Recap

- (Ch 11) Learning to classify
  - Evaluating a classifier
  - Nearest neighbors classifier
  - Naïve Bayes classifier

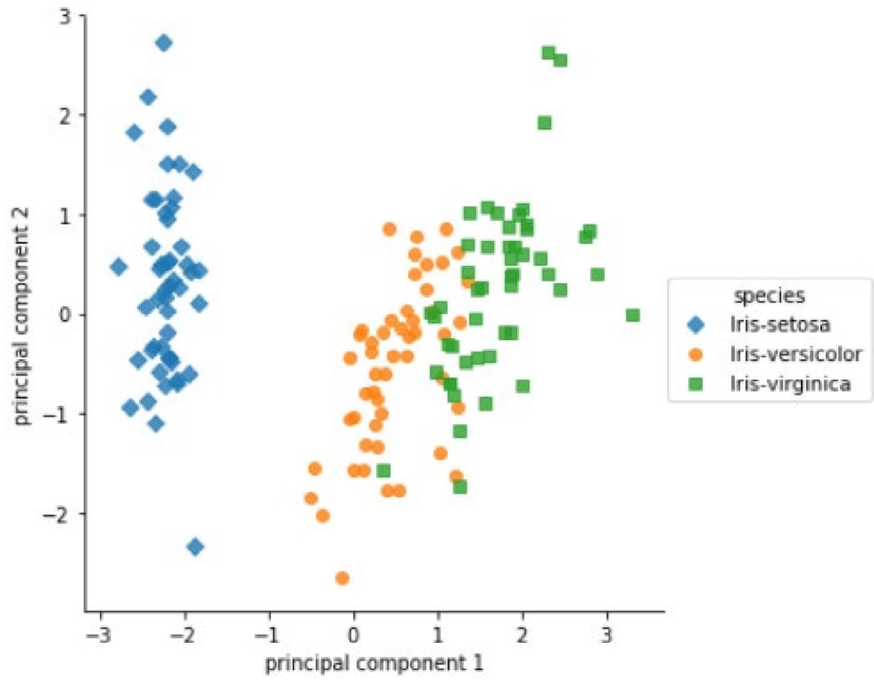
Today

- (Ch 11) Learning to classify
  - Support vector machine (SVM) classifier

Learning to classify

Given a set of **feature vectors** $\mathbf{x}_i$, where each has a **class label** y_i, we want to train a classifier that can map unlabeled vectors to labels

sepal length	sepal width	petal length	petal width	species
5.1	3.5	1.4	0.2	Iris-setosa
7.0	3.2	4.7	1.4	Iris-versicolor
6.3	3.3	6.0	2.5	Iris-virginica



How do we know if our classifier is good?

- We want the classifier to avoid making classification mistakes on unlabeled data that we will only see at run-time

- Problem 1: some mistakes may be more costly than others

  We can tabulate different types of error and define a **loss function**

- Problem 2: we will never know the true labels of the run-time data

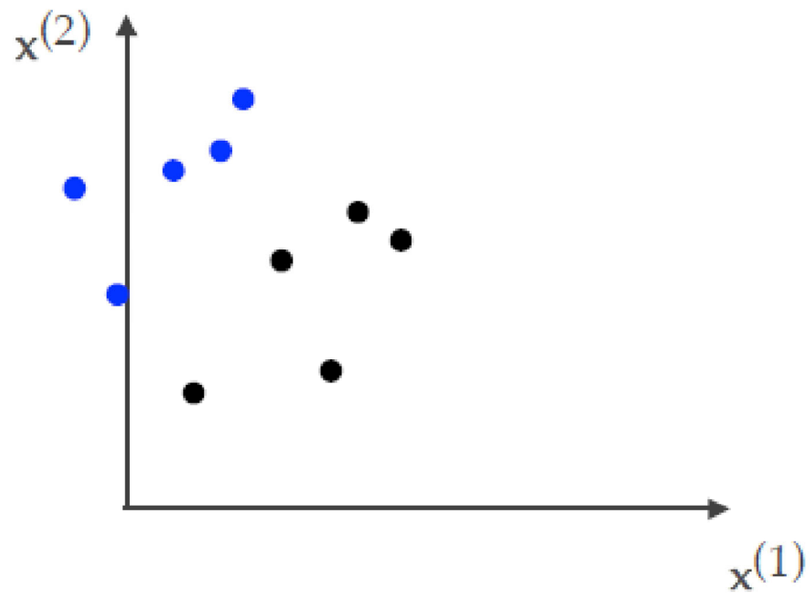
  We must separate the labeled data into a **training set** and a **validation/test set**

Support vector machine (SVM) overview

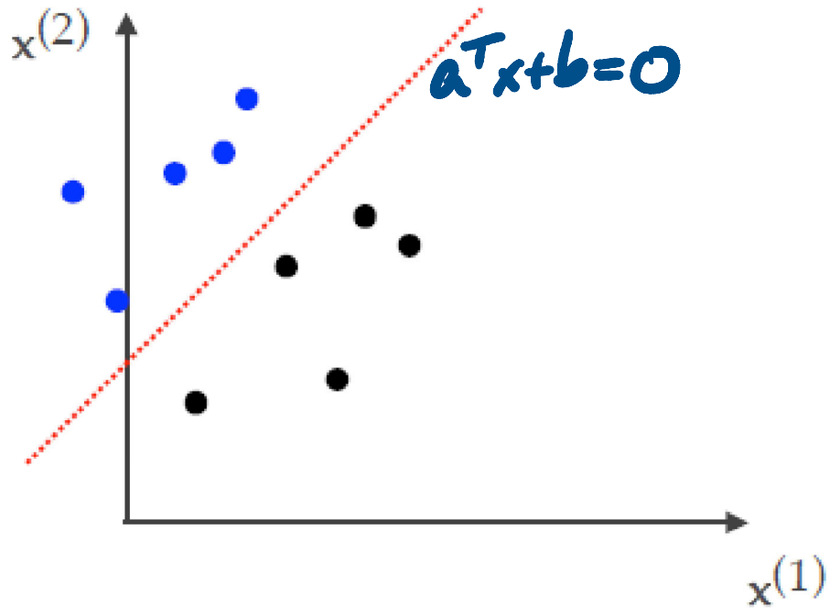
- Decision boundary and classification function
- Loss function
- Training
- Validation/testing
- Extension to multiclass classification

SVM problem formulation

- At first we assume a binary classification problem
- The training set consists of N items
  - Feature vectors $\mathbf{x}_i$ of dimension d
  - Corresponding class labels $y_i \in \{\pm 1\}$
- We can picture the training data as a d-dimensional scatter plot with colored labels



Decision boundary



- An SVM uses a hyperplane as its **decision boundary**

- Mathematically, the decision boundary is

$$a_1 x^{(1)} + a_2 x^{(2)} + \cdots + a_d x^{(d)} + b = 0$$

- In vector notation, the decision boundary is

$$\mathbf{a}^T \mathbf{x} + b = 0$$

Classification function (or prediction function)

- The SVM assigns a label to a feature vector $\mathbf{x}_i$ according to this rule

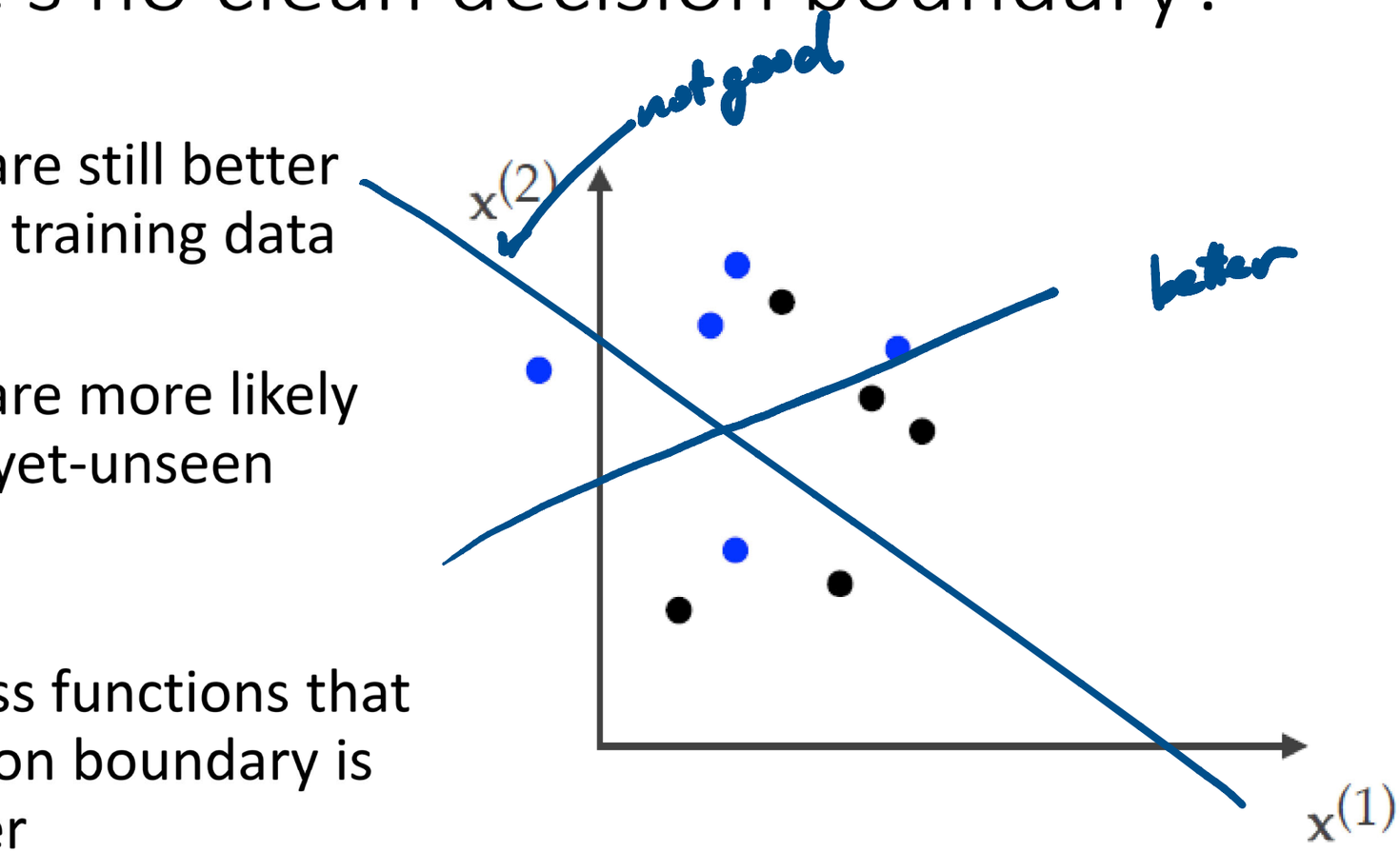
$$\begin{array}{ll} +1 & \text{if } \mathbf{a}^T\mathbf{x}_i + b \geq 0 \\ -1 & \text{if } \mathbf{a}^T\mathbf{x}_i + b < 0 \end{array}$$

- In other words, the classification function is: $\text{sign}\left(\mathbf{a}^T\mathbf{x}_i + b\right)$

- Note that
  - If $\left|\mathbf{a}^T\mathbf{x}_i + b\right|$ is small, then $\mathbf{x}_i$ was close to the decision boundary
  - If $\left|\mathbf{a}^T\mathbf{x}_i + b\right|$ is large, then $\mathbf{x}_i$ was far from the decision boundary

What if there's no clean decision boundary?

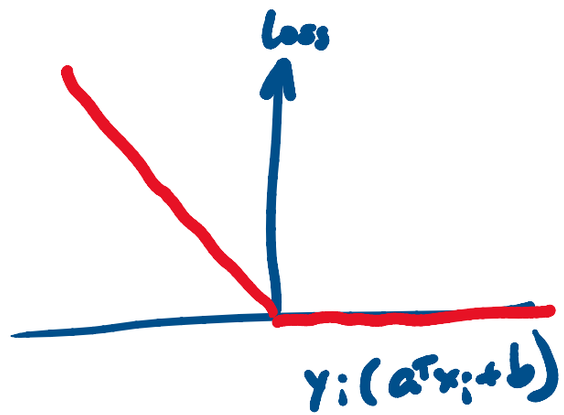
- Some boundaries are still better than others on the training data
- Some boundaries are more likely to be robust to as-yet-unseen run-time data
- We will develop loss functions that tell us if one decision boundary is better than another



Loss function 1

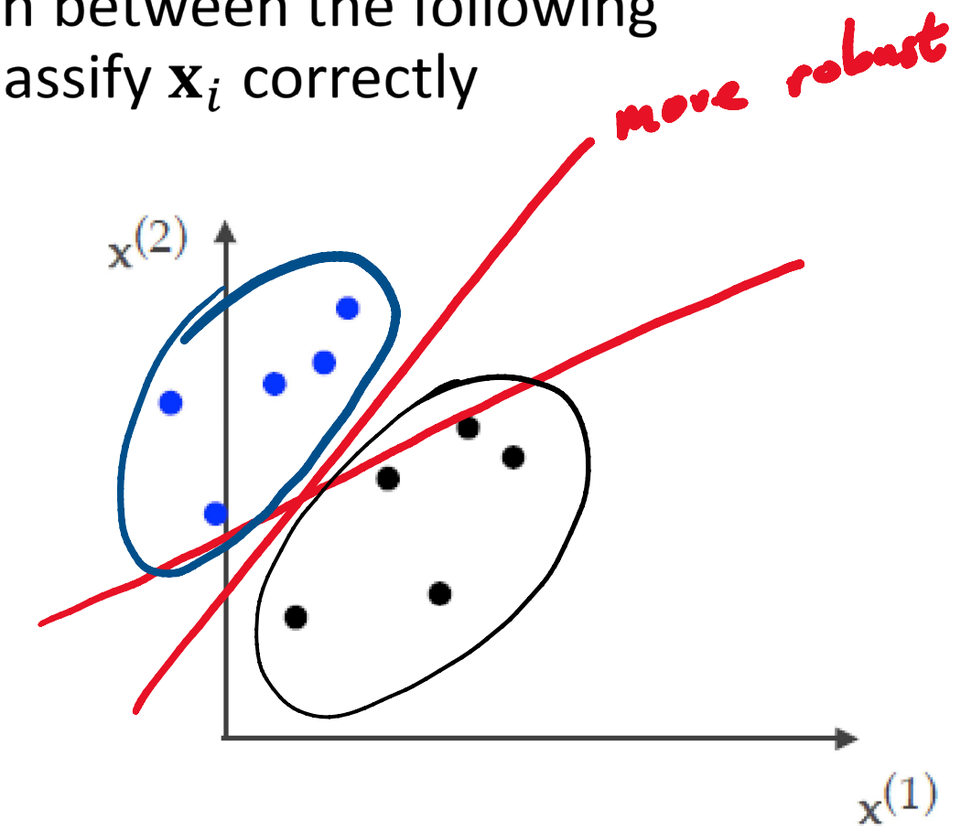
- For any given feature vector $\mathbf{x}_i$ with class label $y_i \in \{\pm 1\}$, we want
  - Zero loss if $\mathbf{x}_i$ is classified correctly: $\text{sign}\left(\mathbf{a}^T\mathbf{x}_i + b\right) = y_i$
  - Positive loss if $\mathbf{x}_i$ is misclassified: $\text{sign}\left(\mathbf{a}^T\mathbf{x}_i + b\right) \neq y_i$
  - If $\mathbf{x}_i$ is misclassified, more loss the further away it is from the boundary
- Loss function 1 meets the requirements: $\max(0, -y_i\left(\mathbf{a}^T\mathbf{x}_i + b\right))$
- Training error cost

$$S(\mathbf{a}, \mathrm{b}) = \frac{1}{N}\sum_{i=1}^{N} \max\left(0, -y_i\left(\mathbf{a}^T\mathbf{x}_i + b\right)\right)$$



The problem with loss function 1

- Loss function 1 does not distinguish between the following decision boundaries if they both classify $\mathbf{x}_i$ correctly
  - One that passes close to $\mathbf{x}_i$
  - One that leaves a wide margin
- But leaving a margin gives robustness to run-time data



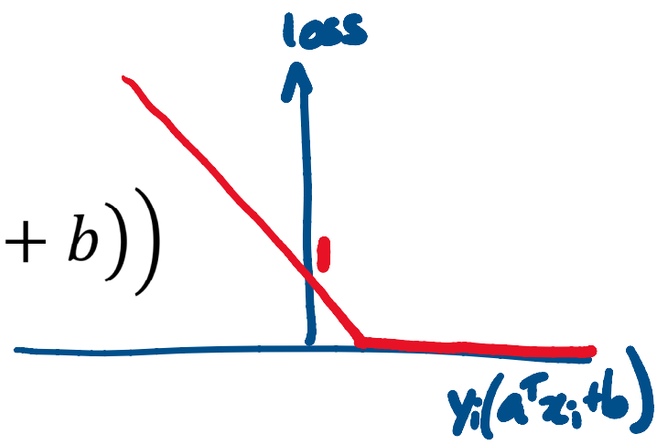
Loss function 2: hinge loss

- We want to impose a small positive loss if $\mathbf{x}_i$ is correctly classified but close to the boundary

- The **hinge loss** meets the requirements: $\max(0, 1 - y_i(\mathbf{a}^T\mathbf{x}_i + b))$

- Training error cost

$$S(\mathbf{a}, \mathrm{b}) = \frac{1}{N}\sum_{i=1}^{N} \max\left(0, 1 - y_i\left(\mathbf{a}^T\mathbf{x}_i + b\right)\right)$$



The problem with loss function 2

- Loss function 2 favors decision boundaries with large $\|\mathbf{a}\|$ because increasing $\|\mathbf{a}\|$ can artificially zero-out the loss for a correctly classified $\mathbf{x}_i$ near the boundary

- But large $\|\mathbf{a}\|$ makes the classification function $\text{sign}\left(\mathbf{a}^T\mathbf{x}_i + b\right)$ extremely sensitive to small changes in $\mathbf{x}_i$, which is bad for robustness to run-time data

- So small $\|\mathbf{a}\|$ is better

Hinge loss with regularization penalty

- We add a penalty on the square magnitude $\|\mathbf{a}\|^2 = \mathbf{a}^T\mathbf{a}$
- Training error cost

$$S(\mathbf{a}, \mathrm{b}) = \left[\frac{1}{N}\sum_{i=1}^{N} \max\left(0, 1 - y_i\left(\mathbf{a}^T\mathbf{x}_i + b\right)\right)\right] + \lambda\left(\frac{\mathbf{a}^T\mathbf{a}}{2}\right)$$

- The **regularization parameter** λ trades off between the two objectives

Training procedure

The training error cost $S(\mathbf{a}, \mathrm{b})$ is a function of the decision boundary parameters $(\mathbf{a}, \mathrm{b})$, so it can help us find the best decision boundary

- Fix λ and set some initial values of $(\mathbf{a}, \mathrm{b})$
- Search iteratively for $(\mathbf{a}, \mathrm{b})$ that minimize $S(\mathbf{a}, \mathrm{b})$ on the training set
- Repeat the previous steps for several values of λ and choose the one that gives the decision boundary with best accuracy on a validation set

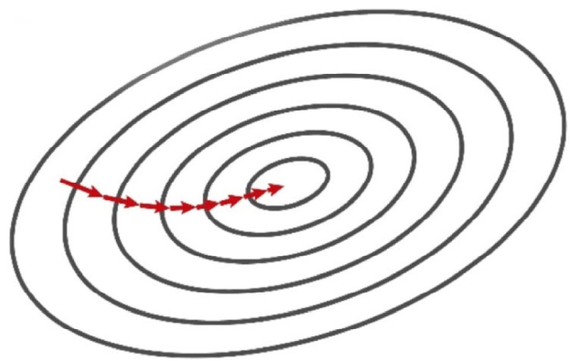
Iterative minimization by gradient descent

- For simplicity, we write

$$S(\mathbf{a}, \mathrm{b}) = \left[\frac{1}{N}\sum_{i=1}^{N} \max\left(0, 1 - y_i\left(\mathbf{a}^T\mathbf{x}_i + b\right)\right)\right] + \lambda\left(\frac{\mathbf{a}^T\mathbf{a}}{2}\right) = \left[\frac{1}{N}\sum_{i=1}^{N} S_i\left(\mathbf{a}, \mathrm{b}\right)\right] + S_0(\mathbf{a}, \mathrm{b})$$

- The direction of steepest descent is

$$-\nabla S(\mathbf{a}, \mathrm{b}) = -\left[\frac{1}{N}\sum_{i=1}^{N} \nabla S_i(\mathbf{a}, \mathrm{b})\right] - \nabla S_0(\mathbf{a}, \mathrm{b})$$



Source: Chris Fregly

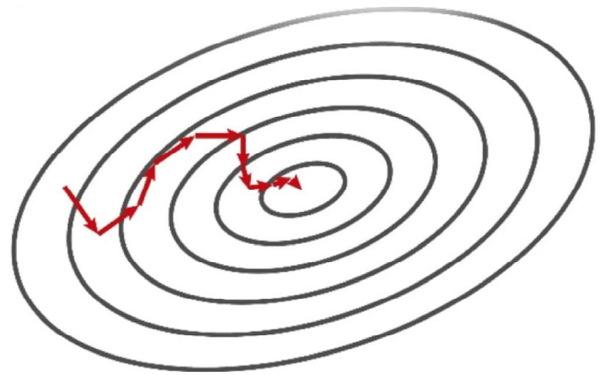
Stochastic gradient descent

- The exact $-\nabla S(\mathbf{a}, \mathrm{b})$ is too cumbersome to evaluate for large N, so we approximate in each step by randomly sampling a single vector $\mathbf{x}_k \in \{\mathbf{x}_i\}$ with replacement

Direction of descent: $-\nabla S(\mathbf{a}, \mathrm{b}) \approx -\nabla S_k(\mathbf{a}, \mathrm{b}) - \nabla S_0(\mathbf{a}, \mathrm{b})$

- Recall that

$$\nabla S_k(\mathbf{a}, \mathrm{b}) = \begin{bmatrix} \frac{\partial S_k}{\partial a_1} \\ \vdots \\ \frac{\partial S_k}{\partial b} \end{bmatrix}$$



Source: Chris Fregly

Update equations for parameters $(\mathbf{a}, \mathrm{b})$

Since $S_k(\mathbf{a}, \mathrm{b}) = \max\left(0, 1 - y_k\left(\mathbf{a}^T\mathbf{x}_k + b\right)\right)$ and $S_0(\mathbf{a}, \mathrm{b}) = \lambda\left(\frac{\mathbf{a}^T\mathbf{a}}{2}\right)$, we have the following update equations

If $y_k\left(\mathbf{a}^T\mathbf{x}_k + b\right) \geq 1$	If $y_k\left(\mathbf{a}^T\mathbf{x}_k + b\right) < 1$
• $\mathbf{a} \leftarrow \mathbf{a} - \eta(\lambda\mathbf{a})$ • $b \leftarrow b - 0$	• $\mathbf{a} \leftarrow \mathbf{a} - \eta(\lambda\mathbf{a} - y_k\mathbf{x}_k)$ • $b \leftarrow b - \eta(-y_k)$

An **epoch** is a certain number of steps in the descent (usually set to be approximately the size N of the training set)

In the eth epoch, it is common to set the **steplength** $\eta = \frac{m}{e+n}$ where m and n are constants chosen by experiment

Validation/testing

- Split the labeled data into **training**, **validation** and **test** sets
- For each choice of λ, run stochastic gradient descent to find the best decision boundary parameters $(\mathbf{a}, \mathrm{b})$ using the **training** set
- Choose the best λ based on accuracy on the **validation** set
- Finally evaluate the SVM's accuracy on the **test** set
- This process avoids overfitting $(\mathbf{a}, \mathrm{b})$ and λ

Extension to multiclass classification

- All vs. all
  - Train a separate binary classifier for each pair of classes
  - To classify, run all classifiers and see which class label is chosen most
  - Computational complexity scales quadratically with the number of classes
- One vs. all
  - Train a separate binary classifier for each class against all other classes
  - To classify, run all classifiers and see which class label gets the highest score
  - Computational complexity scales linearly with the number of classes
- Both approaches can work well in practice