WELCOME

Introduction to Machine Learning

DBDA.X408.(33)

Instructor:

Bill Chen

UCSC Silicon Valley Extension
PROFESSIONAL EDUCATION

UCSC Silicon Valley Extension

E: xchen375@ucsc.edu

Week 6

Random Forest.

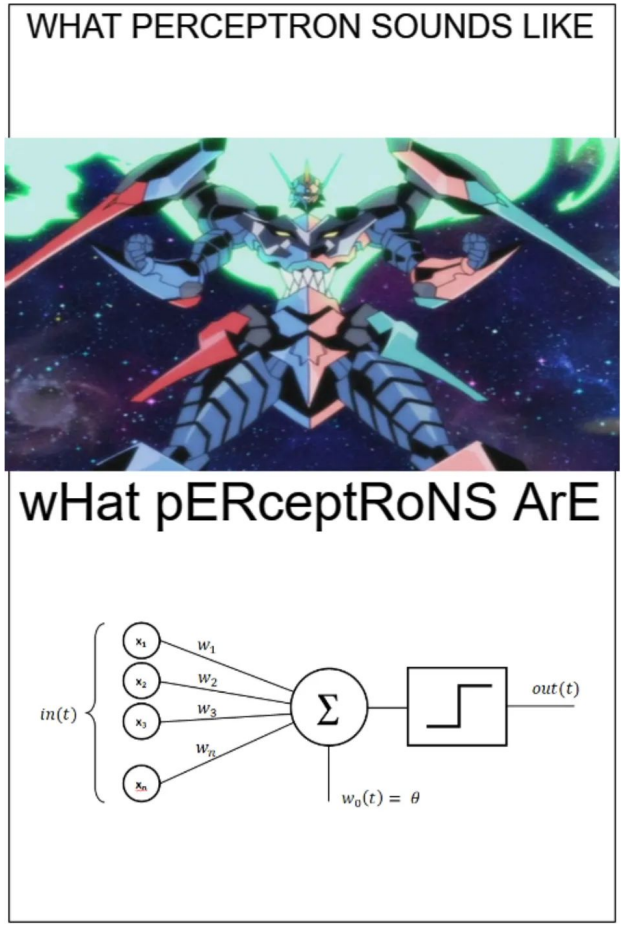
Extremely Randomized Trees.

Gradient Boosted Regression Trees.

SVM revisiting.

Perceptron.

Neural Network.

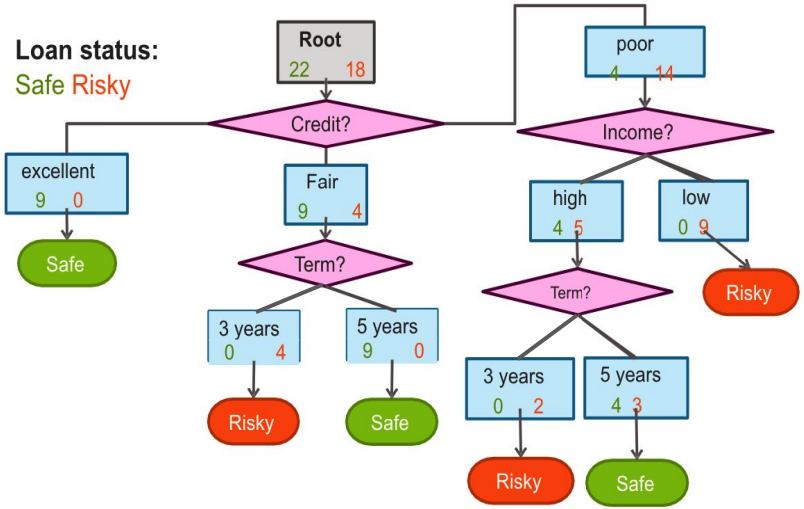


Random Forest

Decision trees (Breiman et al., 1984)

Algorithm:

- Step-1: Begin the tree with the root node, says S, which contains the complete dataset.
- Step-2: Find the best attribute in the dataset using **Attribute Selection Measure (ASM)**.
- Step-3: Divide the S into subsets that contains possible values for the best attributes.
- Step-4: Generate the decision tree node, which contains the best attribute.
- Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.



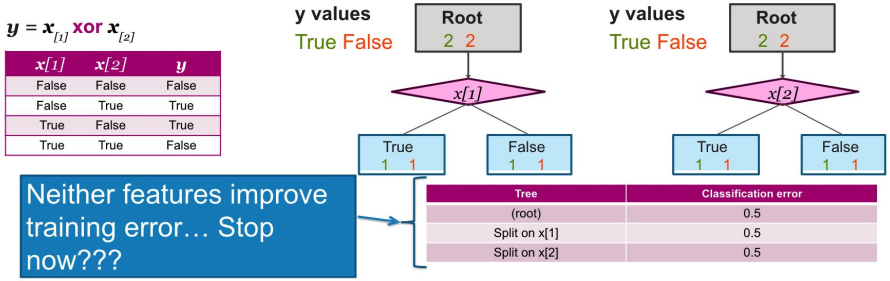
Decision Tree Pruning

- **Pre-pruning or early stopping**

This means stopping before the full tree is even created. We can keep making the tree until the next node doesn't change the error rate change drastically. Whenever we see a very minor change in the error rate after building the next node, we can stop building further.

- **Post Pruning**

Creating the full tree up until we have very few data points in each set and then track backing such that the change in the error rate is not much.



Do you remember last time we try NOT to stop split when the classification error didn't decrease?

https://colab.research.google.com/drive/1rB1xsYvWZ4Z-peFHJE1zahK9LTCSvwx9#scrollTo=2PLA2uuGlyGx&line=1&uniqifier=1

sklearn.tree

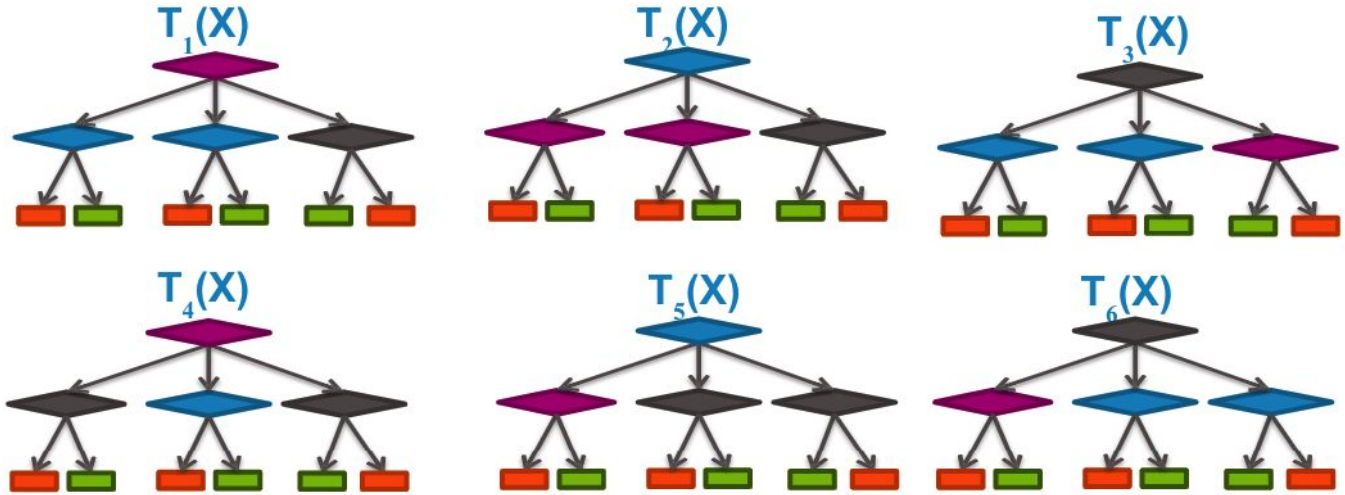
```
# Fit a decision tree
from sklearn.tree import DecisionTreeRegressor
estimator = DecisionTreeRegressor(criterion="mse",    # Set i(t) function
                                  max_leaf_nodes=5)
estimator.fit(X_train, y_train)

# Predict target values
y_pred = estimator.predict(X_test)

# MSE on test data
from sklearn.metrics import mean_squared_error
score = mean_squared_error(y_test, y_pred)
>>> 0.572049826453
```

```
# Display tree
from sklearn.tree import export_graphviz
export_graphviz(estimator, out_file="tree.dot",
                feature_names=feature_names)
```

Intuition of Random Forest

You can have many decision trees.

But how can we have many different trees?



Random Forests (Breiman, 2001; Geurts et al., 2006)

1. For $b = 1$ to B:

   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.

   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.

      i. Select m variables at random from the p variables.

      ii. Pick the best variable/split-point among the m.

      iii. Split the node into two daughter nodes.

2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

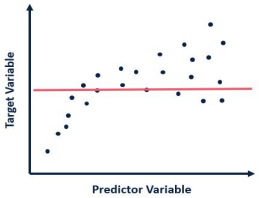
Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

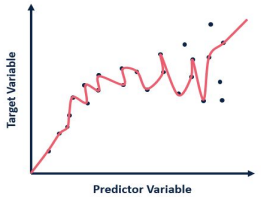
Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

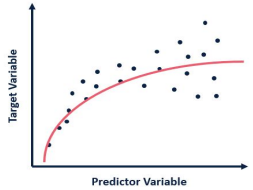
Differences to standard tree:

- Train each tree on bootstrap resample of data (Bootstrap resample of data set with N samples:
- Make new data set by drawing with replacement N samples; i.e., some samples will probably occur multiple times in new data set)
- For each split, consider only m randomly selected variables
- Don’t prune
- Fit B trees in such a way and use average or majority voting to aggregate results

Tree vs. Random Forest





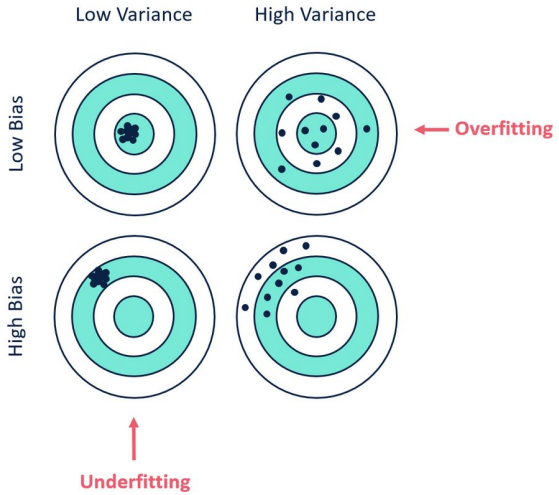


Tree

+ Trees yield insight into decision rules.
+ Rather fast.
+ Easy to tune parameters.
- Prediction of trees tend to have a high variance.

Random Forest

+ RF as smaller prediction variance and therefore usually a better general performance.
+ Easy to tune parameters.
- Rather slow.
- “Black box”: rather hard to get insights into decision rules.



Extremely Randomized Trees

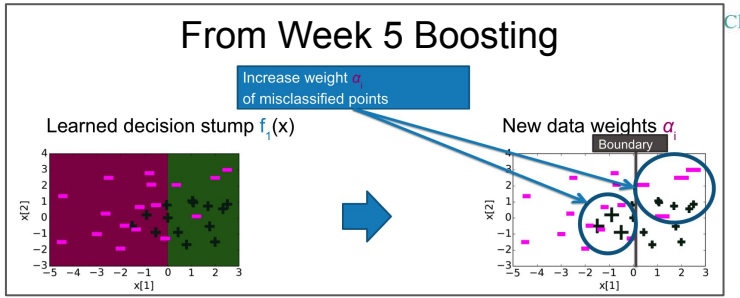
Extremely Randomized Trees, also known as Extra Trees, construct multiple trees like RF algorithms during training time over the entire dataset. During training, the ET will construct trees over every observation in the dataset but with different subsets of features.

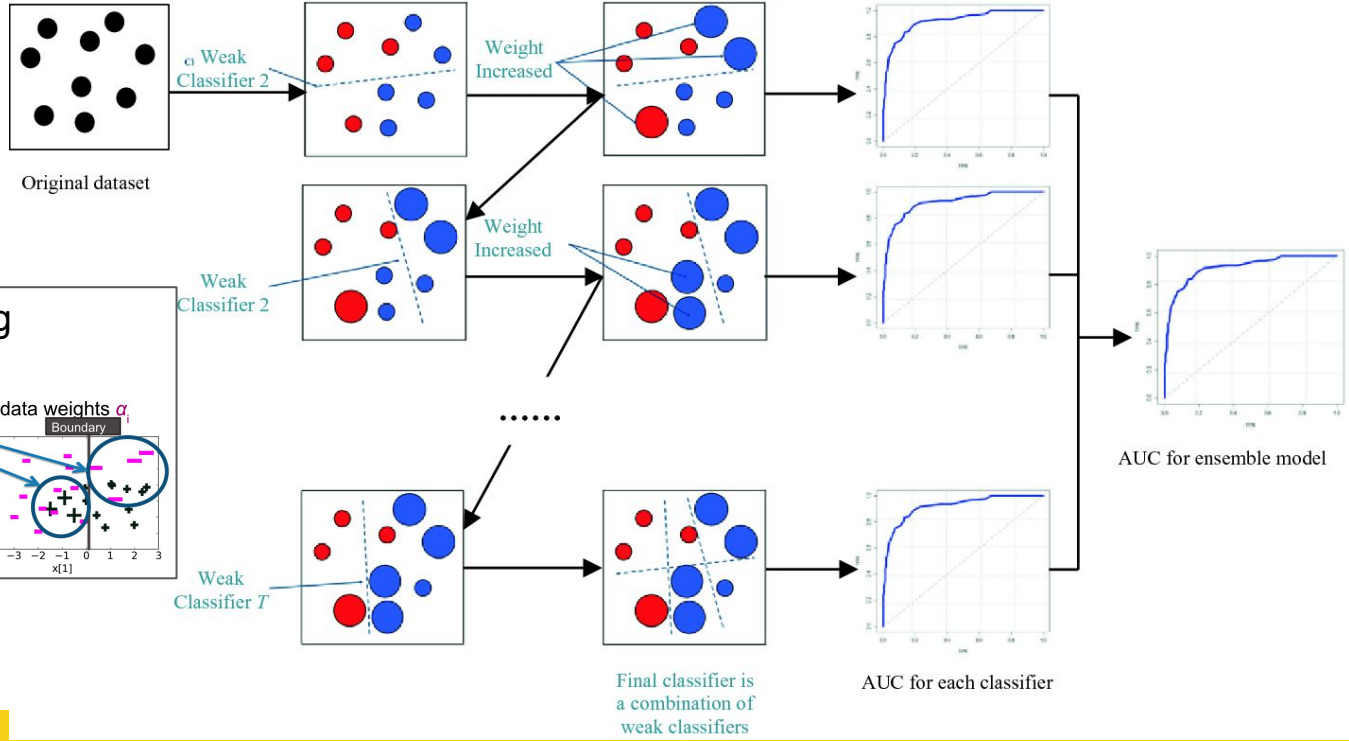
It is important to note that although bootstrapping is not implemented in ET's original structure, we can add it in some implementations. Furthermore, when constructing each decision tree, the ET algorithm splits nodes randomly.

Random Forest	Extremely Randomized Trees
Samples subsets through bootstrapping	Samples the entire dataset
Nodes are split looking at the best split	Randomized node split
Medium Variance	Low Variance
It takes time to find the best node to split on	Faster since node splits are random

https://colab.research.google.com/drive/1rB1xsYvWZ4Z-peFHJE1zahK9LTCSvwx9#scrollTo=jnOG-i0XqQyG&line=1&uniqifier=1

Gradient Boosted Regression Trees (Friedman, 2001)





Strengths and Weaknesses of GBRT

- Often more accurate than random forests.
- Flexible framework, that can adapt to arbitrary loss functions.
- Fine control of under/overfitting through regularization (e.g., learning rate, subsampling, tree structure, penalization term in the loss function, etc).
- Careful tuning required.
- Slow to train, fast to predict.

https://colab.research.google.com/drive/1rB1xsYvWZ4Z-peFHJE1zahK9LTCSvwx9#scrollTo=45SEV3Js1PTf&line=1&uniqifier=1

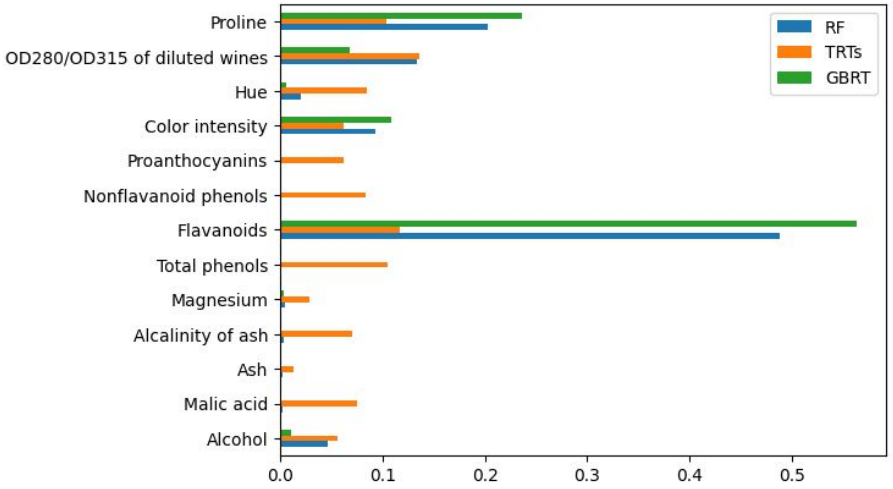
Variable Importances

Feature (Variable) Selection/Ranking/Exploration

Tree-based models come with built-in methods for variable selection, ranking or exploration.

The main goals are:

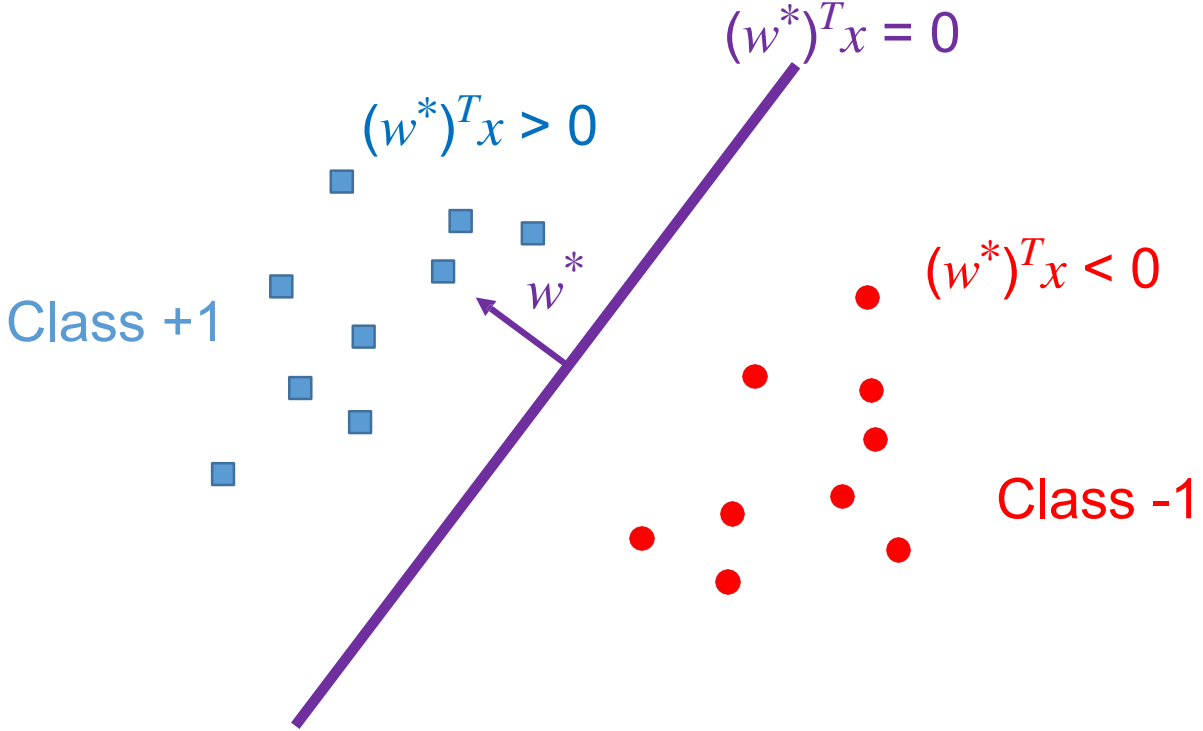
- To reduce training times;
- To enhance generalisation by reducing overfitting;
- To uncover relations between variables and ease model interpretation.



https://colab.research.google.com/drive/1rB1xsYvWZ4Z-peFHJE1zahK9LTCSvwx9#scrollTo=vScYMt173lf5

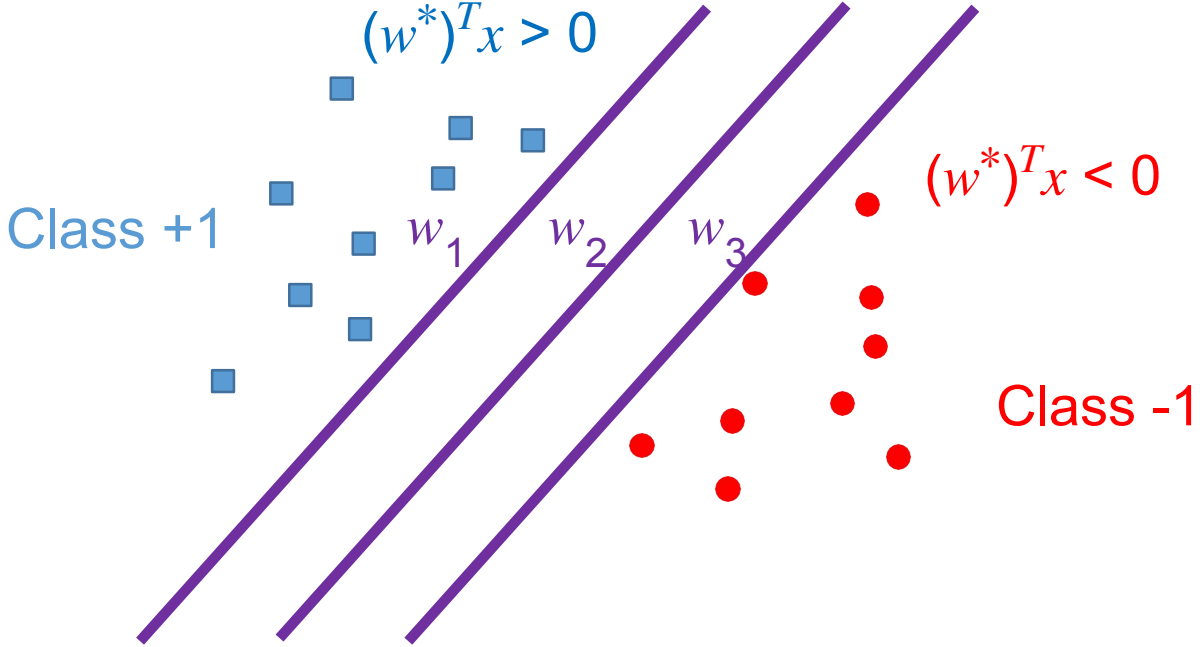
SVM Revisiting

Motive: Linear Classification



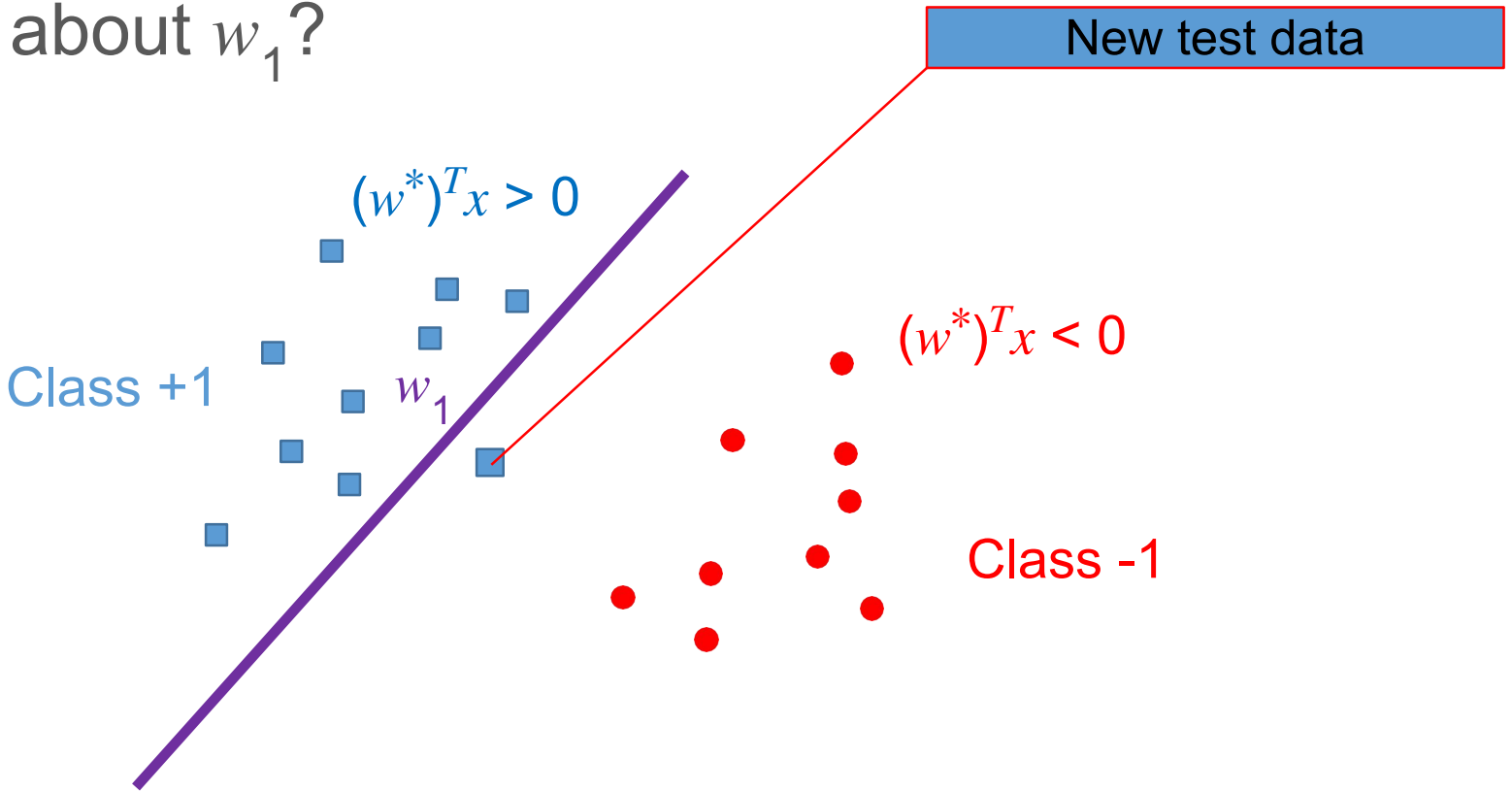
Assume perfect separation between the two classes

Motive: Multiple Optimal Solutions?



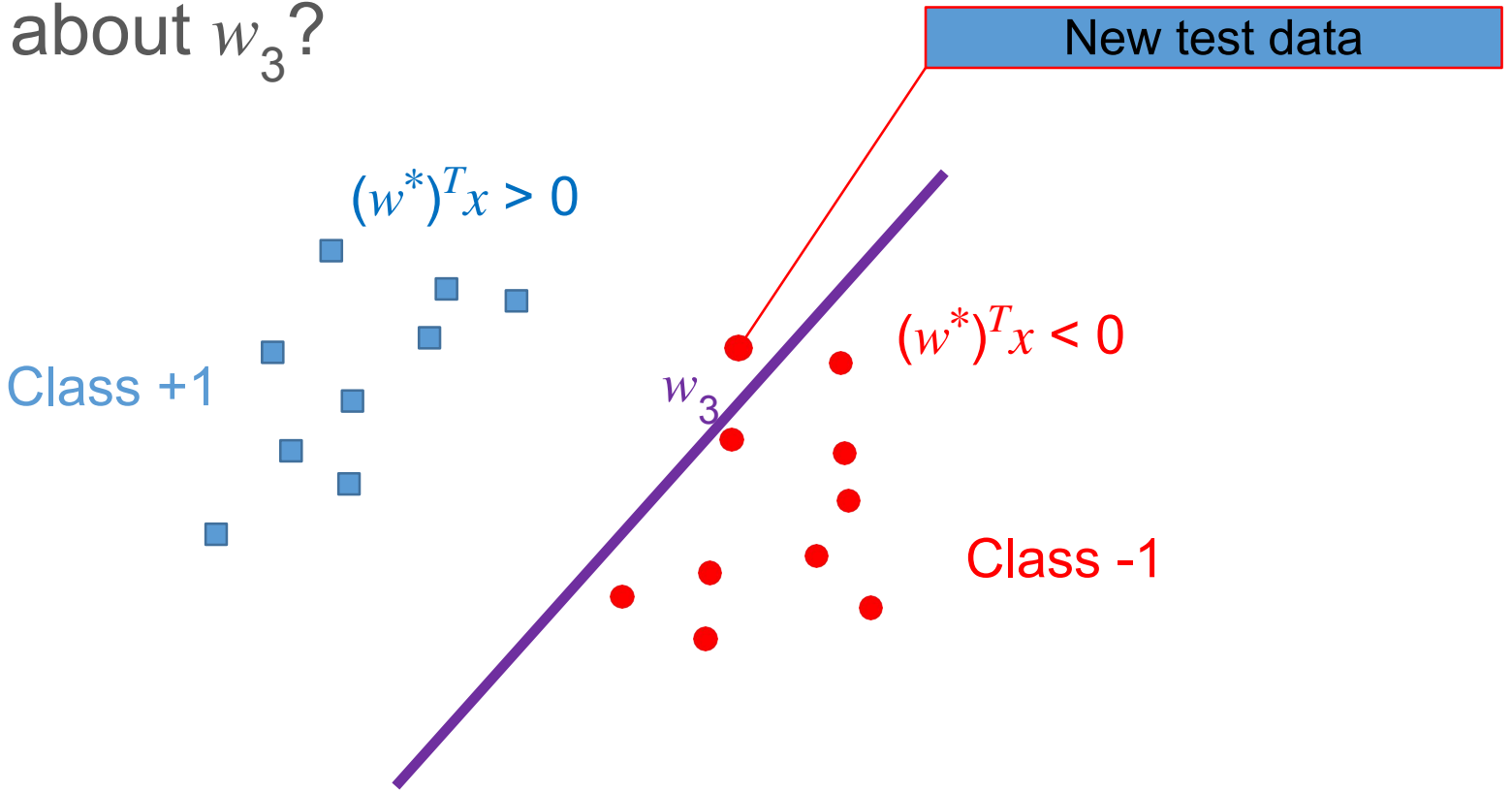
Same on empirical loss; Different on test/expected loss

What about w_1?



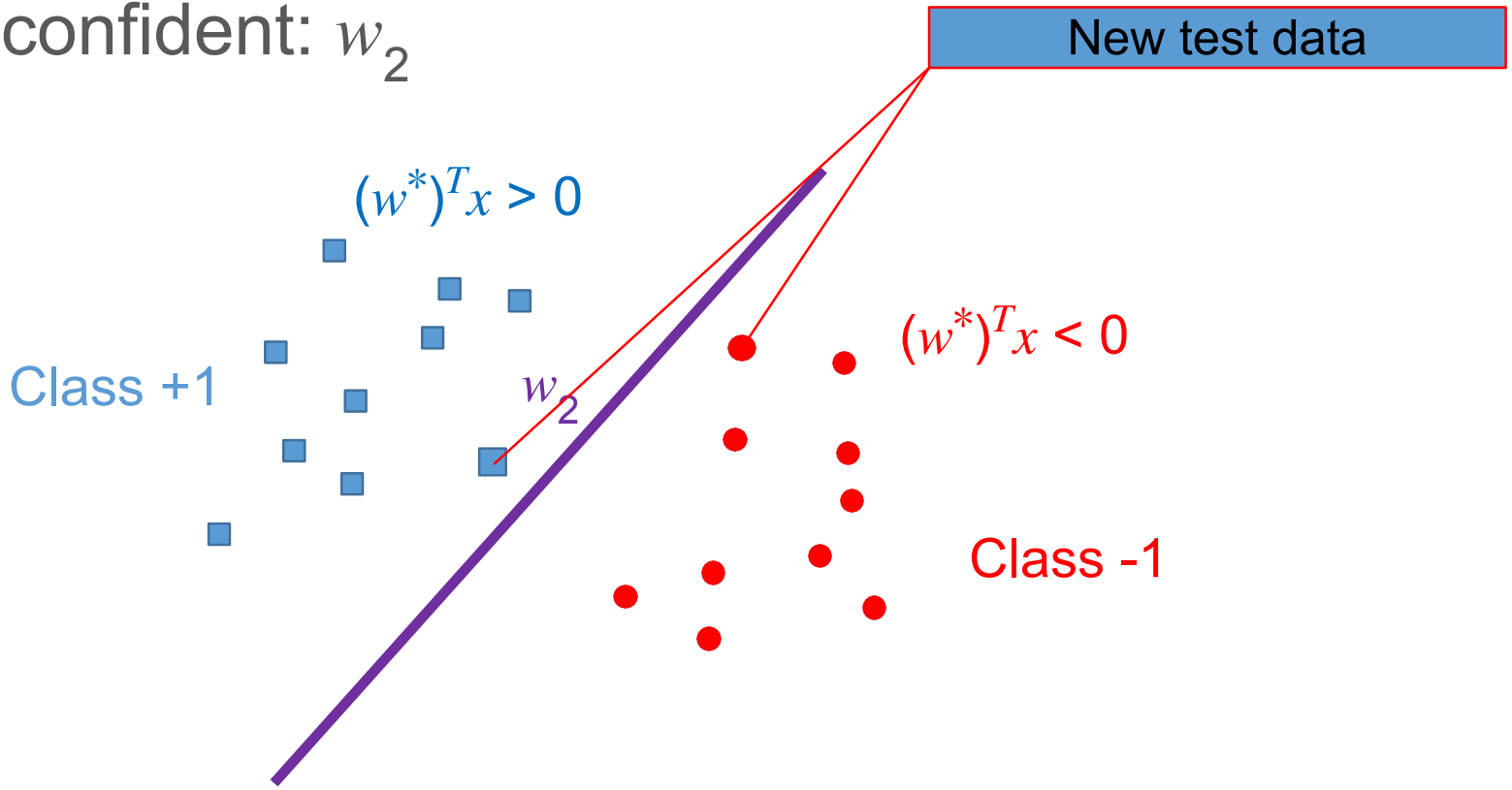
Same on empirical loss; Different on test/expected loss

What about w_3?



Same on empirical loss; Different on test/expected loss

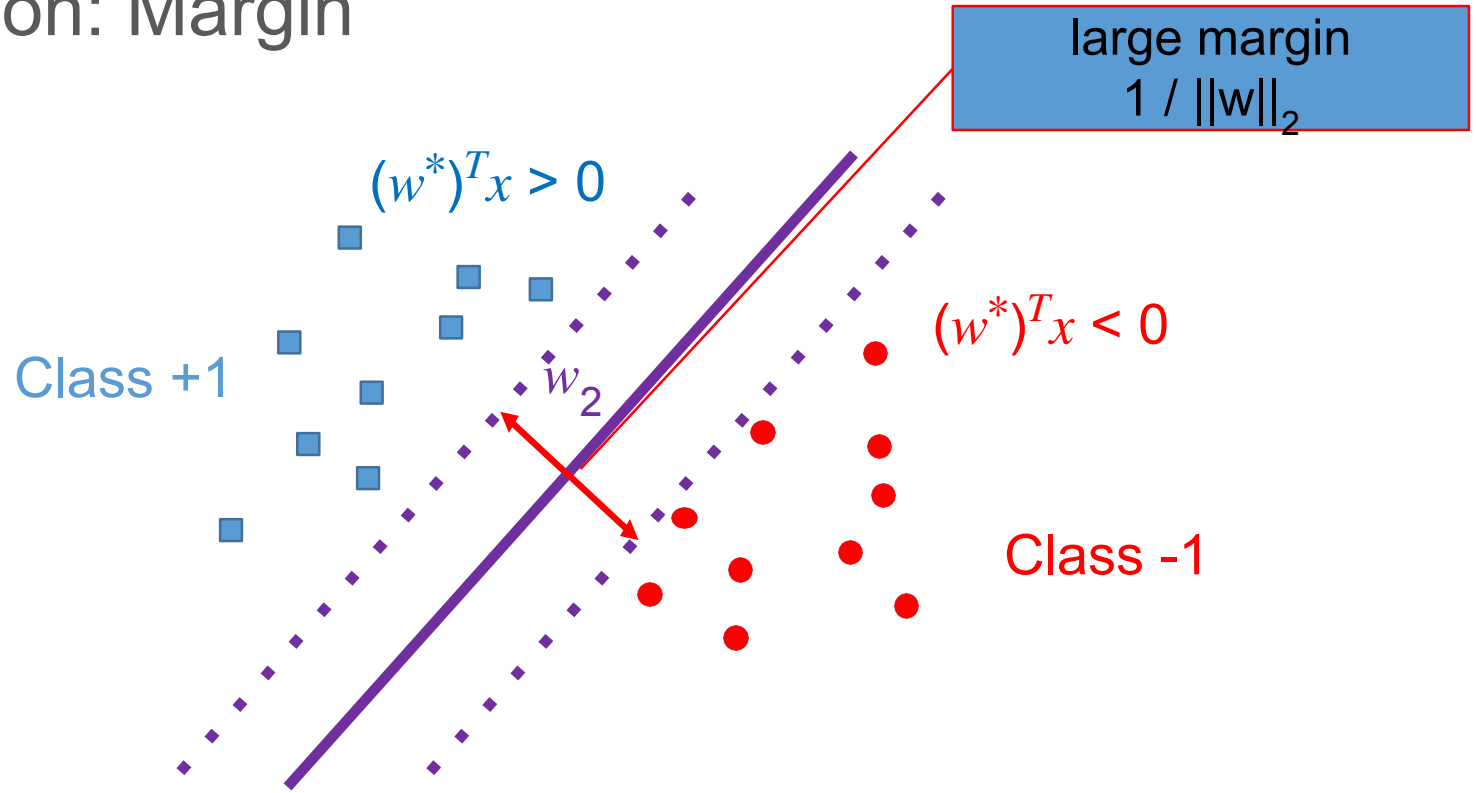
Most confident: w_2



Same on empirical loss; Different on test/expected loss

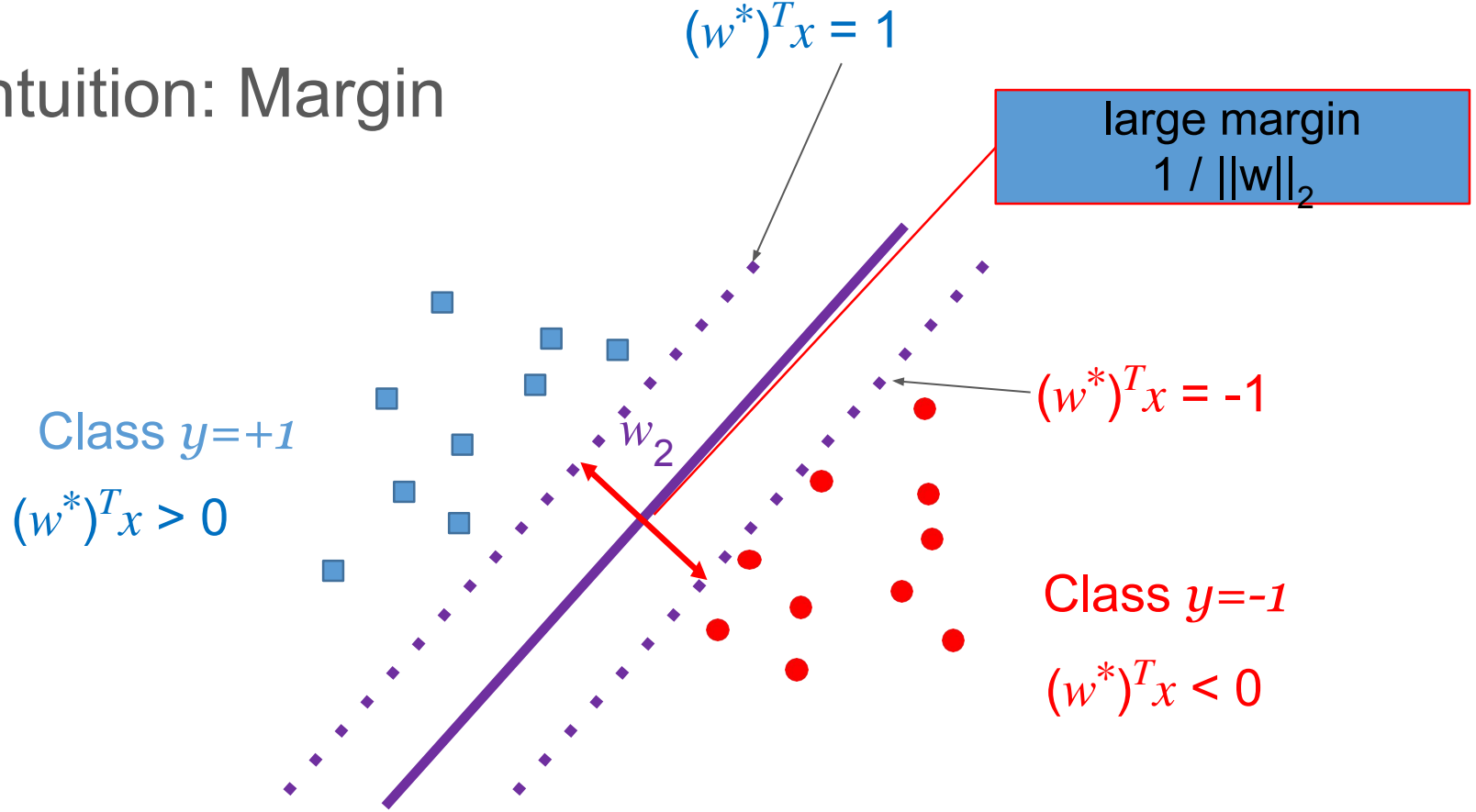
$(w^*)^T x > 0$

Intuition: Margin



Same on empirical loss; Different on test/expected loss

Intuition: Margin

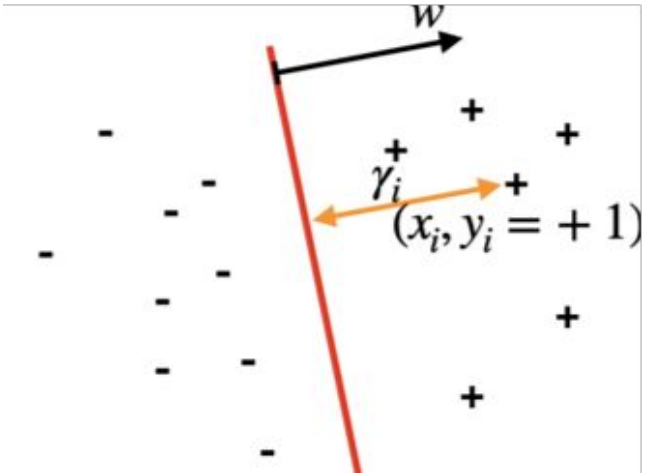


Same on empirical loss; Different on test/expected loss

SVM: Geometric Margin

- Given a set of training examples $\{(x_i, y_i)\}^n_{i=1}$
- A linear classifier will be $(w^*)^T x = 0$
- We define functional margin of w with respect to a training example (x_i, y_i) as the distance from the point (x_i, y_i) to the decision boundary, which is

$$\gamma_i = y_i \frac{(w^T x_i + b)}{\|w\|_2}$$



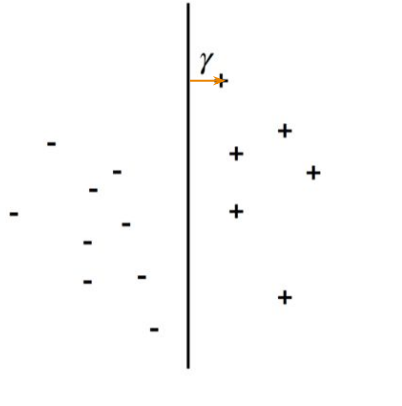
Maximum Margin Classifier

We propose the following optimization problem, maximize the margin.

$$\gamma_i = y_i \frac{(w^T x_i + b)}{\|w\|_2}$$

This is hard to do:

- maximize the numerator
- minimize the denominator



How about we just try to do this if we assume γ is a lower bound on the margin?

$$\frac{y_i(w^T x_i + b)}{\|w\|_2} \geq \gamma \quad \text{for all } i \in \{1, \ldots, n\}$$

Maximum Margin Classifier

Still hard to optimize, so we reparameterization if we condition the numerator is >=1:

$$\frac{y_i(w^T x_i + b)}{\|w\|_2} \geq \gamma \quad \text{for all } i \in \{1,\ldots,n\} \longrightarrow \frac{y_i(w^T x_i + b)}{\|w\|_2} \geq \gamma \quad \text{for all } i \in \{1,\ldots,n\}$$
$$\|w\|_2 = \frac{1}{\gamma}$$

Conditioning on $y_i(w^T x_i + b) \geq 1 \quad \text{for all } i \in \{1,\ldots,n\}$

So we can see we want to maximize $\frac{1}{\|w\|_2}$, which is minimize $\|w\|_2^2$

This is is a quadratic problem with linear constraints, easy to solve

Don’t Forget It’s Still a Classifier

We want to have a loss function that conditioning on the $\ell(\hat{y}, y)$

So the overall SVM algorithm is try to optimize:

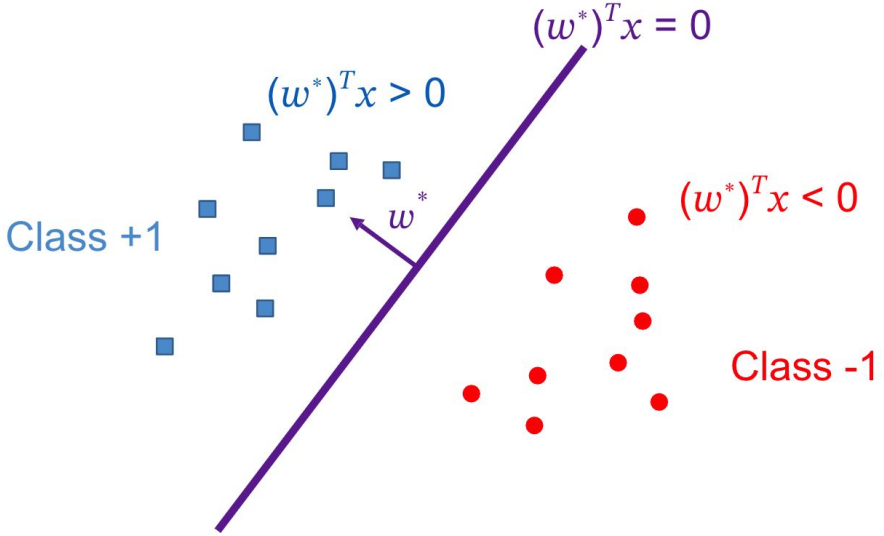
$$\lambda \|w\|_2^2 + \sum_{i=1}^{n} \ell(\hat{y}, y)$$

Perceptron

Perceptron

One of the most primitive form of learning and it is used to classify linearly-separable datasets.

Given two data points, Perceptron tries to find a hyperplane that separates the data. There can be numerous hyperplane that separates two data points in Perceptron.

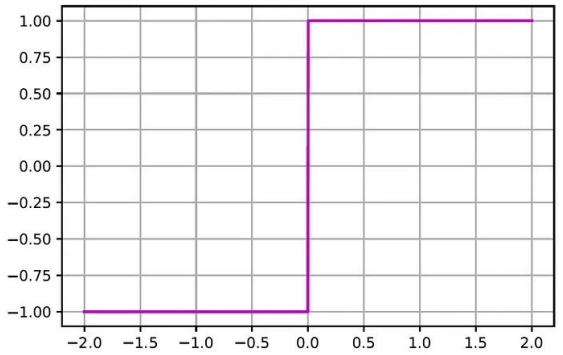


Similar to SVM but simpler.

https://www.ling.upenn.edu/courses/cogs501/Rosenblatt1958.pdf

Attempt

- Given training data $\{(x_i, y_i): 1 \le i \le n\}$ i.i.d. from distribution D
- Hypothesis $f_w(x) = w^T x$
  - $y = +1$ if $w^T x > 0$
  - $y = -1$ if $w^T x < 0$
- Prediction: $y = \boldsymbol{sign}(f_w(x)) = \boldsymbol{sign}(w^T x)$
- Goal: minimize classification error



The sign function is a function whose value is -1 for negative inputs and 1 for non-negative inputs

Perceptron Algorithm

Assume for simplicity: all x_i has length 1

1. Start with the all-zeroes weight vector $\mathbf{w}_1 = \mathbf{0}$, and initialize t to 1.
2. Given example $\mathbf{x}$, predict positive iff $\mathbf{w}_t \cdot \mathbf{x} > 0$.
3. On a mistake, update as follows:
   - Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \mathbf{x}$.
   - Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \mathbf{x}$.

   $t \leftarrow t + 1$.

Perceptron: figure from the lecture note of Nina Balcan

Perceptron vs. SVM

- Perceptron learning algorithm works better with linear data, but not better than SVM algorithm.
- There can be different hyperplane that a Perceptron can generate in different experiments. And it solely depends upon the initial weights.
- SVM keeps a classification margin on each side so that it classifies test data points that come near to the boundary properly.
- The margin of the SVM makes SVM more robust in getting more closer to the real boundary (target function) of the datasets

https://colab.research.google.com/drive/1rB1xsYvWZ4Z-peFHJE1zahK9LTCSvwx9#scrollTo=cwVmdSQMfEUO&line=1&uniqifier=1

Neural Network

Neural Network

Origins: Neural Network was trying to mimic the brain.

Was widely used in 80s and early 90s; popularity diminished in late 90s.

Recent resurgence: State-of-the-art technique for many applications

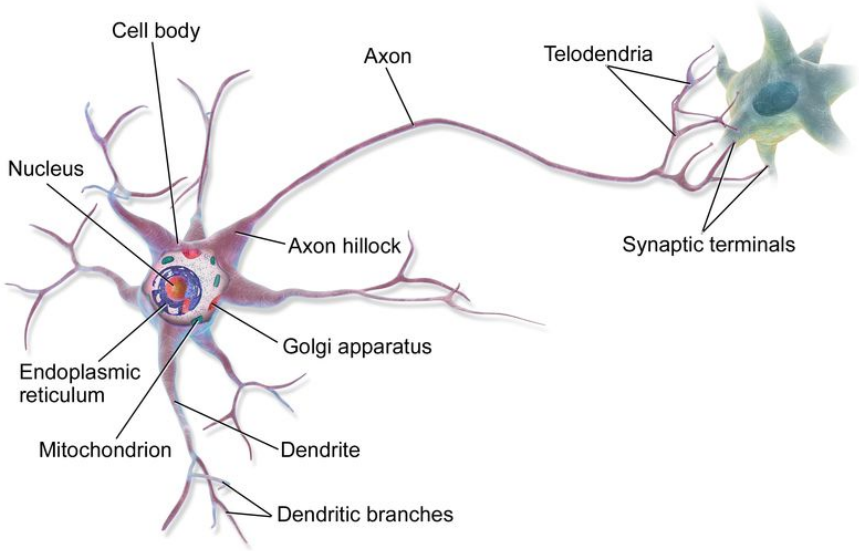
A single-layer perceptron is the basic unit of a neural network.

Biological Neural Networks

A biological neural network (such as the one we have in our brain) is composed of a large number of nerve cells called **neurons**.

Each neuron receives electrical signals (impulses) from its neighboring neurons via fibers called **dendrites**. When the total sum of its incoming signals exceeds some threshold, the neuron “fires” its own signal via long fibers called axons that are connected to the dendrites of other neurons.

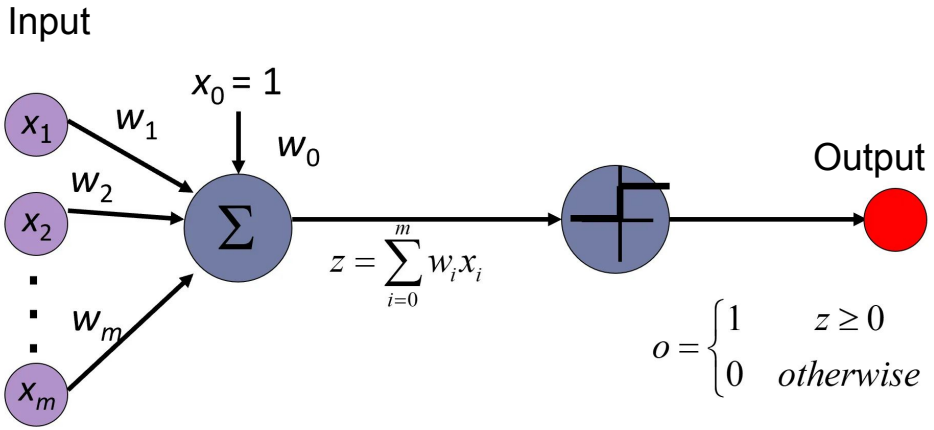
The junction between two neurons is called a **synapse**. On average, each neuron is connected to about 7,000 synapses, which demonstrates the high connectivity of the network we have in our brain. When we learn new associations between two concepts, the synaptic strength between the neurons that represent these concepts is strengthened. This phenomenon is known as Hebb’s rule (1949) that states “Cells that fire together wire together”.

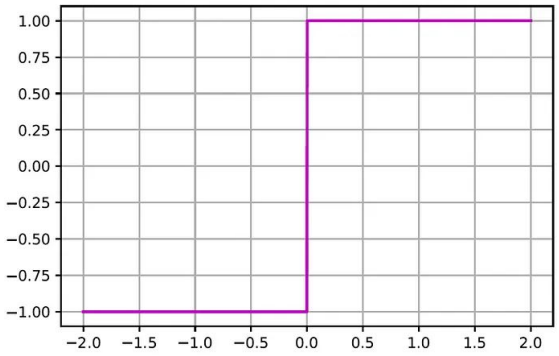


Perceptron Model

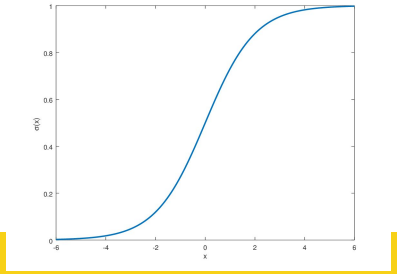
Each input neuron x_i is connected to the perceptron via a link whose strength is represented by a weight w_i. Inputs with higher weights have a larger influence on the perceptron's output.

More generally, the perceptron applies an activation function f(z) on the net input that generates its output.

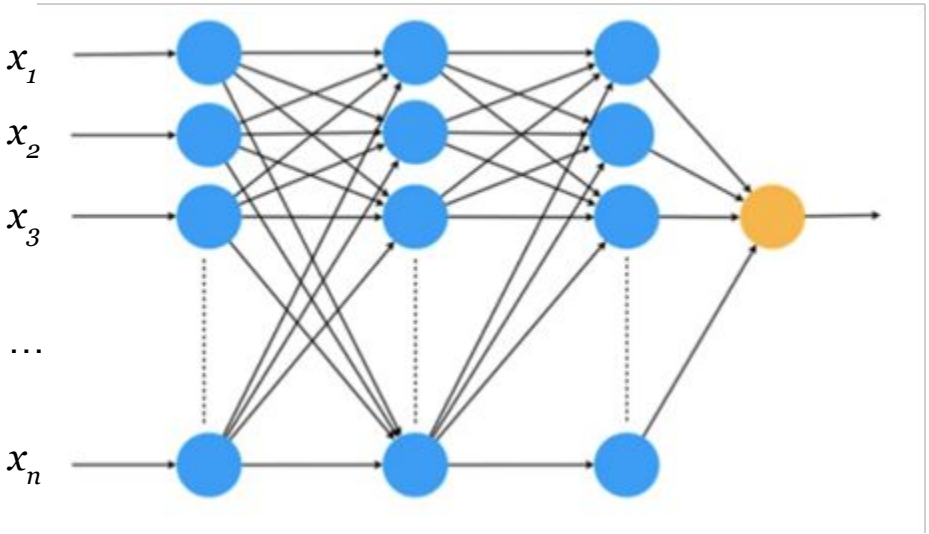




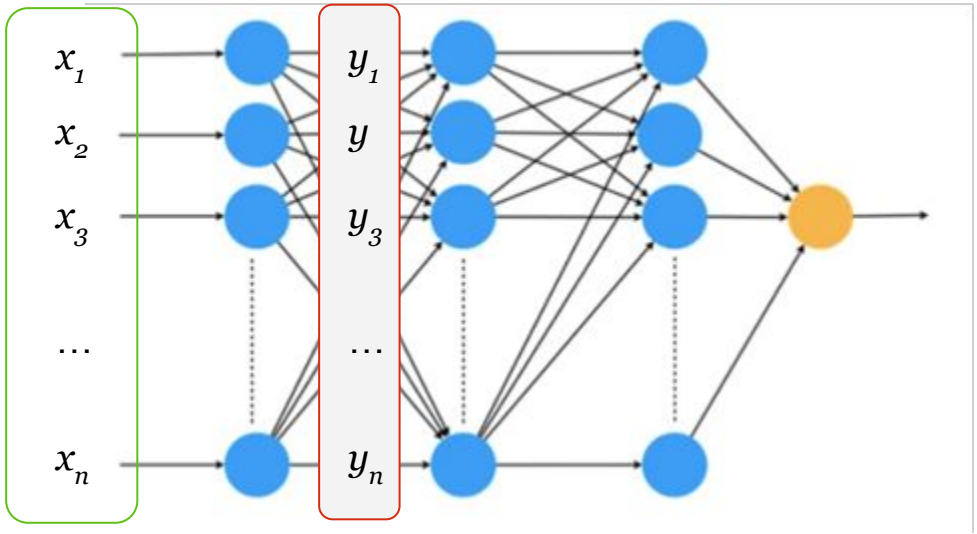
The sign function is a function whose value is -1 for negative inputs and 1 for non-negative inputs. We can also use sigmoid function.



Neural Model: Stack Neurons Together by Layers

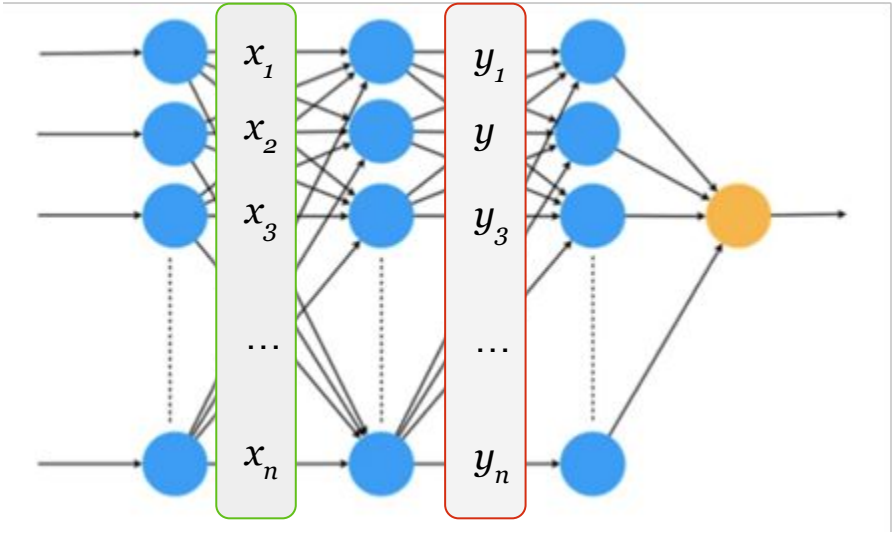


Neural Model: Stack Neurons Together by Layers

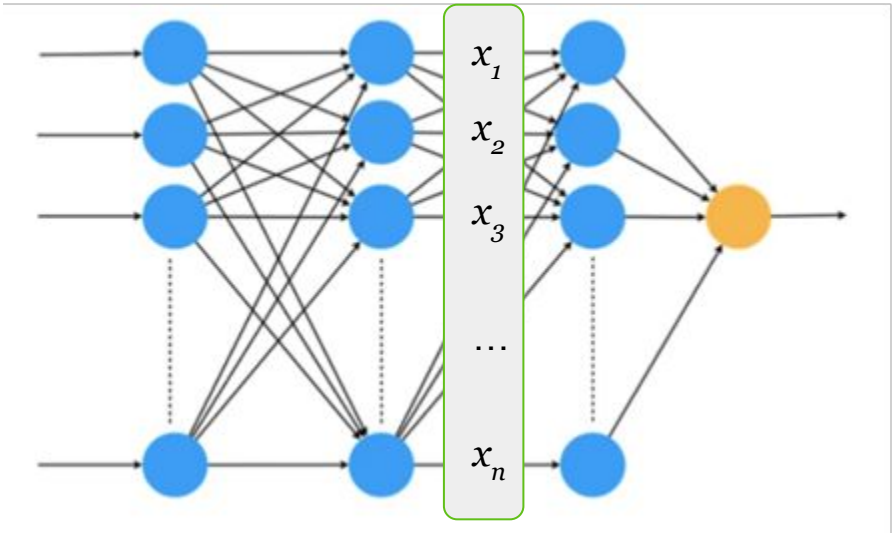


Inputs Outputs

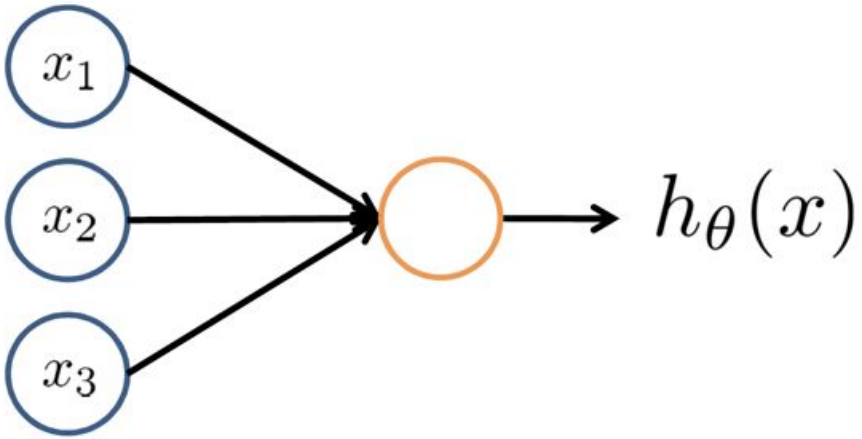
Neural Model: Stack Neurons Together by Layers



Neural Model: Stack Neurons Together by Layers



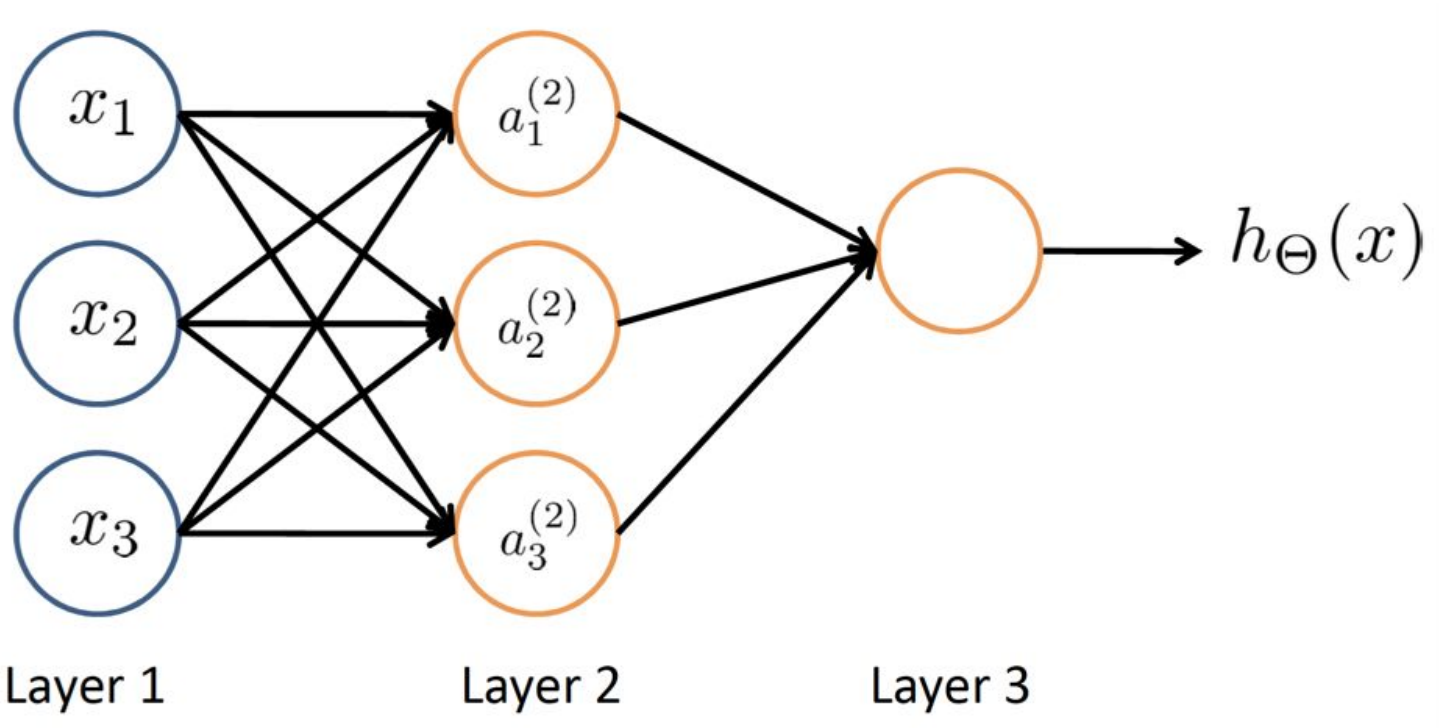
Neuron Model:



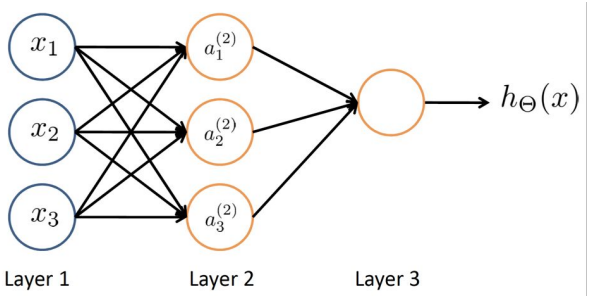
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

Sigmoid (logistic) activation function.

Neural Network



Neural Network



$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer $j+1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

If network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$.