Large-Scale Machine Learning

Big Data Analytics CSCI 4030

New Topic: Machine Learning!

High dim. data	Graph data	Infinite data	Machine learning	Apps
Locality sensitive hashing	**PageRank,** SimRank	Filtering data streams	SVM	Recommender systems
Clustering	Community Detection	Web advertising	Decision Trees	Association Rules
Dimensionality reduction	Spam Detection	Queries on streams	Perceptron, kNN	Duplicate document detection

Machine Learning

- Many algorithms are today classified as **machine learning** ones.
  - These algorithms extract information from data.
  - Produce summary of data, from which decision is made.
- Machine learning algorithms learn a **model** from the data.
  - Discover something about data that will be seen in the future..

Unsupervised learning

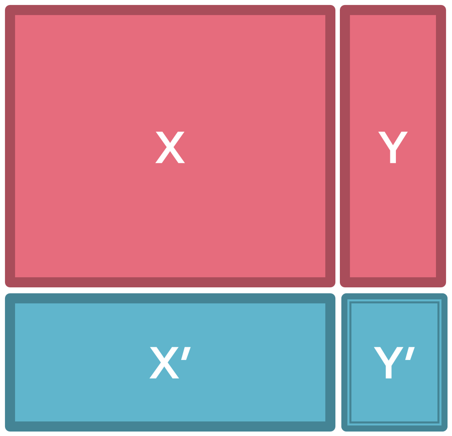
- E.g., the clustering algorithms allow us to classify **future data** into one of the clusters.
  - Machine learning enthusiast call it **unsupervised learning**.
- Unsupervised means that the input data does not tell the clustering algorithm what the clusters should be.

Supervised Learning

- In **supervised machine learning:**
  - The available data includes information about the correct way to classify at least some of the data.
  - The data classified already is called the **training set**.
- This is the main subject of today’s lecture.

Supervised Learning

- **Would like to do prediction:**
  estimate a function **f(x)** so that ***y = f(x)***
- **Where *y* can be:**
  - **Real number**: Regression
  - **Categorical**: Classification



Training and test set

Estimate *y = f(x)* on *X,Y.*
Hope that the same *f(x)*
also works on unseen X', Y'

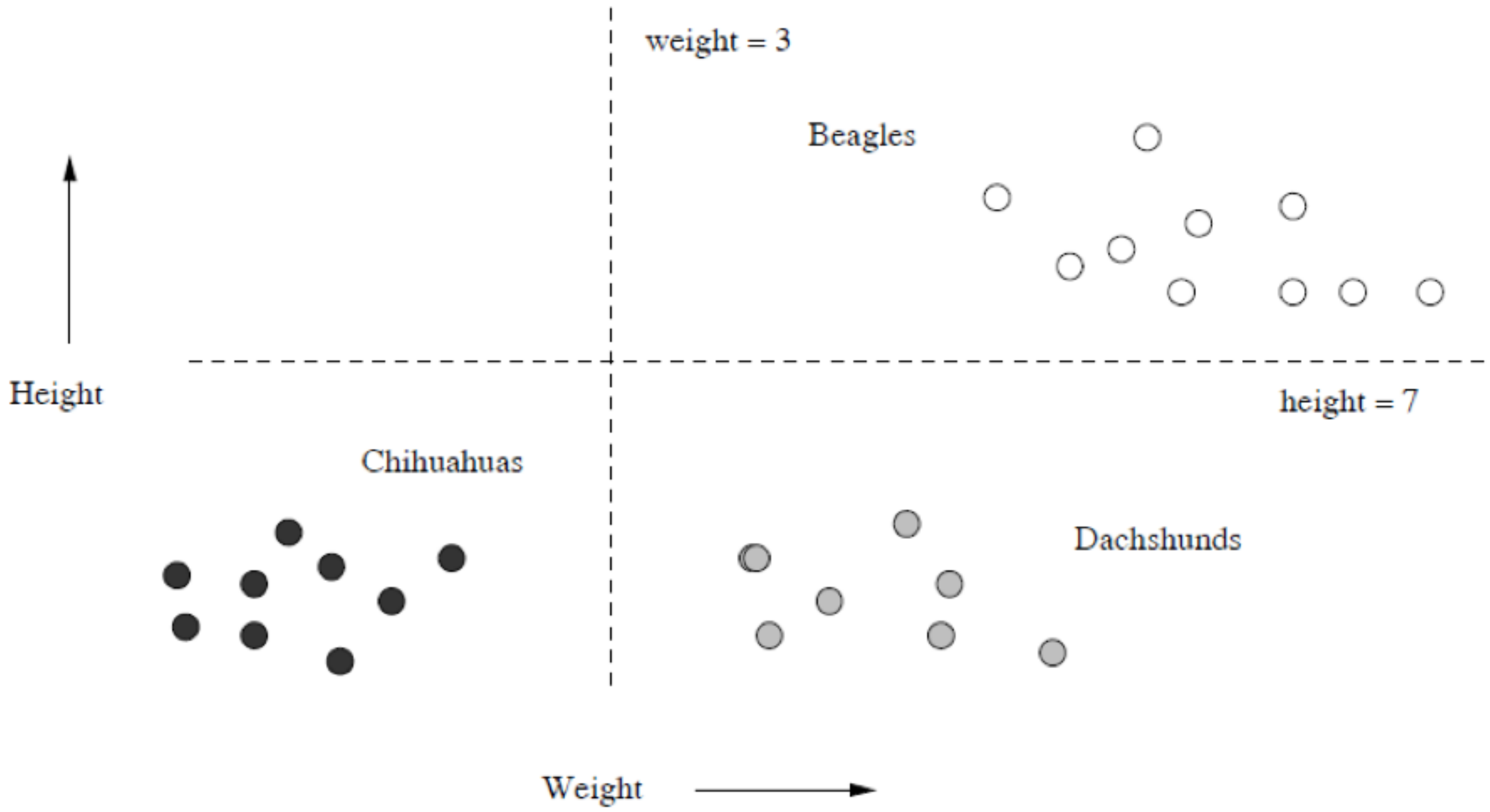
- **Data is labeled:**
  - Have many pairs **{(x, y)}**
    - **x** … vector of binary, categorical, real valued features
    - **y** … class ({+1, -1}, or a real number)

Chihuahua, Beagles, Dachshunds..

Learning Illustrative Example

- Plot the *height and weight* of dogs in three classes: **Beagles, Chihuahuas, and Dachshunds.**
- Each pair (**x**, y) in the training set consists of:
  - Feature vector **x** of the form **[height, weight].**
  - The associated label *y* is the variety of the dog.
- An example of a training-set pair would be **([5 inches, 2 pounds], Chihuahua).**

Heights and Weights of Certain Dogs



Decision Function

- The **horizontal line** represents a height of 7 inches and separates Beagles from Chihuahuas and Dachshunds.
- The **vertical line** represents a weight of 3 pounds and separates Chihuahuas from Beagles and Dachshunds.

Decision Function

- The algorithm that implements function *f* is:

```
if (height > 7) print Beagle
else if (weight < 3) print Chihuahua
else print Dachshund;
```

- Is it supervised on unsupervised learning?

Decision Function

- The algorithm that implements function *f* is:

```
if (height > 7) print Beagle
else if (weight < 3) print Chihuahua
else print Dachshund;
```

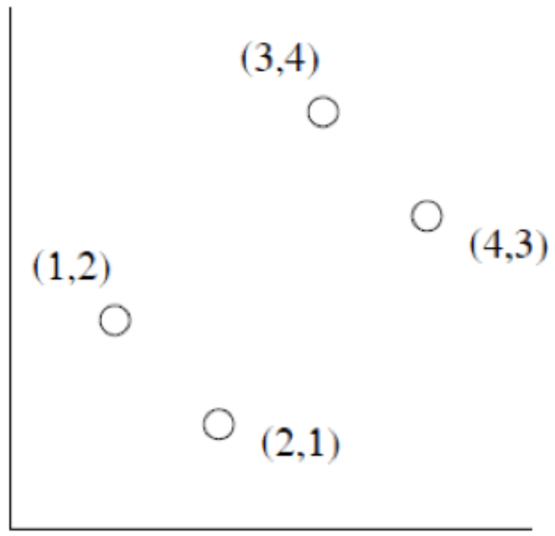
- Is it supervised on unsupervised learning?
  - Here, we are performing **supervised learning** with the same data (weight and height) augmented by classifications (variety) for the training data.

Type of *y* is Arbitrary

- *y* is a real number. ML problem is called **regression.**
- *y* is a boolean value true-or-false (+1 and −1). The problem is **binary classification.**
- *y* is a member of some finite set (classes). The problem is **multiclass classification.**

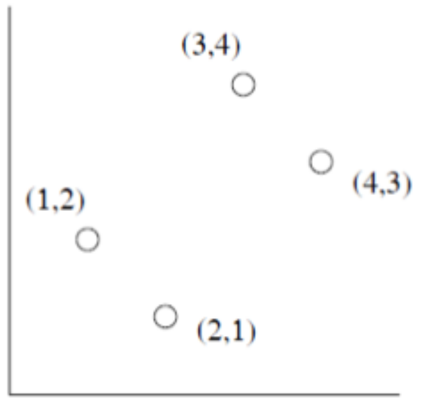
Example with Data Points

- Assume four data points: (1, 2), (2, 1), (3, 4) and (4, 3).



Interpretation of Data Points

- Let these points be a training dataset, where the vectors are one-dimensional.
- i.e., (1,2) can be thought as a pair ([1], 2), where [1] is feature vector **x** and 2 is the associated label *y*.
  - The other points are interpreted, accordingly.



RMSE

- Suppose we want to learn the linear function *f(x) = ax + b*
  - That best represents the point of the training set.
  - What is the appropriate value of *a* and *b*?
- A natural interpretation of best is **root-mean-square error (RMSE).**
  - the value of *f(x)* compared with given value of *y*.

Minimizing RMSE

- That is we want to minimize RMSE:

$$\sum_{x=1}^{4}(ax + b - y_x)^2$$

- This sum is:

$$(a + b - 2)^2 + (2a + b - 1)^2 + (3a + b - 4)^2 + (4a + b - 3)^2$$

- Simplifying the sum is:

$$30a^2 + 4b^2 + 20ab - 56a - 20b + 30$$

Minimizing RMSE

- If we then take the *derivatives* wrt *a* and *b* and set them to 0, we get:

$$\begin{aligned} 60a + 20b - 56 &= 0 \\ 20a + 8b - 20 &= 0 \end{aligned}$$

- Therefore, *a* = 3/5 and *b* = 1,
  - i.e., *f(x) = (3/5)x + 1*.
  - For these values the RMSE is 3.2.

Large Scale Machine Learning

- **We will talk about the following methods:**
  - Decision trees
  - Perceptrons
  - Support Vector Machines
  - Neural nets (Neural Networks)
  - Instance based learning

- **Main question:**
  How to efficiently train
  (build a model/find model parameters)**?**

Decision Trees

- The form of function *f* is a **tree.**
- Each node of the tree has a function of **x** that determines to which child or children the search must proceed.
- Decision trees are suitable for binary and multiclass classification.
  - Especially when the dimension of the feature vector is not too large.
  - Large numbers of features can lead to overfitting.

Perceptrons

- Perceptrons are threshold functions applied to the vector **x** = [x_1, x_2, . . ., x_i,… , x_n].
- A weight w_i is associated with the i-th component and there is a threshold **θ**
  - The output is +1 if

$$\sum_{i=1}^{n} w_i x_i \geq \theta$$

  - and the output is -1 otherwise.
- Suitable for binary classification.

Neural Nets

- **Neural nets** are acyclic networks of perceptrons
  - the outputs of some perceptrons are used as inputs to others.
- These are suitable for binary or multiclass classification.
  - since there can be several perceptrons used as output, with one or more indicating each class.

Instance Based Learning

- Uses the entire training set to represent the function *f*.
- An example is **k-nearest-neighbor.**
- For instance, 1-nearest-neighbor classifies data by giving it the same class as that of its nearest training example.

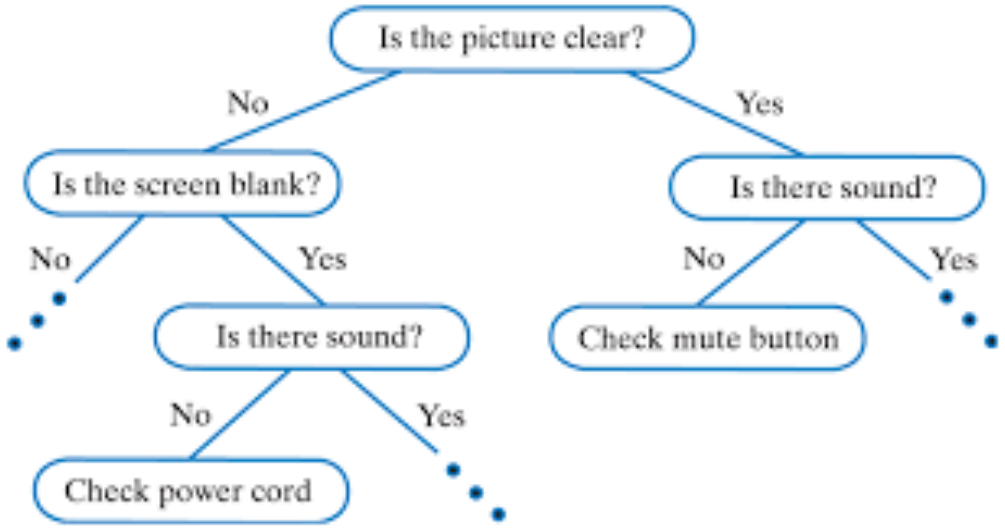
Support-Vector Machines

- **Support-vector machines** are an advance over the algorithms traditionally used to select the weights and threshold.
- The result is a classifier that tends to be more accurate on unseen data.

Decision Trees

Decision Trees

- A decision tree is a decision support tool that uses a tree-like graph.



TF-IDF

- The formal measure of how concentrated into relatively few documents are the occurrences of a given word is called TF.IDF (Term Frequency times Inverse Document Frequency).
- Suppose we have a collection of N documents. Define f_{ij} to be the frequency (number of occurrences) of term (word) *i* in document *j*.

TF-IDF

- Then, define the term frequency TF_{ij} to be:

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

- Suppose term *i* appears in n_i of the N documents in the collection.

$$IDF_i = \log_2(N/n_i)$$

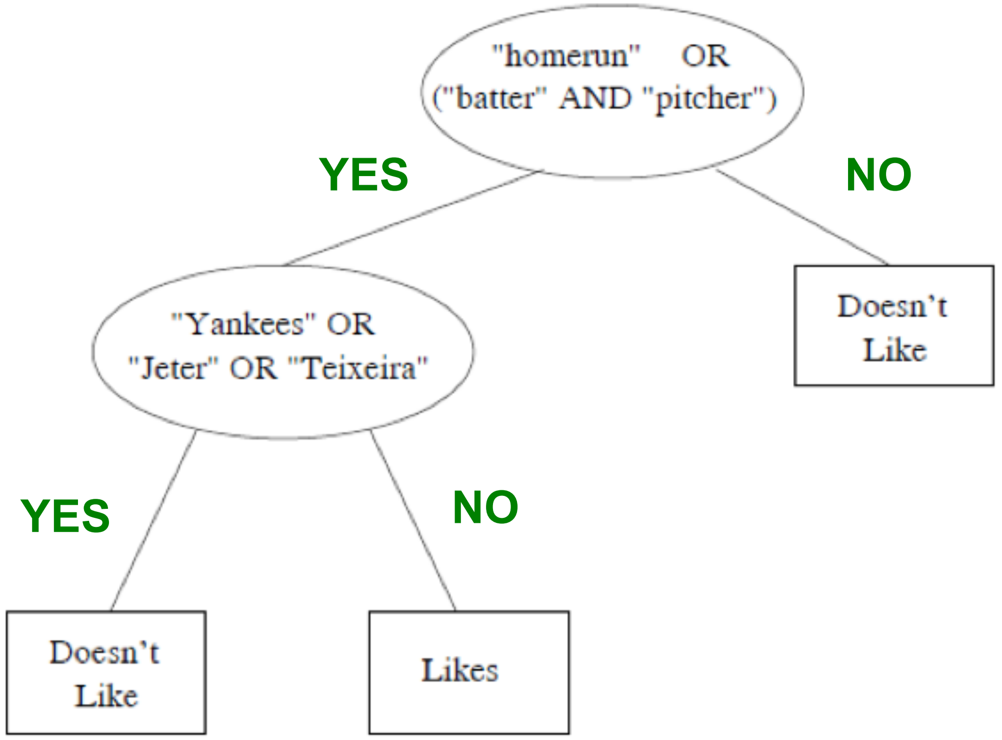
- Then, TF-IDF is defined as:

$$TF_{ij} \times IDF$$

News Articles: Likes and Dislikes

- Suppose items are news articles, and features are the high-TF.IDF words in those documents.
- Also suppose there is a user who likes articles about baseball, except articles about the New York Yankees..
- The row of the utility matrix for user has 1 if he has read the article and is blank if not.
  - We shall take the 1's as "like" and the blanks as "doesn't like."
- Predicates will be boolean expressions of words.

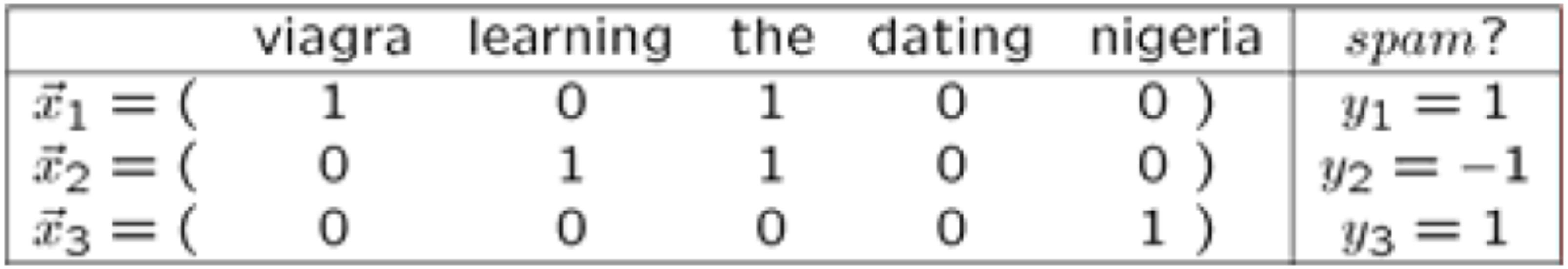
Decision Tree


"homerun" OR ("batter" AND "pitcher")
YES
NO
"Yankees" OR "Jeter" OR "Teixeira"
Doesn't Like
YES
NO
Doesn't Like
Likes

Perceptrons

Linear models: Perceptron

- Example: Spam filtering



	viagra	learning	the	dating	nigeria	*spam?*
$\vec{x}_1 = ($	1	0	1	0	0)	$y_1 = 1$
$\vec{x}_2 = ($	0	1	1	0	0)	$y_2 = -1$
$\vec{x}_3 = ($	0	0	0	0	1)	$y_3 = 1$

- Instance space x ∈ X (|**X**|= **n** data points)
  - **Binary or real-valued feature vector *x* of word occurrences**
  - ***d*** features (words + other things, **d**~100,000)
- Class y ∈ Y
  - ***y***: Spam (+1), Not Spam (-1)

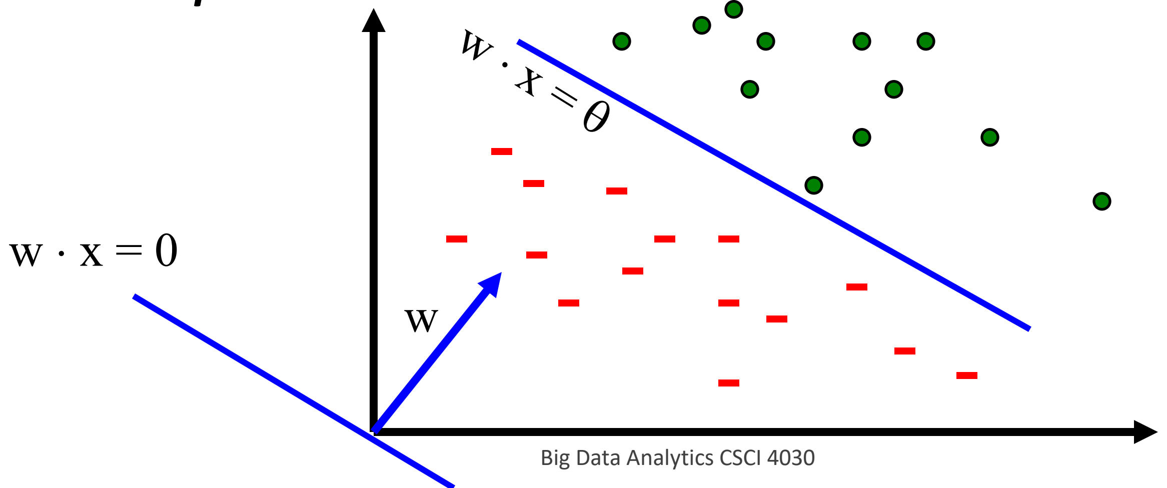
Linear models for classification

- **Binary classification:**

$$f(\mathbf{x}) = \begin{cases} +1 & \text{if} \quad \mathbf{w_1 x_1 + w_2 x_2 + \ldots w_d x_d} \geq \theta \\ -1 & \text{otherwise} \end{cases}$$

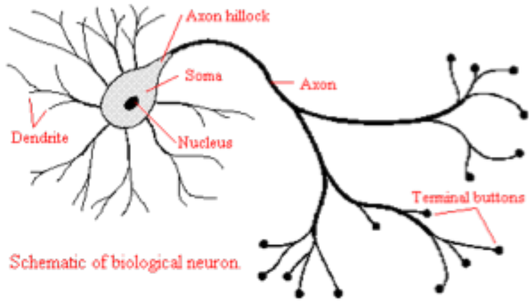
Decision boundary is **linear**

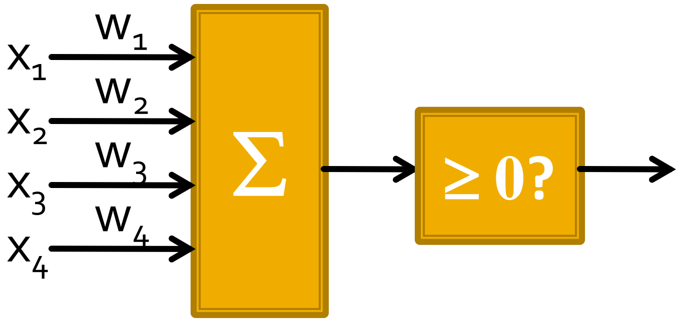
- **Input:** Vectors $\boldsymbol{x}^{(j)}$ and labels $\boldsymbol{y}^{(j)}$
  - Vectors $\boldsymbol{x}^{(j)}$ are real valued
- **Goal:** Find vector $w = (w_1, w_2, \ldots, w_d)$
  - Each $\boldsymbol{w}_i$ is a real number



Perceptron

- (very) Loose motivation: **Neuron**
- Inputs are feature values
- Each feature has a weight $\boldsymbol{w_i}$
- **Activation is the sum:**
  - $f(x) = \Sigma_i\, w_i\, x_i = w \cdot x$
- If the ***f(x)*** is:
  - **Positive:** Predict **+1**
  - **Negative:** Predict **-1**



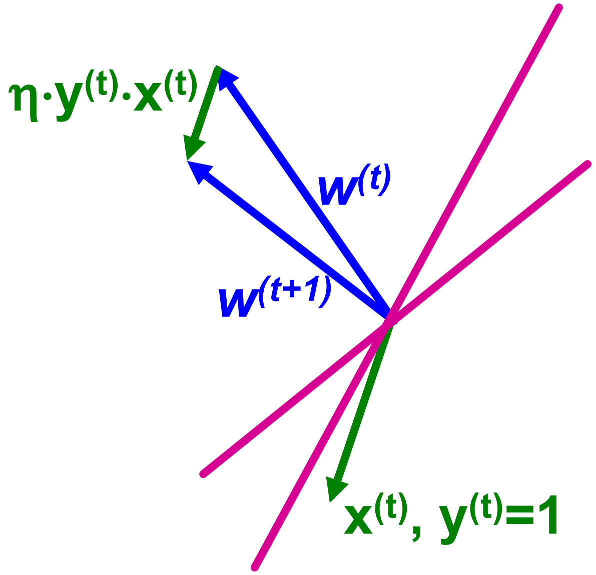


Perceptron: Estimating *w*

- **Perceptron:** $y' = sign(w \cdot x)$
- **How to find parameters *w*?**
  - Start with $w_0 = 0$
  - Pick training examples $x^{(t)}$ **one by one (from disk)**
  - Predict class of $x^{(t)}$ using current weights
    - $y' = sign(w^{(t)} \cdot x^{(t)})$
  - **If y' is correct (i.e., $y_t = y'$)**
    - No change: $w^{(t+1)} = w^{(t)}$
  - **If y' is wrong:** adjust $w^{(t)}$

$$w^{(t+1)} = w^{(t)} + \eta \cdot y^{(t)} \cdot x^{(t)}$$

    - η is the learning rate parameter
    - $x^{(t)}$ is the t-th training example
    - $y^{(t)}$ is true t-th class label ({+1, -1})

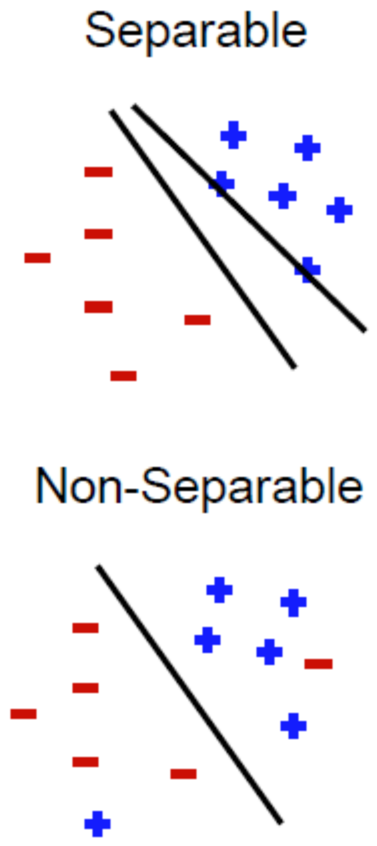


Perceptron Convergence

- **Perceptron Convergence Theorem:**
  - If there exist a set of weights that are consistent (i.e., the data is linearly separable) the Perceptron learning algorithm will converge
- **How long would it take to converge?**
- **Perceptron Cycling Theorem:**
  - If the training data is not linearly separable the Perceptron learning algorithm will eventually repeat the same set of weights and therefore enter an infinite loop

Properties of Perceptron

- **Separability:** Some parameters get training set perfectly
- **Convergence:** If training set is separable, perceptron will converge



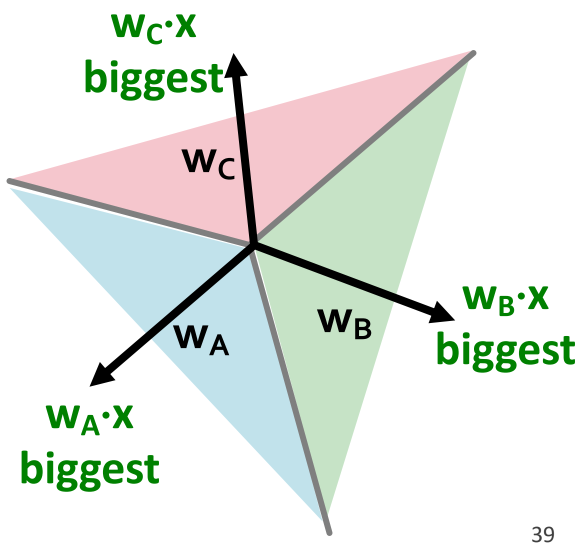
Updating the Learning Rate

- **Perceptron will oscillate and won't converge**
- **When to stop learning?**
  - **(1)** Use fixed η
  - **(2)** Slowly decrease the learning rate η
    - A classic way is to: $\eta = \mathbf{c_1/(t + c_2)}$
      - But, we also need to determine constants $\mathbf{c_1}$ and $\mathbf{c_2}$
  - **(3)** Stop when we reached some maximum number of passes over the data

Multiclass Perceptron

- **What if more than 2 classes?**
- Weight vector $\boldsymbol{w_c}$ for each class **c**
  - **Train one class vs. the rest:**
    - Example: 3-way classification **y = {A, B, C}**
    - Train 3 classifiers: $\mathbf{w_A}$: A vs. B,C; $\mathbf{w_B}$: B vs. A,C; $\mathbf{w_C}$: C vs. A,B
- **Calculate activation for each class**

Highest activation wins



Multiclass Perceptron Example

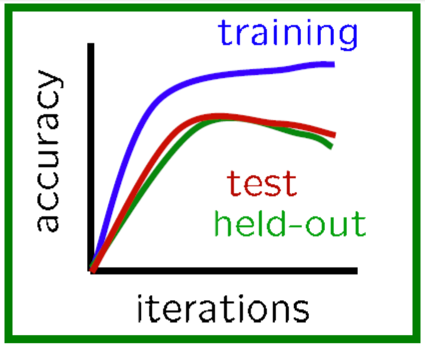
- We want to classify Web pages into a number of topics, such as sports, politics, medicine, and so on.
  - We can represent Web pages by a vector with 1 for each word present in the page and 0 otherwise.
  - Each topic has certain words that tend to indicate that topic.
  - For instance, sports pages would be full of words like “win,” “goal,” “played,” and so on.
- The weight vector for that topic would give higher weights to the words that characterize that topic.

Multiclass Perceptron

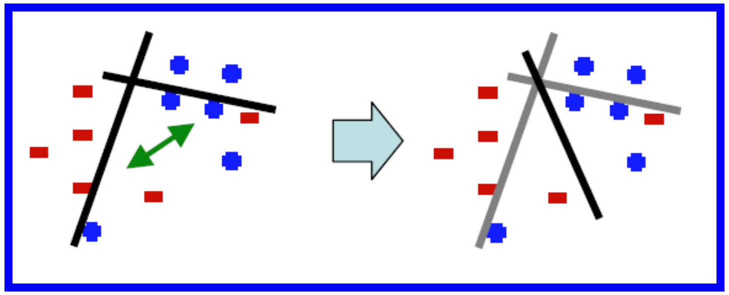
- A new page could be classified as belonging to the topic that gives the highest score
  - when the dot product of the page’s vector and the weight vectors for the topics are computed.

Issues with Perceptrons

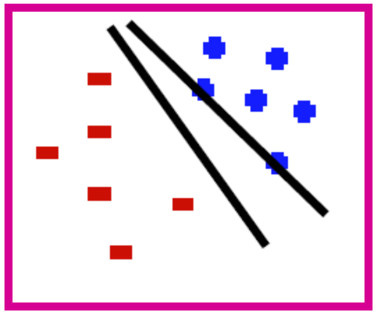
- **Overfitting:**



- **Regularization:** If the data is not separable weights dance around

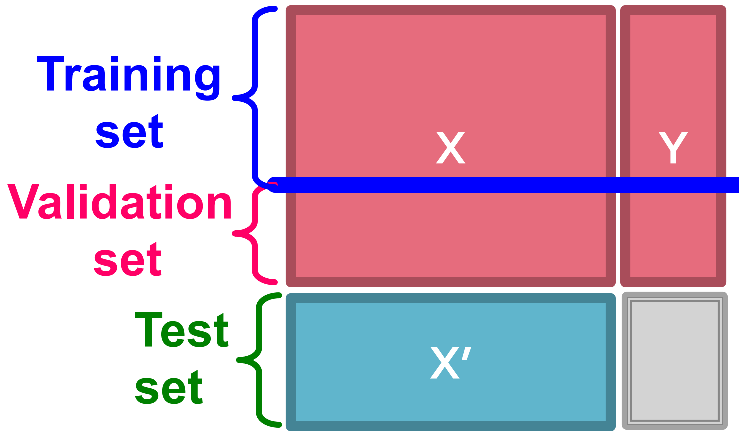


- **Mediocre generalization:**
  - Finds a "barely" separating solution



Supervised Learning

- **Idea: Pretend we do not know the data/labels we actually do know**
  - Build the model **f(x)** on the training data
    See how well **f(x)** does on the test data
    - If it does well, then apply it also to **X'**



- **Refinement: Cross validation**
  - Let's split our data **(X,Y)** into 10-folds (buckets)
  - Take out 1-fold for validation, train on remaining 9
  - Repeat this 10 times, report average performance

Perceptron Summary

Perceptron

- **Online**: Can adjust to changing target, over time
- **Advantages**
  - Simple
  - Guaranteed to learn a linearly separable problem
  - **Advantage with few relevant features per training example**
- **Limitations**
  - Only linear separations
  - Only converges for linearly separable data
  - Not really “efficient with many features”

Online Learning

Online Learning

- **New setting: Online Learning**
  - Allows for modeling problems where we have a continuous stream of data
  - We want an algorithm to learn from it and slowly adapt to the changes in data
- **Idea: Do slow updates to the model**
  - Perceptron makes updates if they misclassify an example
  - **So:** First train the classifier on training data. Then for every example from the stream, if we misclassify, update the model (using small learning rate)

Shipping Service

- **Protocol:**
  - User comes and tell us origin and destination
  - We offer to ship the package for some money (\$10 - \$50)
  - Based on the price we offer, sometimes the user uses our service (**y = 1**), sometimes they don't (**y = -1**)
- **Task:** Build an algorithm to optimize what price we offer to the users
- **Features *x* capture:**
  - Information about user
  - Origin and destination
- **Problem: Will user accept the price?**

Example: Shipping Service

- **Model whether user will accept our price:**
  y = f(x; w)
  - **Accept: y =1**, **Not accept: y=-1**
  - Build this model with say Perceptron
- **The website that runs continuously**
- **Online learning algorithm would do something like**
  - User comes
  - He/She is represented as an **(x,y)** pair where
    - **x:** Feature vector including price we offer, origin, destination
    - **y:** If they chose to use our service or not
  - The algorithm updates **w** using just the **(x,y)** pair
  - Basically, we update the **w** parameters every time we get some new data

Shipping Service Continue

- For a major website where you have a massive stream of data then this kind of algorithm is pretty reasonable
  - Why?

Shipping Service Continue

- For a major website where you have a massive stream of data then this kind of algorithm is pretty reasonable
  - Why?
    - Don’t need to deal with all the training data

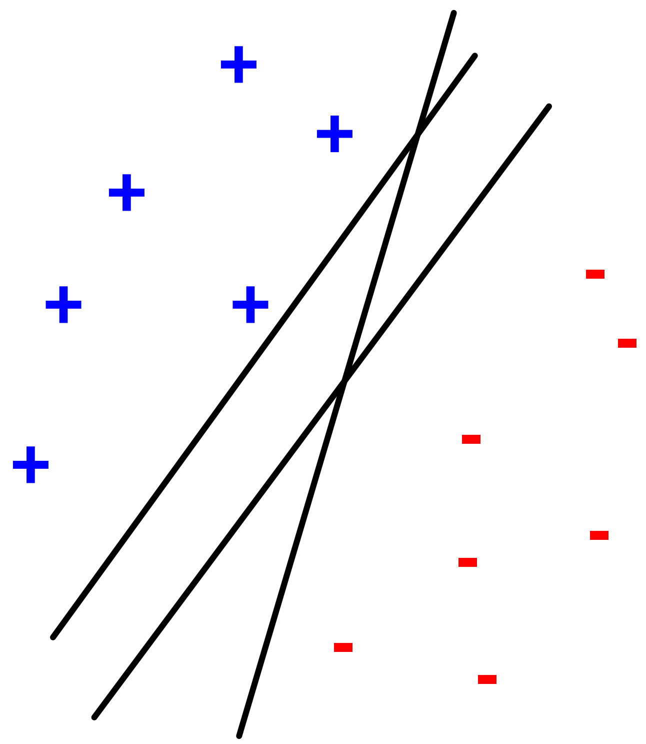
Online Algorithms

- An online algorithm can adapt to changing user preferences
- For example, over time users may become more price sensitive
- **The algorithm adapts and learns this**
- So the system is dynamic

Support Vector Machines

Support Vector Machines

- Want to separate "+" from "-" using a line



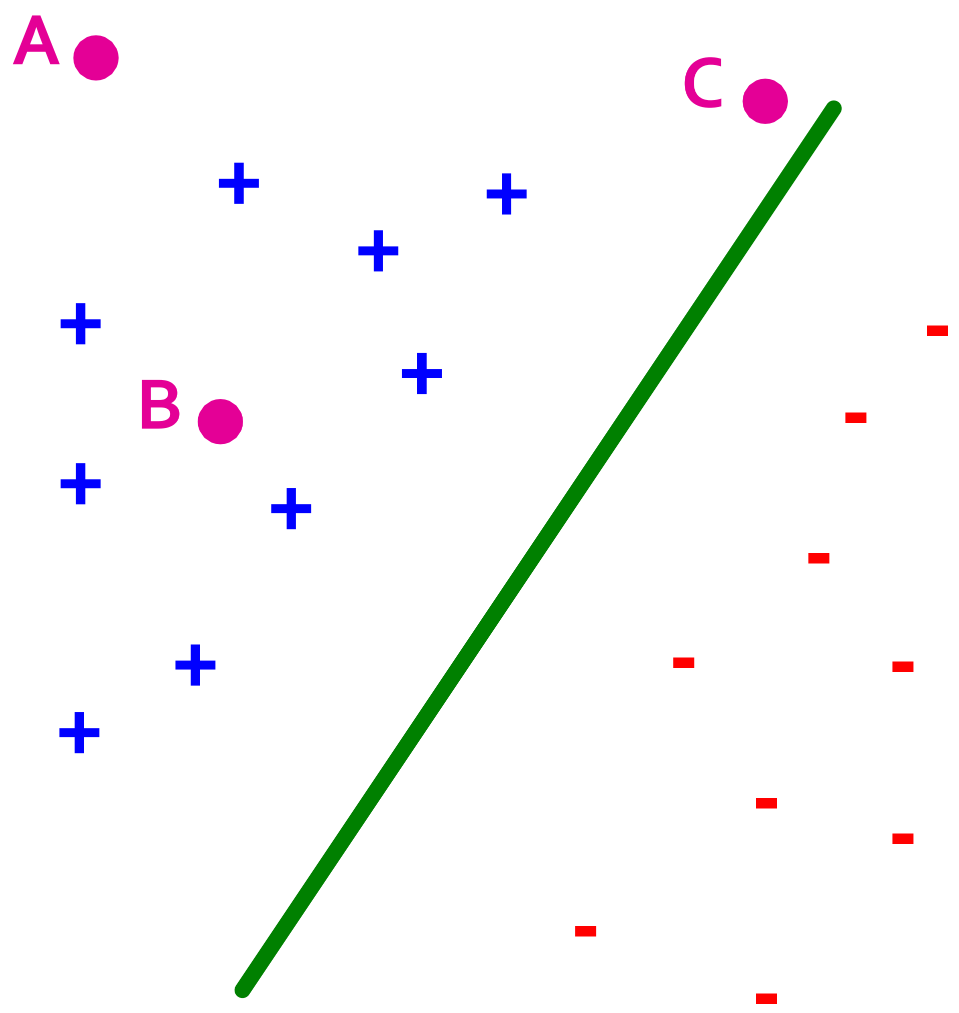
Data:

- Training examples:
  - $(x_1, y_1) \dots (x_n, y_n)$
- Each example *i*:
  - $x_i = (x_i^{(1)},\dots , x_i^{(d)})$
    - $x_i^{(j)}$ is real valued
  - $y_i \in \{ -1, +1 \}$
- Inner product:

$$\boldsymbol{w} \cdot \boldsymbol{x} = \sum_{j=1}^{d} w^{(j)} \cdot x^{(j)}$$

Which is best linear separator (defined by *w*)?

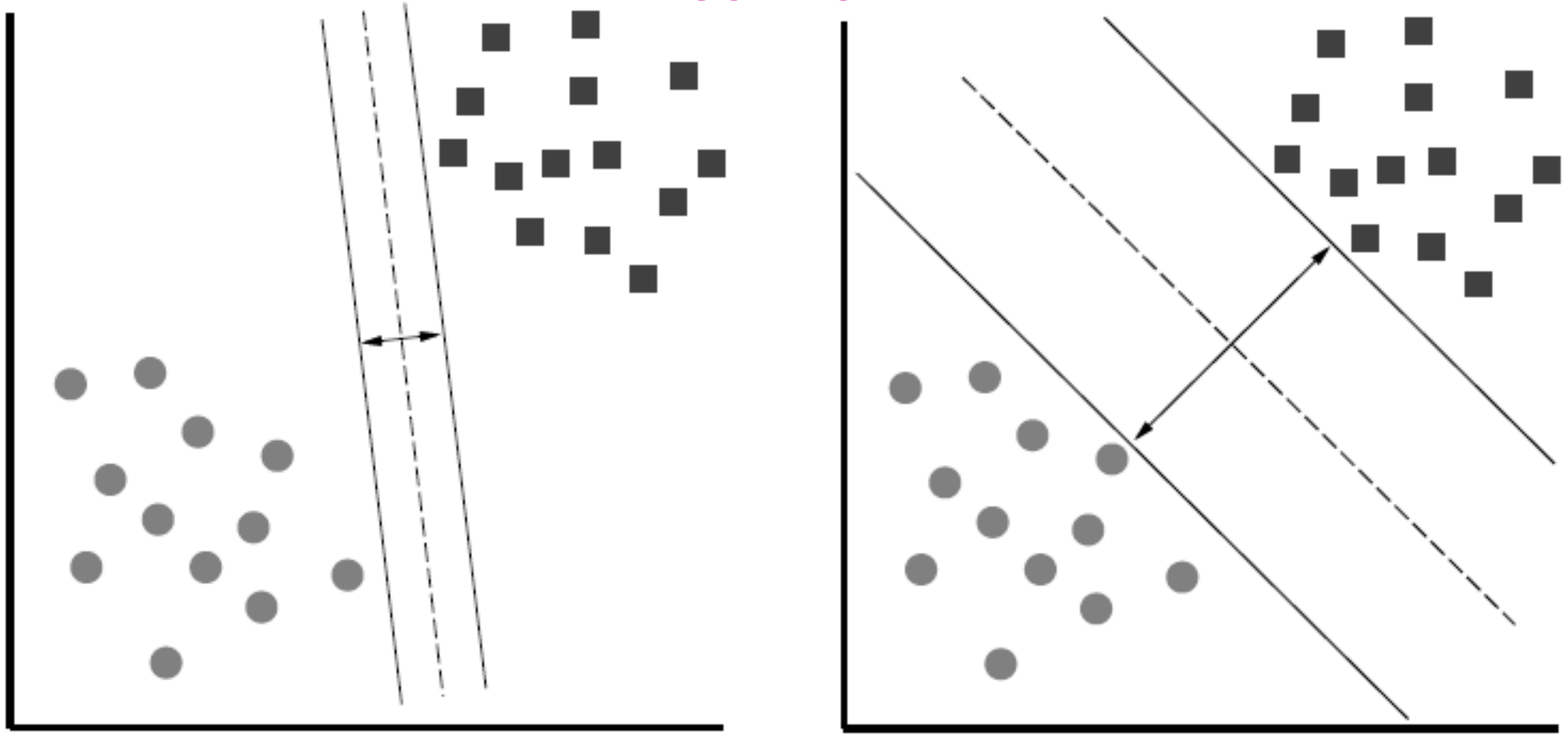
Largest Margin



- **Distance from the separating hyperplane corresponds to the "confidence" of prediction**
- **Example:**
  - We are more sure about the class of **A** and **B** than of **C**

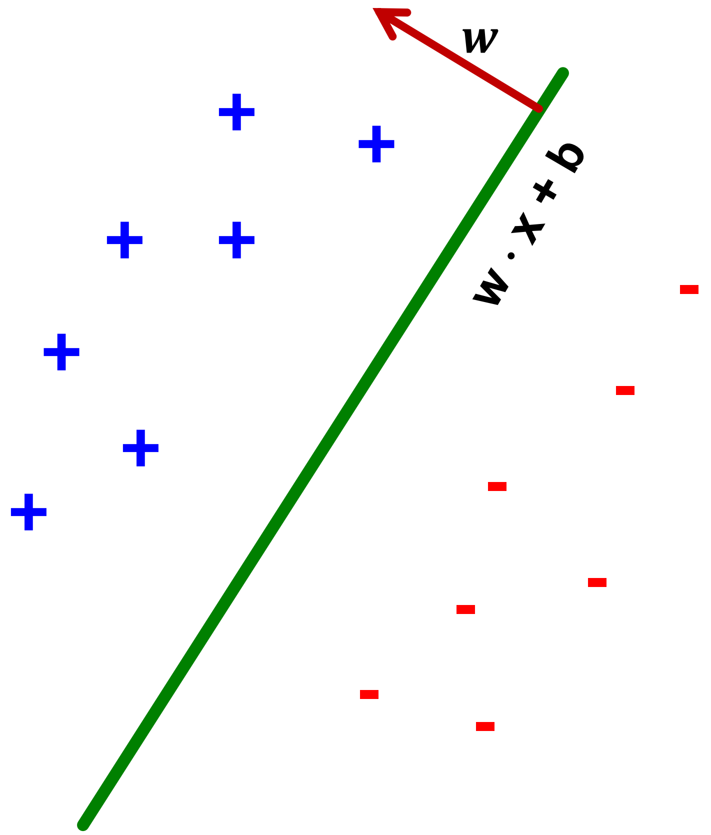
Largest Margin

- **Margin γ: Distance of closest example from the decision line/hyperplane**



The reason we define margin this way is due to theoretical convenience and existence of generalization error bounds that depend on the value of margin.

Largest Margin



- Prediction = $\mathbf{sign}(w \cdot x + b)$
- "Confidence" = $(w \cdot x + b)\, y$
- For i-th datapoint:

  $$\gamma_i = (w \cdot x_i + b) y_i$$

- Want to solve

$$\max_{w,\gamma} \gamma$$

$$s.t. \forall i, y_i (w \cdot x_i + b) \geq \gamma$$

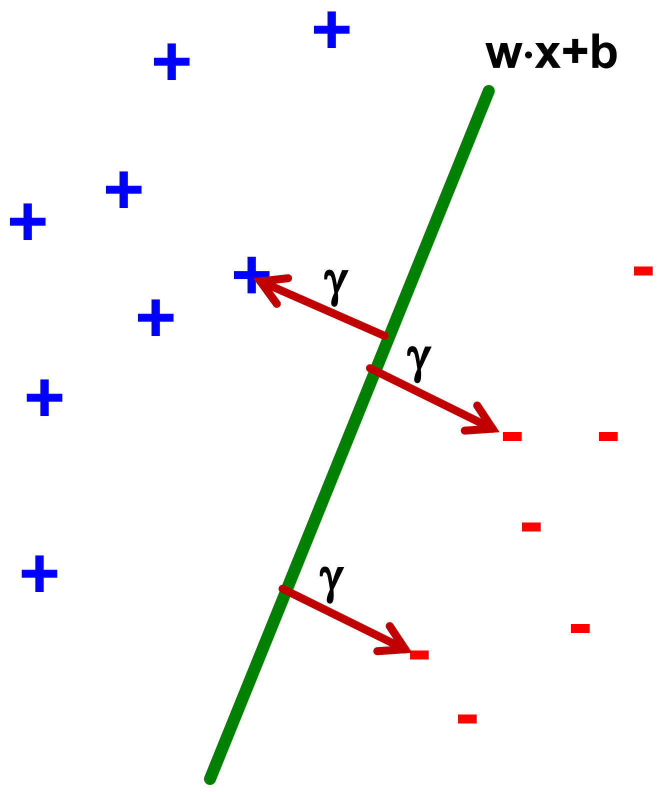
Support Vector Machine

- **Maximize the margin:**
  - **Good according to intuition, theory**

$$\max_{w,\gamma} \gamma$$

$$s.t. \forall i, y_i (w \cdot x_i + b) \geq \gamma$$

  - ***γ* is margin … distance from the separating hyperplane**



Maximizing the margin

SVM: How to estimate *w*?

- **Want to estimate w and b!**
  - **Standard way:** Use a solver!
    - **Solver:** software for finding solutions to "common" optimization problems, e.g., **WEKA**
- **Use Stochastic Gradient Descent (SGD)**

Support Vector Machines: Example

Text Categorization

- **Example:**
  - **Reuters RCV1** document corpus
    - Predict a category of a document
      - One **vs.** the rest classification
  - ***n* = 781,000** training examples (documents)
  - 23,000 test examples
  - ***d* = 50,000** features
    - One feature per word
    - Remove stop-words
    - Remove low frequency words

Example: Text categorization

- **Questions:**
  - **(1)** Is **SGD** successful at minimizing ***f(w,b)***?
  - **(2)** How quickly does **SGD** find the min of ***f(w,b)***?
  - **(3)** What is the error on a test set?

	Training time	***Value of f(w,b)***	***Test error***
Standard SVM	23,642 secs	0.2275	6.02%
"Fast SVM"	66 secs	0.2278	6.03%
SGD SVM	1.4 secs	0.2275	6.02%

(1) SGD-SVM is successful at minimizing the value of ***f(w,b)***
(2) SGD-SVM is super fast
(3) SGD-SVM test set error is comparable

Instance Based Learning

Instance Based Learning

- **Instance based learning**
- **Example: Nearest neighbor**
  - Keep the whole training dataset: **{(x, y)}**
  - A query example (vector) ***q*** comes
  - Find closest example(s) $\mathbf{x}^*$
  - Predict $\mathbf{y}^*$
- **Works both for regression and classification**
  - **Collaborative filtering** is an example of k-NN classifier
    - Find ***k*** most similar people to user **x** that have rated movie **y**
    - Predict rating $\mathbf{y}_x$ of **x** as an average of $\mathbf{y}_k$

1-Nearest Neighbor

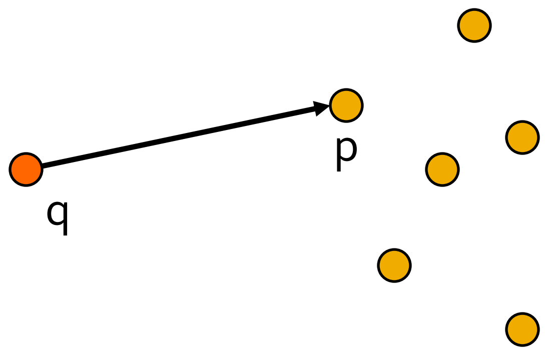
- **To make Nearest Neighbor work we need 4 things:**
  - **Distance metric:**
    - Euclidean
  - **How many neighbors to look at?**
    - One
  - **Weighting function (optional):**
    - Unused
  - **How to fit with the local points?**
    - Just predict the same output as the nearest neighbor

k-Nearest Neighbor

- **Distance metric:**
  - Euclidean
- **How many neighbors to look at?**
  - ***k***
- **Weighting function (optional):**
  - Unused
- **How to fit with the local points?**
  - Just predict the average output among *k* nearest neighbors

How to find nearest neighbors?

- **Given:** a set ***P*** of ***n***
- **Goal: Given a query point *q***
  - **NN:** Find the *nearest neighbor* ***p*** of ***q*** in ***P***
  - **Range search:** Find one/all points in ***P*** within distance ***r*** from ***q***



Algorithms for NN

- **Main memory:**
  - **Linear scan**
  - **Tree based:**
    - Quadtree
  - **Hashing:**
    - Locality-Sensitive Hashing
- **Secondary storage:**
  - R-trees

Summary

- Nearest-Neighbor Learning: In this approach to machine learning, the entire training set is used as the model.
- For each ("query") point to be classified, we search for its k nearest neighbors in the training set.
- The classification of the query point is some function of the labels of these k neighbors.
- The simplest case is when k = 1, in which case we can take the label of the query point to be the label of the nearest neighbor.

Summary

- Perceptrons: This machine-learning method assumes the training set has only two class labels, positive and negative.
- Perceptrons work when there is a hyperplane that separates the feature vectors of the positive examples from those of the negative examples.
- We converge to that hyperplane by
- adjusting our estimate of the hyperplane by a fraction – the learning rate – of the direction that is the average of the currently misclassified points.

Summary

- Support-Vector Machines: The SVM improves upon perceptrons by finding a separating hyperplane that not only separates the positive and negative
- points, but does so in a way that maximizes the margin – the distance perpendicular to the hyperplane to the nearest points.
- The points that lie exactly at this minimum distance are the support vectors.
- Alternatively, the SVM can be designed to allow points that are too close to the hyperplane, or even on the wrong side of the hyperplane, but minimize the error due to such misplaced points.

Quiz

Quiz: Perceptrons

- Consider training data for spam emails
  - **+1 indicates spam**

	and	viagra	the	of	nigeria	y
a	1	1	0	1	1	+1
b	0	0	1	1	0	−1
c	0	1	1	0	0	+1
d	1	0	0	1	0	−1
e	1	0	1	0	1	+1
f	1	0	1	1	0	−1

- Apply Perceptron algorithm
  - with learning rate **η = ½**
  - and we shall visit each training example a-f once (stopping criteria)
  - What is the precision of the algorithm given computed weights w?