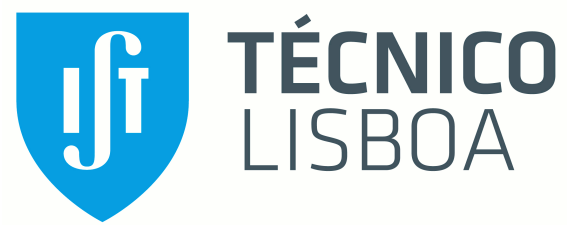
Lecture 3: Linear Models II

André Martins, Francisco Melo, Mário Figueiredo



Deep Learning Course, Winter 2023-2024

Today's Roadmap

- Logistic regression
- Regularization and optimization
- Stochastic gradient descent.

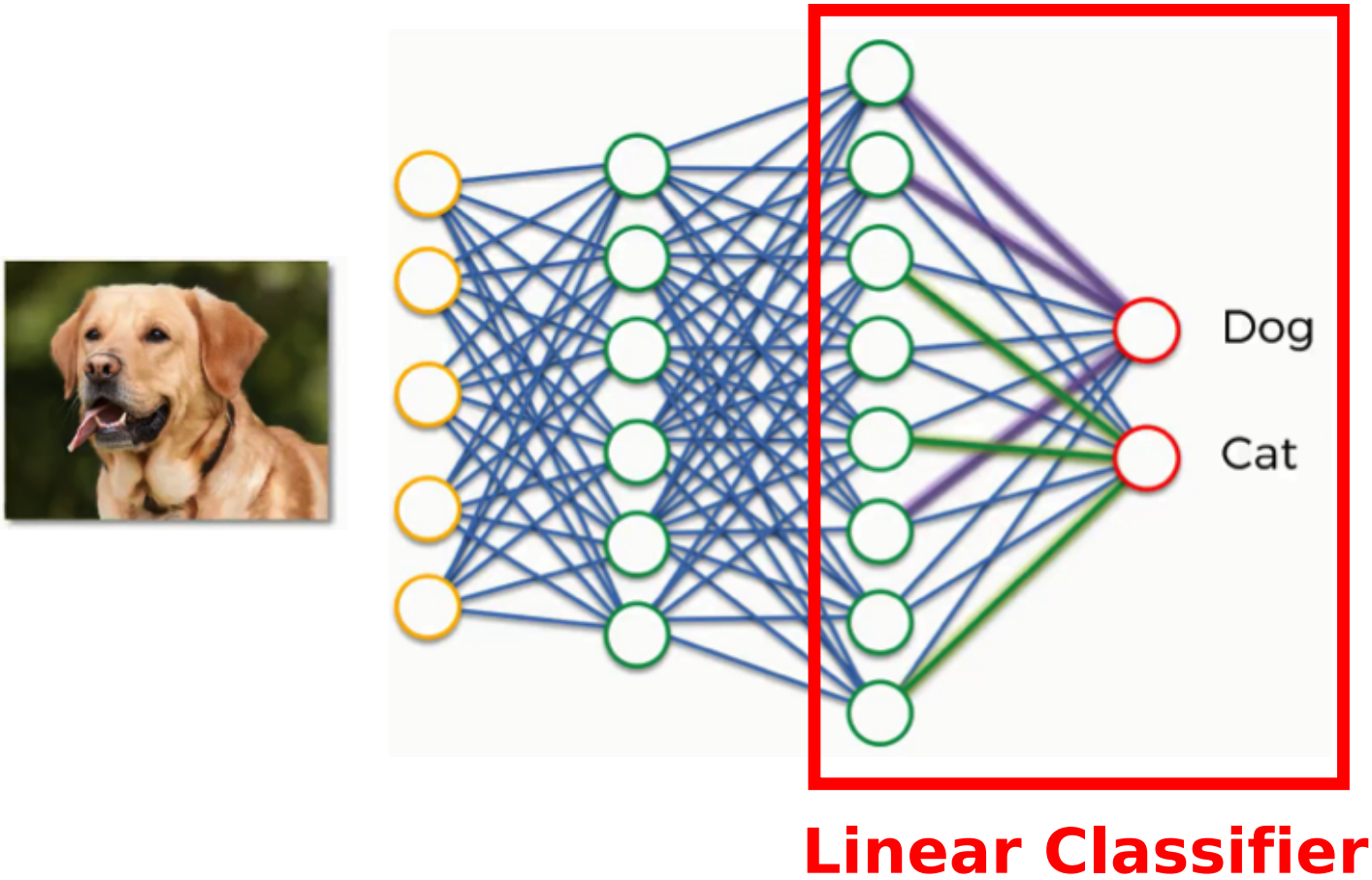
Why Linear Classifiers?

We know the course title promised "deep", ...

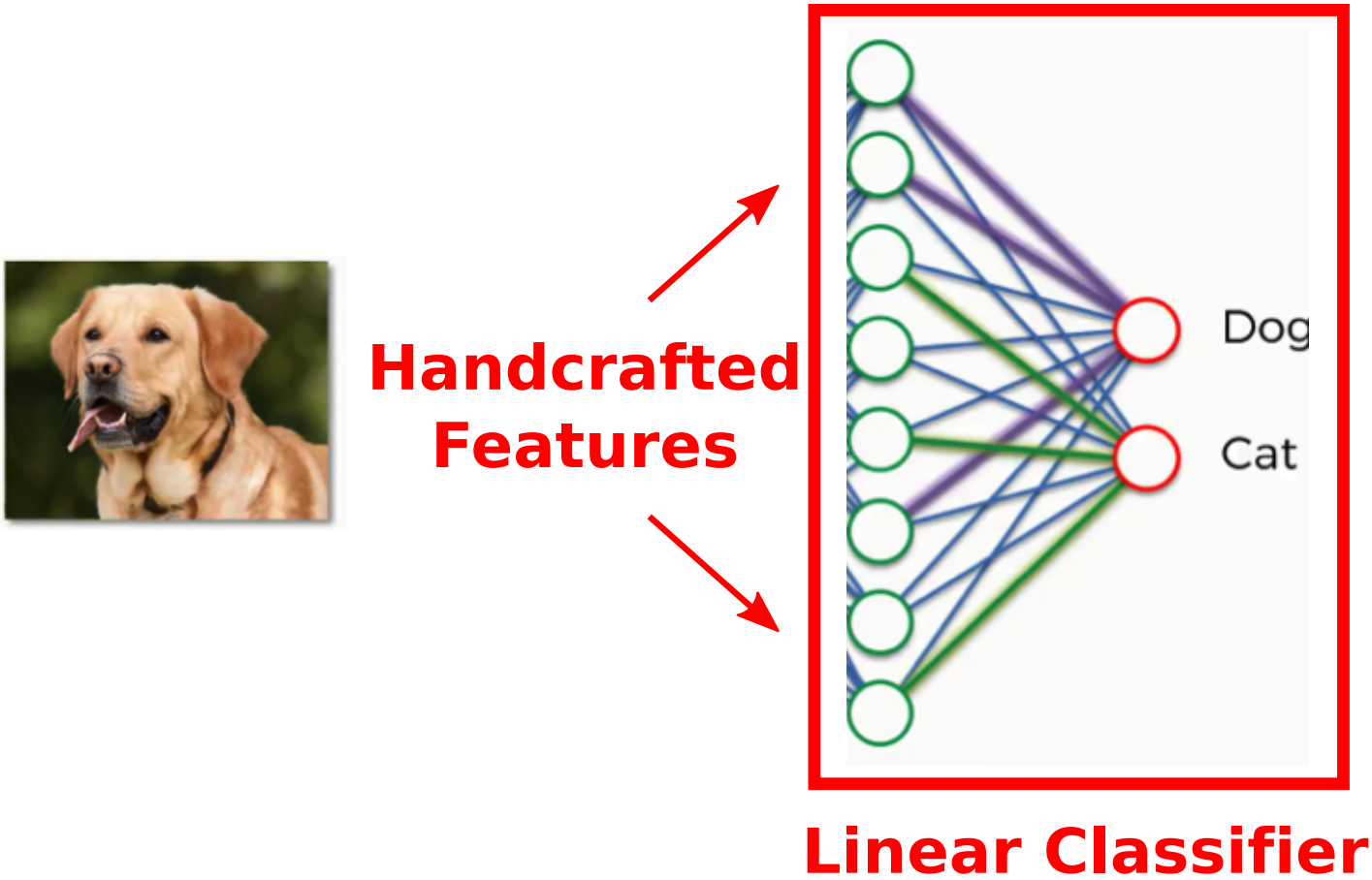
...but (as explained in Lecture 2):

- The underlying machine learning concepts are the same
- The theory (statistics and optimization) are much better understood
- Linear classifiers still widely used (very effective when data is scarce)
- Linear classifiers are **a component of neural networks**.

Linear Classifiers and Neural Networks

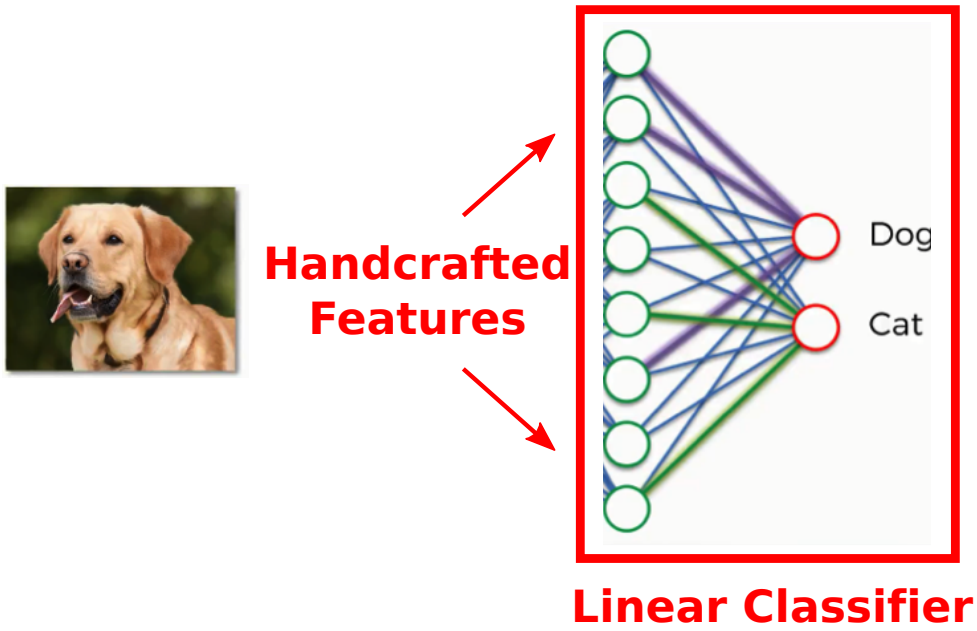


Linear Classifiers and Neural Networks



...in fact, these may also be learned features, from all the previous layers of some deep neural network.

Reminder: Linear Classifier

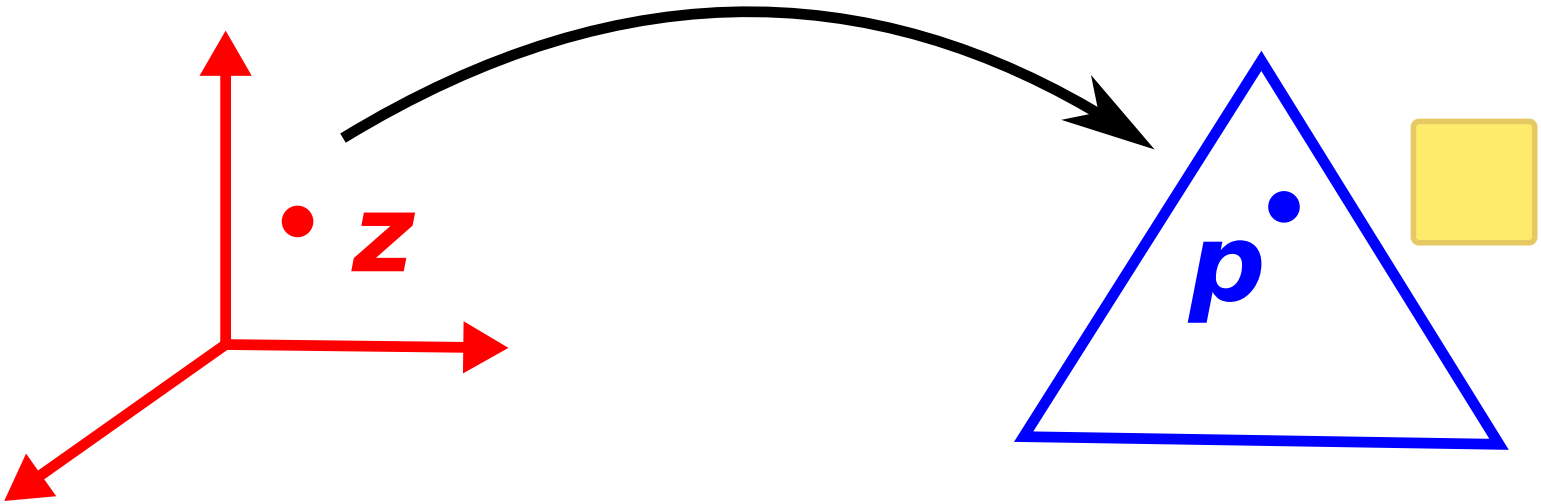


$$\widehat{y} = \arg\max_y \big((\boldsymbol{W}\boldsymbol{\phi}(x))_y \big), \quad \boldsymbol{W} = \begin{bmatrix} w_1^T \\ \vdots \\ w_{|\mathcal{Y}|}^T \end{bmatrix}$$

As before, assume $\phi_0(x) = 1$, absorbing the bias terms.

Key Problem

Mapping from a vector of scores $\mathbb{R}^{|\mathcal{Y}|}$ to a probability distribution over $\mathcal{Y}$?



We will see an important mapping: softmax (next).

Outline

1. Logistic Regression
2. Regularization
3. Non-Linear Classifiers

Logistic Regression

A linear model gives a score for each class y: $w_y^T \phi(x)$,

...from which we may compute a conditional (posterior) probability:

$$P(y|x) = \frac{\exp\left(w_y^T \phi(x)\right)}{Z_x}, \qquad \text{where } Z_x = \sum_{y' \in \mathcal{Y}} \exp\left(w_{y'}^T \phi(x)\right)$$

This is called the softmax transformation

Still a linear classifier, if we choose the most probable class:

$$\begin{aligned} \arg\max_y \; P(y|x) &= \arg\max_y \; \frac{\exp\left(w_y^T \phi(x)\right)}{Z_x} \\ &= \arg\max_y \; \exp\left(w_y^T \phi(x)\right) \\ &= \arg\max_y \; w_y^T \phi(x) \end{aligned}$$

The decision boundaries are still hyperplanes w.r.t. $\phi(x)$.

Binary Logistic Regression

Binary labels ($\mathcal{Y} = \{\pm 1\}$)

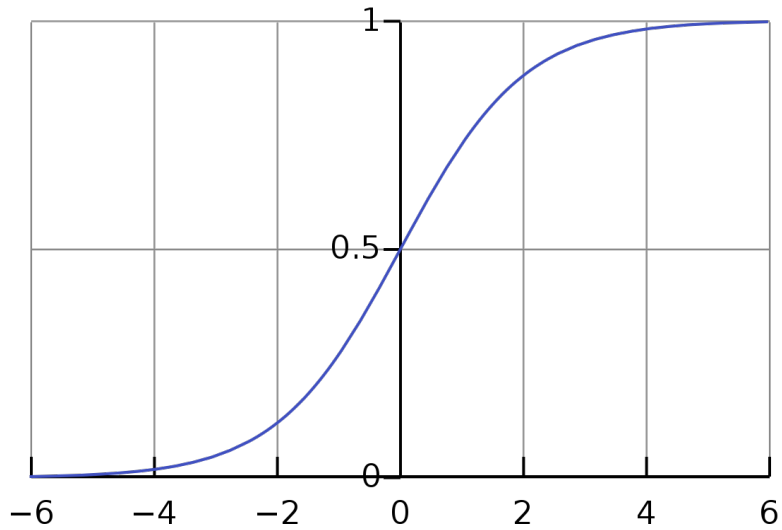
Since adding a constant to all scores does not change the probabilities, we can use score 0 for the -1 class, and $w^T \phi(x)$ for $+1$ class,

$$\begin{aligned} P(y = +1 \mid x) &= \frac{\exp\big(w^T \phi(x)\big)}{1 + \exp\big(w^T \phi(x)\big)} \\ &= \frac{1}{1 + \exp\big(-w^T \phi(x)\big)} \\ &\equiv \sigma\big(w^T \phi(x)\big). \end{aligned}$$

This is called a sigmoid transformation.

Sigmoid Transformation

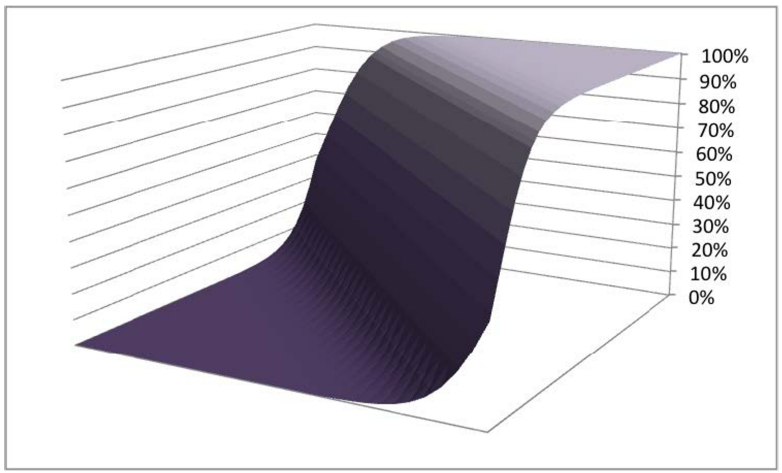
$$\sigma(u) = \frac{e^u}{1 + e^u}$$

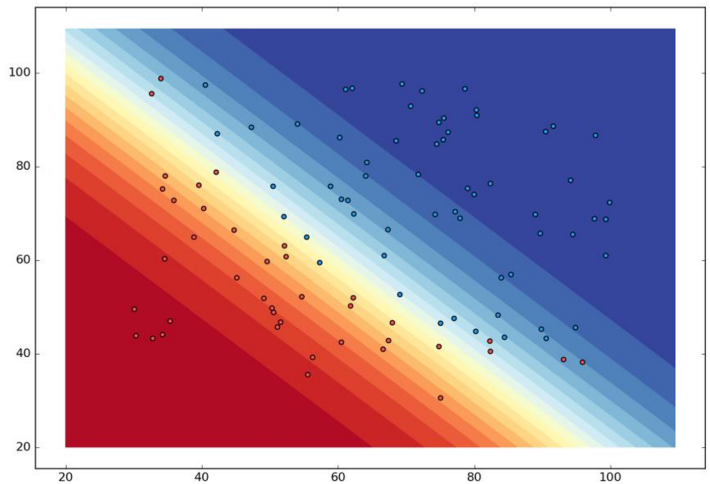


- Widely used in neural networks.
- Maps $\mathbb{R}$ into $[0, 1]$.
- The output can be interpreted as a probability.
- Positive, bounded, strictly increasing.

Binary Logistic Regression

- In two dimensions, i.e., $\boldsymbol{w}$, $\boldsymbol{\phi}(x) \in \mathbb{R}^2$





- MAP boundary, $P(y = +1 \mid x) = 1/2 \Leftrightarrow \boldsymbol{w}^T \boldsymbol{\phi}(x) = 0$, is linear w.r.t. $\boldsymbol{\phi}(x)$.
- Some other threshold, $P(y = +1 \mid x) = \tau \Leftrightarrow \boldsymbol{w}^T \boldsymbol{\phi}(x) = \log(\frac{\tau}{1-\tau})$; linear w.r.t. $\boldsymbol{\phi}(x)$.

Multinomial Logistic Regression

$$P_{\boldsymbol{W}}(y \mid x) = \frac{\exp\left(w_y^T \phi(x)\right)}{Z_x}$$

- How to learn weights $\boldsymbol{W}$?
- Maximize the conditional log-likelihood of training data:

$$\begin{aligned}
\widehat{\boldsymbol{W}} &= \arg\max_{\boldsymbol{W}} \log \left(\prod_{t=1}^{N} P_{\boldsymbol{W}}(y_t|x_t) \right) \\
&= \arg\min_{\boldsymbol{W}} - \sum_{t=1}^{N} \log P_{\boldsymbol{W}}(y_t|x_t) \\
&= \arg\min_{\boldsymbol{W}} \sum_{t=1}^{N} \left(\log \underbrace{\sum_{y'} \exp\left(w_{y'}^T \phi(x_t)\right)}_{Z_{x_t}} - w_{y_t}^T \phi(x_t) \right),
\end{aligned}$$

- i.e., choose $\boldsymbol{W}$ to maximize the probability of the true labels.

Logistic Regression

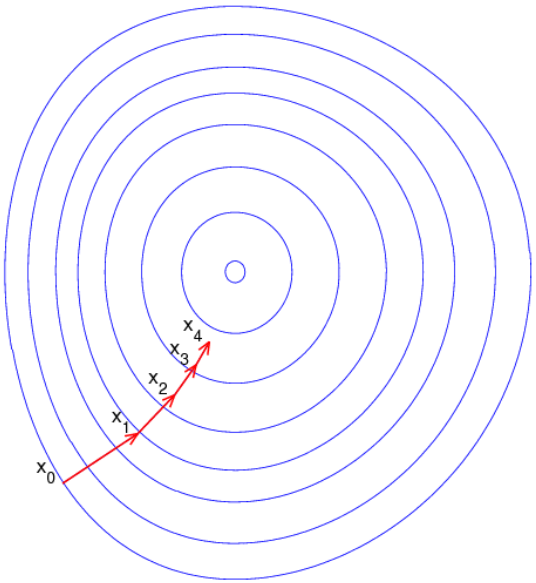
- This objective function is strictly convex.
- Proof left as exercise! (hint, compute the Hessian).
- Therefore, any local minimum is a global minimum.
- No closed-form solution; many numerical methods have been proposed
  - ✓ Gradient methods (gradient descent, conjugate gradient).
  - ✓ Quasi-Newton methods (L-BFGS, ...).

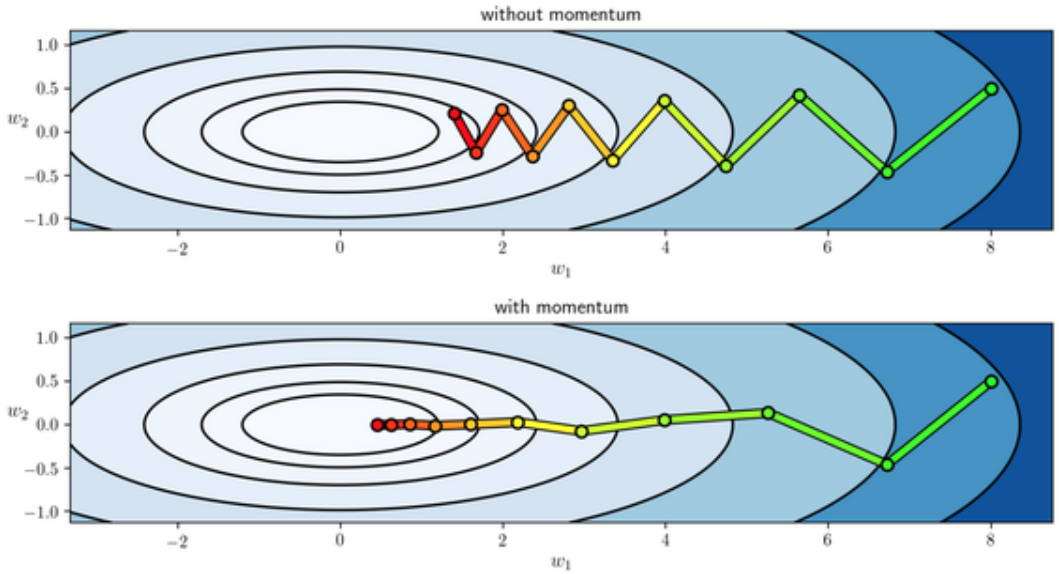
Recap: Gradient Descent

- Goal: minimize $f : \mathbb{R}^d \to \mathbb{R}$, for differentiable f
- Take small steps in the negative gradient direction until a stopping criterion is met:

$$x^{(t+1)} \leftarrow x^{(t)} - \eta_t \nabla f(x^{(t)})$$

- Choosing the step-size: crucial for convergence and performance.
- GD may work well, or not so well. There are many ways to improve it.





Gradient Descent

- Loss function in logistic regression is

$$L(\boldsymbol{W};(x,y)) = \log \sum_{y'} \exp\left(w_{y'}^T \phi(x)\right) - w_y^T \phi(x).$$

- We want to find: $\arg\min_{\boldsymbol{W}} \sum_{t=1}^{N} L(\boldsymbol{W};(x_t,y_t))$
  - ✓ Set $\boldsymbol{W}^{(0)} = 0$
  - ✓ Iterate until convergence (for suitable stepsize η_k):

$$\begin{aligned} \boldsymbol{W}^{(k+1)} &= \boldsymbol{W}^{(k)} - \eta_k \nabla_{\boldsymbol{W}} \left(\sum_{t=1}^{N} L(\boldsymbol{W}^{(k)};(x_t,y_t)) \right) \\ &= \boldsymbol{W}^{(k)} - \eta_k \sum_{t=1}^{N} \nabla_{\boldsymbol{W}} L(\boldsymbol{W}^{(k)};(x_t,y_t)) \end{aligned}$$

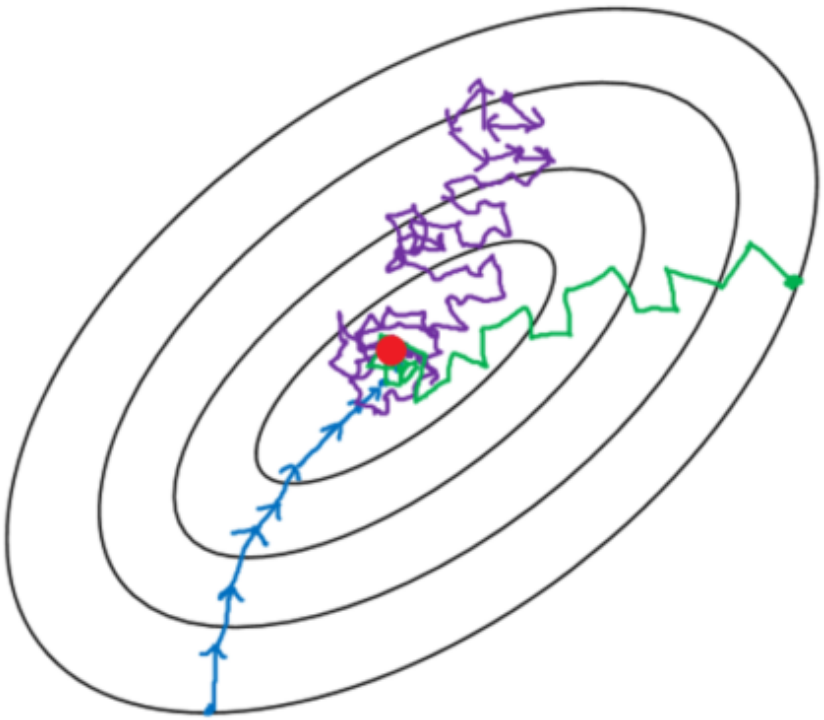
- $L(\boldsymbol{W})$ convex $\Rightarrow$ gradient descent will reach the global optimum $\boldsymbol{W}$, with appropriate choice of step-size η_k.

Stochastic Gradient Descent

Monte Carlo approximation of the gradient (more frequent updates, convenient with large datasets):

- Set $\boldsymbol{W}^{(0)} = 0$
- Iterate until convergence:
  - Pick (x_t, y_t) randomly.
  - Update $\boldsymbol{W}^{(k+1)} = \boldsymbol{W}^{(0)} - \eta_k \nabla_{\boldsymbol{W}} L(\boldsymbol{W}^{(k)}; (x_t, y_t))$
- *i.e.* approximate the gradient with a noisy, unbiased, version based on a single sample
- Variant between GD and SGD: mini-batch (pick B, rather than 1 pair)
- All guaranteed to find the optimal $\boldsymbol{W}$ (for suitable step sizes)

Stochastic vs Batch Gradient Descent



Batch gradient descent
Mini-batch gradient Descent
Stochastic gradient descent

Computing the Gradient

- We need to compute $\nabla_{\boldsymbol{W}} L(\boldsymbol{W}; (x_t, y_t))$, where

$$L(\boldsymbol{W}; (x, y)) = \log \sum_{y'} \exp\left(w_{y'}^T \phi(x)\right) - w_y^T \phi(x)$$

- Some reminders (we next omit the subscript $\boldsymbol{W}$ for simplicity)
  1. $\nabla \log F(\boldsymbol{W}) = \dfrac{\nabla F(\boldsymbol{W})}{F(\boldsymbol{W})}$
  2. $\nabla \exp F(\boldsymbol{W}) = \exp(F(\boldsymbol{W}))\nabla F(\boldsymbol{W})$

- We denote by

$$e_y = [0, \ldots, 0, 1, 0, \ldots, 0]^T, \quad \text{1 in } y\text{-th position}$$

the one-hot vector representation of class y.

Computing the Gradient

$$
\begin{aligned}
\nabla L(\boldsymbol{W};(x,y)) &= \nabla\left(\log\sum_{y'}\exp(w_{y'}^T\phi(x)) - w_y^T\phi(x)\right) \\
&= \nabla\log\sum_{y'}\exp(w_{y'}^T\phi(x)) - \nabla w_y^T\phi(x) \\
&= \frac{1}{\sum_{y'}\exp(w_{y'}^T\phi(x))}\sum_{y'}\nabla\exp(w_{y'}^T\phi(x)) - e_y\phi(x)^T \\
&= \frac{1}{Z_x}\sum_{y'}\exp(w_{y'}^T\phi(x))\nabla w_{y'}^T\phi(x) - e_y\phi(x)^T \\
&= \sum_{y'}\frac{\exp(w_{y'}^T\phi(x))}{Z_x}e_{y'}\phi(x)^T - e_y\phi(x)^T \\
&= \sum_{y'}P_{\boldsymbol{W}}(y'|x)e_{y'}\phi(x)^T - e_y\phi(x)^T \\
&= \left(\begin{bmatrix} P_{\boldsymbol{W}}(1|x) \\ \vdots \\ P_{\boldsymbol{W}}(|\mathcal{Y}|\,|\,x)\end{bmatrix} - e_y\right)\phi(x)^T.
\end{aligned}
$$

Logistic Regression Summary

- Define conditional probability

$$P_{\boldsymbol{W}}(y|x) = \frac{\exp\left(w_y^T \phi(x)\right)}{Z_x}$$

- Set weights to maximize conditional log-likelihood of training data:

$$\boldsymbol{W} = \arg\max_{\boldsymbol{W}} \sum_t \log P_{\boldsymbol{W}}(y_t|x_t) = \arg\min_{\boldsymbol{W}} \sum_t L(\boldsymbol{W}; (x_t, y_t))$$

- Run gradient descent, or another gradient-based algorithm, using

$$\nabla L(\boldsymbol{W}; (x, y)) = \sum_{y'} P_{\boldsymbol{W}}(y'|x) \boldsymbol{e}_{y'} \phi(x)^\top - \boldsymbol{e}_y \phi(x)^\top$$

The Story So Far

The Story So Far

- Logistic regression is discriminative: maximizes conditional likelihood
  - ✓ also called log-linear model and max-entropy classifier
  - ✓ no closed-form solution
  - ✓ stochastic gradient updates look like

$$\boldsymbol{W}^{(k+1)} = \boldsymbol{W}^{(k)} + \eta_k \left(\boldsymbol{e}_y \boldsymbol{\phi}(x)^\top - \sum_{y'} P_{\boldsymbol{W}}(y'|x) \boldsymbol{e}_{y'} \boldsymbol{\phi}(x)^\top \right)$$

- The perceptron is also discriminative, but non-probabilistic classifier
  - ✓ perceptron's updates look like

$$\boldsymbol{W}^{(k+1)} = \boldsymbol{W}^{(k)} + \boldsymbol{e}_y \boldsymbol{\phi}(x)^\top - \boldsymbol{e}_{\hat{y}} \boldsymbol{\phi}(x)^\top$$

- SGD updates for logistic regression and the perceptron look similar!

Other Options: Maximizing Margin

- For a training set $\mathcal{D} = \{(x_t, y_t),\ t = 1, ..., N\}$

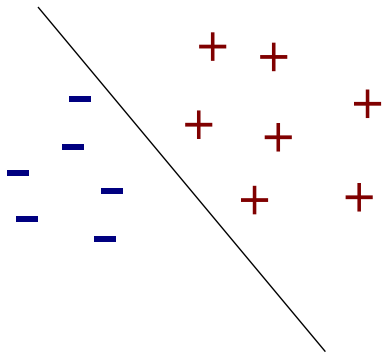
- Margin achieved by a weight matrix $\boldsymbol{W}$: smallest γ such that, for every training instance $(x_t, y_t) \in \mathcal{D}$,

$$w_{y_t}^T \phi(x_t) \geq w_{y'}^T \phi(x_t) + \gamma, \quad \text{for any } \ y' \neq y_t$$

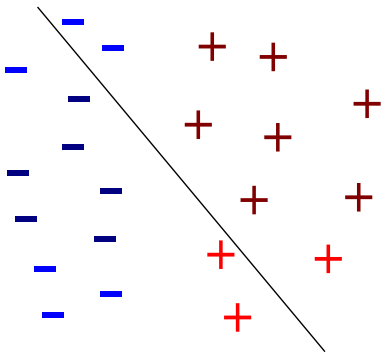
- i.e., score of the correct label is at least γ higher than that of any other label.

Margin

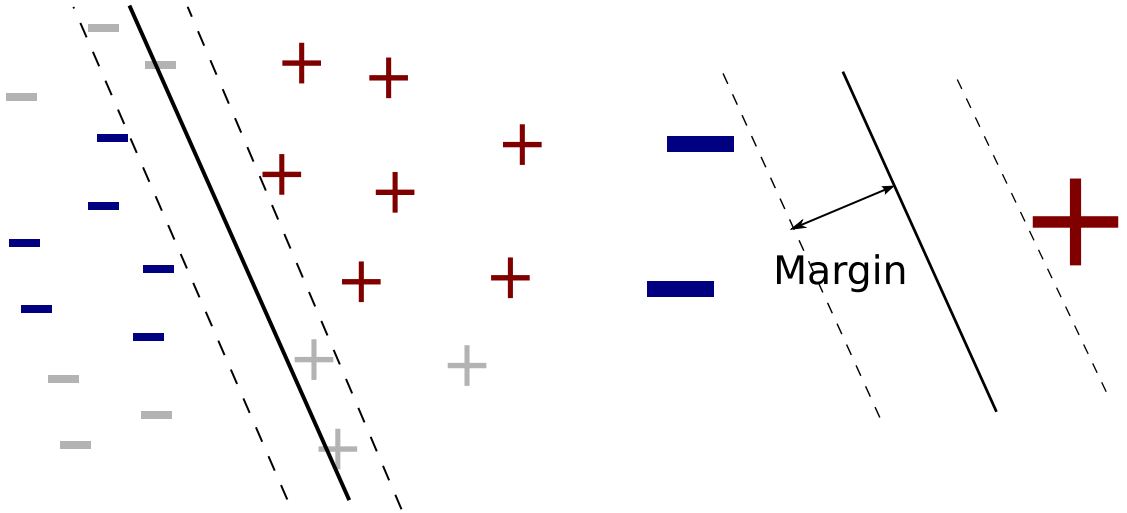
Training



Testing



Denote the value of the margin by γ



Maximizing Margin

- Intuitively maximizing margin makes sense
- More importantly, generalization error to unseen test data is proportional to the inverse of the margin

$$\epsilon \propto \frac{R^2}{\gamma^2 N}$$

- **Perceptron**:
  - ✓ If a training set is separable by some margin, the perceptron will find a $\boldsymbol{W}$ that separates the data
  - ✓ However, the perceptron does not pick $\boldsymbol{W}$ to maximize the margin!
  - ✓ Support vector machines (SVM) do exactly this (not covered)

Summary

What we saw:

- Linear Classifiers
  - ✓ Logistic regression
  - ✓ Perceptron
  - ✓ Support vector machines (not covered)

Next:

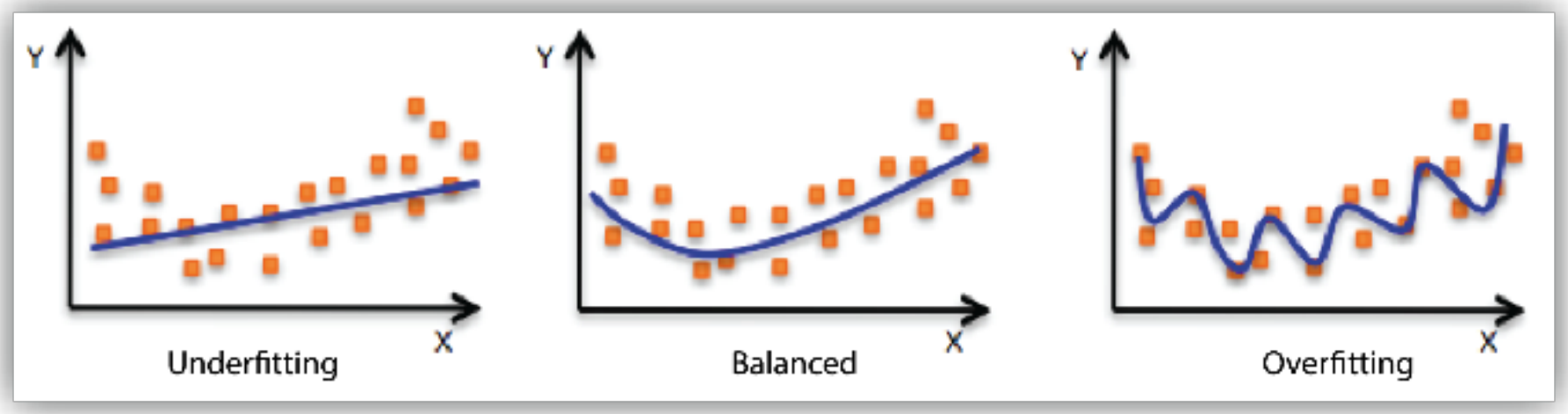
- Regularization
- Softmax
- Non-linear classifiers

Outline

1. Logistic Regression
2. Regularization
3. Non-Linear Classifiers

Overfitting

If the model is too complex (too many parameters) and the data is scarce, we run the risk of overfitting:



Regularization

In practice, we regularize to prevent overfitting

$$\arg\min_{\boldsymbol{W}} \sum_{t=1}^{N} L(\boldsymbol{W}; (x_t, y_t)) + \lambda \Omega(\boldsymbol{W}),$$

$\Omega(\boldsymbol{W})$ is the regularization function, and λ controls its weight.

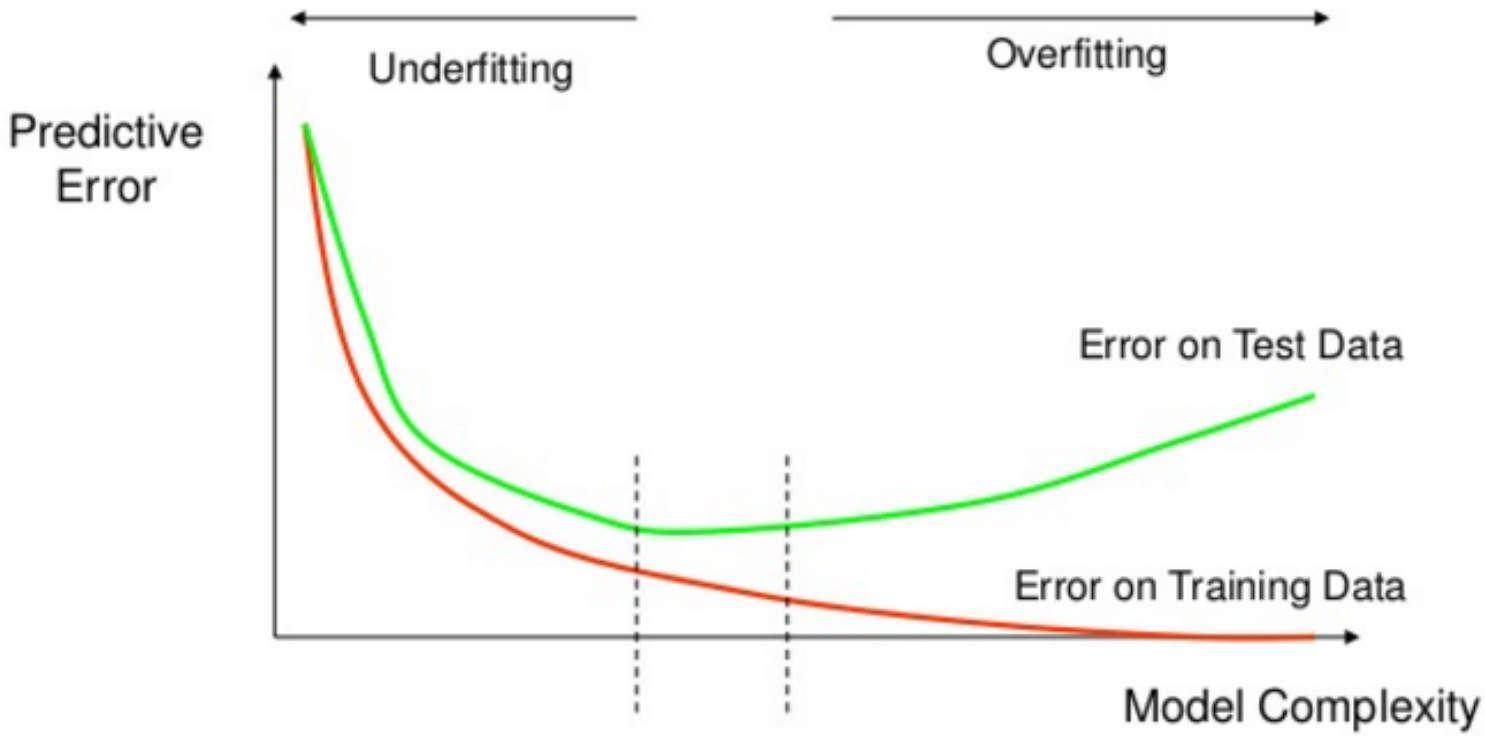
- ℓ_2 regularization promotes smaller weights:

$$\Omega(\boldsymbol{W}) = \sum_y \|w_y\|_2^2 = \sum_y \sum_i w_{y,i}^2.$$

- ℓ_1 regularization promotes smaller and sparse weights!

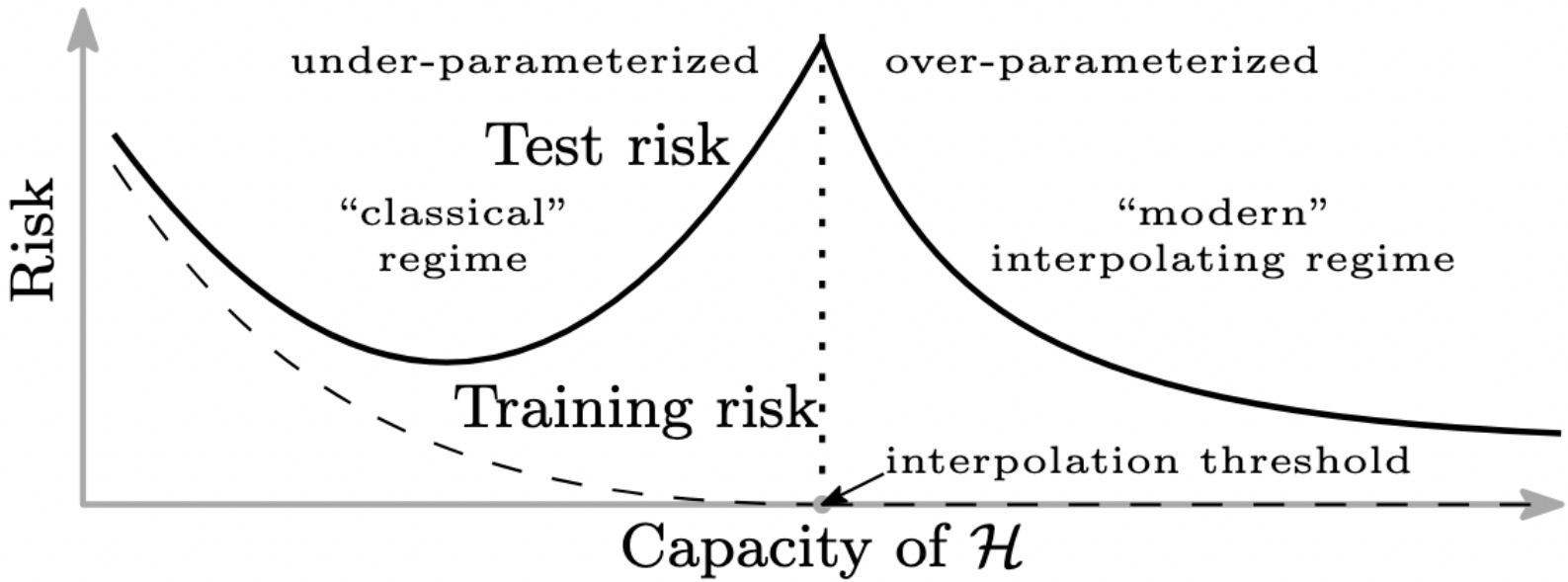
$$\Omega(\boldsymbol{W}) = \sum_y \|w_y\|_1 = \sum_y \sum_i |w_{y,i}|$$

Empirical Risk Minimization: The Classical View



...doesn't explain the excellent performance of deep networks, which are very complex (have many parameters).

Empirical Risk Minimization: Double Descent



Fully understanding double descent is an open research topic.

Logistic Regression with ℓ_2 Regularization

- ℓ_2-regularized (a.k.a. ridge or weight decay) logistic regression

$$\min -\sum_{t=1}^{N} \underbrace{\log\left(\exp(w_{y_t}^T \phi(x_t))/Z_x\right)}_{L(\boldsymbol{W};(x_t,y_t))} + \frac{\lambda}{2}\|\boldsymbol{W}\|_F^2$$

- The gradient is now

$$\sum_{t=1}^{N} \nabla L(\boldsymbol{W};(x_t,y_t)) + \frac{\lambda}{2}\nabla\|\boldsymbol{W}\|_2^2$$

- For gradient descent, we also need $\nabla\|\boldsymbol{W}\|_2^2 = 2\boldsymbol{W}$
- GD step (similarly for SGD) with weight decay:

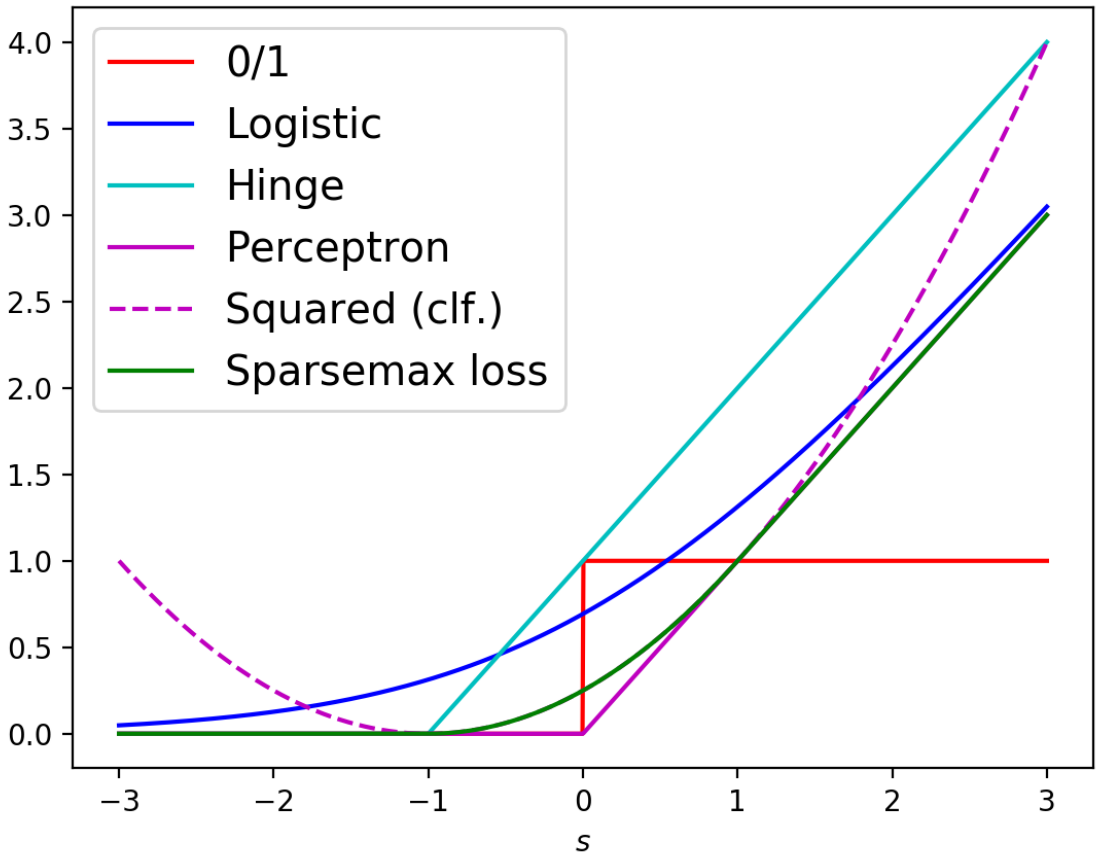
$$\boldsymbol{W}^{(k+1)} = \underbrace{(1-\eta_k\lambda)}_{\substack{\text{weight decay} \\ (1-\eta_k\lambda)<1}} \boldsymbol{W}^{(k)} - \eta_k \sum_{t=1}^{N} \nabla L(\boldsymbol{W};(x_t,y_t))$$

Loss Functions

- Should match the metric we want to optimize at test time
- Should be well-behaved (convex, hopefully smooth) for optimization (rules out the 0/1 loss)
- Some examples:
  - ✓ Squared loss for regression
  - ✓ Negative log-likelihood (a.k.a. cross-entropy) for multinomial logistic regression
  - ✓ Hinge lossfor support vector machines (not covered)
  - ✓ Sparsemax loss for multi-class and multi-label classification (not covered)

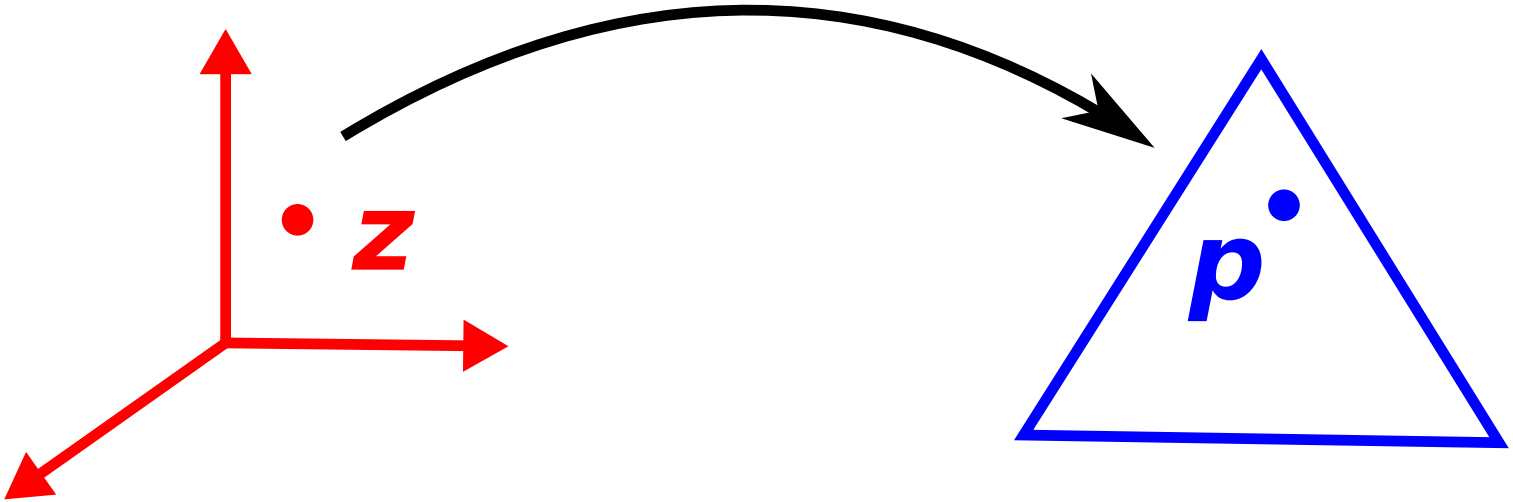
Classification Losses (Binary Case)

- The horizontal axis shows $s = -y\ w^T \phi(x)$.



Recap

Mapping a label score vector $z \in \mathbb{R}^{|\mathcal{Y}|}$ to a probability distribution in $\mathcal{Y}$?



We already saw one example: softmax.

Another example is sparsemax (not covered): Martins and Astudillo (2016)

Recap: Softmax Transformation

The typical transformation for multi-class classification:

$$\text{softmax} : \mathbb{R}^{|\mathcal{Y}|} \to \Delta_{|\mathcal{Y}|-1}$$

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_c \exp(z_c)}, \ldots, \frac{\exp(z_{|\mathcal{Y}|})}{\sum_c \exp(z_c)} \right]^T$$

- Underlies multinomial logistic regression
- Strictly positive, sums to 1
- Resulting distribution has full support: $\text{softmax}(z) > 0, \forall \boldsymbol{z}$

Recap: Multinomial Logistic Regression

- The common choice for a softmax output layer
- The classifier estimates $P(y \mid x; \boldsymbol{W})$
- We minimize the negative log-likelihood:

$$\begin{aligned} L(\boldsymbol{W}; (x, y)) &= -\log P(y \mid x; \boldsymbol{W}) \\ &= -\log \left(\text{softmax}(z(x))\right)_y, \end{aligned}$$

  where $z_c(x) = w_c^T \phi(x)$ is the score of class c for sample x.

- Loss gradient:

$$\nabla L(\boldsymbol{W}; (x, y)) = \left(\text{softmax}(z(x)) - e_y\right) \phi(x)^T$$

Outline

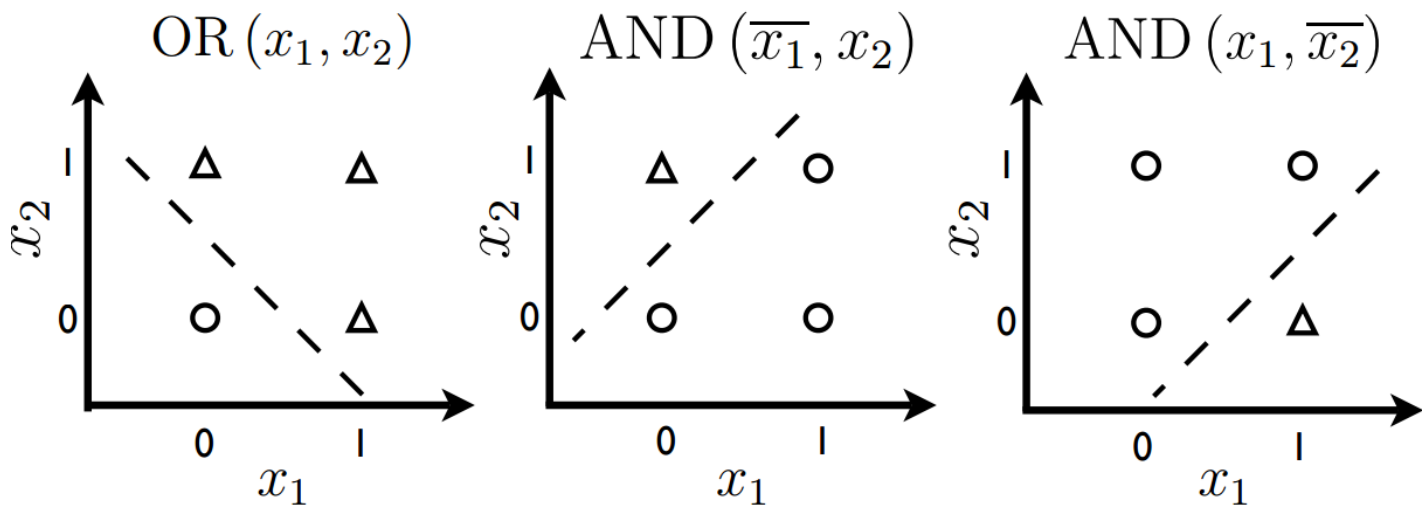
1 Logistic Regression

2 Regularization

3 Non-Linear Classifiers

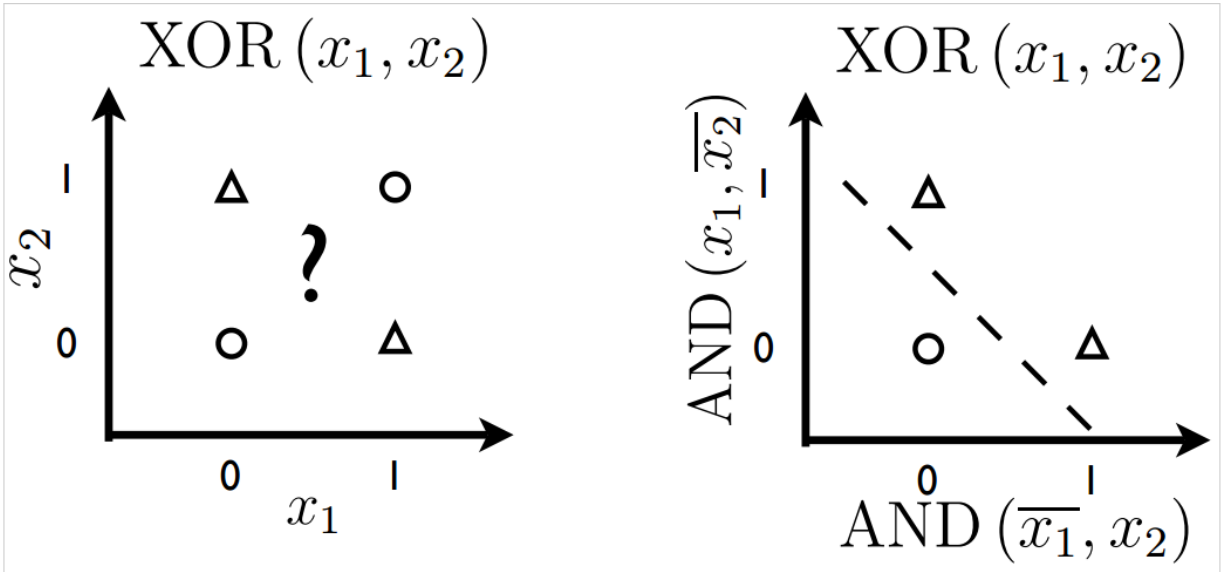
Recap: What a Linear Classifier Can Do

- It **can** solve linearly separable problems (OR, AND)



Recap: What a Linear Classifier Can't Do

- ... but it **cannot** solve non-linearly separable problems such as simple XOR (unless input is transformed into a better representation):



- This was observed by Minsky and Papert (1969) (for the perceptron) and motivated strong criticisms

Summary: Linear Classifiers

- We have seen
  - ✓ Perceptron
  - ✓ Logistic regression
  - ✓ Support vector machines (not covered)

Summary: Linear Classifiers

- We have seen
  - ✓ Perceptron
  - ✓ Logistic regression
  - ✓ Support vector machines (not covered)
- All lead to convex optimization problems: no issues with local minima.

Summary: Linear Classifiers

- We have seen
  - ✓ Perceptron
  - ✓ Logistic regression
  - ✓ Support vector machines (not covered)
- All lead to convex optimization problems: no issues with local minima.
- All assume the features are well-engineered such that the data is (nearly) linearly separable

What If Data Are Not Linearly Separable?

What If Data Are Not Linearly Separable?

Engineer better features (often works!)

What If Data Are Not Linearly Separable?

Engineer better features (often works!)

Kernel methods:

- works implicitly in a high-dimensional feature space
- ... but still need to choose/design a good kernel
- model capacity confined to positive-definite kernels

What If Data Are Not Linearly Separable?

Engineer better features (often works!)

Kernel methods:

- works implicitly in a high-dimensional feature space
- ... but still need to choose/design a good kernel
- model capacity confined to positive-definite kernels

Neural networks (next class!)

- embrace non-convexity and local minima
- instead of engineering features/kernels, engineer the model architecture

Two Views of Machine Learning

There's two main ways of building machine learning systems:

1. Feature-based: describe object properties (features) and build models that manipulate them.
   - ✓ What that we have seen so far.

Two Views of Machine Learning

There's two main ways of building machine learning systems:

1. Feature-based: describe object properties (features) and build models that manipulate them.
   - ✓ What that we have seen so far.

2. Similarity-based: don't describe objects by their properties; rather, build systems based on **comparing** objects to each other
   - ✓ k-th nearest neighbors; kernel methods; Gaussian processes.

Two Views of Machine Learning

There's two main ways of building machine learning systems:

1. Feature-based: describe object properties (features) and build models that manipulate them.
   - ✓ What that we have seen so far.

2. Similarity-based: don't describe objects by their properties; rather, build systems based on **comparing** objects to each other
   - ✓ k-th nearest neighbors; kernel methods; Gaussian processes.

Sometimes the two are equivalent!

Nearest Neighbor(s) (NN) Classifier

- Not a linear classifier!
- In its simplest version, doesn't require any parameters

Nearest Neighbor(s) (NN) Classifier

- Not a linear classifier!
- In its simplest version, doesn't require any parameters
- No training, just memorize the data $\mathcal{D} = \{(x_i, y_i),\ _i = 1, ..., N\}$

Nearest Neighbor(s) (NN) Classifier

- Not a linear classifier!
- In its simplest version, doesn't require any parameters
- No training, just memorize the data $\mathcal{D} = \{(x_i, y_i),\ _i = 1, ..., N\}$
- Given a new input x, find its nearest data point x_i and predict

$$\widehat{y}(x) = y_n, \quad \text{where } n = \arg\min_i \|x - x_i\|$$

Nearest Neighbor(s) (NN) Classifier

- Not a linear classifier!
- In its simplest version, doesn't require any parameters
- No training, just memorize the data $\mathcal{D} = \{(x_i, y_i),\ _i = 1, ..., N\}$
- Given a new input x, find its nearest data point x_i and predict

$$\widehat{y}(x) = y_n, \quad \text{where } n = \arg\min_i \|x - x_i\|$$

- Variants: k-NN returns the majority class in the k nearest neighbours.

Nearest Neighbor(s) (NN) Classifier

- Not a linear classifier!
- In its simplest version, doesn't require any parameters
- No training, just memorize the data $\mathcal{D} = \{(x_i, y_i),\ _i = 1, ..., N\}$
- Given a new input x, find its nearest data point x_i and predict

$$\widehat{y}(x) = y_n, \quad \text{where } n = \arg\min_i \|x - x_i\|$$

- Variants: k-NN returns the majority class in the k nearest neighbours.
- Disadvantage: requires searching over the entire training data

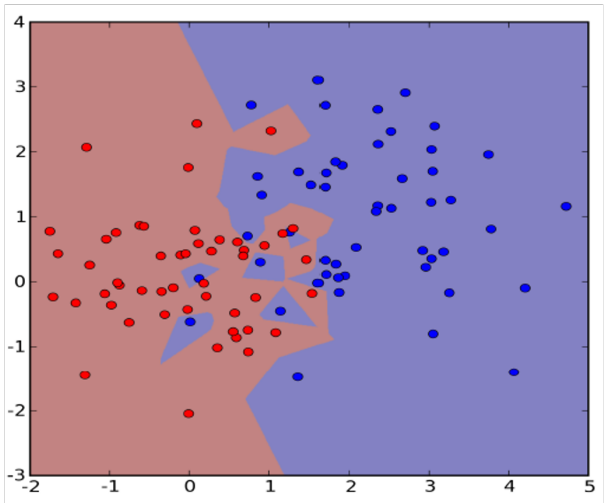
Nearest Neighbor(s) (NN) Classifier

- Not a linear classifier!
- In its simplest version, doesn't require any parameters
- No training, just memorize the data $\mathcal{D} = \{(x_i, y_i),\ _i = 1, ..., N\}$
- Given a new input x, find its nearest data point x_i and predict

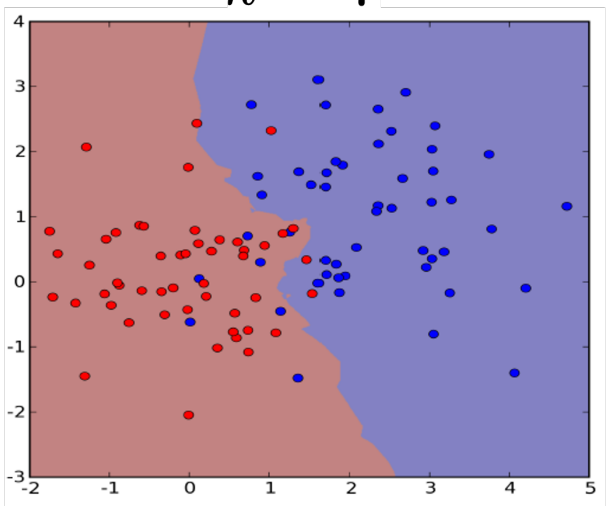
$$\widehat{y}(x) = y_n, \quad \text{where } n = \arg\min_i \|x - x_i\|$$

- Variants: k-NN returns the majority class in the k nearest neighbours.
- Disadvantage: requires searching over the entire training data
- Specialized data structures can be used to speed up search.

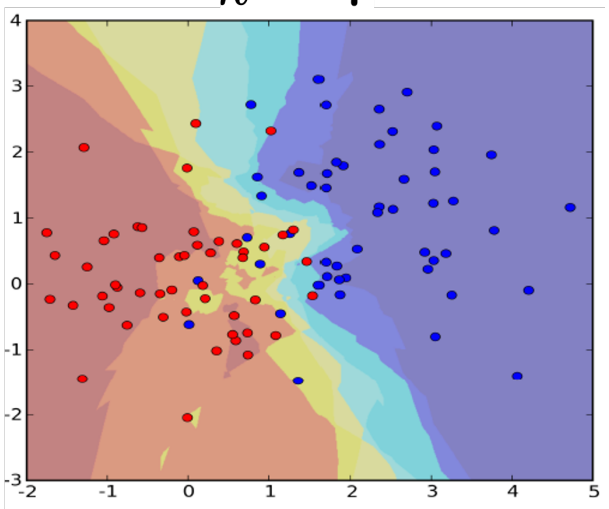
Nearest Neighbour(s) Classifiers



$k = 7$



$k = 7$



class probability estimates

Kernels

- A kernel is a similarity function between two points that is **symmetric** and **positive semi-definite**, denote by:

$$\kappa(x_i, x_j) \in \mathbb{R}$$

Kernels

- A kernel is a similarity function between two points that is **symmetric** and **positive semi-definite**, denote by:

$$\kappa(x_i, x_j) \in \mathbb{R}$$

- Given dataset $\mathcal{D} = \{(x_i, y_i),\ i = 1, ..., N\}$, the Gram matrix $\boldsymbol{K}$ is the $N \times N$ matrix defined as:

$$K_{i,j} = \kappa(x_i, x_j)$$

Kernels

- A kernel is a similarity function between two points that is **symmetric** and **positive semi-definite**, denote by:

$$\kappa(x_i, x_j) \in \mathbb{R}$$

- Given dataset $\mathcal{D} = \{(x_i, y_i),\ i = 1, ..., N\}$, the Gram matrix $\boldsymbol{K}$ is the $N \times N$ matrix defined as:

$$K_{i,j} = \kappa(x_i, x_j)$$

- **Symmetric:**

$$\kappa(x_i, x_j) = \kappa(x_j, x_i)$$

Kernels

- A kernel is a similarity function between two points that is **symmetric** and **positive semi-definite**, denote by:

$$\kappa(x_i, x_j) \in \mathbb{R}$$

- Given dataset $\mathcal{D} = \{(x_i, y_i),\ i = 1, ..., N\}$, the Gram matrix $\boldsymbol{K}$ is the $N \times N$ matrix defined as:

$$K_{i,j} = \kappa(x_i, x_j)$$

- **Symmetric:**

$$\kappa(x_i, x_j) = \kappa(x_j, x_i)$$

- **Positive definite:** for all non-zero $\mathbf{v}$

$$\mathbf{v}\boldsymbol{K}\mathbf{v}^T \geq 0$$

Kernels

- **Mercer's Theorem**: for any kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there exists some feature mapping $\phi : \mathcal{X} \to \mathcal{H}$, s.t.:

$$\kappa(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_{\mathcal{H}}$$

Kernels

- **Mercer's Theorem**: for any kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there exists some feature mapping $\phi : \mathcal{X} \to \mathcal{H}$, s.t.:

$$\kappa(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_{\mathcal{H}}$$

- A kernel corresponds to a mapping into an **implicit** feature space.

Kernels

- **Mercer's Theorem**: for any kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there exists some feature mapping $\phi : \mathcal{X} \to \mathcal{H}$, s.t.:

$$\kappa(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_{\mathcal{H}}$$

- A kernel corresponds to a mapping into an **implicit** feature space.
- The kernel is equivalent to an inner product in that feature space.

Kernels

- **Mercer's Theorem**: for any kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there exists some feature mapping $\phi : \mathcal{X} \to \mathcal{H}$, s.t.:

$$\kappa(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_{\mathcal{H}}$$

- A kernel corresponds to a mapping into an **implicit** feature space.
- The kernel is equivalent to an inner product in that feature space.
- Kernel trick: take a feature-based algorithm (SVM, perceptron, logistic regression) and replace all explicit feature computations by **kernel evaluations**:

$$w_y^T \phi(x) = \sum_{i=1}^{N} \sum_{y \in \mathcal{Y}} \alpha_{i,y} \kappa(x, x_i) \quad \text{for some } \alpha_{i,y} \in \mathbb{R}$$

Kernels

- **Mercer's Theorem**: for any kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there exists some feature mapping $\phi : \mathcal{X} \to \mathcal{H}$, s.t.:

$$\kappa(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_{\mathcal{H}}$$

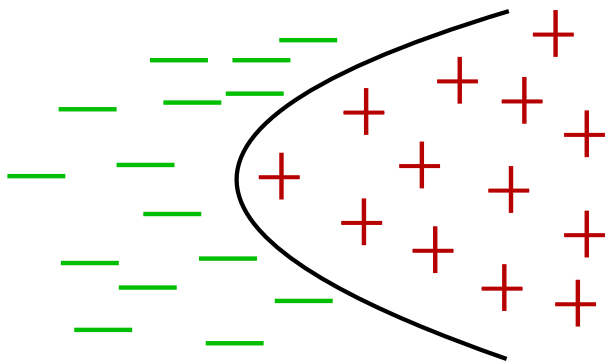
- A kernel corresponds to a mapping into an **implicit** feature space.
- The kernel is equivalent to an inner product in that feature space.
- Kernel trick: take a feature-based algorithm (SVM, perceptron, logistic regression) and replace all explicit feature computations by **kernel evaluations**:

$$w_y^T \phi(x) = \sum_{i=1}^{N} \sum_{y \in \mathcal{Y}} \alpha_{i,y} \kappa(x, x_i) \quad \text{for some } \alpha_{i,y} \in \mathbb{R}$$

- Extremely popular idea in the 1990-2000s!

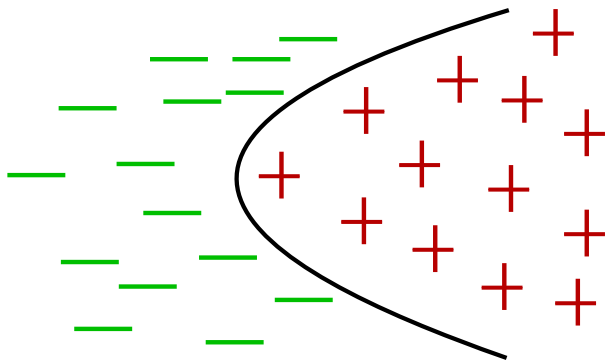
Kernels = Tractable Non-Linearity

- A linear classifier in a higher dimensional feature space is a non-linear classifier in the original space



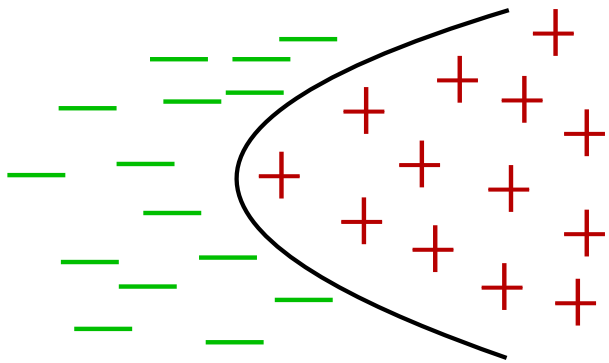
Kernels = Tractable Non-Linearity

- A linear classifier in a higher dimensional feature space is a non-linear classifier in the original space
- Computing a non-linear kernel is often cheaper than calculating the corresponding inner product in the high-dimensional feature space.



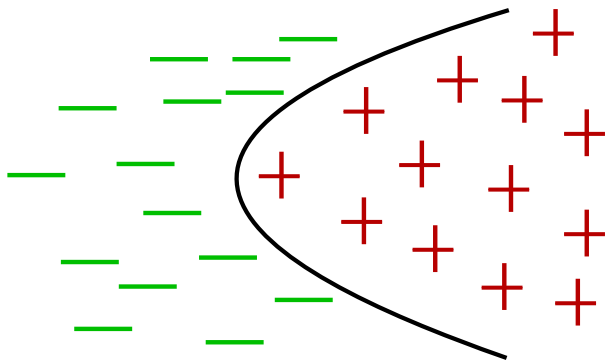
Kernels = Tractable Non-Linearity

- A linear classifier in a higher dimensional feature space is a non-linear classifier in the original space
- Computing a non-linear kernel is often cheaper than calculating the corresponding inner product in the high-dimensional feature space.
- Many models can be "kernelized".

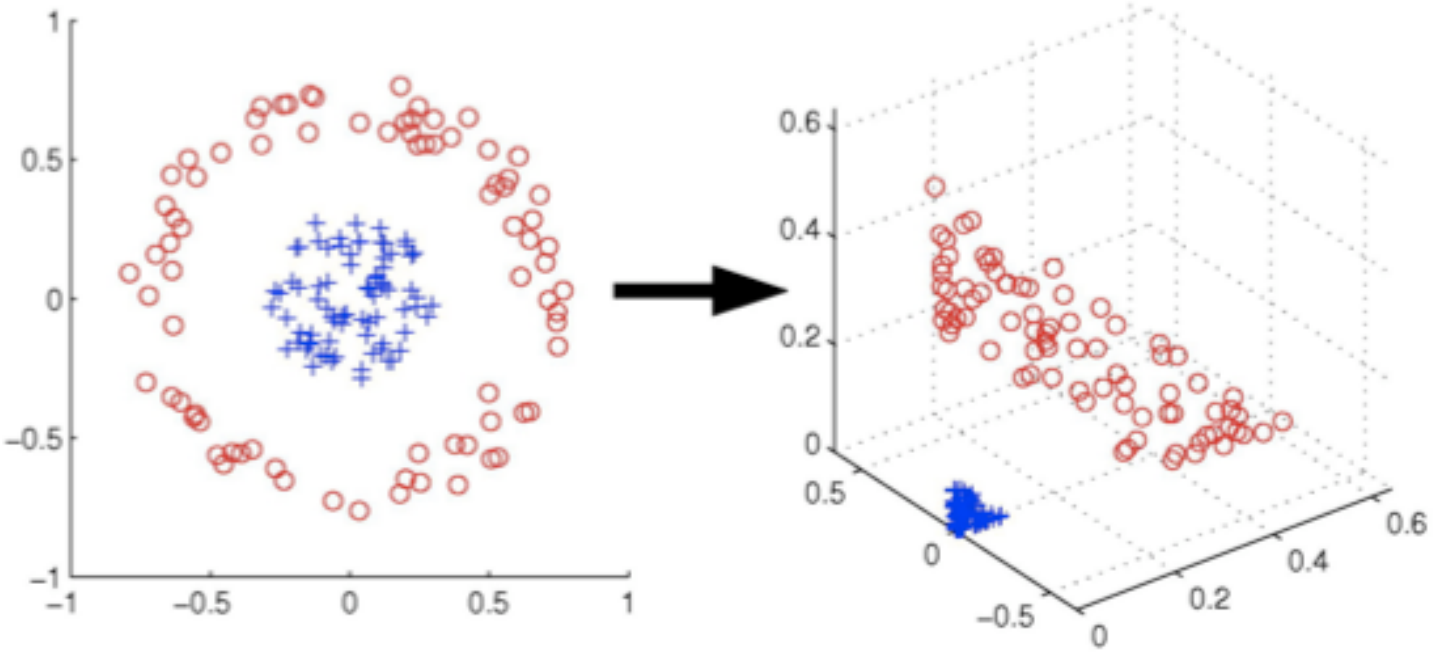


Kernels = Tractable Non-Linearity

- A linear classifier in a higher dimensional feature space is a non-linear classifier in the original space
- Computing a non-linear kernel is often cheaper than calculating the corresponding inner product in the high-dimensional feature space.
- Many models can be "kernelized".
- Drawback: quadratic dependency on dataset size



Linear Classifiers in High Dimension



- Example: $\phi([x_1, x_2]) = [x_1^2, x_2^2, \sqrt{2}\, x_1\, x_2]$
- Kernel trick; let $x = [x_1, x_2]$ and $x' = [x_1', x_2']$;

$$\langle \phi(x), \phi(x') \rangle = \langle [x_1, x_2], [x_1', x_2'] \rangle^2 = k(x, x')$$

Popular Kernels

- Polynomial kernel of order d:

$$\kappa(x, x') = \langle \phi(x), \phi(x') \rangle + 1)^d$$

Popular Kernels

- Polynomial kernel of order d:

$$\kappa(x, x') = \langle \phi(x), \phi(x') \rangle + 1)^d$$

- Gaussian radial basis kernel

$$\kappa(x, x') = exp(\frac{-\|\phi(x) - \phi(x')\|^2}{2\sigma^2})$$

Popular Kernels

- Polynomial kernel of order d:

$$\kappa(x, x') = \langle \phi(x), \phi(x') \rangle + 1)^d$$

- Gaussian radial basis kernel

$$\kappa(x, x') = exp(\frac{-\|\phi(x) - \phi(x')\|^2}{2\sigma^2})$$

- String kernels (Lodhi et al., 2002; Collins and Duffy, 2002)

Popular Kernels

- Polynomial kernel of order d:

$$\kappa(x, x') = \langle \phi(x), \phi(x') \rangle + 1)^d$$

- Gaussian radial basis kernel

$$\kappa(x, x') = exp(\frac{-\|\phi(x) - \phi(x')\|^2}{2\sigma^2})$$

- String kernels (Lodhi et al., 2002; Collins and Duffy, 2002)

- Tree kernels (Collins and Duffy, 2002)

Conclusions

- Linear classifiers are a broad class including well-known methods such as perceptrons, logistic regression, support vector machines.
- They all involve manipulating weights and features.
- They either lead to closed-form solutions or convex optimization problems (no local minima)
- Stochastic gradient descent algorithms are useful if training datasets are large.
- However, they require manual specification of feature representations

References I

Collins, M. and Duffy, N. (2002). Convolution kernels for natural language. *Advances in Neural Information Processing Systems*, 1:625–632.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., and Watkins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444.

Martins, A. F. T. and Astudillo, R. (2016). From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification. In *Proc. of the International Conference on Machine Learning*.

Minsky, M. and Papert, S. (1969). Perceptrons.