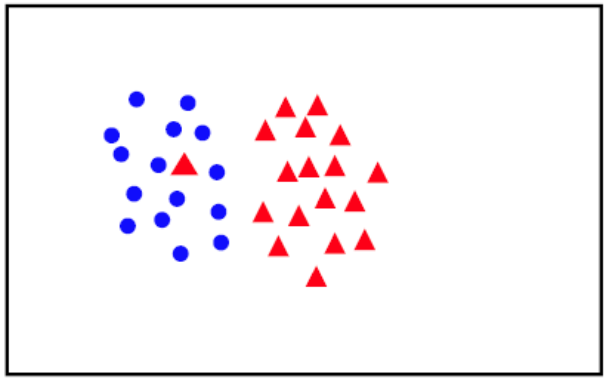
Linear Separability

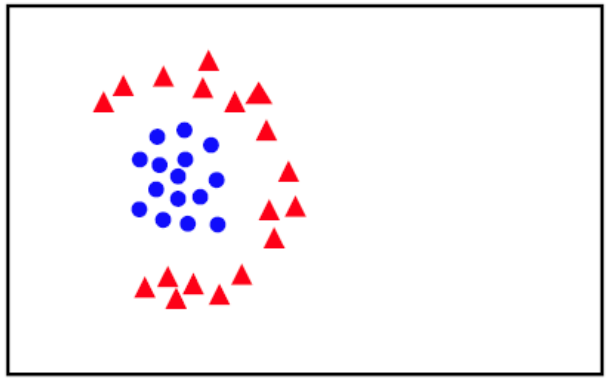
- Linearly separable





- Not linearly separable

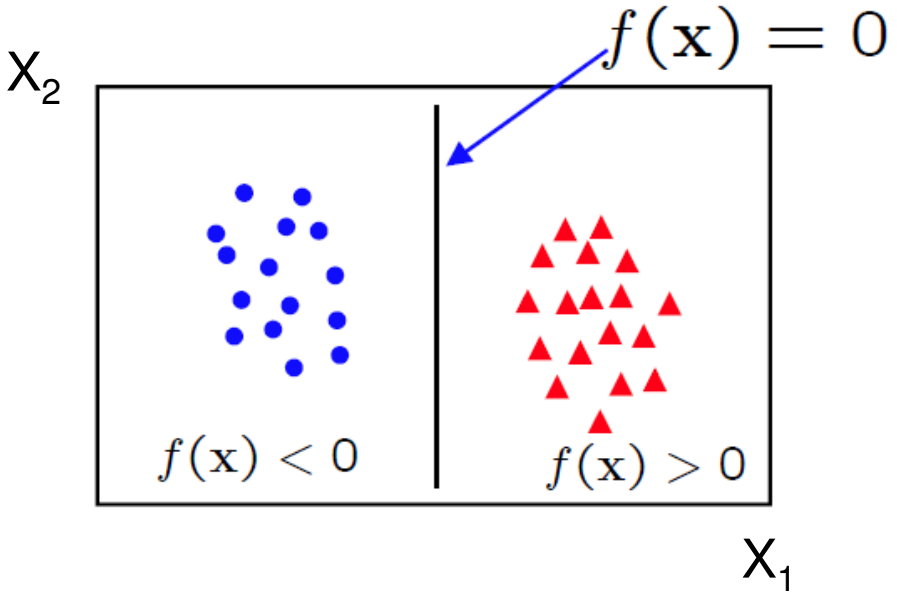




Linear Classifier

- A linear classifier has the form

$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b$$

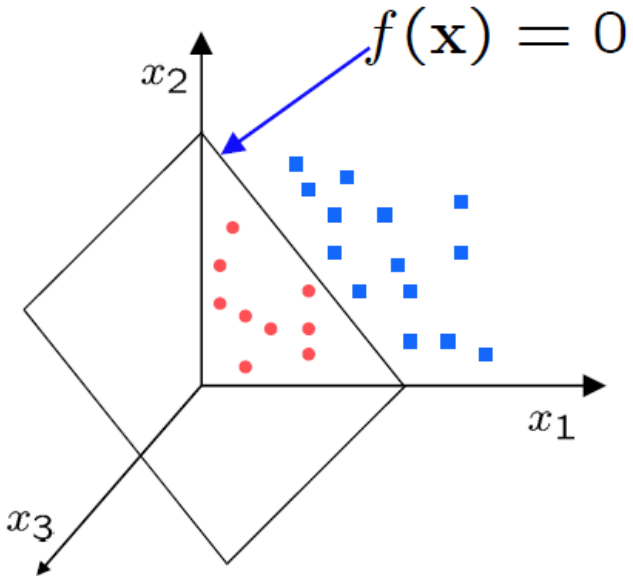


- in 2D the discriminant is a line
- $\mathbf{w}$ is the normal to the line, and b the bias
- $\mathbf{w}$ is known as the weight vector

Linear Classifier

- A linear classifier has the form

$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b$$



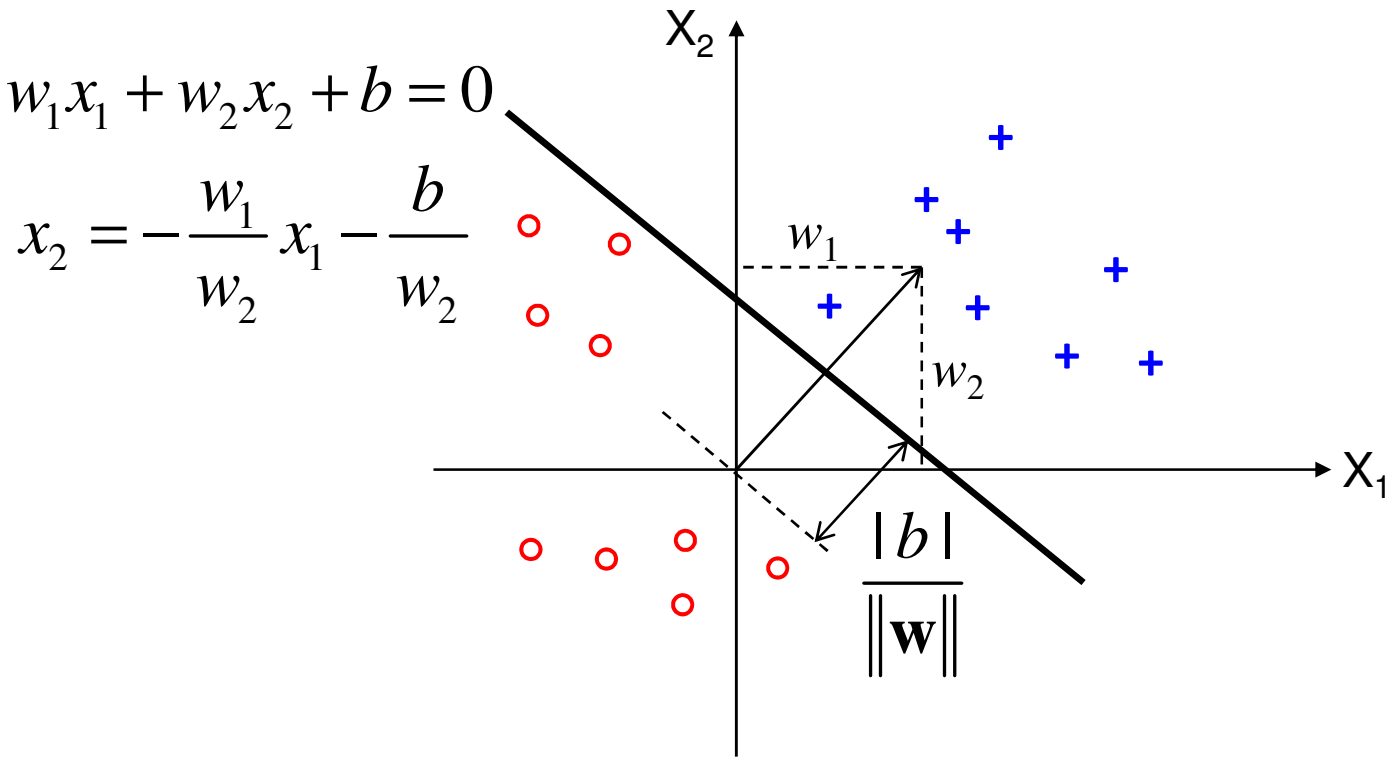
- in 3D the discriminant is a plane, and in nD it is a hyperplane
- For a linear classifier, the training data is used to learn $\mathbf{w}$ and then discarded
- Only $\mathbf{w}$ is needed for classifying new data

Why linear model?

- Simplest model – fewer parameters to learn (requires less training data to learn reliably)
- Intuitively appealing – draw a straight line (for 2D inputs) or a linear hyperplane (for higher dimensional inputs) to separate positive from negative
- Can be used to learn nonlinear models as well

Linear Separability

- The bias is proportional to the offset of the plane from the origin
- The weights determine the slope of the line
- The weight vector is perpendicular to the plane



The Perceptron Classifier

- Given linearly separable data $\mathbf{x}_i$ labelled into two categories $y_i = \{-1, 1\}$, find a weight vector $\mathbf{w}$ such that the discriminant function

$$f(\mathbf{x}_i) = \mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b$$

  separates the categories for $i = 1, \ldots, N$
- How can we find this separating hyperplane?
- Perceptron Learning Algorithm
  - Train the perceptron to classify inputs correctly
  - Accomplished by adjusting the connecting weights and the bias
  - Can only properly handle linearly separable datasets

Perceptron Learning Algorithm

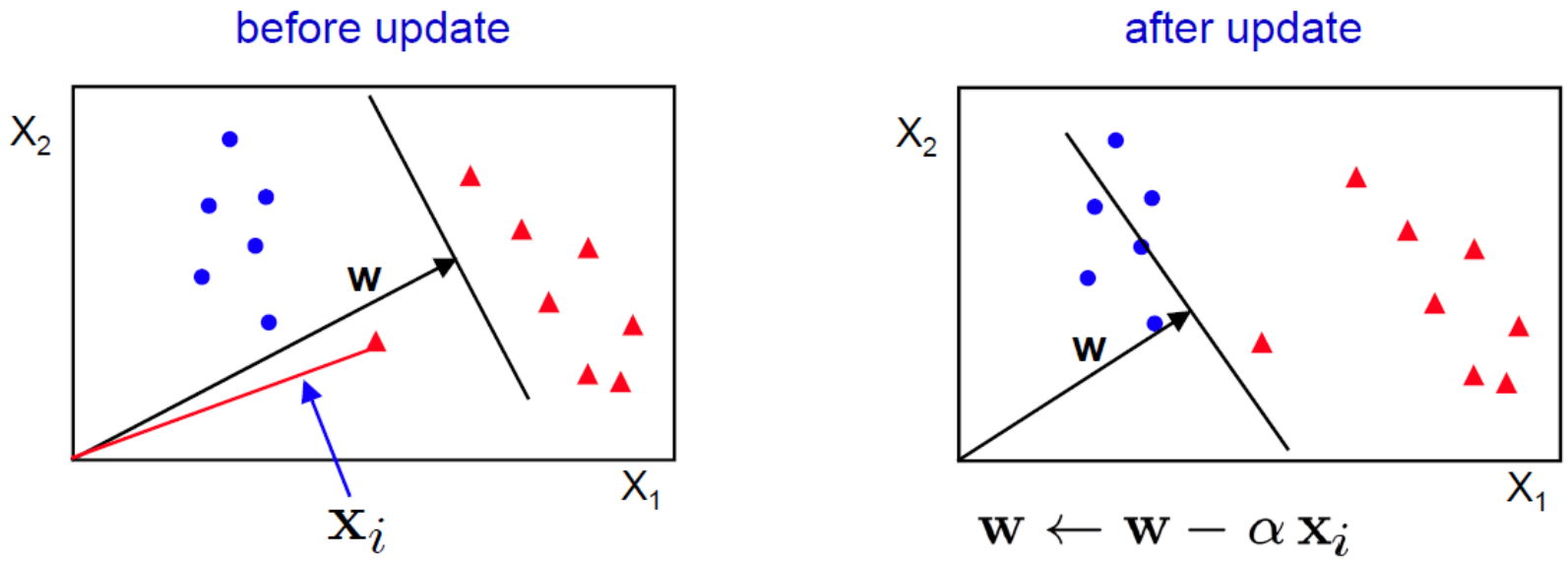
- During training both $\mathbf{w}_i$ and b (bias) are modified for convenience, let $\mathbf{w}_0 = b$ and $\mathbf{x}_0 = 1$
- Let, α, the learning rate, be a small positive number (small steps lessen the possibility of destroying correct classifications)
- Initialize $\mathbf{w}_0$ to some values
- Cycle through the data points $\{\mathbf{x}_i, y_i\}$
  - If classification is correct, do nothing
  - If classification is incorrect, modify the weight vector **w** using

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \text{sign}\left(f(\mathbf{x}_i)\right)\mathbf{x}_i$$

- Repeat this procedure until entire training set is classified correctly

The Perceptron Classifier

- For example in 2D
  - Initialize $\mathbf{w} = 0$
  - Cycle through the data points $\{\mathbf{x}_i, y_i\}$
    - If $\mathbf{x}_i$ is misclassified then $\mathbf{w} \leftarrow \mathbf{w} + \alpha \text{sign}\left(f(\mathbf{x}_i)\right)\mathbf{x}_i$
  - Until all the data points are correctly classified



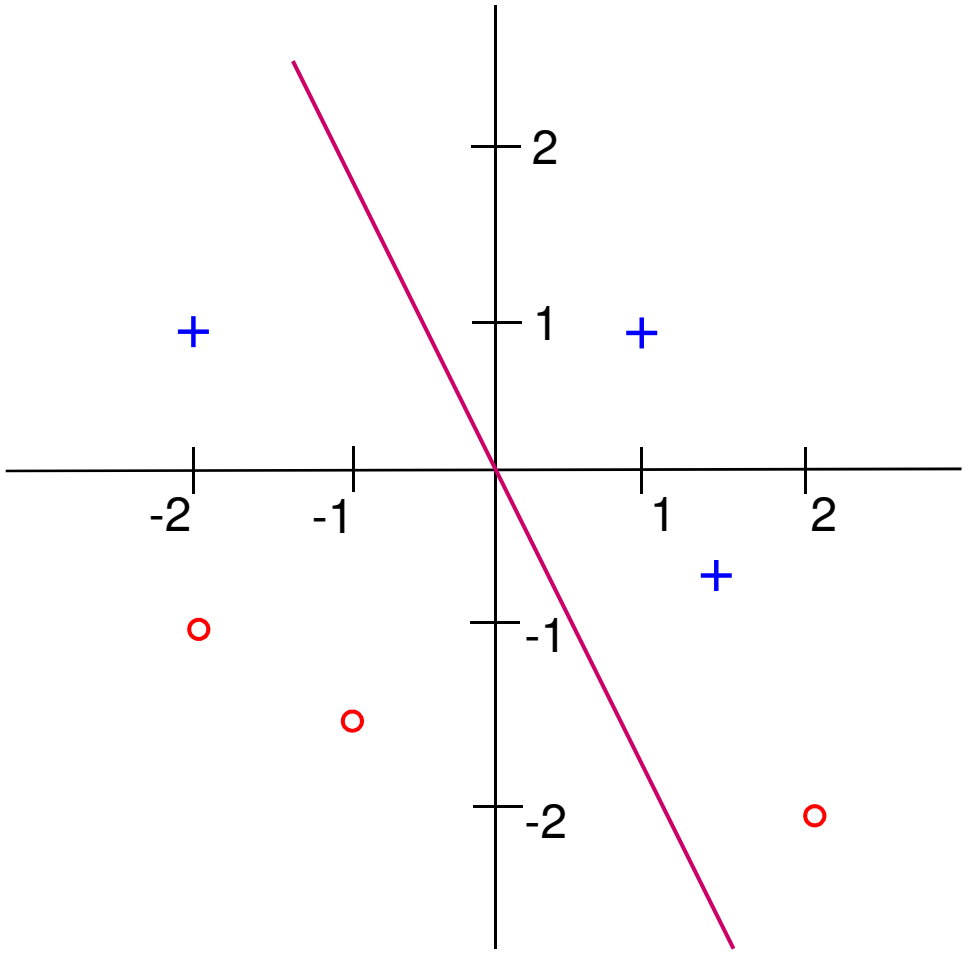
Perceptron learning example

- Initial values

$$\alpha = 0.2$$

$$\mathbf{w} = \begin{pmatrix} 0 \\ 1 \\ 0.5 \end{pmatrix}$$

$$\begin{aligned} 0 &= w_0 + w_1 x_1 + w_2 x_2 \\ &= 0 + x_1 + 0.5 x_2 \\ \Rightarrow x_2 &= -2 x_1 \end{aligned}$$



Example is adopted from: http://130.243.105.49/~lilien/ml/seminars/2007_02_01b-Janecek-Perceptron.pdf

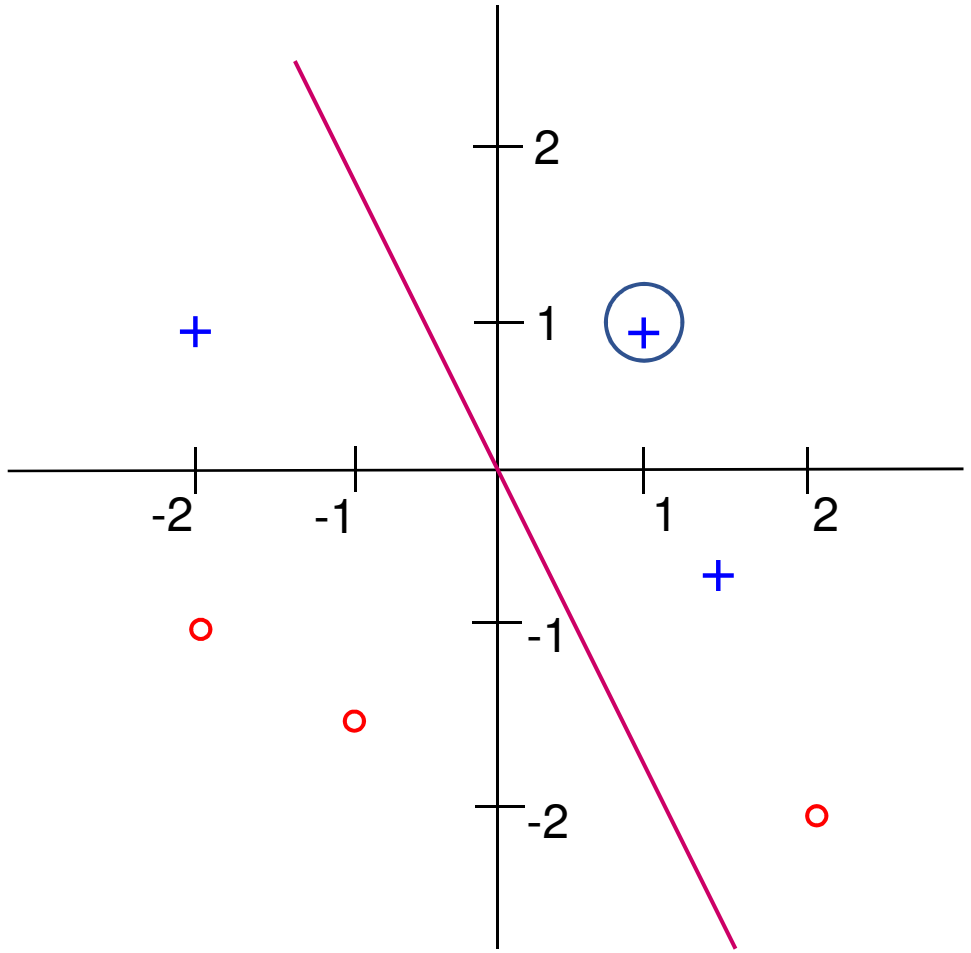
Perceptron learning example

$$\alpha = 0.2$$

$$\mathbf{w} = \begin{pmatrix} 0 \\ 1 \\ 0.5 \end{pmatrix}$$

$$x_1 = 1,\ x_2 = 1$$

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} > 0$$



- Correct classification, no action

Perceptron learning example

$$\alpha = 0.2$$

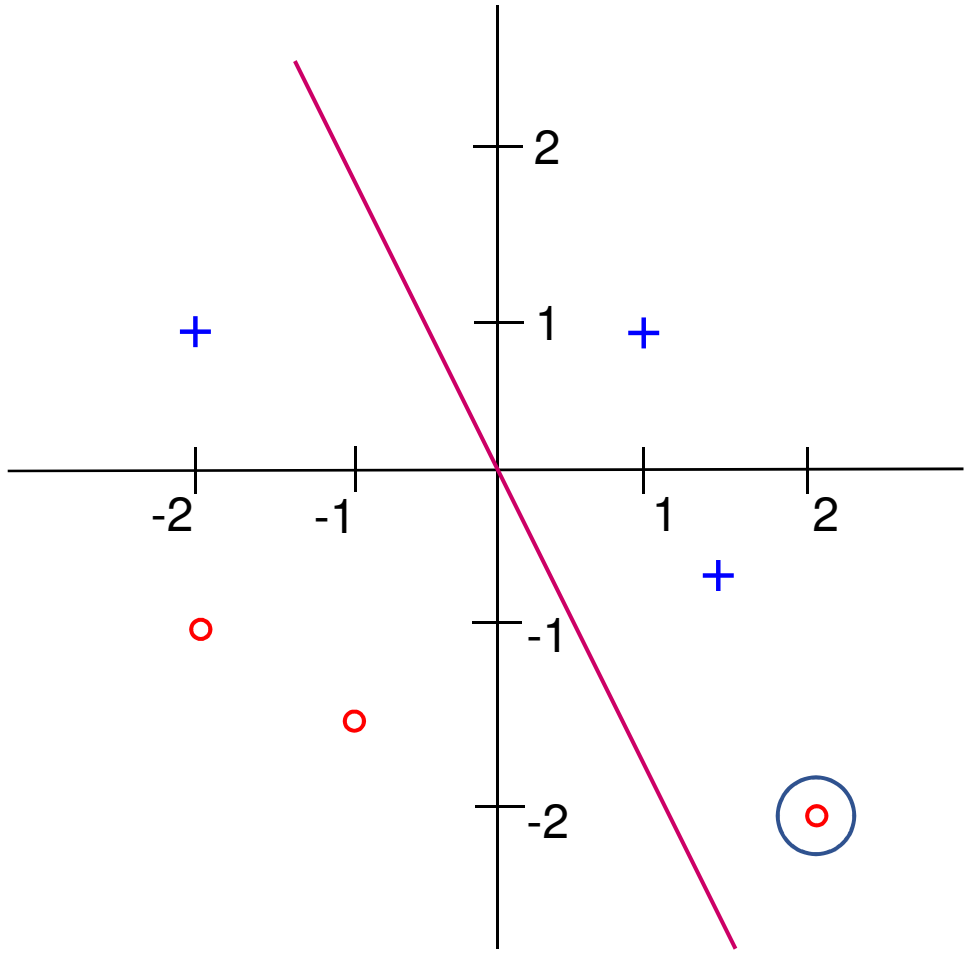
$$\mathbf{w} = \begin{pmatrix} 0 \\ 1 \\ 0.5 \end{pmatrix}$$

$$x_1 = 2,\ x_2 = -2$$

$$w_0 = w_0 - 0.2 \cdot 1$$

$$w_1 = w_1 - 0.2 \cdot 2$$

$$w_2 = w_2 - 0.2 \cdot (-2)$$



Perceptron learning example

$$\alpha = 0.2$$

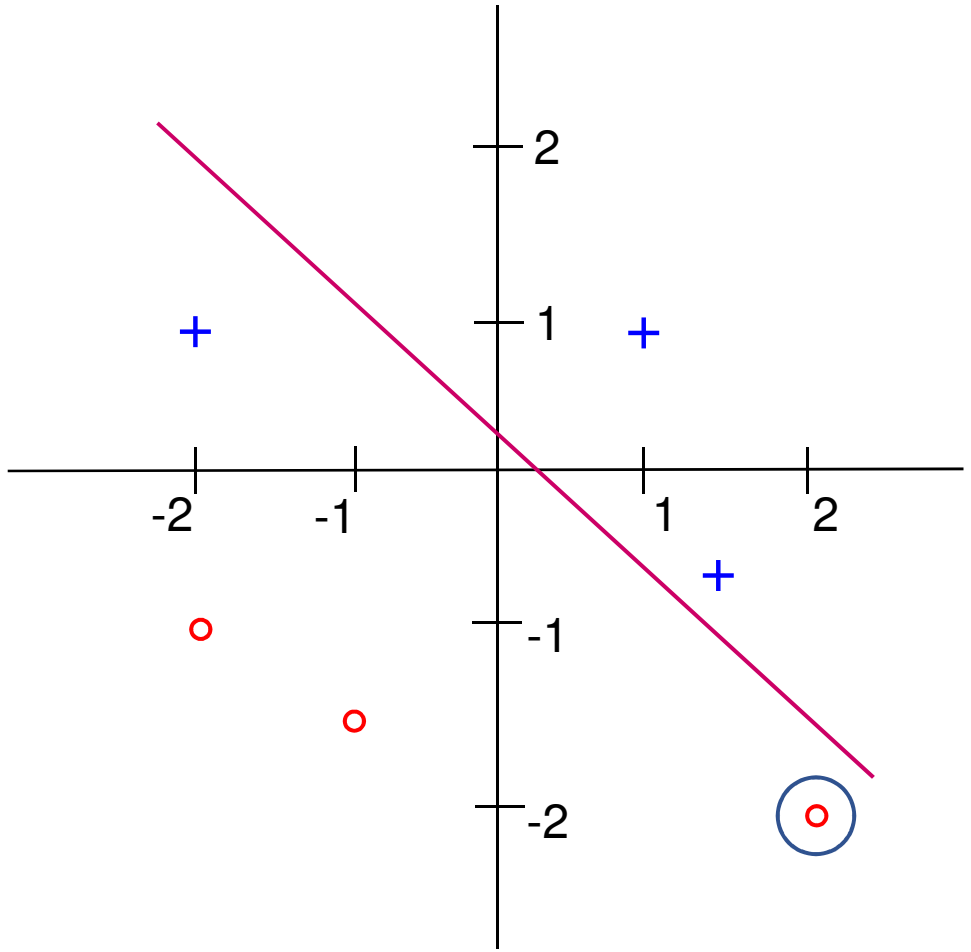
$$\mathbf{w} = \begin{pmatrix} -0.2 \\ 0.6 \\ 0.9 \end{pmatrix}$$

$$x_1 = 2,\ x_2 = -2$$

$$w_0 = w_0 - 0.2 \cdot 1$$

$$w_1 = w_1 - 0.2 \cdot 2$$

$$w_2 = w_2 - 0.2 \cdot (-2)$$



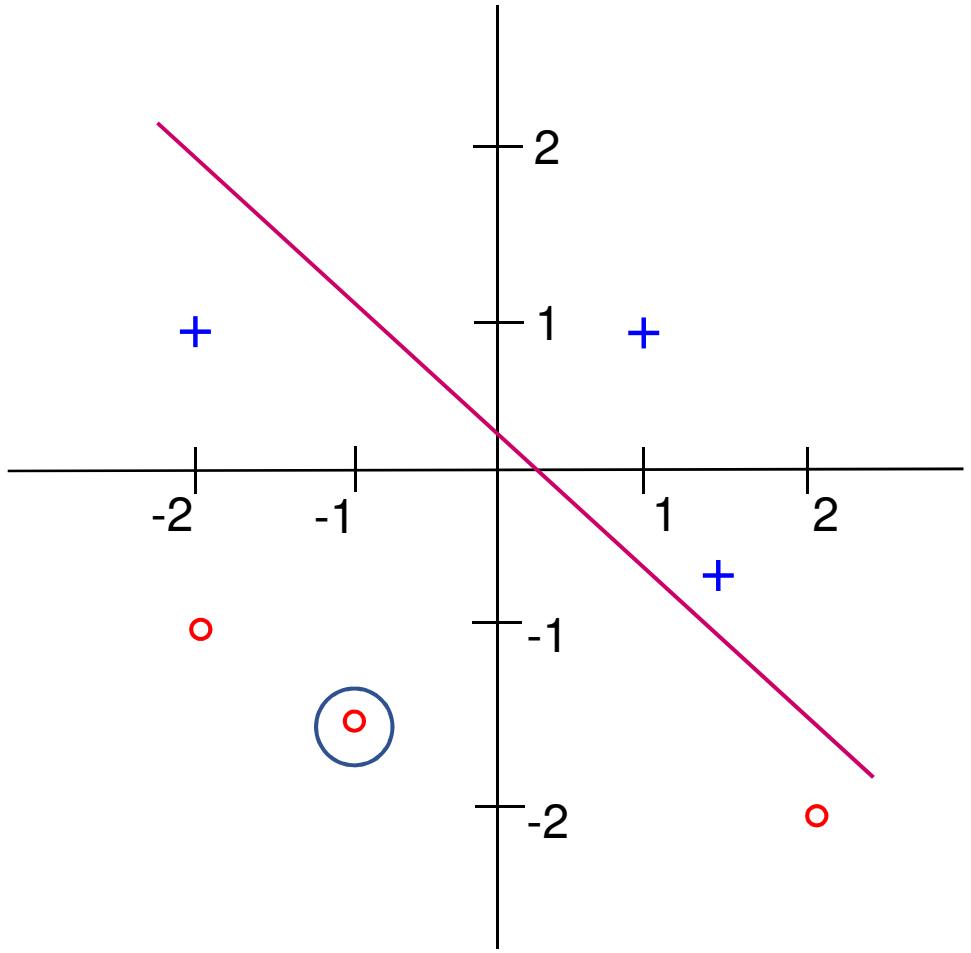
Perceptron learning example

$$\alpha = 0.2$$

$$\mathbf{w} = \begin{pmatrix} -0.2 \\ 0.6 \\ 0.9 \end{pmatrix}$$

$$x_1 = -1, x_2 = -1.5$$

$$\mathbf{w}^T\mathbf{x} < 0$$



- Correct classification, no action

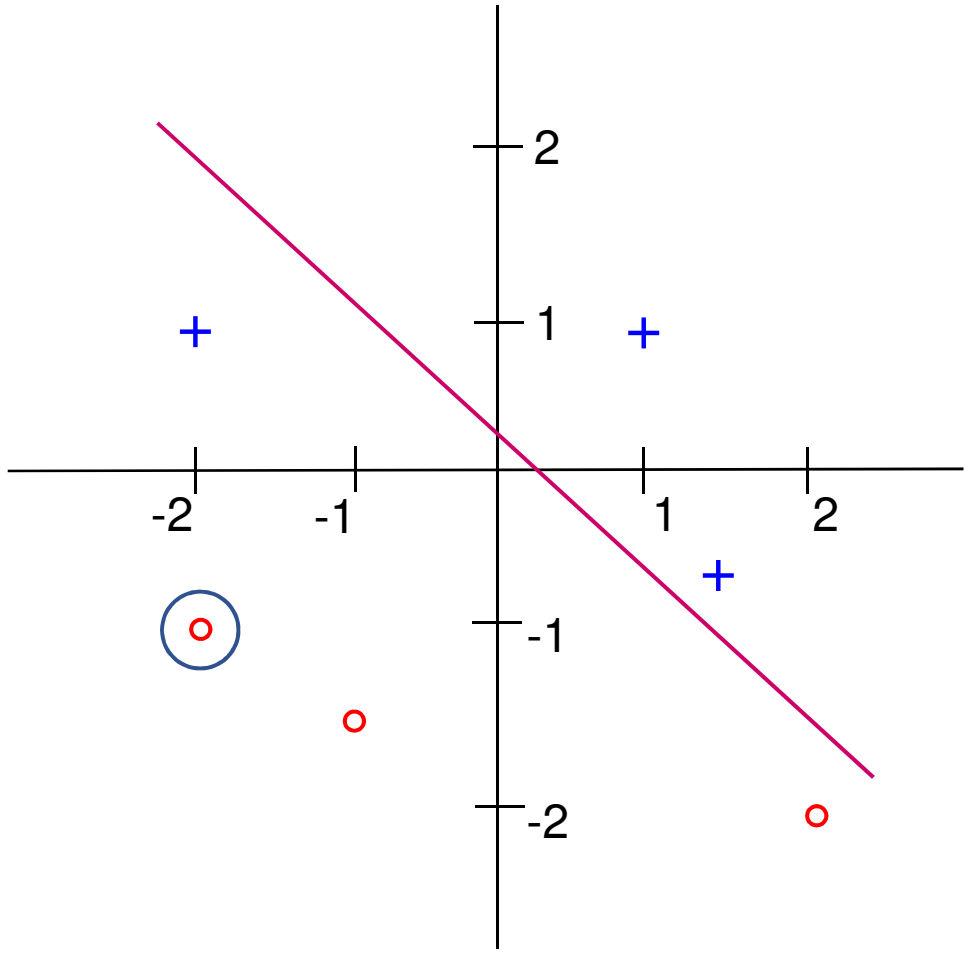
Perceptron learning example

$$\alpha = 0.2$$

$$\mathbf{w} = \begin{pmatrix} -0.2 \\ 0.6 \\ 0.9 \end{pmatrix}$$

$$x_1 = -2,\ x_2 = -1$$

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} < 0$$



- Correct classification, no action

Perceptron learning example

$$\alpha = 0.2$$

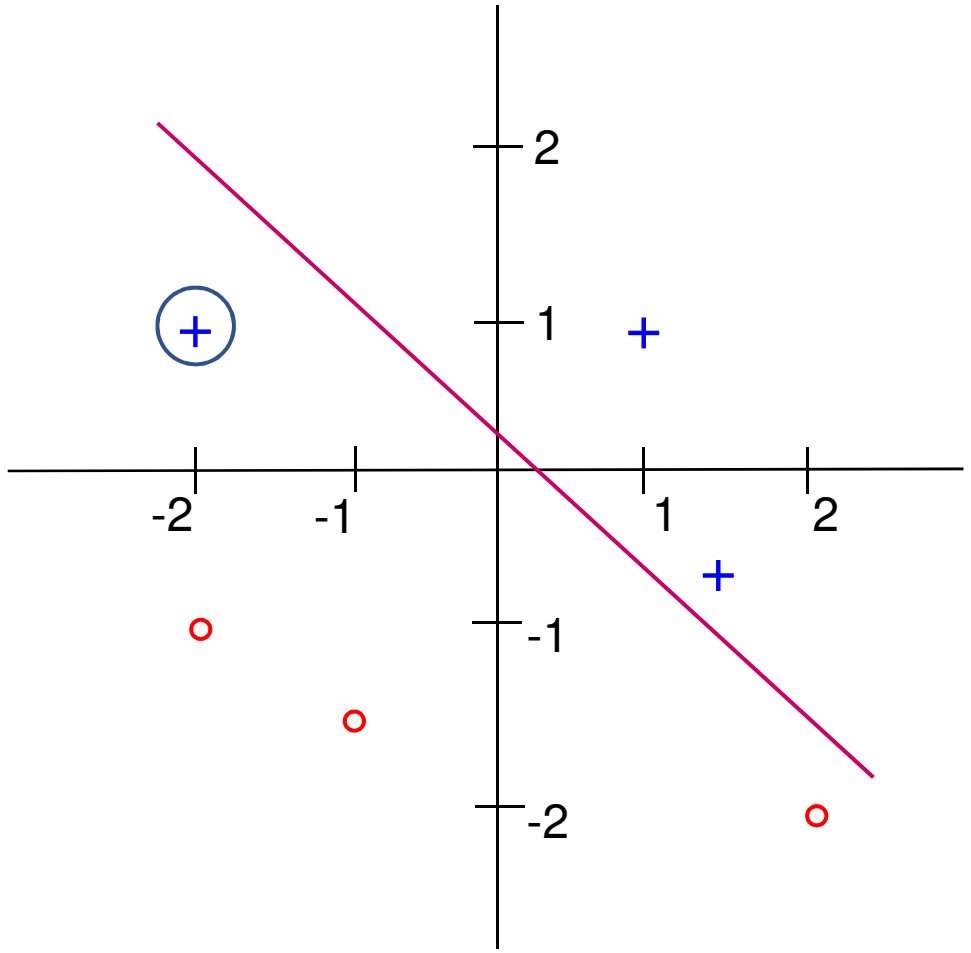
$$\mathbf{w} = \begin{pmatrix} -0.2 \\ 0.6 \\ 0.9 \end{pmatrix}$$

$$x_1 = -2,\ x_2 = 1$$

$$w_0 = w_0 - (-0.2) \cdot 1$$

$$w_1 = w_1 - (-0.2) \cdot (-2)$$

$$w_2 = w_2 - (-0.2) \cdot 1$$



Perceptron learning example

$$\alpha = 0.2$$

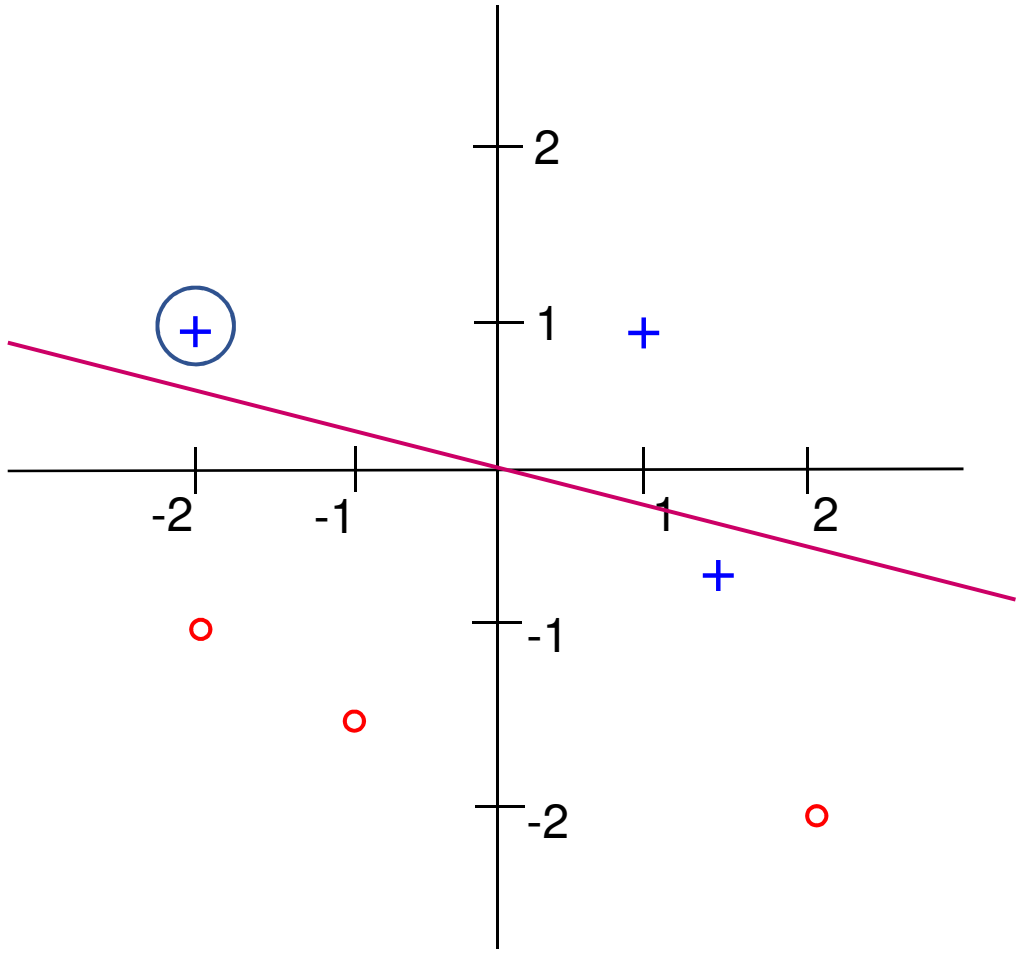
$$\mathbf{w} = \begin{pmatrix} 0 \\ 0.2 \\ 1.1 \end{pmatrix}$$

$$x_1 = -2,\ x_2 = 1$$

$$w_0 = w_0 - (-0.2) \cdot 1$$

$$w_1 = w_1 - (-0.2) \cdot (-2)$$

$$w_2 = w_2 - (-0.2) \cdot 1$$



Perceptron learning example

$$\alpha = 0.2$$

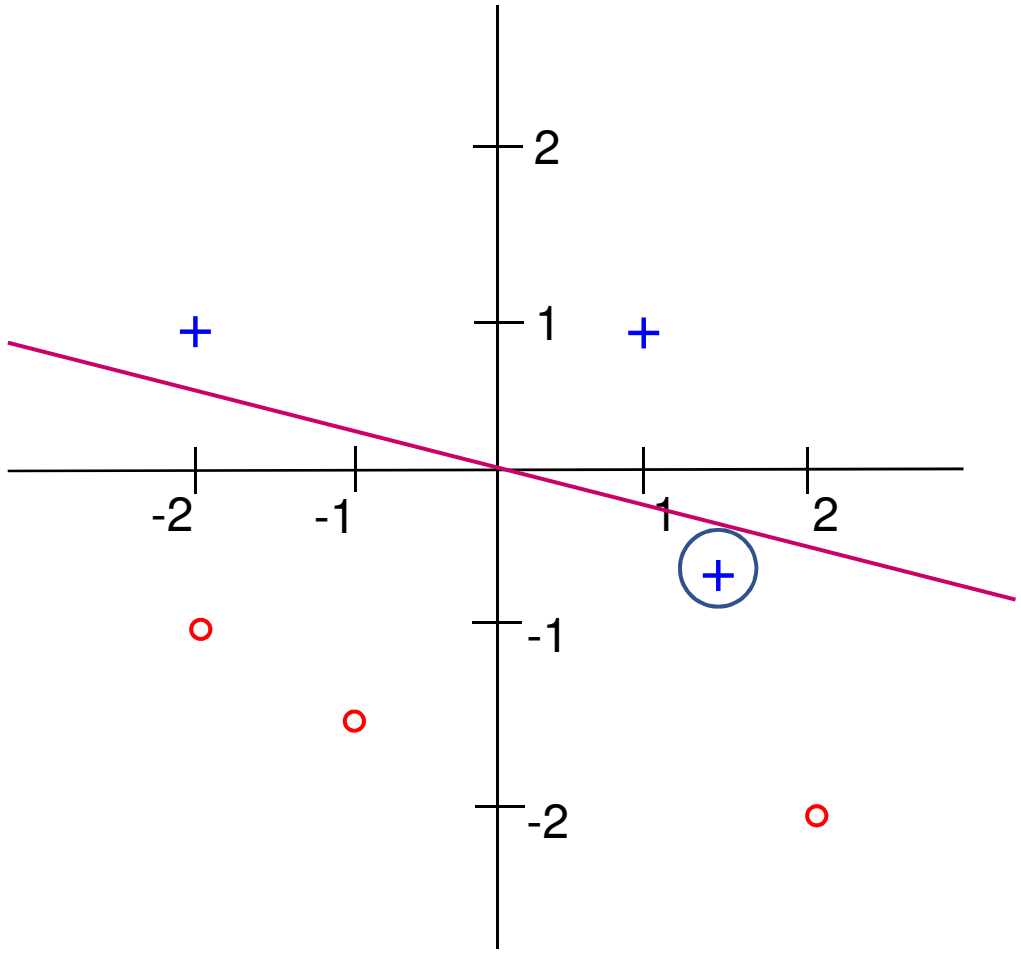
$$\mathbf{w} = \begin{pmatrix} 0 \\ 0.2 \\ 1.1 \end{pmatrix}$$

$$x_1 = 1.5,\ x_2 = 0.5$$

$$w_0 = w_0 - (-0.2) \cdot 1$$

$$w_1 = w_1 - (-0.2) \cdot (1.5)$$

$$w_2 = w_2 - (-0.2) \cdot (-0.5)$$



Perceptron learning example

- Now, all data points are correctly classified, stop

$$\alpha = 0.2$$

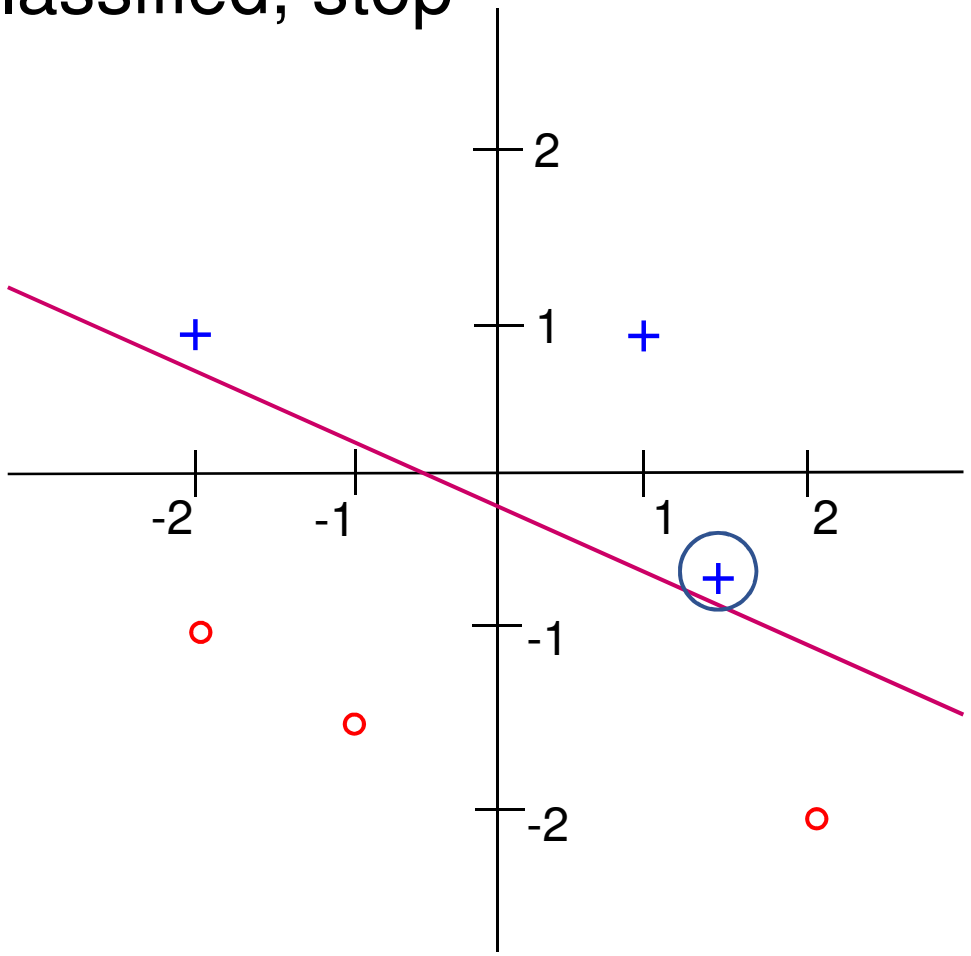
$$\mathbf{w} = \begin{pmatrix} 0.2 \\ 0.5 \\ 1 \end{pmatrix}$$

$$x_1 = 1.5,\ x_2 = 0.5$$

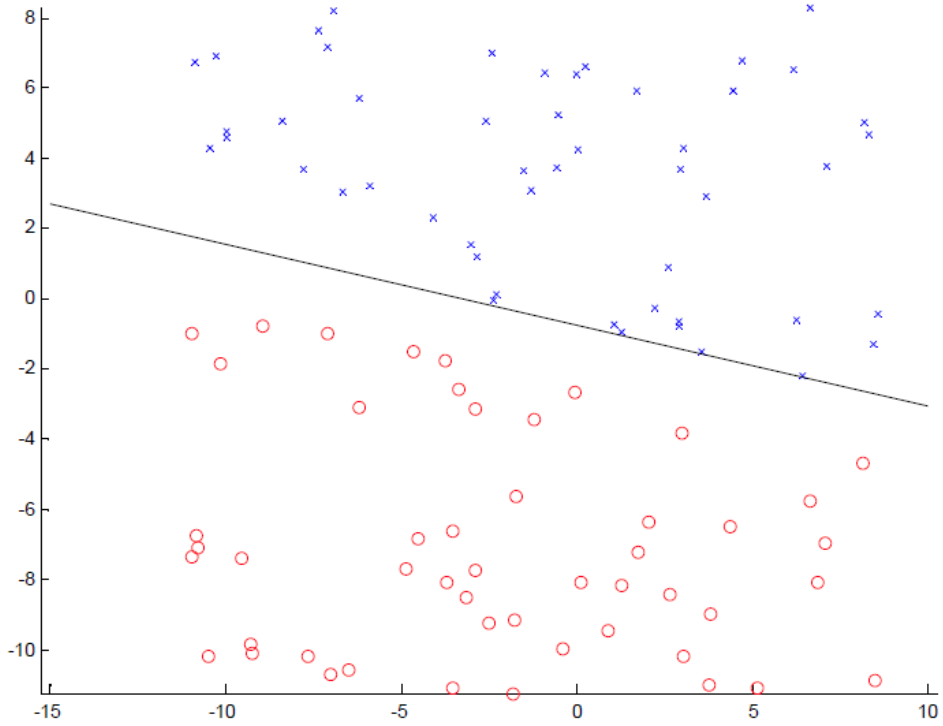
$$w_0 = w_0 - (-0.2) \cdot 1$$

$$w_1 = w_1 - (-0.2) \cdot (1.5)$$

$$w_2 = w_2 - (-0.2) \cdot (-0.5)$$



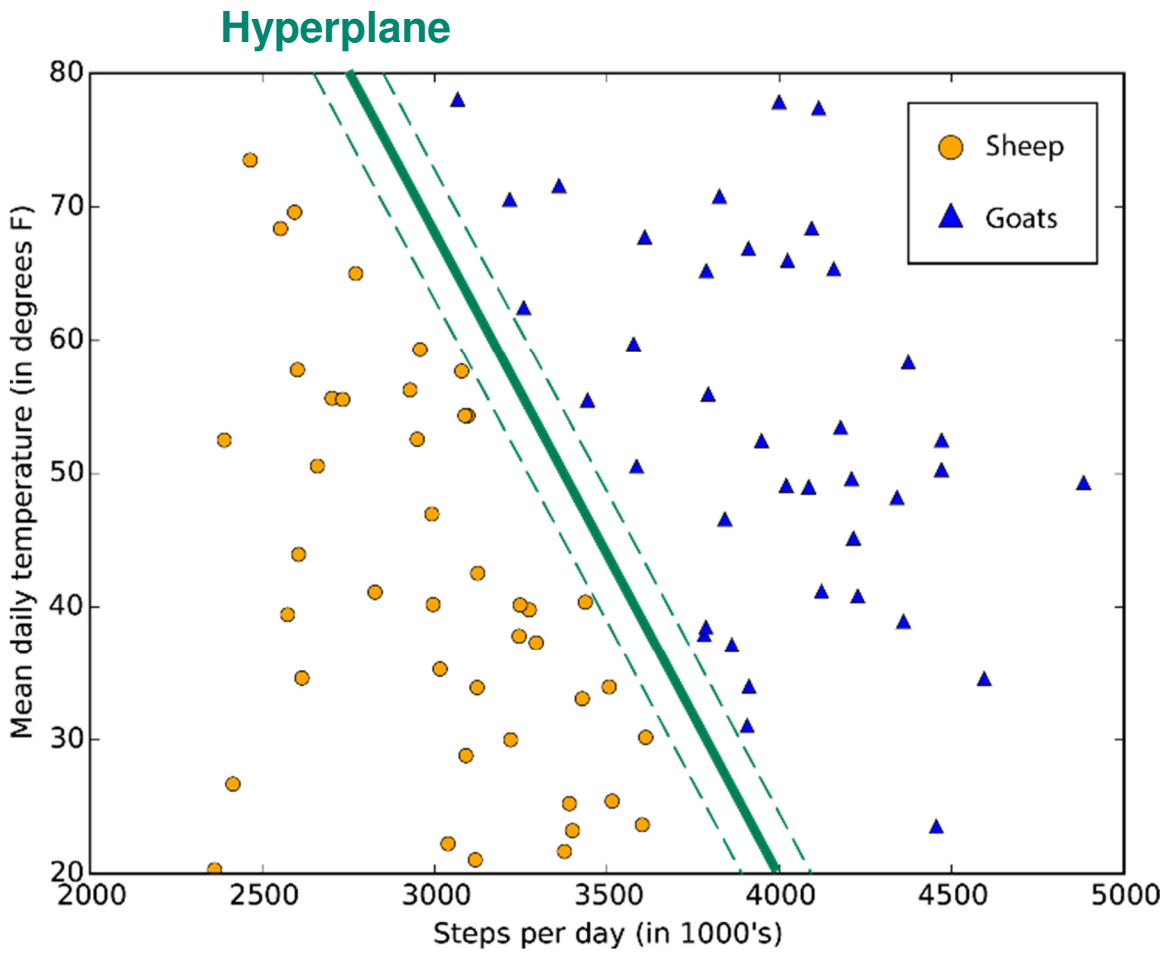
Perceptron learning example



- If data set is linearly separable, then the algorithm will converge, i.e., the perceptron learning rule is guaranteed to find a solution in a finite number of iterations
- Convergence can be slow…
- Separating line close to training data
- A larger margin for generalization is preferred

SVMs are a set of supervised learning methods used for classification, regression and outliers detection

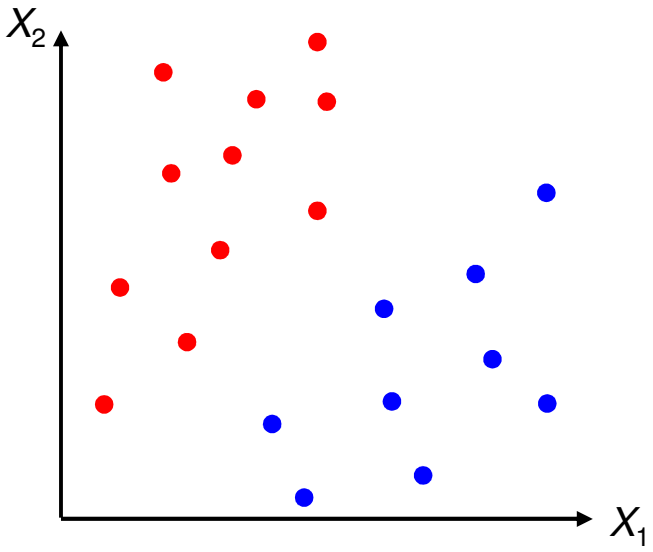
- Classifies data by finding the linear decision boundary (hyperplane) that separates all data points of one class from those of the other class
- The best hyperplane for an SVM is the one with the largest margin between the two classes, when the data is linearly separable
- Support vector machines (SVMs) find the boundary that separates classes by as wide a margin as possible. SVMs are capable of both classification and regression.
- Best used
  - For data that has exactly two classes (you can also use it for multiclass classification with a technique called error- correcting output codes)
  - High-dimensional, nonlinearly separable data
  - When you need a classifier that's simple, easy to interpret, and accurate



https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-choice

How to classify the data set in the following example?

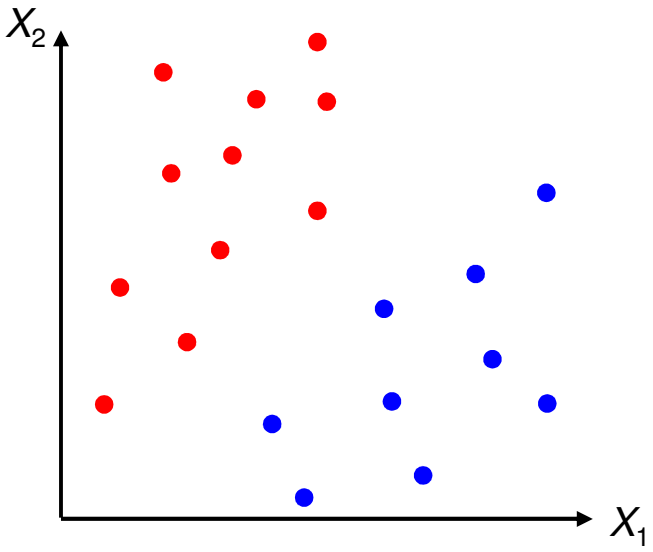
- There is a two dimensional data set with two classes. How to classify it into 2 classes correctly?



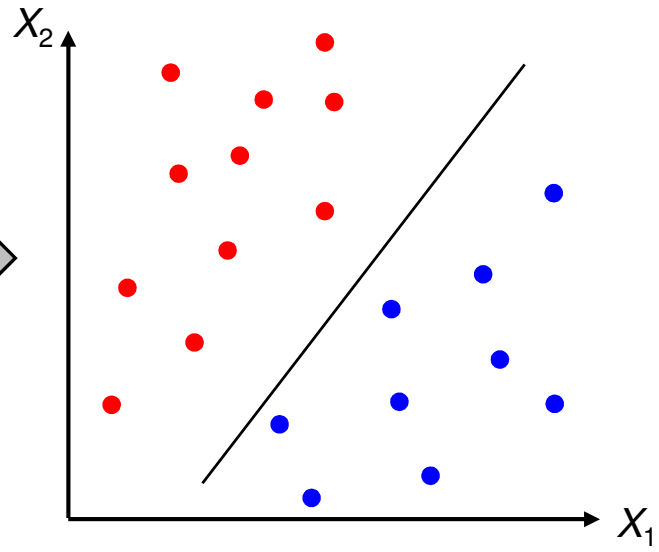
A two dimensional data set with two classes

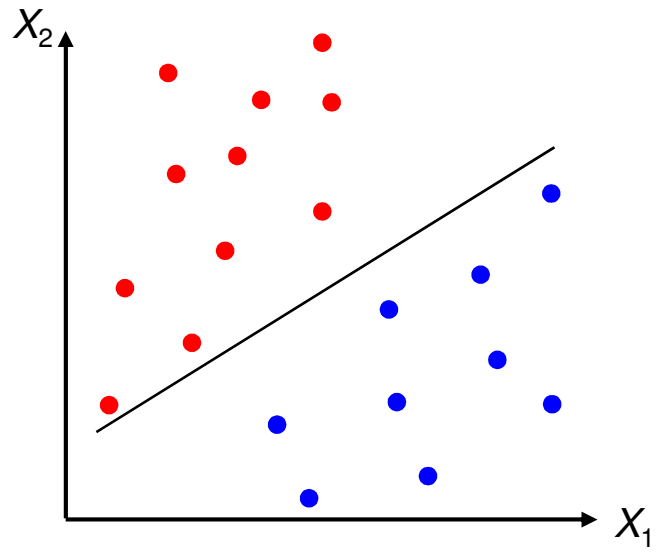
How to classify the data set in the following example?

- There is a two dimensional data set with two classes. How to classify it into 2 classes correctly?



A two dimensional data set with two classes

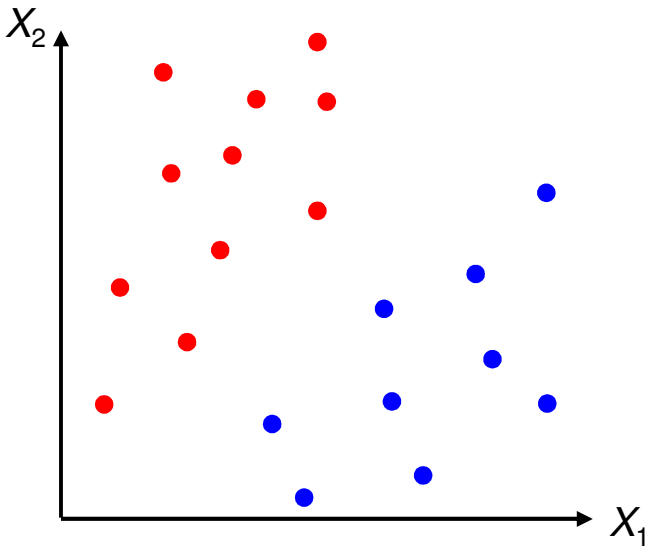




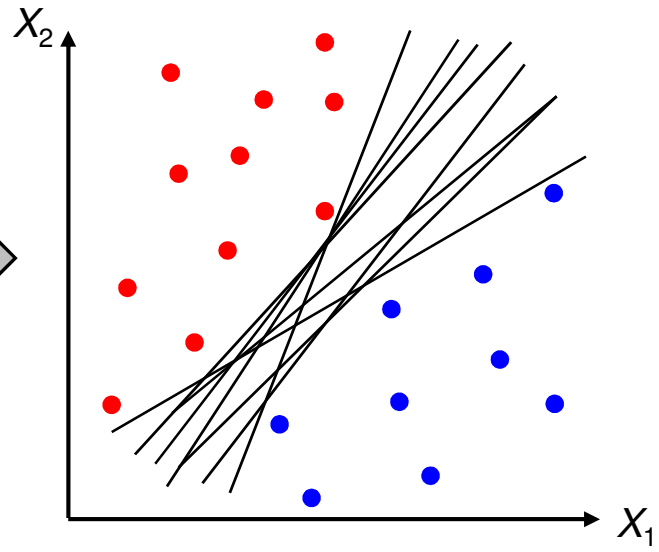
Both lines can be linear classifiers. Actually there are more lines that can be linear classifiers

Which line is the best classifier?

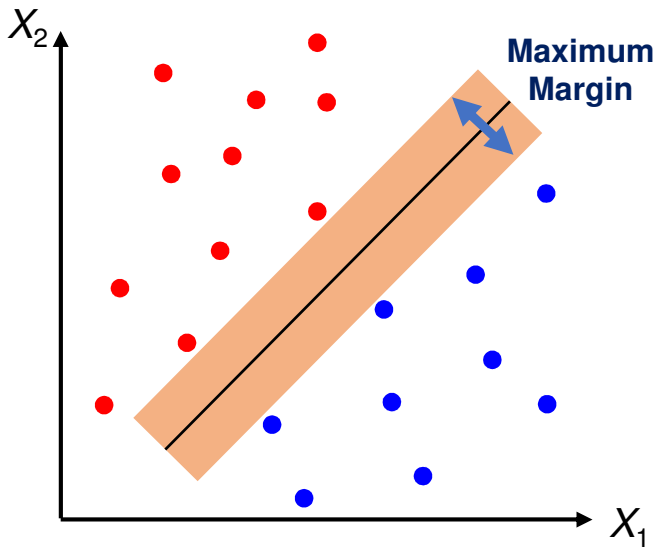
- There is a two dimensional data set with two classes. How to classify it into 2 classes correctly?



A two dimensional data set with two classes



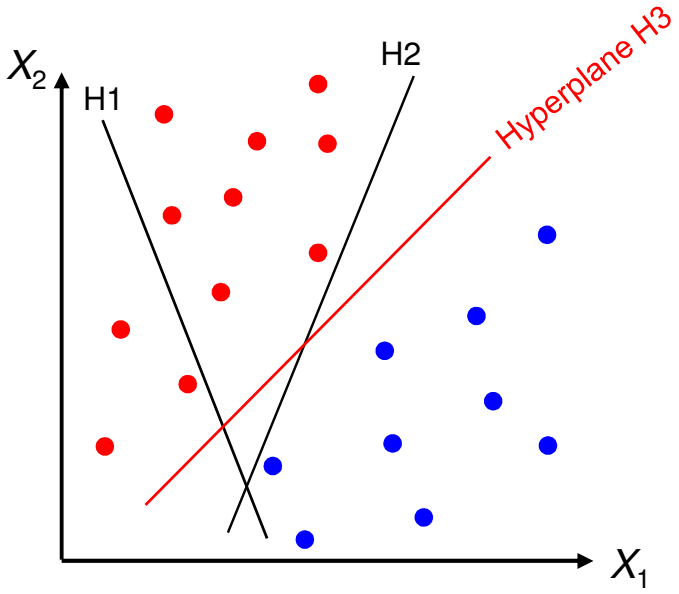
There are many lines that can be linear classifiers.
Which one is the optimal classifier



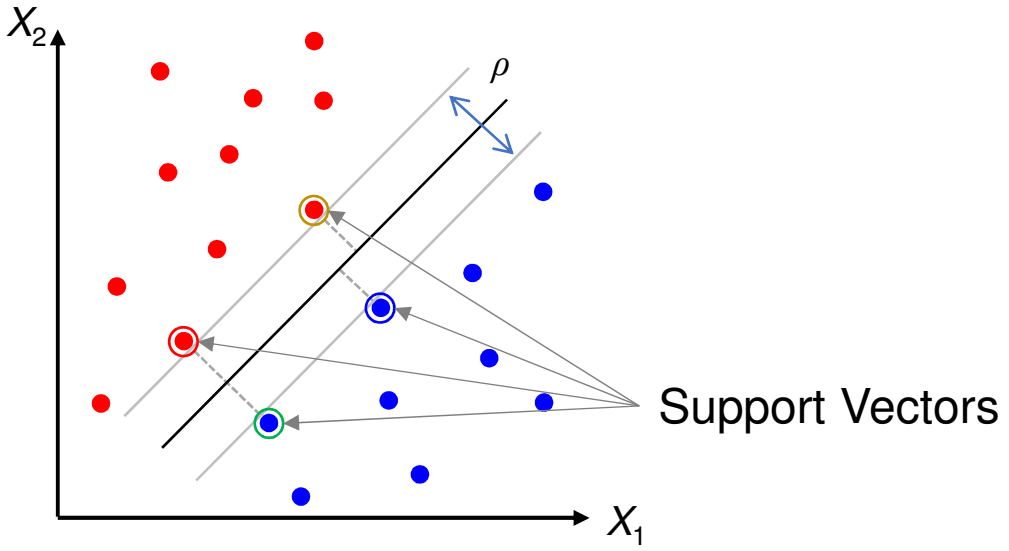
- Define the margin of a linear classifier as the width that the boundary could be increase by before hitting a data point
- The maximum margin linear classifier is the simplest kind of SVM (called Linear SVM)

Select the hyperplane which segregates the two classes better

- A good separation is achieved by the hyper-plane that has the largest distance to the nearest training data points of any class (so-called functional margin)



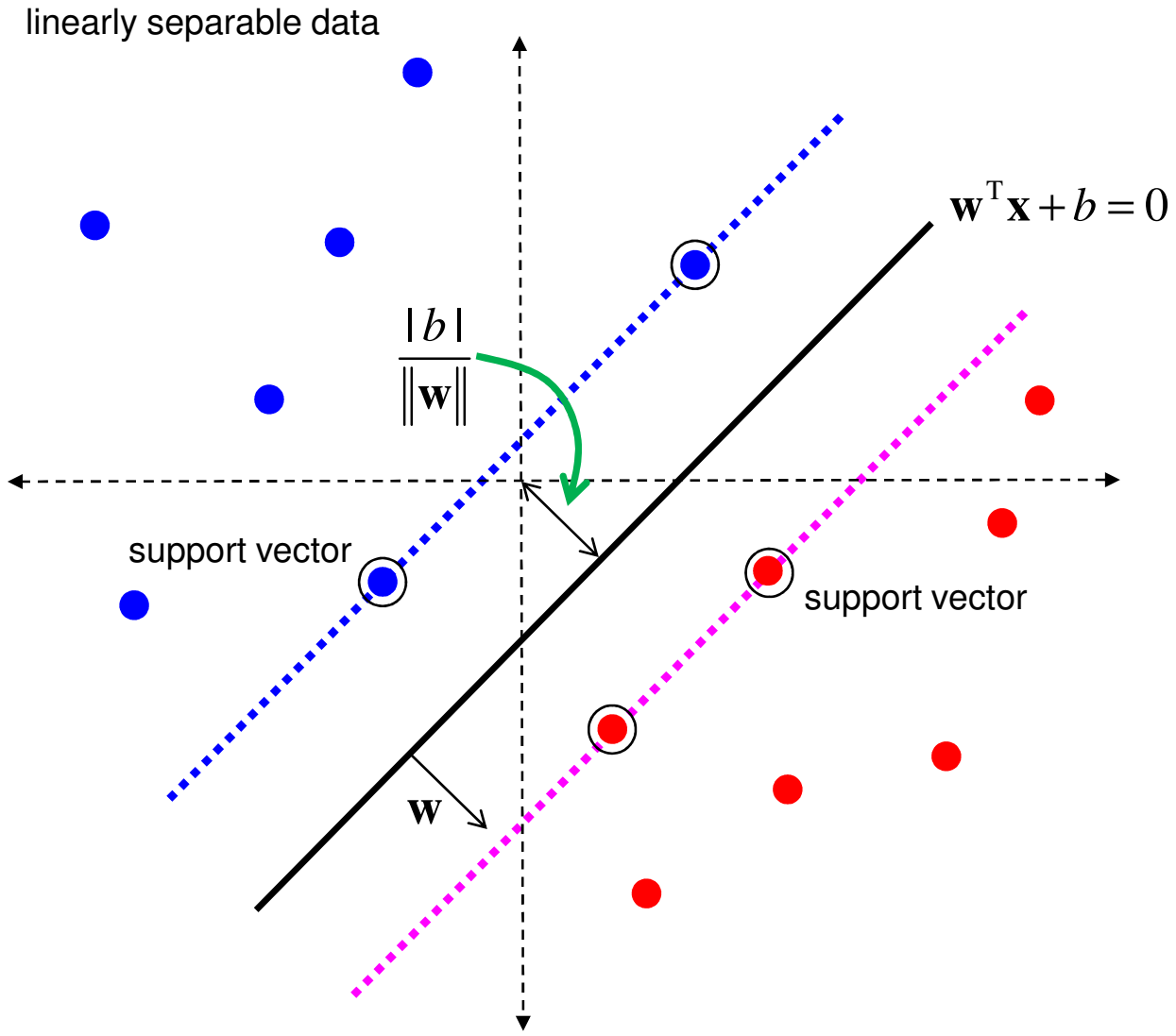
- H1 does not separate the classes
- H2 does, but only with a small margin.
- H3 separates them with the maximum margin.



- Examples closet to the hyperplane are **support vectors**
- Margin ρ of the separator is the distance between support vectors.

- Maximum margin solution: most stable under perturbations of the inputs

SVM -- sketch deviation



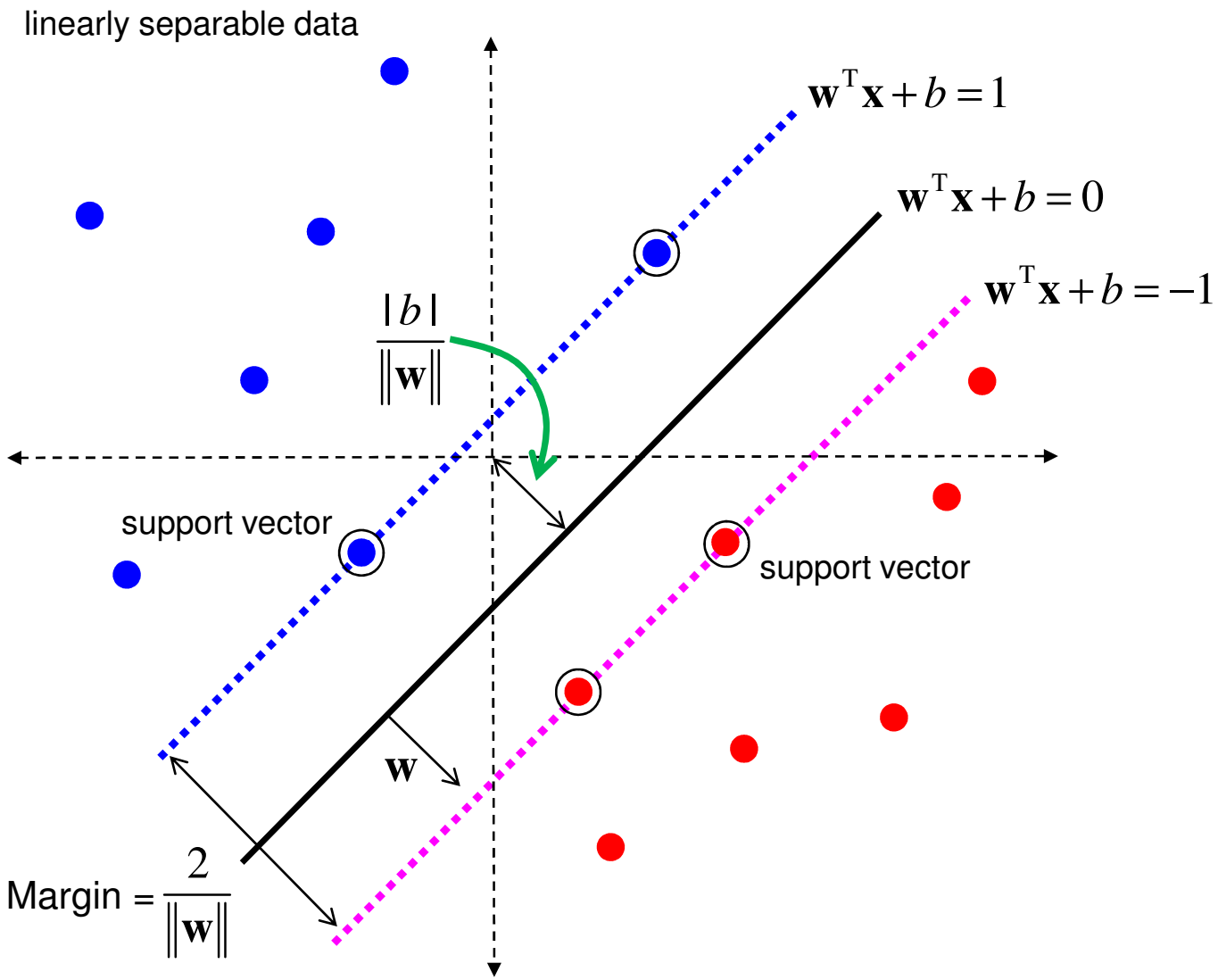
$$f(x)=\sum_i \alpha_i y_i (\mathbf{x}_i{}^T \mathbf{x}) + b$$

support vector

- Since $\mathbf{w}^T\mathbf{x}+b=0$ and $c(\mathbf{w}^T\mathbf{x}+b)=0$ define the same plane, we have the freedom to choose the normalization of $\mathbf{w}$
- Choose normalization such that $\mathbf{w}^T\mathbf{x}_+ + b = +1$ and $\mathbf{w}^T\mathbf{x}_- + b = -1$ for the positive and negative support vectors respectively
- Then the margin is given by

$$\frac{\mathbf{w}}{\|\mathbf{w}\|}\cdot(\mathbf{x}_+ - \mathbf{x}_-) = \frac{\mathbf{w}^T(\mathbf{x}_+ - \mathbf{x}_-)}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}$$

SVM -- sketch deviation



$$f(x)=\sum_i \alpha_i y_i (\mathbf{x}_i{}^T\mathbf{x})+b$$

support vector

- Since $\mathbf{w}^T\mathbf{x}+b=0$ and $c(\mathbf{w}^T\mathbf{x}+b)=0$ define the same plane, we have the freedom to choose the normalization of $\mathbf{w}$
- Choose normalization such that $\mathbf{w}^T\mathbf{x}_+ +b=+1$ and $\mathbf{w}^T\mathbf{x}_- +b=-1$ for the positive and negative support vectors respectively
- Then the margin is given by

$$\frac{\mathbf{w}}{\|\mathbf{w}\|}\cdot(\mathbf{x}_+-\mathbf{x}_-)=\frac{\mathbf{w}^T(\mathbf{x}_+-\mathbf{x}_-)}{\|\mathbf{w}\|}=\frac{2}{\|\mathbf{w}\|}$$

SVM -- Optimization

- Learning the SVM can then be formulated as an optimization,

$$\max_{\mathbf{w}} \frac{2}{\|\mathbf{w}\|}$$

$$\text{subject to } \mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b \begin{cases} \geq +1, & \text{if } y_i = +1 \\ \leq -1, & \text{if } y_i = -1 \end{cases} \text{ for } i = 1, ..., N$$

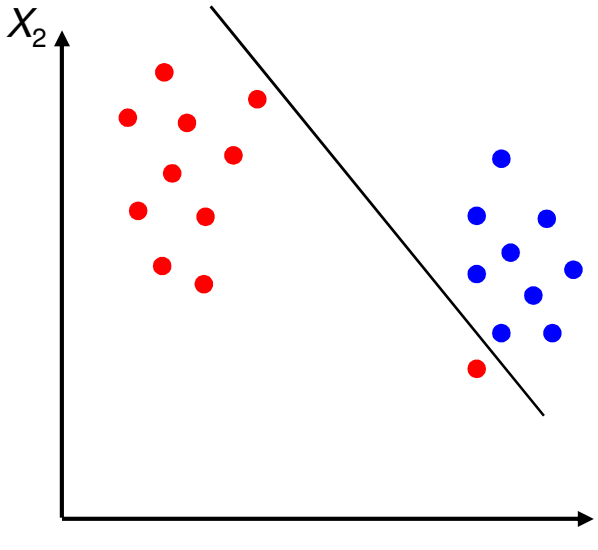
- Or equivalently $\min_{\mathbf{w}} \|\mathbf{w}\|^2$ subject to $y_i\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b\right) \geq 1$ for $i = 1, ..., N$

- This is a quadratic optimization problem subject to linear constraints and there is a unique minimum*

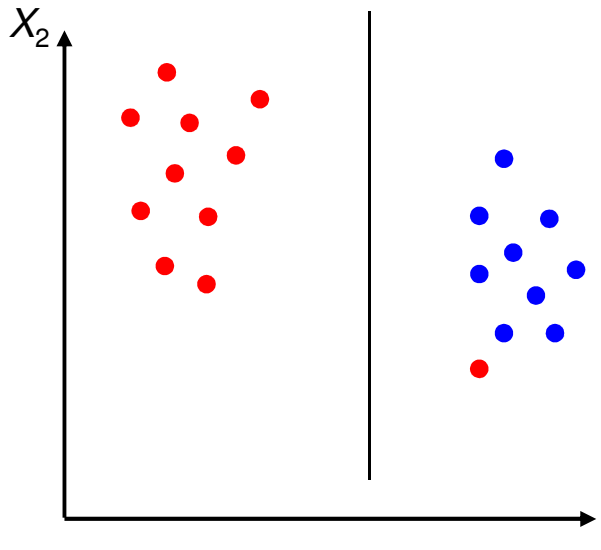
* A list of solvers could be found: https://en.wikipedia.org/wiki/Quadratic_programming

Which line is the best classifier? - again

- What is the best **w**?



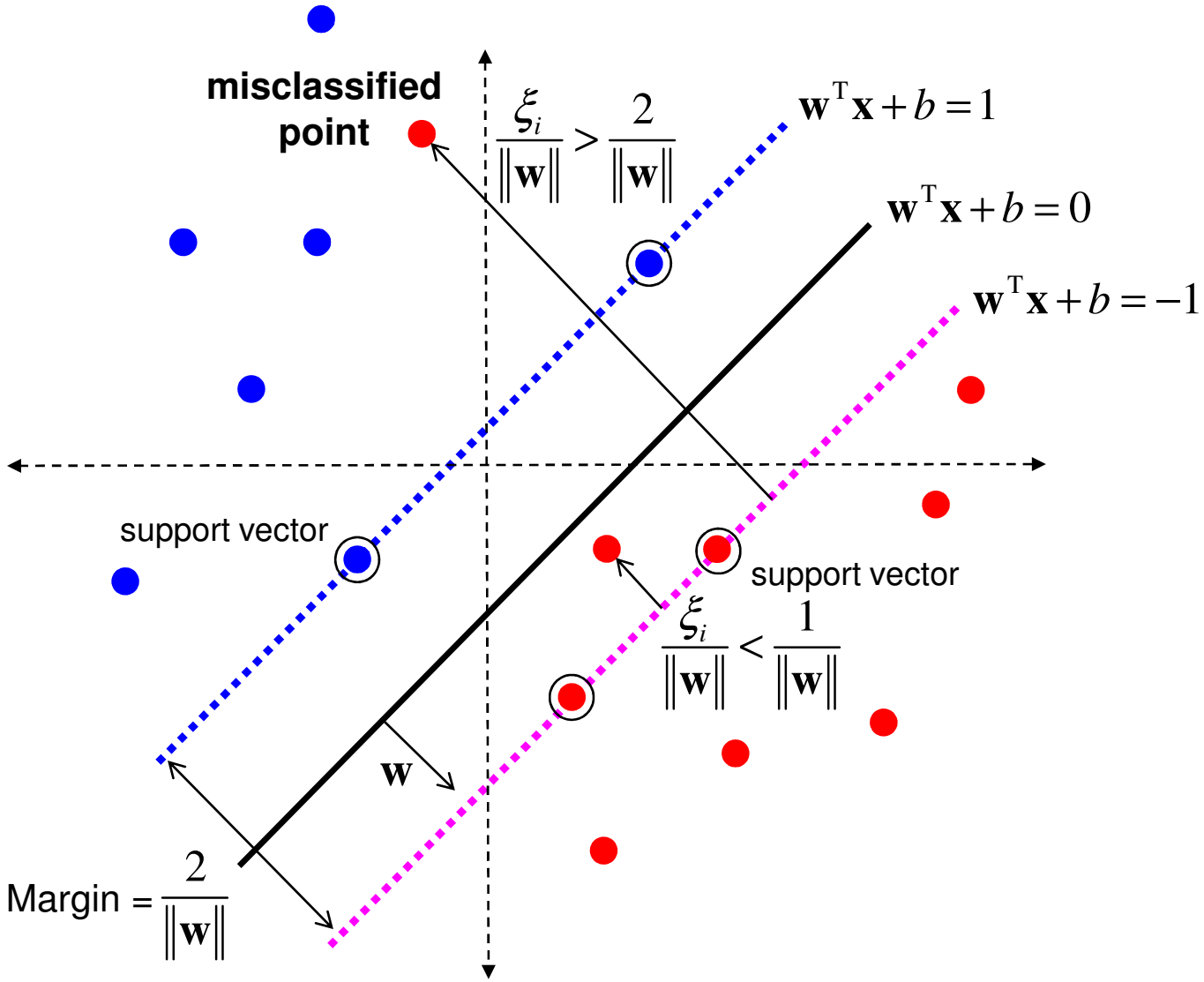
- the points can be linearly separated but there is a very narrow margin



- but possibly the large margin solution is better, even though one constraint is violated

- In general there is a trade off between the margin and the number of mistakes on the training data

Introduce "slack" variables



$$\xi_i \geq 0$$

- for $0 < \xi \leq 1$ point is between margin and correct side of hyperplane. This is a margin violation
- for $\xi > 1$ point is misclassified

“Soft” margin solution

- The optimization problem becomes

$$\min_{\mathbf{w}\in\mathbf{R}^d,\,\xi_i\in\mathbf{R}^+} \|\mathbf{w}\|^2 + C\sum_i^N \xi_i$$

$$\text{subject to } y_i\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b\right) \geq 1-\xi_i \text{ for } i = 1, \ldots, N$$

- Every constraint can be satisfied if ξ_i is sufficiently large
- C is a regularization parameter
  - Small C allows constraints to be easily ignored → large margin
  - Large C makes constraints hard to ignore → narrow margin
  - $C = \infty$ enforces all constraints → hard margin
- Parameter C can be viewed as a way to control *overfitting*: it “trades off” the relative importance of maximizing the margin and fitting the training data.
- This is still a quadratic optimization problem and there is a unique minimum. Note that there is only one parameter, C

Optimization

- Learning an SVM has been formulated as a constrained optimization problem over $\mathbf{w}$ and ξ

$$\min_{\mathbf{w}\in\mathbf{R}^d,\,\xi_i\in\mathbf{R}^+} \|\mathbf{w}\|^2 + C\sum_i^N \xi_i \text{ subject to } y_i\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b\right) \geq 1-\xi_i \text{ for } i = 1, \ldots, N$$

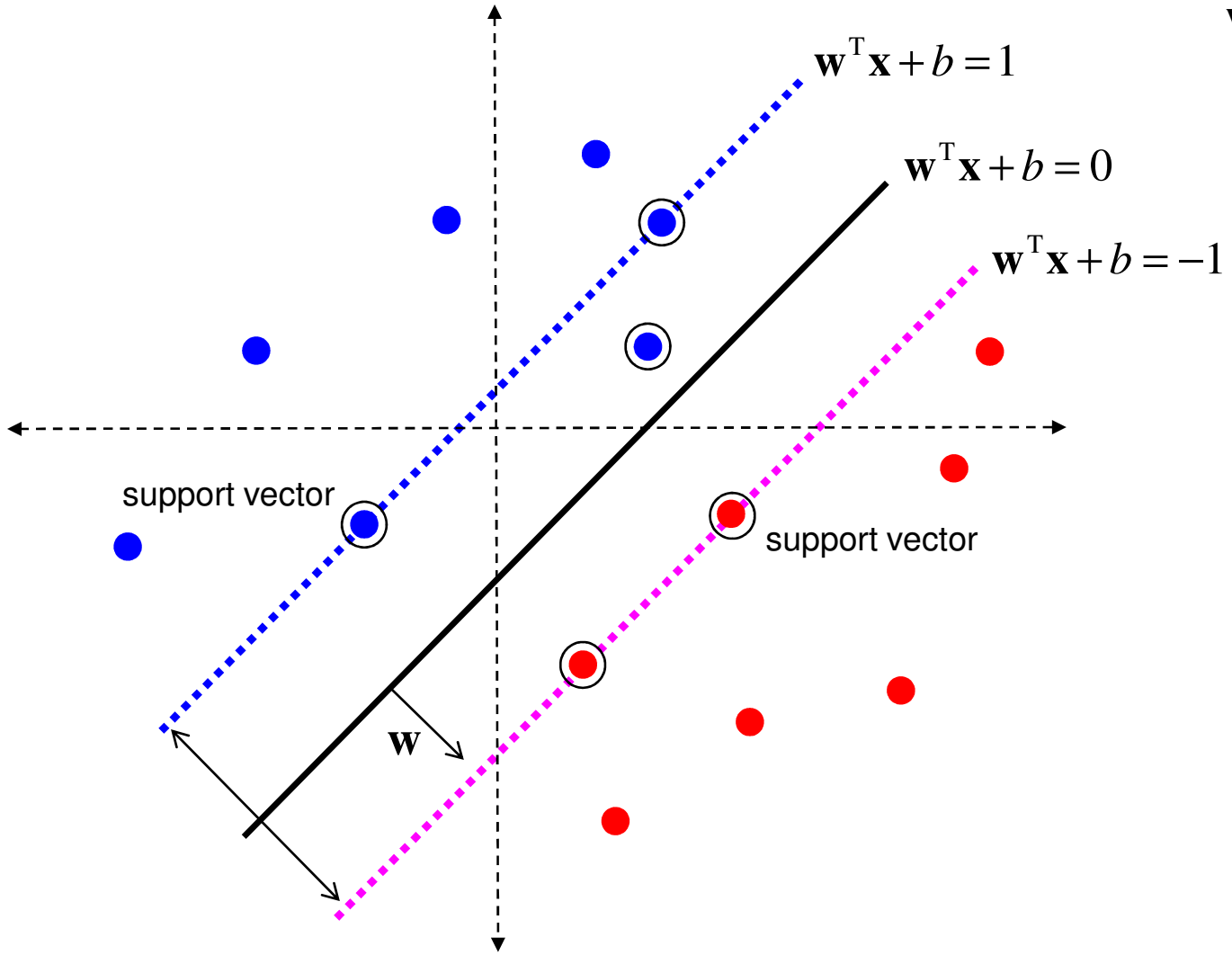
- The constraint $y_i\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b\right) \geq 1-\xi_i$, can be written more concisely as $y_i f\left(\mathbf{x}_i\right) \geq 1-\xi_i$ which, together with $\xi_i \geq 0$, is equivalent to $\xi_i = \max\left(0, 1- y_i f\left(\mathbf{x}_i\right)\right)$

- Hence the learning problem is equivalent to the unconstrained optimization problem over $\mathbf{w}$

$$\min_{\mathbf{w}\in\mathbf{R}^d}\left(\underbrace{\|\mathbf{w}\|^2}_{\text{regularization}} + C\sum_i^N \underbrace{\max\left(0, 1- y_i f\left(\mathbf{x}_i\right)\right)}_{\text{loss function}}\right)$$

Loss function

$$\min_{\mathbf{w}\in\mathbf{R}^d}\left(\|\mathbf{w}\|^2 + C\sum_i^N \underbrace{\max\left(0, 1-y_i f(\mathbf{x}_i)\right)}_{\text{loss function}}\right)$$

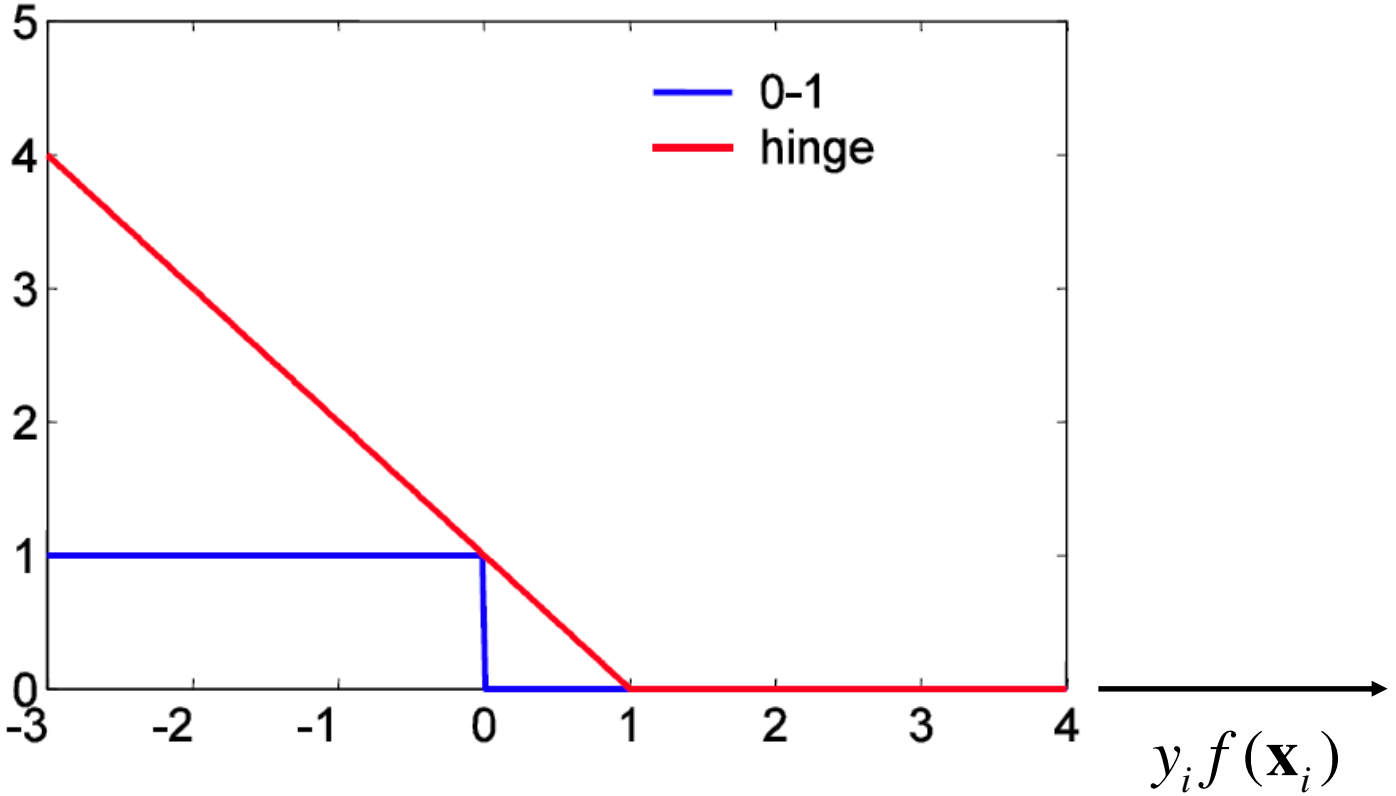


Points are in three categories:

1. $y_i f(x_i) > 1$ point is outside margin and no contribution to loss
2. $y_i f(x_i) = 1$ point is on margin and no contribution to loss as in hard margin case
3. $y_i f(x_i) < 1$ point violates margin constraint and contributes to loss

Loss functions

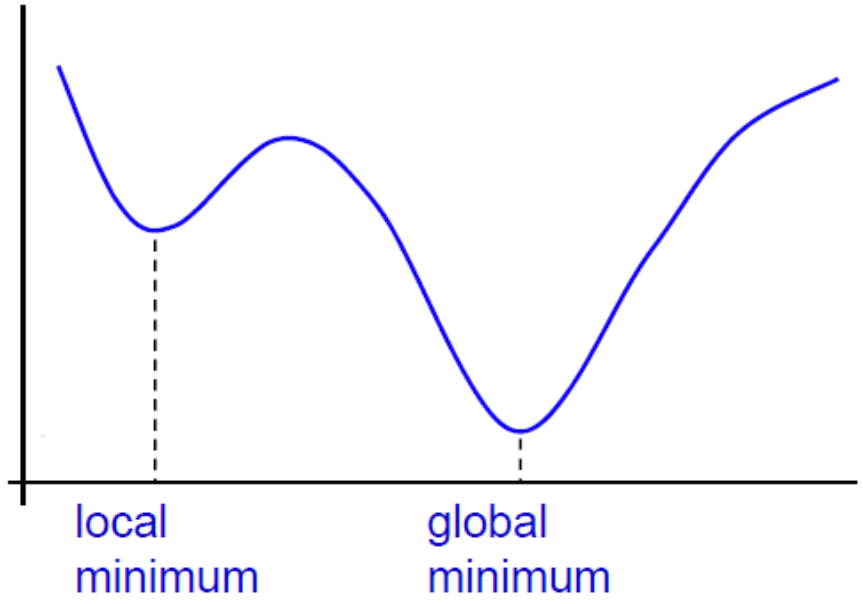
- SVM uses "hinge" loss $\max\left(0, 1 - y_i f(\mathbf{x}_i)\right)$
- An approximation to the 0-1 loss



Optimization (cont.)

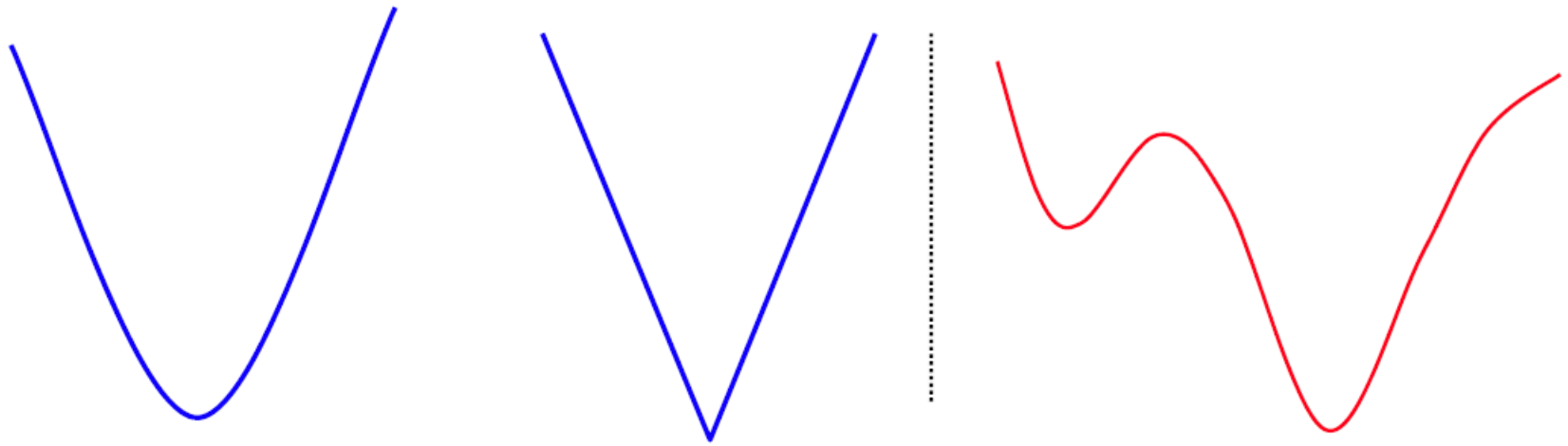
$$\min_{\mathbf{w}\in\mathbf{R}^d}\left(\|\mathbf{w}\|^2 + C\sum_i^N \max\left(0, 1 - y_i f(\mathbf{x}_i)\right)\right)$$

- Does this cost function have a unique solution?
- Does the solution depend on the starting point of an iterative optimization algorithm (such as gradient descent)?
- If the cost function is convex, then a locally optimal point is globally optimal (provided the optimization is over a convex set, which it is in our case)



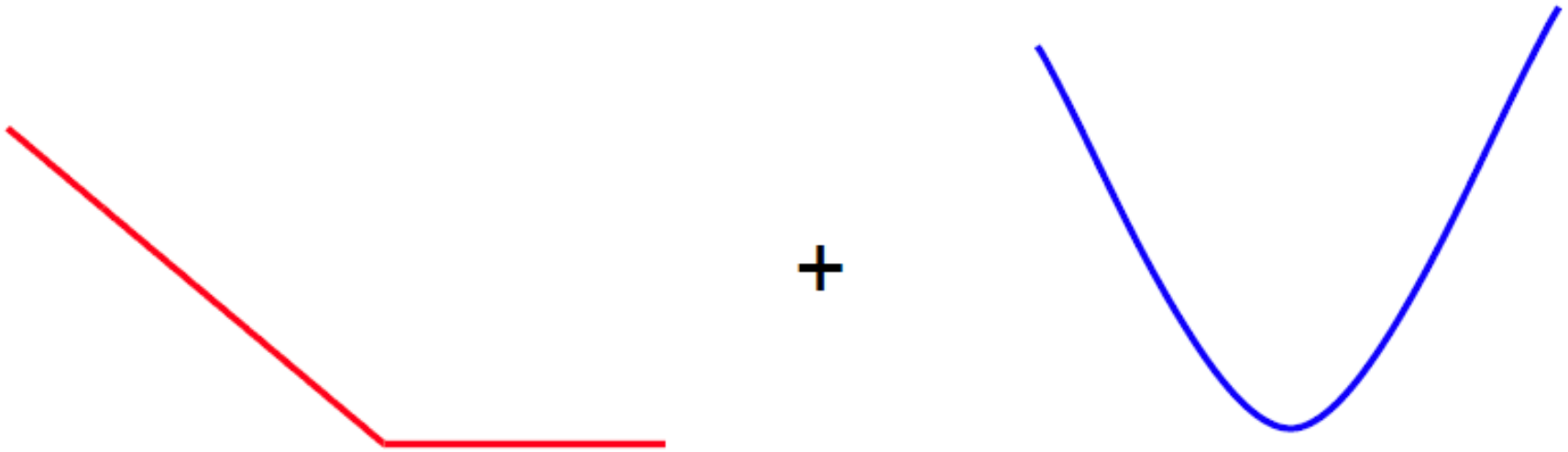
Convex function examples

- A non-negative sum of convex functions is convex



SVM

- $$\min_{\mathbf{w}\in\mathbf{R}^d}\left(\|\mathbf{w}\|^2 + C\sum_i^N \max\left(0, 1 - y_i f(\mathbf{x}_i)\right)\right)$$
- Convex

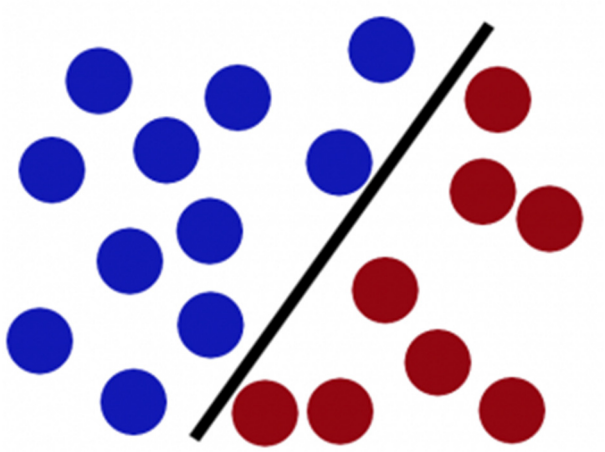


- Gradient (or steepest) descent algorithm for SVM (e.g., Pegasos*)

* Solver download link: https://www.cs.huji.ac.il/~shais/code/pegasos.tgz

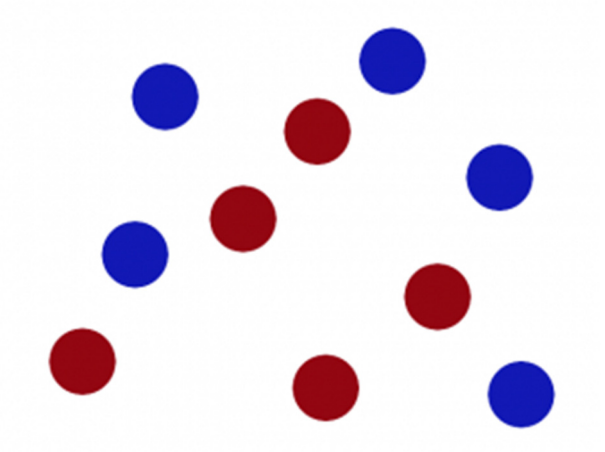
There is a new challenge!

- How to classify the nonlinearly separable data by the hyperplane?



Use Linear SVM algorithm

The linear SVM can work well on classifying this data set



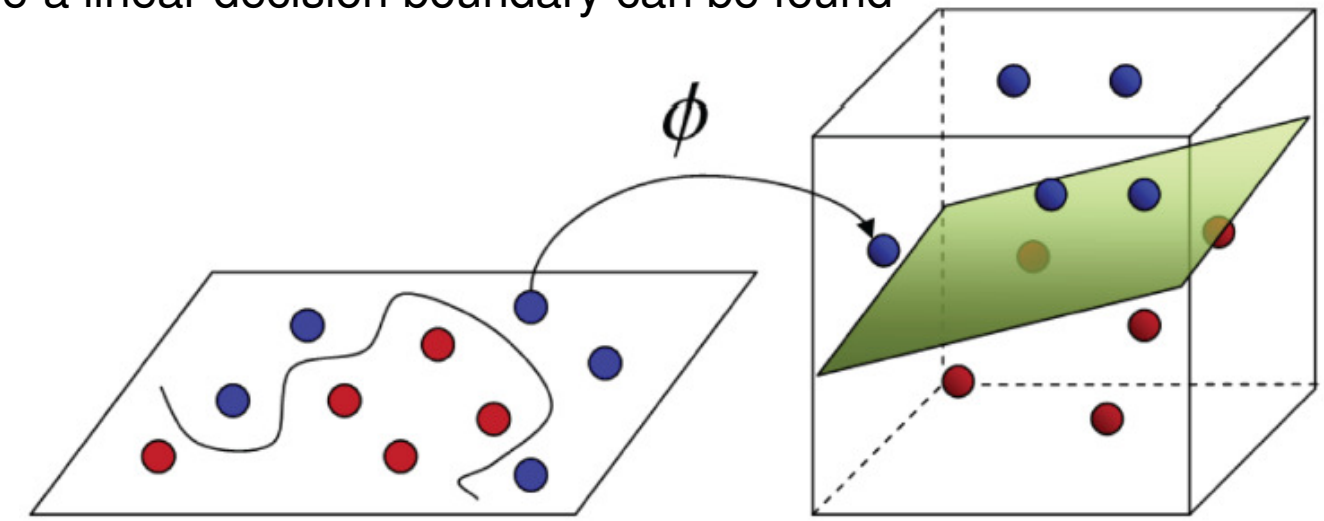
How to classify this data set?

It seems that it's hard to classify this data set via the linear classifier

Image source: http://bytesizebio.net/2014/02/05/support-vector-machines-explained-well/

Create nonlinear classifiers by applying the kernel trick

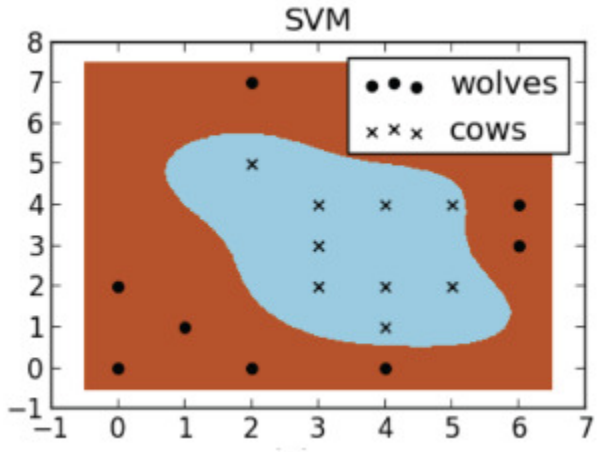
- SVMs sometimes use a kernel transform to transform nonlinearly separable data into higher dimensions where a linear decision boundary can be found

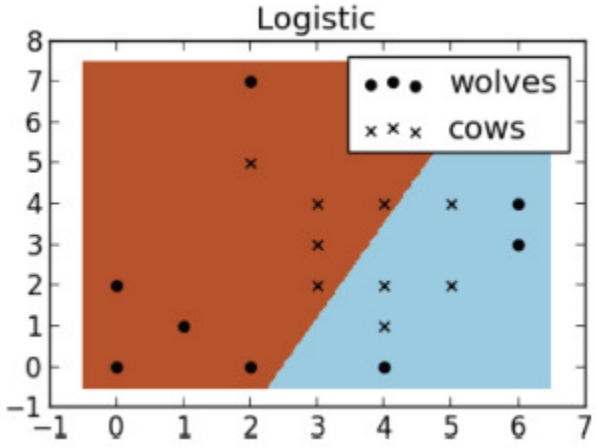


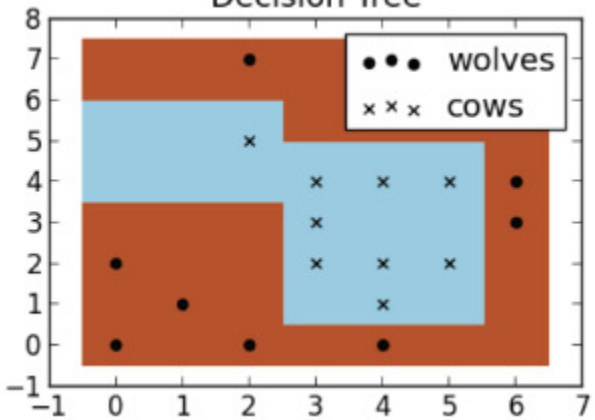
- Kernel trick allows the algorithm to fit the maximum-margin hyperplane in a transformed feature space. Although the classifier is a hyperplane in the transformed feature space, it may be nonlinear in the original input space
- SVM with polynomial kernel visualization
  - https://www.youtube.com/watch?v=3liCbRZPrZA
  - https://www.youtube.com/watch?v=9NrALgHFwTo

Example: SVM using a non-linear kernel

- If you are a farmer and you need to set up a fence to protect your cows from packs of wolves,







- **Where do you build your fence?**
  - If you're a really data driven farmer one way you could do it would be to build a classifier based on the position of the cows and wolves in your pasture. Trying a few different types of classifiers, we see that SVM does a great job at separating your cows from the packs of wolves. I thought these plots also do a nice job of illustrating the benefits of using a non-linear classifiers
  - You can see the logistic and decision tree models both only make use of straight lines

Image source: https://www.kdnuggets.com/2017/02/yhat-support-vector-machine.html

Summary

- $$f(x) = \sum_i \alpha_i y_i (\mathbf{x}_i^{\mathrm{T}} \mathbf{x}) + b$$

- Some online hands-on tutorials about using SVM
  - http://scikit-learn.org/stable/
  - http://scikit-learn.org/stable/modules/svm.html
  - https://www.kdnuggets.com/2017/02/yhat-support-vector-machine.html
  - http://www.semspirit.com/artificial-intelligence/machine-learning/regression/support-vector-regression/support-vector-regression-in-python/

Thanks ! ☺

Questions ?