USC Viterbi
School of Engineering
Department of Computer Science

CSCI 544
Applied Natural Language Processing

Mohammad Rostami
USC Computer Science Department

Logistical Notes

- HW1::
  - Report: explaining how you solve the problem along with the requested output + Jupyter Notebook with printed output + executable .py (different software versions are OK but specify the version clearly in your report)
- Project Group Formation Deadline: 09/12
  - We will form random assignment after this date (30 groups so far)
  - Check slack/Piazza and the Excel sheet (Group of 3 or more on Excel)
  - Meet weekly, helpful for both HW and project
- Paper Selection Deadline: 09/19
  - Focus on project topic
  - Last year projects: YouTube
  - Read papers from venues such as EMNLP, ACL, NAACL, etc. (often needed in industry as well)
  - Pick your paper soon: https://docs.google.com/spreadsheets/d/1_vafG77ijmETCnuVZvKpT35k--5op5wn71GZXgAY7O0

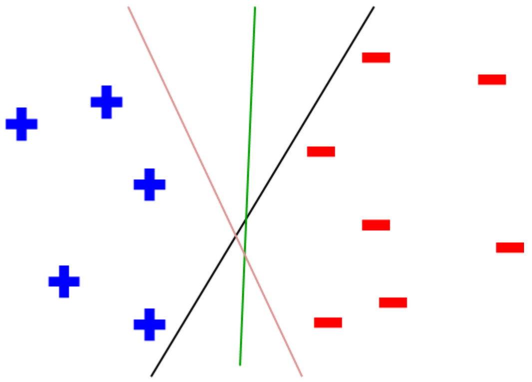
Linear models

- A linear function in *n*-dimensional space (i.e. we have *n* features) is define by *n*+1 weights:

$$Y = \sum_{i=0}^{n} \beta_i X_i$$

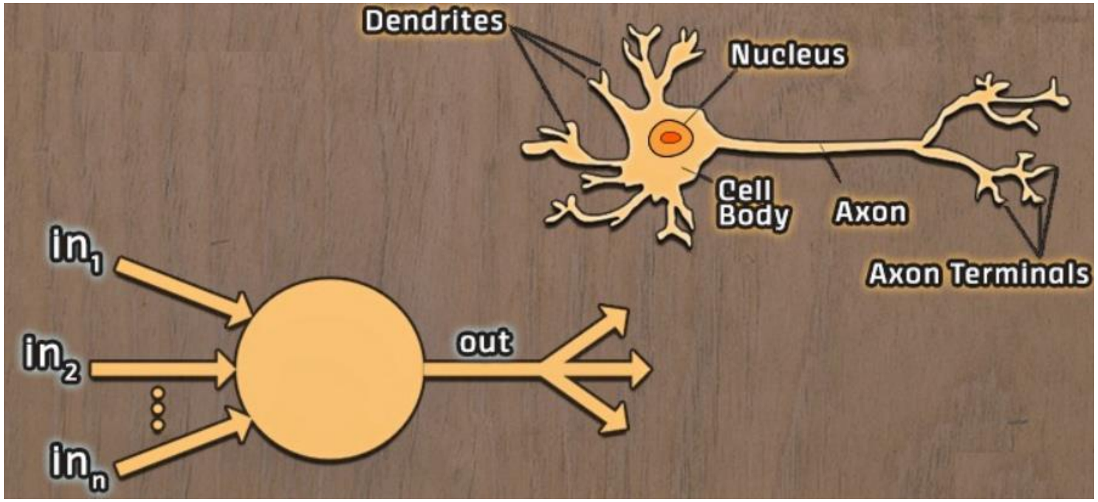
- We find the model weights such that the linear function acts as a good predictive model

- Is not necessarily unique!



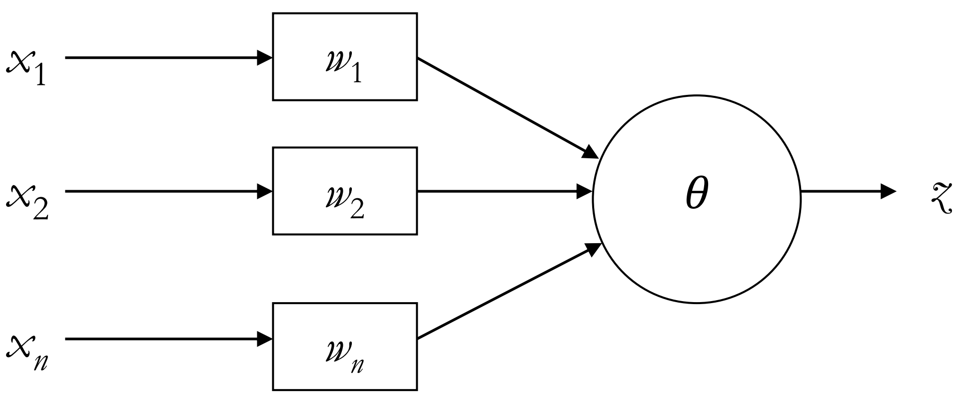
Perceptron

- Invented by Frank Rosenblatt in 1969
- Inspired by the nervous system
- Unit-based: analogous to a neural cell
- Model: neural activity is modeled by mathematical operations



Perceptron

- First neural network learning model in the 1960’s
- Simple and limited (single layer models)
- Basic concepts are similar for multi-layer models so this is a good learning tool
- Still used in current applications (modems, etc.)
- We use discretization to solve classification tasks

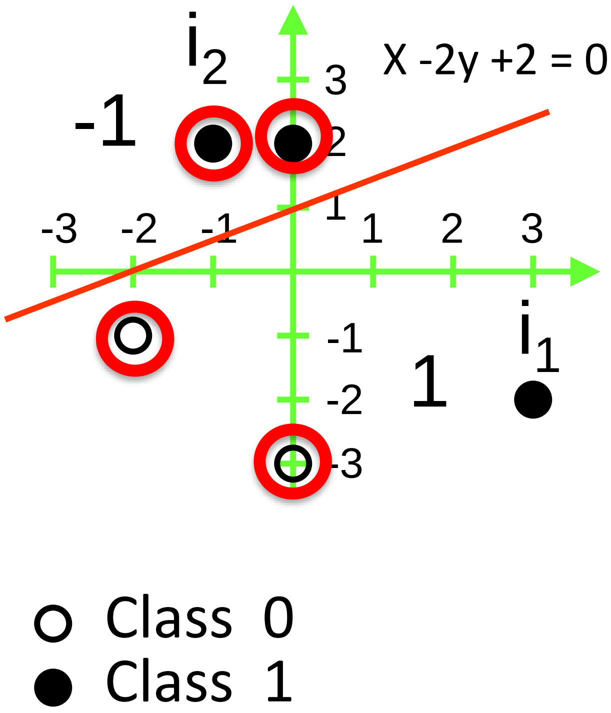


$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$

Perceptron Learning Algorithm

- An iterative algorithm:
- We initialize weights with random values
- We do several pass on the whole training dataset one by one and update the weights
- We will not necessarily find a unique solution

- Least perturbation principle
  - Only change weights if there is an error
  - Use small *learning rate (l)* to change weights sufficiently to make the current pattern correct
- Each iteration through the training set is an *epoch*



Perceptron Learning Algorithm

- Let w_i be the weights vector at iteration, *l* be a constant (the learning rate), *y* be the label for the current iteration, *z* be the current **thresholded** output, and *x* be the input

$$\beta_{i+1} = \beta_i + \Delta\beta$$
$$\Delta\beta = l(y - z)x$$

y	z	$y - z$
0	0	0
0	1	-1
1	0	1
1	1	0

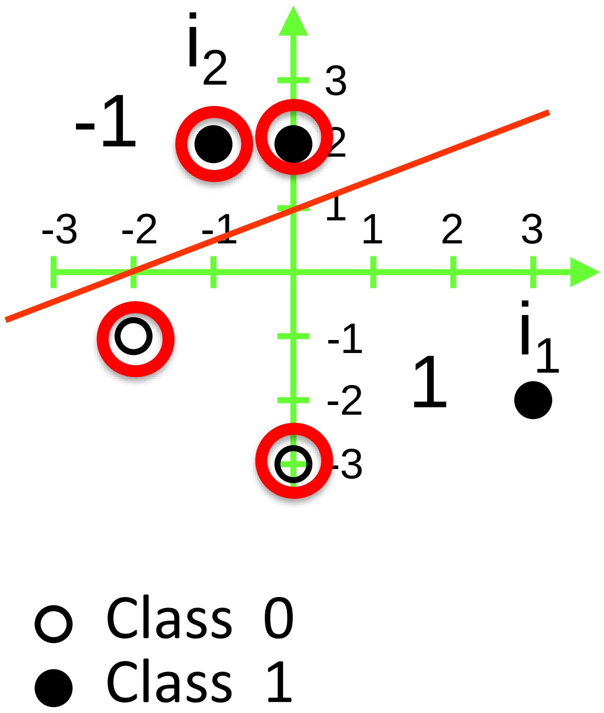
- Continue training until total training set error ceases to improve
- Perceptron Convergence Theorem: Guaranteed to find a solution in finite time if a solution exists

Perceptron Learning Example

We would like a perceptron to correctly classify the five 2-dimensional data points below.

Let the random initial weight vector $\boldsymbol{\beta}_0 = (1, \text{-}2, 2)^T$, why? (ax+by+c = 0 or [a b c][x y 1] ^{T} =0)

Then the dividing line crosses at $(0, 1)^T$ and $(-2, 0)^T$.



Let us pick the misclassified point $(-2, -1)^T$ for learning assuming l = 1:

$\mathbf{x}_1$ = $[-2, -1,1]^T$ (include offset 1)

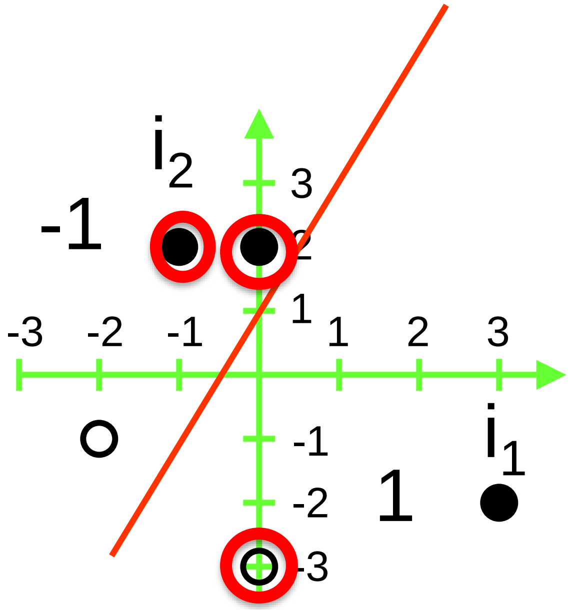
$\mathbf{z}_1$ = $[1,-2,2]\cdot(-2, -1, 1)^T$ = 2 -> 1 ($\mathbf{x}_1$ is in class 0) $(\mathbf{y}_1 - \mathbf{z}_1 = \text{-}1)$

$\boldsymbol{\beta}_1 = \boldsymbol{\beta}_0 - \mathbf{x}_1$ (let us set l = 1 for simplicity)

$\boldsymbol{\beta}_1 = (1, -2, 2)^T - (-2, -1, 1)^T = (3, -1, 1)^T$

The new dividing line crosses at $(0, 1)^T$ and $(-1/3, 0)^T$.

Perceptron Learning Example



○ Class 0
● Class 1

Let us pick the misclassified point $(-1, 2)^T$ for learning:

$\mathbf{x}_2 = [-1, 2, 1]^T$ (include offset 1)

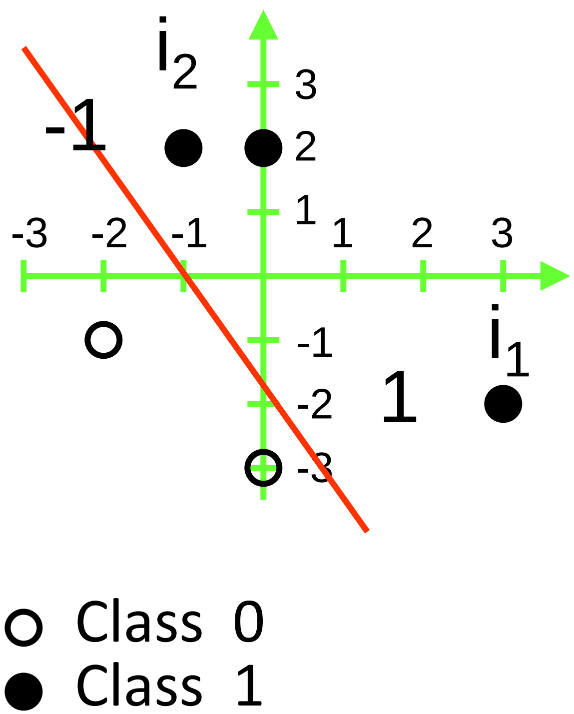
$\mathbf{z}_2 = [3,-1,1] \cdot (-1, 2, 1)^T = -4 \to 0$ ($\mathbf{x}_2$ is in class 1) ($\mathbf{y}_2 - \mathbf{z}_2 = 1$)

$\boldsymbol{\beta}_2 = \boldsymbol{\beta}_1 + \mathbf{x}_2$ (let us set l = 1 for simplicity)

$\boldsymbol{\beta}_2 = (3, -1, 1)^T + (-1, 2, 1)^T = (2, 1, 2)^T$

The new dividing line crosses at $(0, -2)^T$ and $(-1, 0)^T$.

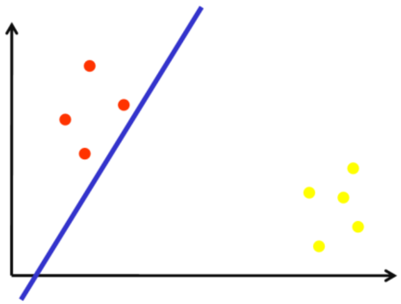
Perceptron Learning Example



- The algorithm has converged in one epoch because all the points are classified correctly
- In general, we may need several epochs
- Learning rate value is important for fast convergence
- At each epoch, we may use a different random order on the data points
- Neural networks, including the state-of-the-art deep networks are extensions of perceptron (potentially using different learning algorithms)

Optimal Boundary

- The perceptron learning algorithm is guaranteed to find a solution if the data points are linearly separable
- But are perceptron solutions optimal?



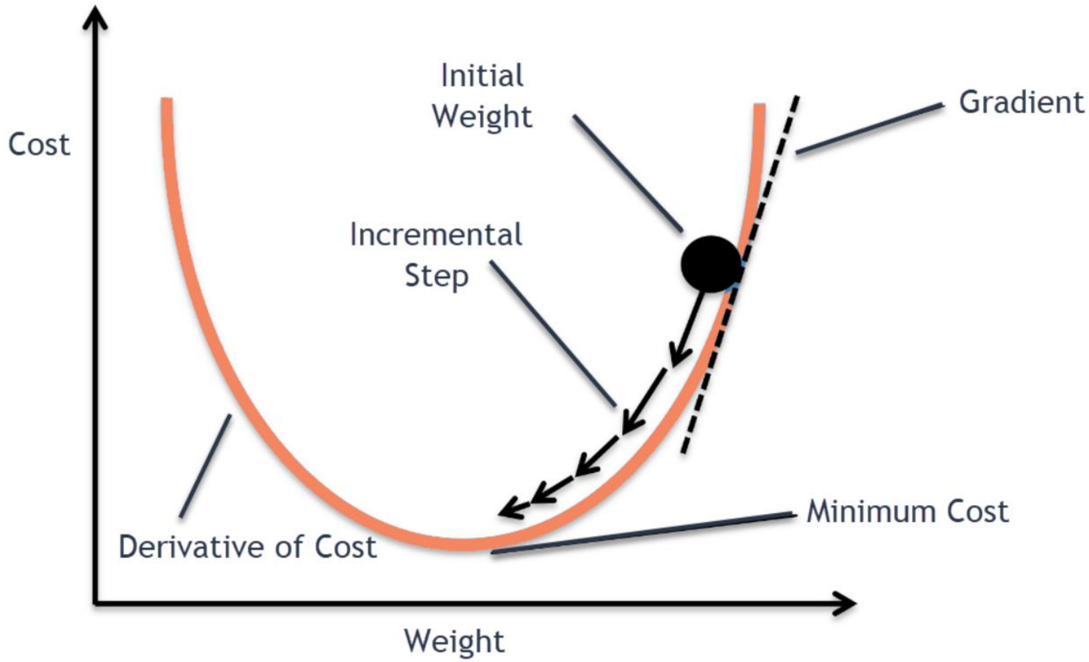
Perfect classification of training samples but may not generalize well to new (untrained) samples.

- Idea: compute a continuous, differentiable error function between input and desired output

- Define an objective function to measure quality of a model
- Find the weights for which the objective function is minimal.
- With a differential function, we can use the gradient descent techniques to find a good solution (at least locally optimal)

Gradient decent

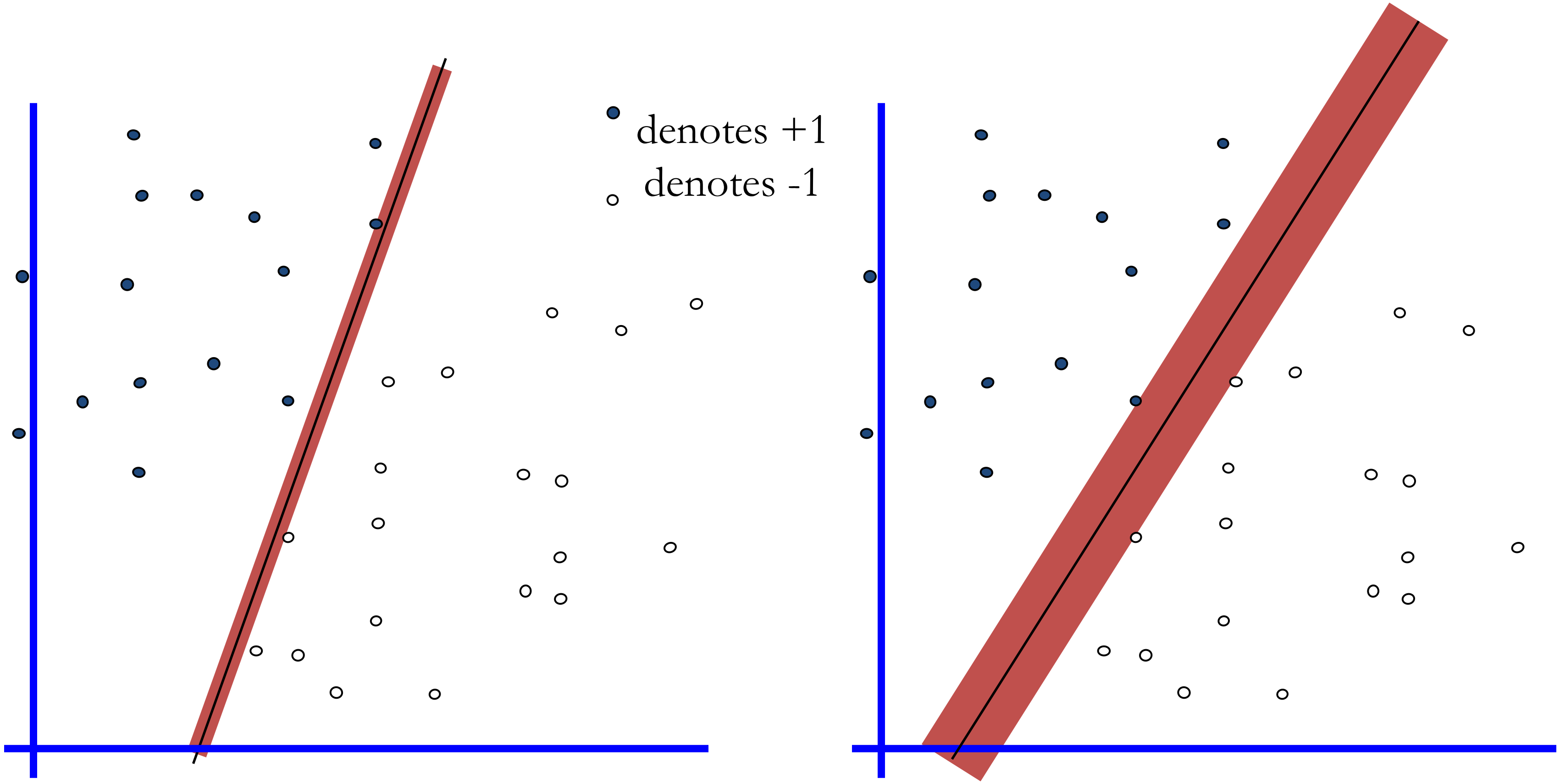
- Suitable for finding minimums of convex differentiable functions (cost functions)



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

Support Vector Machine

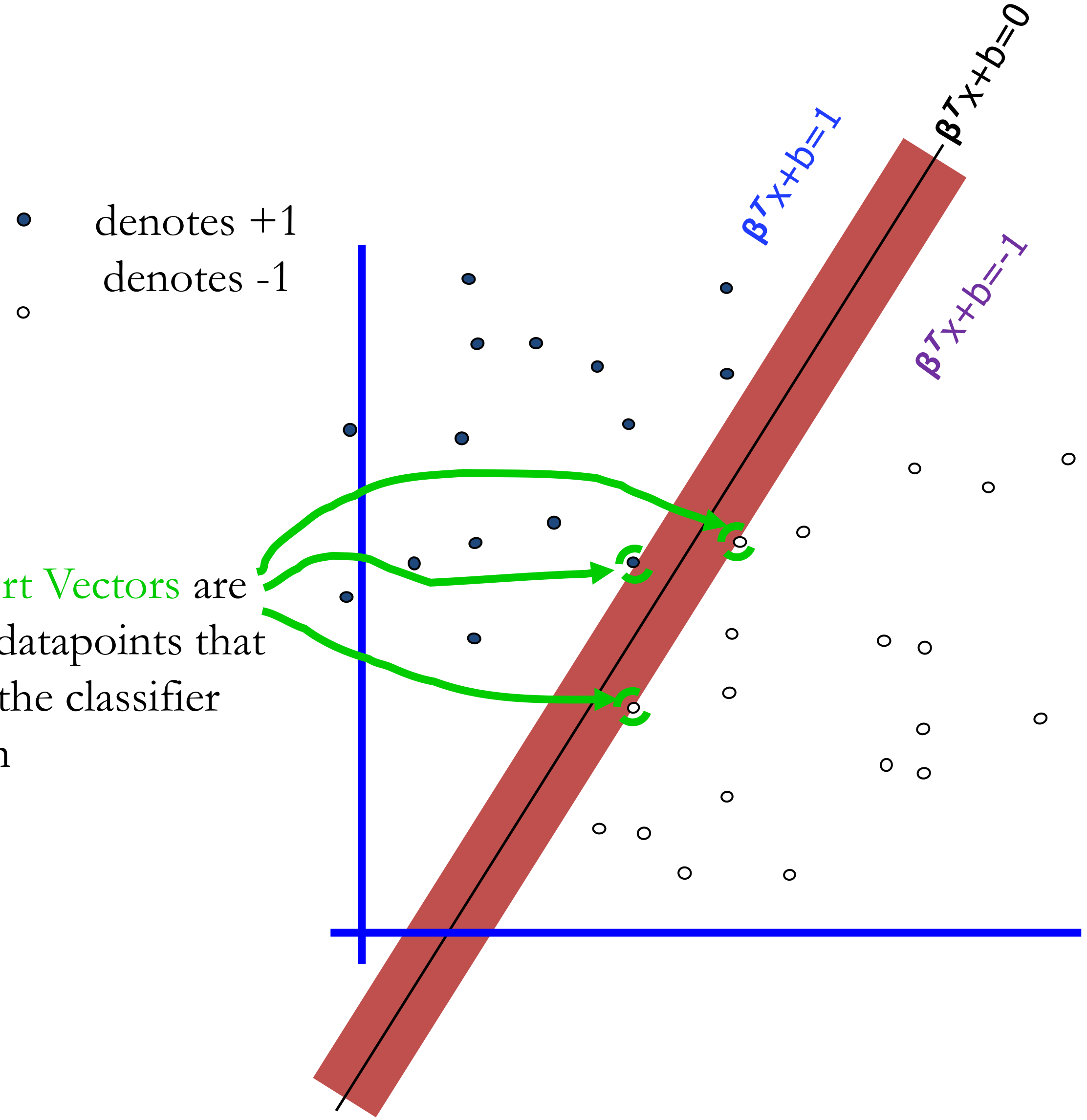
The margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.



Support Vector Machine (SVM): the maximum margin linear classifier

Adopted from slides by Andrew Moore

Support Vector Machine



Support Vectors are those datapoints that lie on the classifier margin

Adopted from slides by Andrew Moore

Computing the Margin Width

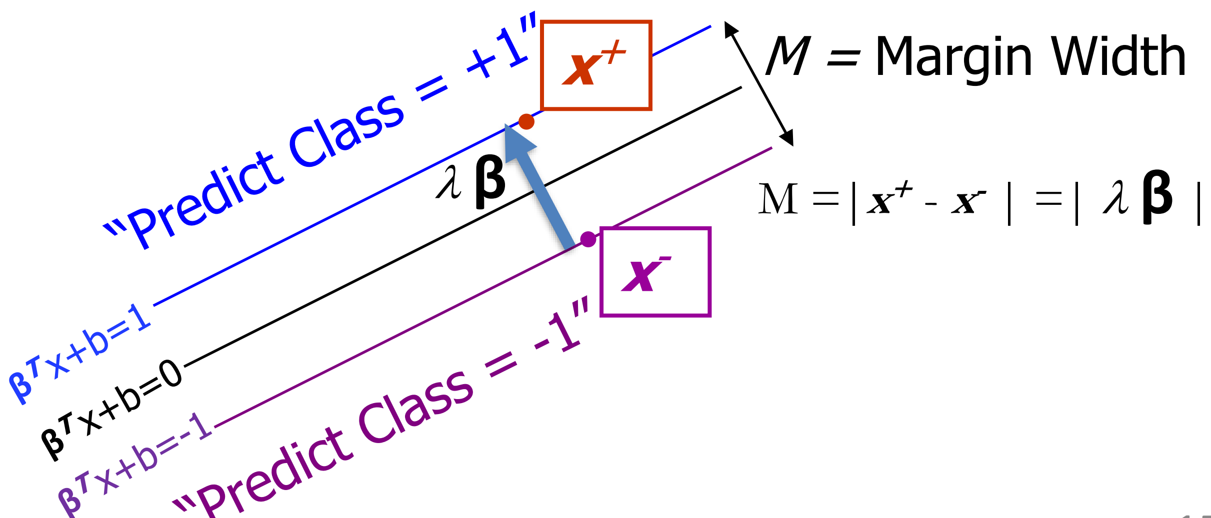
- Plus-plane = $\{ \mathbf{x} : \boldsymbol{\beta}^T . \mathbf{x} + b = +1 \}$
- Minus-plane = $\{ \mathbf{x} : \boldsymbol{\beta}^T . \mathbf{x} + b = -1 \}$
- Let $\mathbf{x}^-$ be any point on the minus plane
- Let $\mathbf{x}^+$ be the closest plus-plane-point to $\mathbf{x}^-$.
- We can deduce:

$$\mathrm{M} = |\, \mathbf{x}^+ - \mathbf{x}^- \,| = |\, \lambda \boldsymbol{\beta} \,|$$

$$\begin{matrix} \boldsymbol{\beta}^T . \mathbf{x}^+ + b = +1 \\ \boldsymbol{\beta}^T . \mathbf{x}^- + b = -1 \end{matrix} \longrightarrow \boldsymbol{\beta}^T . (\mathbf{x}^+ - \mathbf{x}^-) = 2 \longrightarrow \boldsymbol{\beta}^T . (\lambda \boldsymbol{\beta}) = 2 \longrightarrow \lambda = \frac{2}{\boldsymbol{\beta}^T . \boldsymbol{\beta}}$$

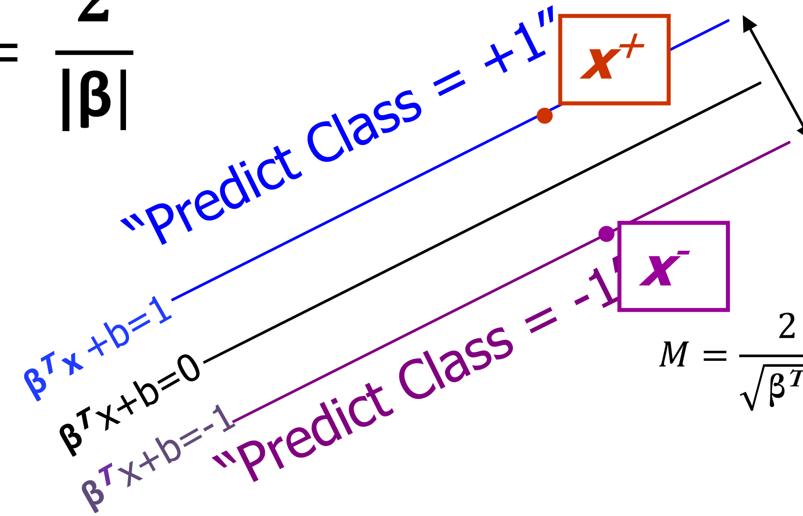
$$\mathrm{M} = |\, \mathbf{x}^+ - \mathbf{x}^- \,|$$

$$\mathbf{x}^+ = \mathbf{x}^- + \lambda \boldsymbol{\beta}$$



Computing the margin width

$$M = |\boldsymbol{x}^{+} - \boldsymbol{x}^{-}| = |\lambda \boldsymbol{\beta}| = \left|\frac{2}{\boldsymbol{\beta}^{T}.\boldsymbol{\beta}}\boldsymbol{\beta}\right| = \frac{2}{|\boldsymbol{\beta}|}$$



$$M = \frac{2}{\sqrt{\boldsymbol{\beta}^{T}.\boldsymbol{\beta}}}$$

Given **β** and *b* (a boundary), we can:

- Classify all data points (only one boundary is optimal)
- Compute the width of the margin

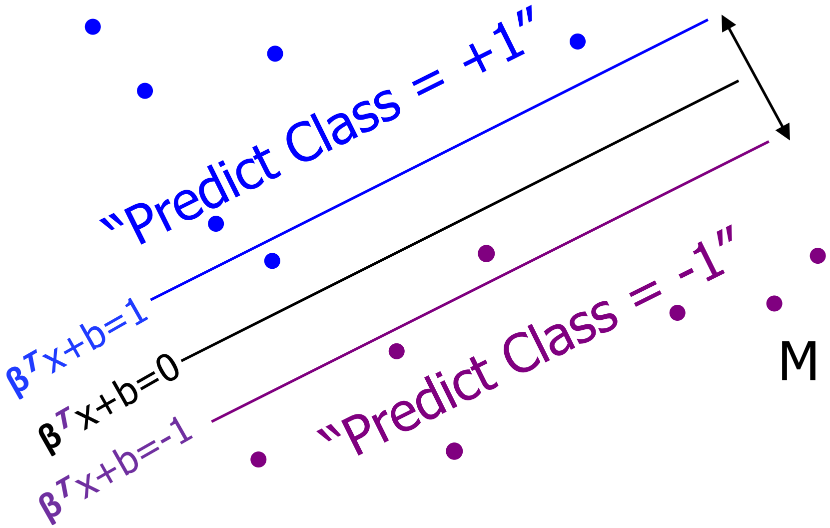
Now we need to write an optimization problem to search for the optimal **β** and *b* with the widest margin that classifies all the training data points correctly

Minimize $|\boldsymbol{\beta}|$ Subject to

$\boldsymbol{\beta}^{T}.\,\boldsymbol{x} + b > 1$; when in class +1
$\boldsymbol{\beta}^{T}.\,\boldsymbol{x} + b < -1$; when in class -1

Adopted from slides by Andrew Moore

Learning the Maximum Margin Classifier



Minimize $\boldsymbol{\beta}^T.\boldsymbol{\beta}$ Subject to

$(\boldsymbol{\beta}^T.\ \mathbf{x}_k + b)\ y_k >= 1$ for all $(\mathbf{x}_k, y_k)$

$$M = \frac{2}{|\beta|}$$

Incorporating soft constraints

Min $\frac{1}{2}\beta^T.\beta + C\sum_{k=1}^{R} \varepsilon_k$ s.t $(\boldsymbol{\beta}^T.\ \mathbf{x}_k + b)\ y_k >= 1-\varepsilon_k$ for all $(\mathbf{x}_k, y_k)$

$0 <= \varepsilon_k <= 1$

Solving the above problem using quadratic programming is a straightforward task

Adopted from slides by Andrew Moore

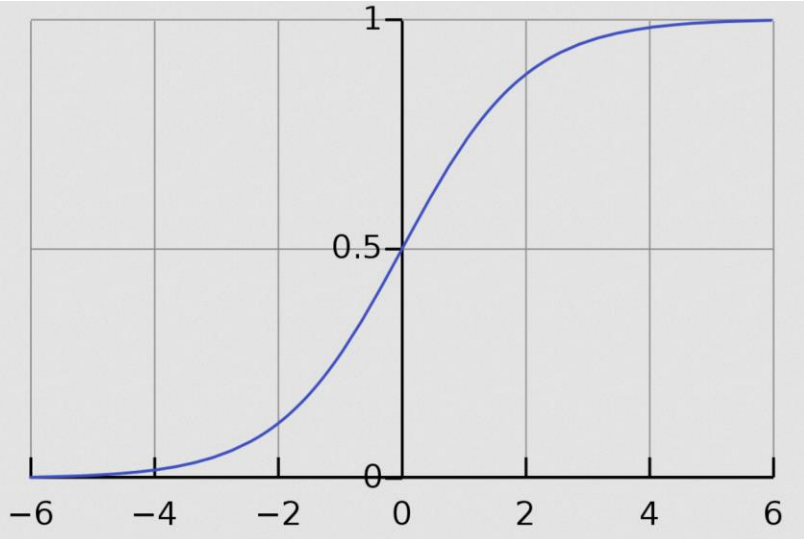
Logistic Regression

- Maximum Likelihood Estimation:

$$\hat{Y} = \arg\max P(Y|X)$$

- We assume that the likelihood function is a logistic function of a linear relationship

$$P(Y=1|X) = \frac{1}{1+e^{-\beta^T X}}$$



Logistic Regression

- Log-likelihood Inference:

Analogous to using Entropy loss

$$\hat{\beta} = \arg\max_{\beta} \log(\Pi_i P(Y_i|X_i)) =$$

$$\arg\max_{\beta} \Sigma_i \log(P(Y_i|X_i)) =$$

$$\arg\max_{\beta} \Sigma_i Y_i \log(P(Y_i = 1|X_i)) + (1 - Y_i) \log(P(Y_i = 0|X_i)) =$$

$$\arg\max_{\beta} \Sigma_i Y_i (-\log(1+e^{\beta^T X_i}) + (1-Y_i)(-\beta^T X_i) - \log(1+e^{\beta^T X_i}) =$$

$$\arg\min_{\beta} \Sigma_i (1 - Y_i)\beta^T X_i + \log(1 + e^{\beta^T X_i})$$

- There is no closed form solution:

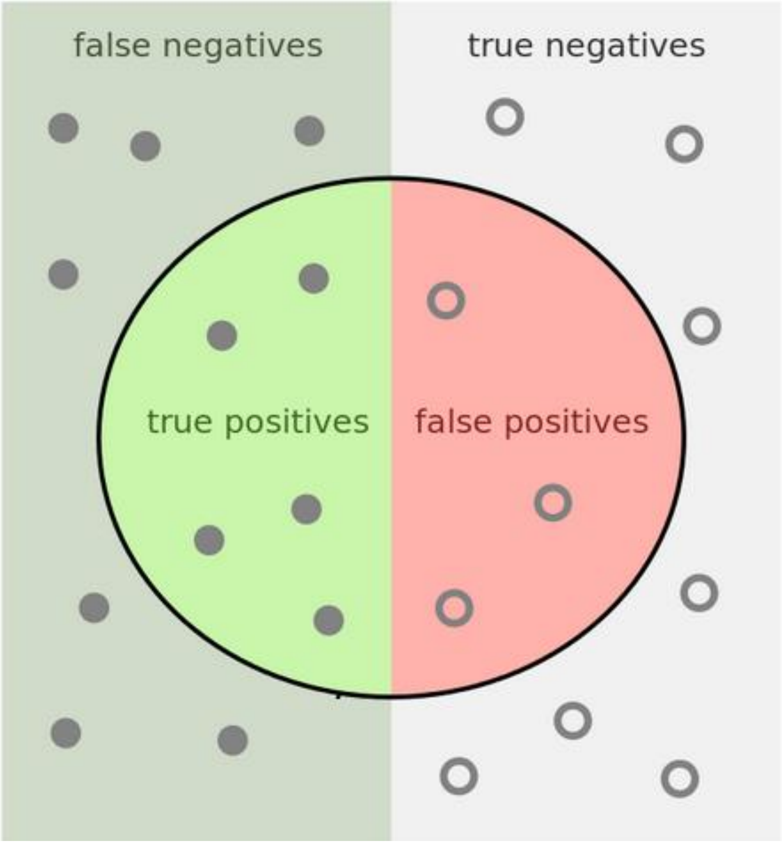
- We can use numerical optimization, e.g., Newton method
- We can approximate the logit term

Model Evaluation Process

- We use a training dataset for model selection
- A **good** parametric model along with a **suitable** training algorithm guarantees training a model that works well on the training data
- We need to validate that trained models **generalize** well on unseen data instances
- We need a second testing dataset which is fully independent of the training dataset
- We randomly split the annotated dataset into testing and training splits (sometimes, a validation set is generated as well)

Evaluation Metrics

- **Accuracy**: proportion of correctly classified items
- Accuracy can be dominated by **true negatives** (items correctly classified as not in a class).
- Sensitive with respect to imbalance
- Precision: $\frac{\text{True Positives}}{\text{True Positives+False Positive}}$
- Recall: $\frac{\text{True Positives}}{\text{True Positives+False negative}}$
- Precision and recall are not useful metrics when used in isolation?
- We want our model to have good performance with respect to both metrics
- Implemented in sklearn



Evaluation Metrics

- Why having one measure is helpful?
- $\text{F1} = \dfrac{2\ \text{Precision Recall}}{\text{Precision} + \text{Recall}}$
- F1 is biased towards the lower of precision and recall:

- harmonic mean < geometric mean < arithmetic mean
- F1=0 when Precision=0 or Recall=0

- Generalized F score:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}.$$