Machine Learning for Robotics
Intelligent Systems Series
Lecture 3

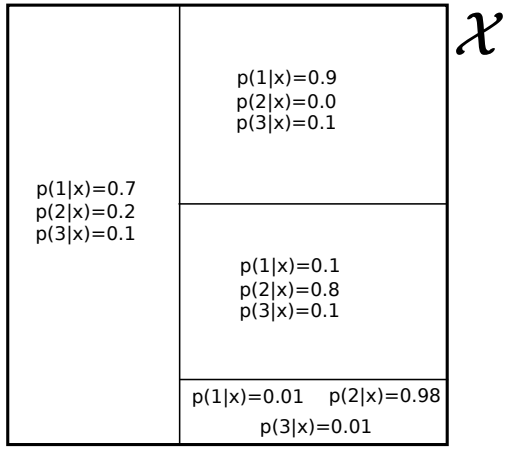
Georg Martius
Slides adapted from Christoph Lampert, IST Austria

MPI for Intelligent Systems, Tübingen, Germany

May 8, 2017

Nonparametric Discriminative Model

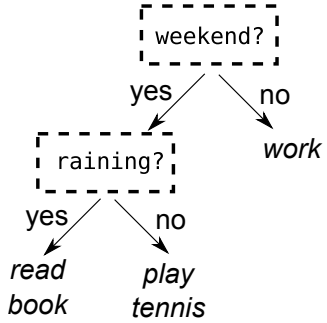
Idea: split $\mathcal{X}$ into regions, for each region store an estimate $\hat{p}(y|x)$.



Use a *decision tree*. (introduced next)

Decision Trees – a short intro (analysis: Breiman 1980s)

Task: decide what to do today



Classifier has a tree structure:

- each *interior node* makes a decision: it picks an attribute within x, branches for each possible value
- each *leaf* has one output label
- to classify a new example, we
  - put it into the root node,
  - follow the decisions until we reach a leaf.
  - use the leaf value as the prediction

Decisions trees ('expert systems') are popular especially for non-experts:

- **easy to use**, and **interpretable.**

How to automatically build a decision tree

Given: training set $\mathcal{D} = \{(x^1, y^1), \ldots, (x^n, y^n)\}$.

Convention:

- each node contains a subset of examples,
- its label is the majority label of the examples in this node (any of the majority labels, if there's a tie)

Decision Tree – Training

initialize: put all examples in root node
mark root as *active*
repeat
  pick active node with largest number of misclassified examples
  mark the node as *inactive*
  for each attributes, check error rate of splitting along this attribute
  keep the split with smallest error, if any, and mark children as *active*
until no more active nodes.

How to automatically build a decision tree

Decision Tree – Classification

input decision tree, example x
  assign x to root node
  while x not in leaf node **do**
    move x to child according to the test in node
  end while
output label of the leaf that x is in

Decision Trees Example - Training phase

Training data: Zoe (our customer) selected whom she would like to date from a list of profiles

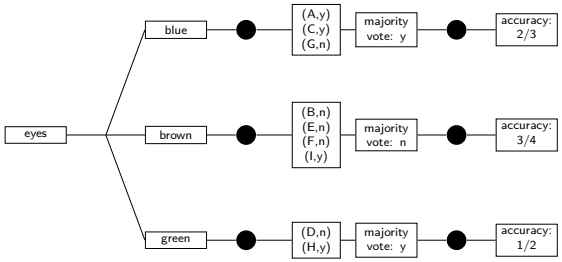
	$\mathcal{X}$					$\mathcal{Y}$
person	eyes	handsome	height	sex	soccer	date?
Apu	blue	yes	tall	M	no	yes
Bernice	brown	yes	short	F	no	no
Carl	blue	no	tall	M	no	yes
Doris	green	yes	short	F	no	no
Edna	brown	no	short	F	yes	no
Prof. Frink	brown	yes	tall	M	yes	no
Gil	blue	no	tall	M	yes	no
Homer	green	yes	short	M	no	yes
Itchy	brown	no	short	M	yes	yes

Decision Trees Example - Training phase

Step 1: put all all training examples into the root node

$$root = \{\ (A,y),(B,n),(C,y),(D,n),(E,n),(F,n),(G,n),(H,y),(I,y)\ \}$$

For each feature, check the classification accuracy of this single feature:



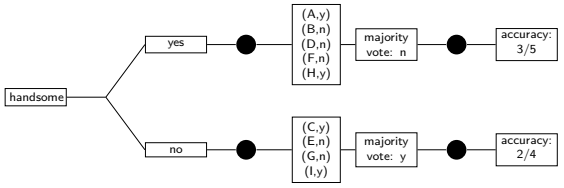
Total accuracy eyes: 6/9

Decision Trees Example - Training phase

Step 1: put all all training examples into the root node

$$root = \{\ (A,y),(B,n),(C,y),(D,n),(E,n),(F,n),(G,n),(H,y),(I,y)\ \}$$

For each feature, check the classification accuracy of this single feature:



Total accuracy handsome: 5/9

Decision Trees Example - Training phase

Step 1: put all all training examples into the root node

$$root = \{ \text{ (A,y),(B,n),(C,y),(D,n),(E,n),(F,n),(G,n),(H,y),(I,y) } \}$$

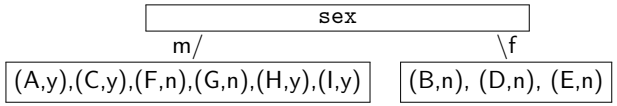
For each feature, check the classification accuracy of this single feature:

feature	accuracies	→	total
`eyes`	blue: (2/3), brown: (3/4), green: (1/2)	→	total: (6/9)
`handsome`	yes: (3/5), no: (2/4)	→	total: (5/9)
`height`	tall: (2/4), short: (3/5)	→	total: (5/9)
`sex`	male: (4/6), female: (3/3)	→	**total: (7/9)**
`soccer`	yes: (3/4), no: (3/6)	→	total: (6/9)

Best feature: `sex`.

Decision Trees Example - Training phase

Step 1 result: first split ist along `sex` feature



Right node: no mistakes, no more splits
Left node: run checks again for remaining data

Step 2:

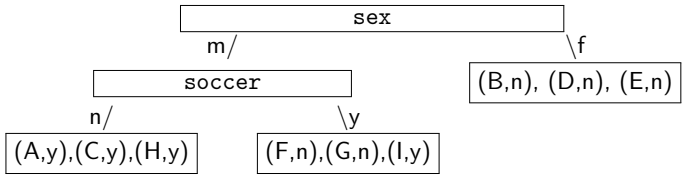
person	`eyes`	`handsome`	`height`	`sex`	`soccer`	`date?`
Apu	blue	yes	tall	male	no	yes
Carl	blue	no	tall	male	no	yes
Frink	brown	yes	tall	male	yes	no
Gil	blue	no	tall	male	yes	no
Homer	green	yes	short	male	no	yes
Itchy	brown	no	short	male	yes	yes

feature	accuracies	→	total
`eyes`	blue: (2/3), brown: (1/2), green: (1/1)	→	total: (4/6)
`handsome`	yes: (2/3), no: (2/3)	→	total: (4/6)
`height`	tall: (2/4), short: (2/2)	→	total: (4/6)
`sex`	male: (4/6)	→	total: (4/6)
`soccer`	yes: (2/3), no: (3/3)	→	**total: (5/6)**

Best feature: `soccer`.

Decision Trees Example - Training phase

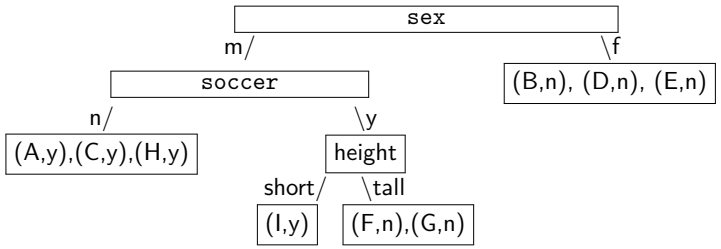
Step 2 result: second split ist along `soccer` feature



Left node: no mistakes, no more splits
Right node: run checks again for remaining data

Decision Trees Example - Training phase

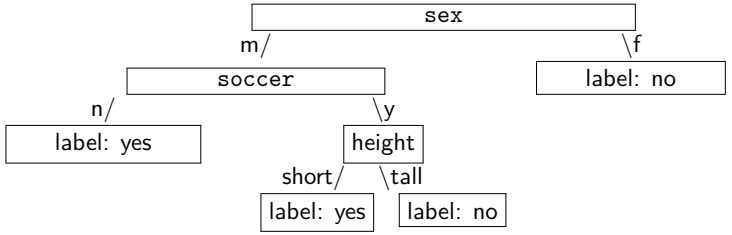
Step 3 result: third split is along `height` feature



Left node: no mistakes, no more splits
Right node: no mistakes, no more splits

Decision Trees Example - Training phase

Step 3 result: third split is along `height` feature



Left node: no mistakes, no more splits
Right node: no mistakes, no more splits

→ Decision tree learning complete.

Decision Trees Example - How good is this classifier?

On all training examples it is correct by construction!
What if we check on new data of the same kind?

person	eyes	handsome	height	sex	soccer	date?	
Jimbo	blue	no	tall	M	no	yes	
Krusty	green	yes	short	M	yes	no	
Lisa	blue	yes	tall	F	no	no	
Moe	brown	no	short	M	no	no	
Ned	brown	yes	short	M	no	yes	
Quimby	blue	no	tall	M	no	yes	

Decision Trees Example - How good is this classifier?

On all training examples it is correct by construction!
What if we check on new data of the same kind?

person	eyes	handsome	height	sex	soccer	date?	tree
Jimbo	blue	no	tall	M	no	yes	**yes**
Krusty	green	yes	short	M	yes	no	**yes**
Lisa	blue	yes	tall	F	no	no	**no**
Moe	brown	no	short	M	no	no	**yes**
Ned	brown	yes	short	M	no	yes	**yes**
Quimby	blue	no	tall	M	no	yes	**yes**

2 mistakes in 6, hm...

Observation
Decision trees don't generalize very well.

Random forest
combines many trees, with random set of splitting features

Decision Trees

- Categorial data can often be handled nicely by a tree.
- For continuous data, $\mathcal{X} = \mathbb{R}^d$, one typically uses splits by comparing any coordinate by a threshold: $[\![x_i \geq \theta]\!]$?
- Finding a split consists of checking all $i = 1, \ldots, d$ and all (reasonable) thresholds, e.g. all $x_i^1, \ldots, x_i^n$
- If d is large, and all dimension are roughly of equal importance (e.g. time series), this is tedious, and the resulting tree might not be good.

Back to: Nonparametric Discriminative Model

Idea: split $\mathcal{X}$ into regions, for each region store an estimate $\hat{p}(y|x)$.

For example, using a **decision tree**:

- training: build a tree
- prediction: for new example x, find its leaf
- output $\hat{p}(y|x) = \frac{n_y}{n}$, where
  - n is the number of examples in the leaf,
  - n_y is the number of example of label y in the leaf.

Back to: Nonparametric Discriminative Model

Idea: split $\mathcal{X}$ into regions, for each region store an estimate $\hat{p}(y|x)$.

For example, using a **decision tree**:

- training: build a tree
- prediction: for new example x, find its leaf
- output $\hat{p}(y|x) = \frac{n_y}{n}$, where
  - n is the number of examples in the leaf,
  - n_y is the number of example of label y in the leaf.

Note: prediction rule

$$c(x) = \operatorname*{\mathbf{argmax}}_y \hat{p}(y|x)$$

is predicts the most frequent label in each leaf.

Parametric Discriminative Model: Logistic Regression

Setting. We assume $\mathcal{X} \subseteq \mathbb{R}^d$ and $\mathcal{Y} = \{-1, +1\}$.

Definition (Logistic Regression (LogReg) Model)
Modeling

$$\hat{p}(y|x; w) = \frac{1}{1 + \exp(-y\langle w, x\rangle)},$$

with parameter vector $w \in \mathbb{R}^d$ is called a *logistic regression* model.

Parametric Discriminative Model: Logistic Regression

Setting. We assume $\mathcal{X} \subseteq \mathbb{R}^d$ and $\mathcal{Y} = \{-1, +1\}$.

Definition (Logistic Regression (LogReg) Model)

Modeling

$$\hat{p}(y|x; w) = \frac{1}{1 + \exp(-y\langle w, x\rangle)},$$

with parameter vector $w \in \mathbb{R}^d$ is called a *logistic regression* model.

Lemma

$\hat{p}(y|x; w)$ *is a well defined probability density w.r.t.* y *for any* $w \in \mathbb{R}^d$.

Proof. elementary.

How to set the weight vector w (based on $\mathcal{D}$)

Logistic Regression Training

Given a training set $\mathcal{D} = \{(x^1, y^1), \ldots, (x^n, y^n)\}$, *logistic regression training* sets the free parameter vector as

$$w_{LR} = \operatorname*{\mathbf{argmin}}_{w \in \mathbb{R}^d} \sum_{i=1}^{n} \log\left(1 + \exp(-y^i \langle w, x^i\rangle)\right)$$

Lemma (Conditional Likelihood Maximization)

w_{LR} *from Logistic Regression training maximizes the conditional data likelihood w.r.t. the LogReg model,*

$$w_{LR} = \operatorname*{\mathbf{argmax}}_{w \in \mathbb{R}^d} \hat{p}(y^1, \ldots, y^n | x^1, \ldots, x^n, w)$$

Proof.

Maximizing

$$\hat{p}(\mathcal{D}^Y | \mathcal{D}^X, w) \overset{i.i.d.}{=} \prod_{i=1}^{n} \hat{p}(y^i | x^i, w)$$

is equivalent to minimizing its negative logarithm

$$\begin{aligned}
-\log\ \hat{p}(\mathcal{D}^Y | \mathcal{D}^X, w) &= -\log \prod_{i=1}^{n} \hat{p}(y^i | x^i, w) = -\sum_{i=1}^{n} \log \hat{p}(y^i | x^i, w) \\
&= -\sum_{i=1}^{n} \log \frac{1}{1 + \exp(-y^i \langle w, x^i\rangle)}, \\
&= -\sum_{i=1}^{n} [\log 1 - \log(1 + \exp(-y^i \langle w, x^i\rangle)], \\
&= \sum_{i=1}^{n} \log(1 + \exp(-y^i \langle w, x^i\rangle).
\end{aligned}$$

□

Alternative Explanation

Definition (Kullback-Leibler (KL) divergence)

Let p and q be two probability distributions (for discrete $\mathcal{Z}$) or probabilitiy densities with respect to a measure $\mathrm{d}\lambda$ (for continuous $\mathcal{Z}$).
The **Kullbach-Leibler (KL)-divergence** between p and q is defined as

$$\mathrm{KL}(p \,\|q) = \sum_{z \in \mathcal{Z}} p(z) \log \frac{p(z)}{q(z)}, \quad \text{or} \quad \mathrm{KL}(p \,\|q) = \int_{z \in \mathcal{Z}} p(z) \log \frac{p(z)}{q(z)}\ \mathrm{d}\lambda(\mathsf{z}),$$

(with convention $0 \log 0 = 0$, and $a \log \frac{a}{0} = \infty$ for $a > 0$).

Definition (Kullback-Leibler (KL) divergence)

Let p and q be two probability distributions (for discrete $\mathcal{Z}$) or probabilitiy densities with respect to a measure $\mathrm{d}\lambda$ (for continuous $\mathcal{Z}$).
The **Kullbach-Leibler (KL)-divergence** between p and q is defined as

$$\mathrm{KL}(p\,\|q) = \sum_{z\in\mathcal{Z}} p(z)\log\frac{p(z)}{q(z)}, \quad \text{or} \quad \mathrm{KL}(p\,\|q) = \int_{z\in\mathcal{Z}} p(z)\log\frac{p(z)}{q(z)}\,\mathrm{d}\lambda(\mathrm{z}),$$

(with convention $0\log 0 = 0$, and $a\log\frac{a}{0} = \infty$ for $a > 0$).
KL is a similarity measure between probability distributions. It fulfills

$$0 \le KL(p\,\|q) \le \infty, \qquad \text{and} \qquad KL(p\,\|q) = 0 \;\Leftrightarrow\; p = q.$$

However, KL is **not a metric**.

- it is in general not symmetric, $KL(q\,\|p) \neq KL(p\,\|q)$,
- it does not fulfill the triangle inequality.

Definition (Expected Kullback-Leibler (KL) divergence)

Let $p(x,y)$ be a probability distribution over $(x,y) \in \mathcal{X}\times\mathcal{Y}$ and let $\hat{p}(y|x)$ be an approximation of $p(y|x)$.
We measure the approximation quality by the **expected KL-divergence between** p **and** q over all $x \in \mathcal{X}$:

$$\mathrm{KL}_{exp}(p\,\|q) = \mathbb{E}_{x\sim p(x)}\{\ \mathrm{KL}(p(\cdot|x)\|q(\cdot|x))\ \}$$

Theorem

The parameter w_{LR} obtained by logistic regression training approximately minimizes the KL divergence between $\hat{p}(y|x;w)$ and $p(y|x)$.

Proof.

We show how maximimzing the conditional likelihood relates to KL_{exp}:

$$\begin{aligned}\mathrm{KL}_{exp}(p\|\hat{p}) &= \mathbb{E}_{x\sim p(x)}\sum_{y\in\mathcal{Y}} p(y|x)\log\frac{p(y|x)}{\hat{p}(y|x,w)}\\ &= \underbrace{\mathbb{E}_{(x,y)\sim p(x,y)}\log p(y|x)}_{\text{indep. of } w} - \mathbb{E}_{(x,y)\sim p(x,y)}\log\hat{p}(y|x,w)\end{aligned}$$

We can't maximize $\mathbb{E}_{(x,y)\sim p(x,y)}\log\hat{p}(y|x,w)$ directly, because $p(x,y)$ is unknown. But we can maximimize its empirical estimate based on $\mathcal{D}$:

$$\mathbb{E}_{(x,y)\sim p(x,y)}\log\hat{p}(y|x,w) \;\approx\; \underbrace{\sum_{(x^i,y^i)\in\mathcal{D}} \log\hat{p}(y^i|x^i,w)}_{\text{log of conditional data likelihood}}\ .$$

The approximation will get better the more data we have. □

Theorem

Logistic Regression training,

$$w_{LR} \;=\; \underset{w\in\mathbb{R}^d}{\mathbf{argmin}}\ \mathcal{L}(w) \quad \textit{for} \quad \mathcal{L}(w) = \sum_{i=1}^{n}\log\left(1+\exp(-y^i\langle w, x^i\rangle)\right),$$

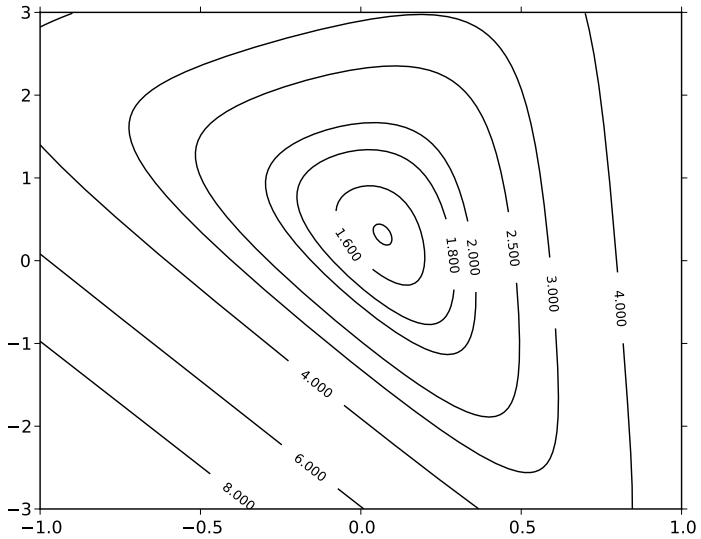
is a C^∞-smooth, unconstrained, convex optimization problem.

Proof.

1. it's an optimization problem,
2. it's unconstrained,
3. it's smooth (the objective function is C^∞ differentiable),
4. remains to show: the objective function is a *convex* function.
   Since $\mathcal{L}$ is smooth, it's enough to show that its *Hessian matrix* (the matrix of 2nd partial derivatives) is everywhere *positive definite.*

Exercise!

Example plot: LogReg objective for three examples in $\mathbb{R}^2$



Numeric Optimization

Convex optimization is a well understood field. We can use, e.g., *gradient descent* will converge to the globally optimal solution!

Steepest Descent Minimization with Line Search

input $\epsilon > 0$ tolerance (for stopping criterion)

1: $w \leftarrow 0$
2: **repeat**
3: $\quad v \leftarrow -\nabla_w \mathcal{L}(w)$ {descent direction}
4: $\quad \eta \leftarrow \mathbf{argmin}_{\eta>0} \mathcal{L}(w + \eta v)$ {1D line search}
5: $\quad w \leftarrow w + \eta d$
6: **until** $\|v\| < \epsilon$

output $w \in \mathbb{R}^d$ learned weight vector

Faster conference from methods that use second-order information, e.g., *conjugate gradients* or *(L-)BFGS* $\rightarrow$ *convex optimization lecture*

Binary classification with a LogReg Models

A discriminative probability model, $\hat{p}(y|x)$, is enough to make decisions:

$$c(x) = \mathbf{argmax}_{y \in \mathcal{Y}} \hat{p}(y|x) \quad \text{or} \quad c(x) = \mathbf{argmin}_{y \in \mathcal{Y}} \mathbb{E}_{\bar{y} \sim \hat{p}(y|x)} \ell(\bar{y}, y).$$

For Logistic Regression, this is particularly simple:

Lemma

The LogReg classification rule for $0/1$*-loss is*

$$c(x) = \text{sign } \langle w, x \rangle.$$

For a loss function $\ell = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ *the rule is*

$$c_\ell(x) = \text{sign}[\ \langle w, x \rangle + \log \frac{c-d}{b-a}\ ],$$

In particular, ***the decision boundaries is linear (or affine).***

Proof. Elementary, since $\log \frac{\hat{p}(+1|x;w)}{p(-1|x;w)} = \langle w, x \rangle$

Multiclass Logistic Regression

For $\mathcal{Y} = \{1, \ldots, M\}$, we can do two things:

- Parametrize $\hat{p}(y|x; \vec{w})$ using $M-1$ vectors, $w_1, \ldots, w_{M-1} \in \mathbb{R}^d$, as

$$\hat{p}(y|x, w) = \frac{\exp(\langle w_y, x \rangle)}{1 + \sum_{j=1}^{M-1} \exp(\langle w_j, x \rangle)} \quad \text{for } y = 1, \ldots, M-1,$$
$$\hat{p}(M|x, w) = \frac{1}{1 + \sum_{j=1}^{M-1} \exp(\langle w_j, x \rangle)}.$$

- Parametrize $\hat{p}(y|x; \vec{w})$ using M vectors, $w_1, \ldots, w_M \in \mathbb{R}^d$, as

$$\hat{p}(y|x, w) = \frac{\exp(\langle w_y, x \rangle)}{\sum_{j=1}^{M} \exp(\langle w_j, x \rangle)} \quad \text{for } y = 1, \ldots, M,$$

Second is more popular, since it's easier to implement and analyze.

Decision boundaries are still *piecewise linear*, $c(x) = \mathbf{argmax}_y \langle w_y, x \rangle$.

Summary: Discriminative Models

Discriminative models treats the input data, x, as fixed and only model the distribution of the output labels $p(y|x)$.

Discriminative models, in particular LogReg, are popular, because

- they often need less training data than generative models,
- they provide an estimate of the uncertainty of a decision $p(c(x)|x)$,
- training them is often efficient,
  e.g. Yahoo trains LogReg models routinely from billions of examples.

But: they also have drawbacks

- often $\hat{p}_{LR}(y|x) \not\to p(y|x)$, even for $n \to \infty$,
- they usually are good for *prediction*, but they do not reflect the actual *mechanism*.

Note: there are much more complex discriminative models than LogReg, e.g. Conditional Random Fields (maybe later).

Maximum Margin Classifiers

Let's use $\mathcal{D}$ to estimate a classifier $c : \mathcal{X} \to \mathcal{Y}$ directly.

Maximum Margin Classifiers

Let's use $\mathcal{D}$ to estimate a classifier $c : \mathcal{X} \to \mathcal{Y}$ directly.

For a start, we fix

- $\mathcal{D} = \{(x^1, y^1), \ldots, (x^n, y^n)\}$,
- $\mathcal{Y} = \{+1, -1\}$,
- we look for classifiers with linear decision boundary.

Several of the classifiers we saw had *linear* decision boundaries:

- Generative classifiers for Gaussian class-conditional densities with shared covariance matrix
- Logistic Regression
- Perceptron (didn't introduce yet)

What's the **best linear classifier**?

Linear classifiers

Definition

Let

$$\mathcal{F} = \{\ f : \mathbb{R}^d \to \{\pm 1\} \text{ with } f(x) = b + a_1 x_1 + \cdots + a_d x_d = b + \langle w, x \rangle\ \}$$

be the set of linear (affine) function from $\mathbb{R}^d \to \mathbb{R}$.

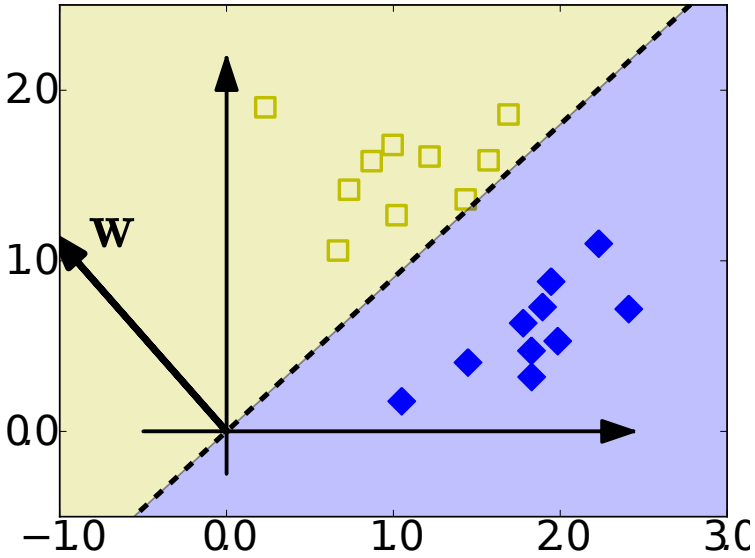
A classifier $g : \mathcal{X} \to \mathcal{Y}$ is called **linear**, if it can be written as

$$g(x) = \operatorname{sign} f(x)$$

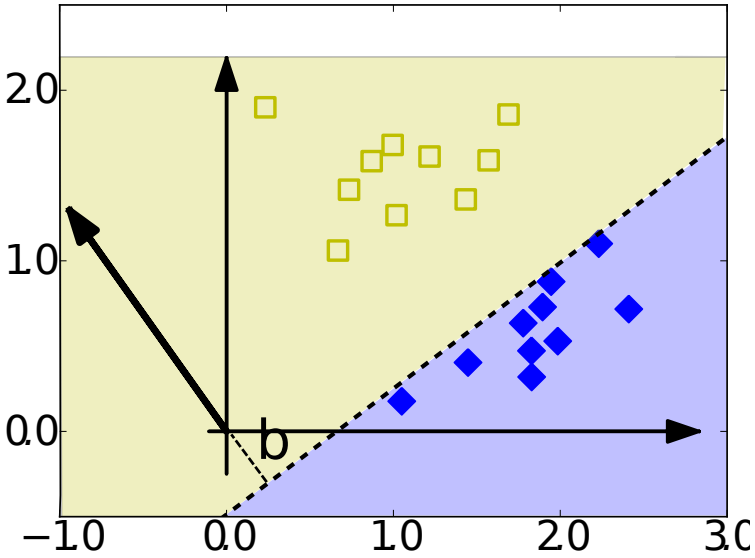
for some $f \in \mathcal{F}$.

We write $\mathcal{G}$ for the set of all linear classifiers.

A linear classifier, $g(x) = \text{sign}\langle w, x\rangle$, with $b = 0$



A linear classifier $g(x) = \text{sign}(\langle w, x\rangle + b)$, with $b > 0$



Linear classifiers

Definition
We call a classifier, g, **correct** (for a training set $\mathcal{D}$), if it predicts the correct labels for all training examples:

$$g(x^i) = y^i \qquad \text{for } i = 1, \ldots, n.$$

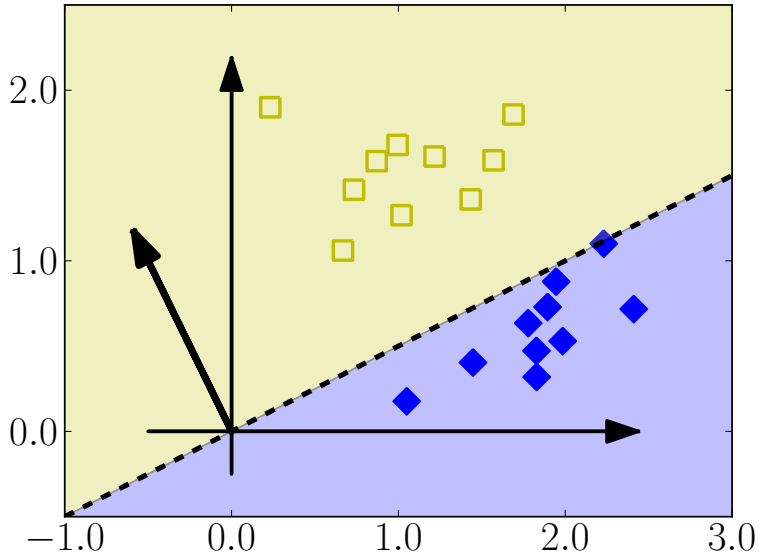
Linear classifiers

Definition
We call a classifier, g, **correct** (for a training set $\mathcal{D}$), if it predicts the correct labels for all training examples:

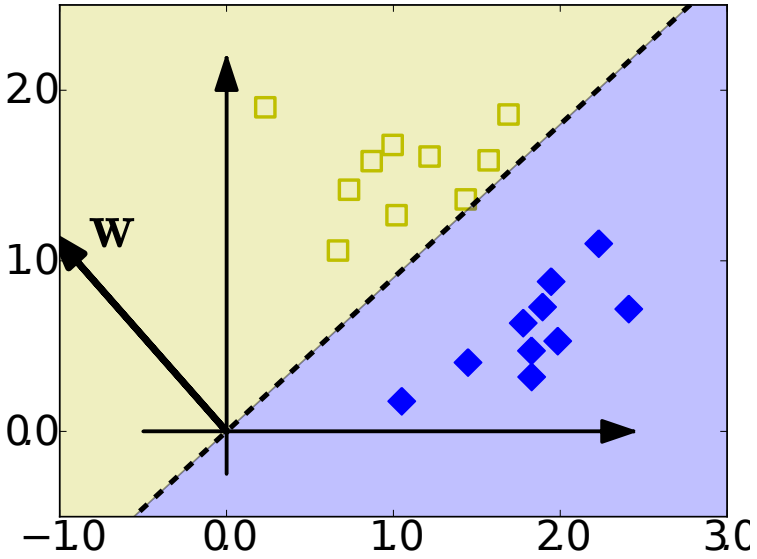
$$g(x^i) = y^i \qquad \text{for } i = 1, \ldots, n.$$

Definition (Linear Separability)
A training set $\mathcal{D}$ is called **linearly separable**, if it allows a correct linear classifier (i.e. the classes can be separated by a hyperplane).

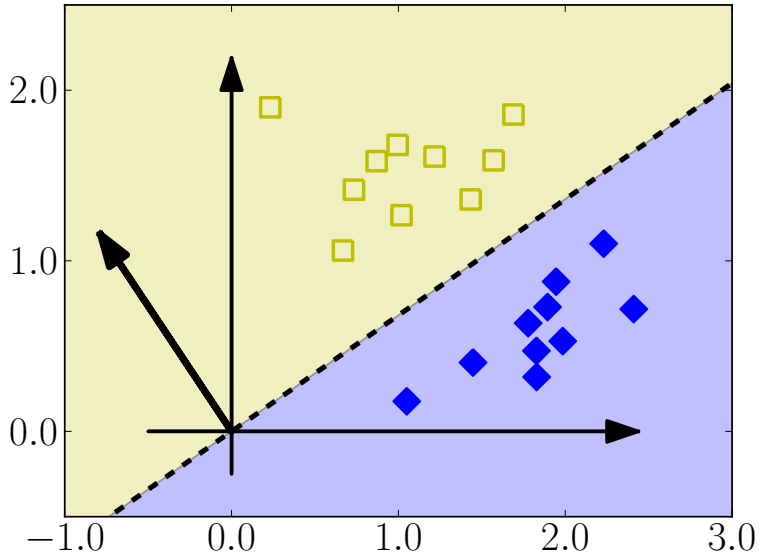
A linearly separable dataset and a correct classifier



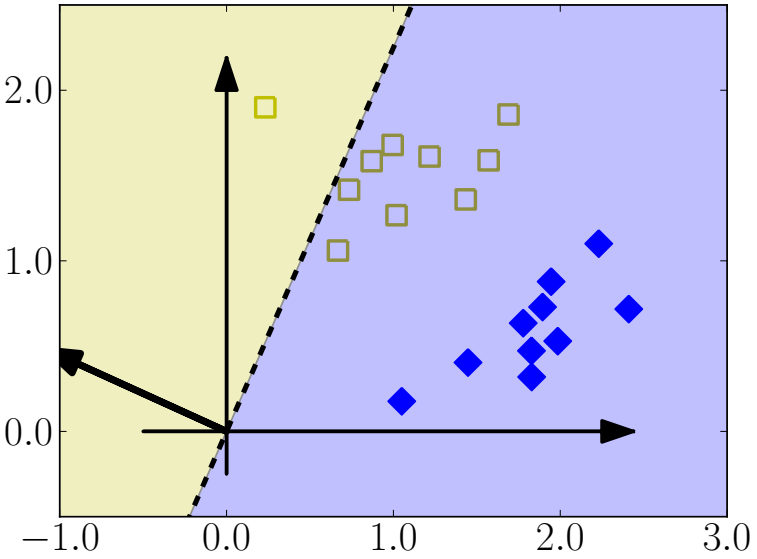
A linearly separable dataset and a correct classifier



A linearly separable dataset and a correct classifier



An incorrect classifier



Linear Classifiers

Definition

The **robustness** of a classifier g (with respect to $\mathcal{D}$) is the largest amount, ρ, by which we can perturb the training samples without changing the predictions of g.

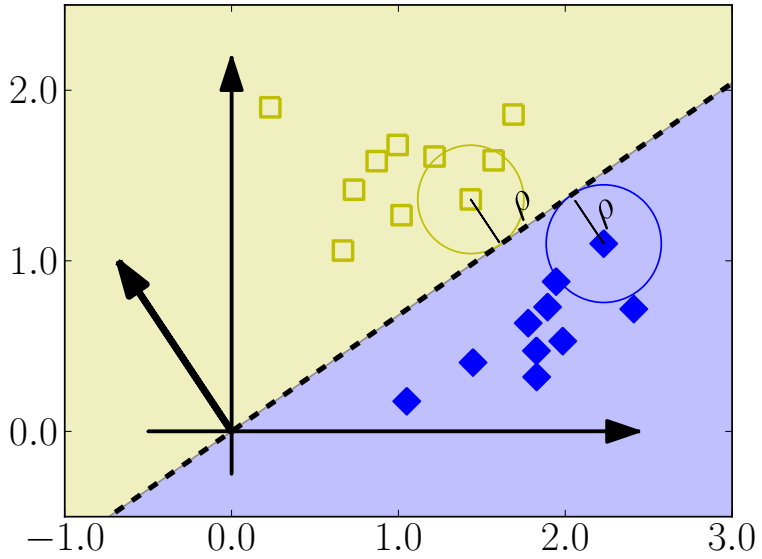
$$g(x^i + \epsilon) = g(x^i), \qquad \text{for all } i = 1, \ldots, n.$$

for any $\epsilon \in \mathbb{R}^d$ with $\|\epsilon\| < \rho$.

Example

- constant classifier, e.g. $c(x) \equiv 1$: very robust ($\rho = \infty$),
  (but it is not *correct*, in the sense of the previous definition)
- robustness of the *Perceptron*: can be arbitrarily small
  (see Exercise...)

Robustness, ρ, of a linear classifier



Definition (Margin)

Let $f(x) = \langle w, x \rangle + b$ define a *correct* linear classifier.
Then the smallest (Euclidean) distance of any training example from the decision hyperplane is called the **margin** of f (with respect to $\mathcal{D}$).

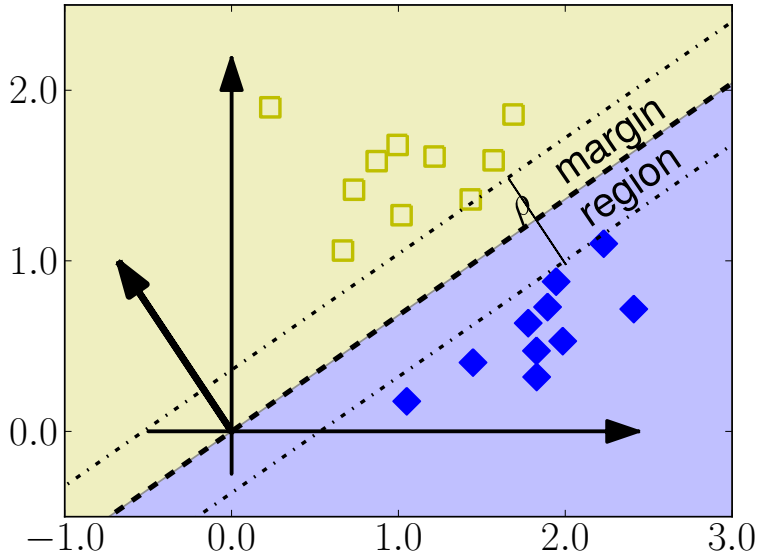
Lemma

We can compute the margin of a linear classifier $f = \langle w, x \rangle + b$ as

$$\rho = \mathbf{min}_{i=1,\ldots,n} \left| \left\langle \frac{w}{\|w\|}, x^i \right\rangle + b \right|.$$

Proof.
High school maths: distance between a points and a hyperplane in *Hessian normal form*.

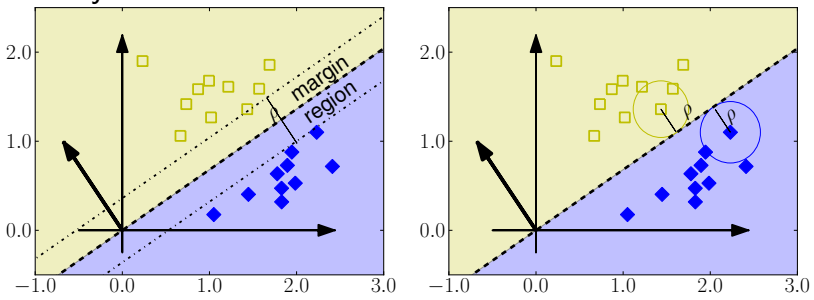
Margin, ρ, of a linear classifier



Theorem

The robustness of a linear classifier function $g(x) = \operatorname{sign} f(x)$ with $f(x) = \langle w, x \rangle + b$ is identical to the margin of f.

Proof by Picture



Maximum-Margin Classifier

Theorem

*Let $\mathcal{D}$ be a linearly separable training set. Then the **most robust, correct classifier** is given by $g(x) = \operatorname{sign}\langle w^*, x \rangle + b^*$ where (w^*, b^*) are the solution to*

$$\mathbf{min}_{w \in \mathbb{R}^d} \ \frac{1}{2}\|w\|^2$$

subject to

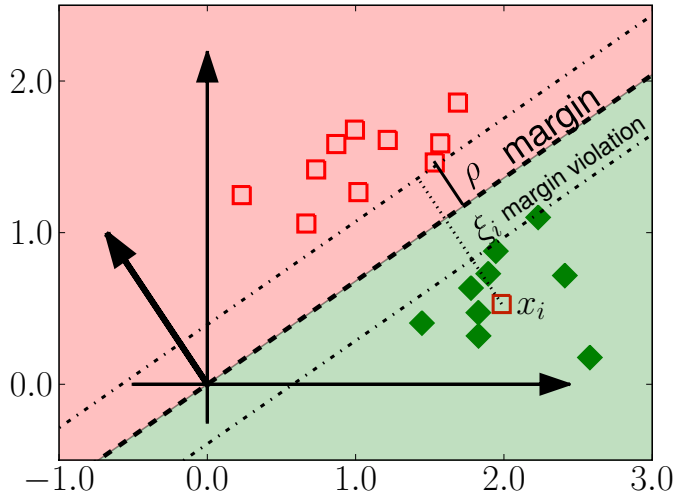
$$y^i(\langle w, x^i \rangle + b) \geq 1, \quad \text{for } i = 1, \ldots, n.$$

Remark

- The classifier defined above is call **Maximum (Hard) Margin Classifier**, or **Hard-Margin Support Vector Machine (SVM)**
- It is unique (follows from strictly convex optimization problem).

Non-Separable Training Sets

Observation (Not all training sets are linearly separable.)



Definition (Maximum Soft-Margin Classifier)

Let $\mathcal{D}$ be a training set, not necessarily linearly separable. Let $C > 0$. Then the classifier $g(x) = \operatorname{sign}\langle w^*, x \rangle$ where (w^*, b^*) are the solution to

$$\min_{w \in \mathbb{R}^d, \xi \in \mathbb{R}^n} \ \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{n} \xi^i$$

subject to

$$y^i(\langle w, x^i \rangle + b) \geq 1 - \xi^i, \quad \text{for } i = 1, \ldots, n.$$
$$\xi^i \geq 0, \quad \text{for } i = 1, \ldots, n.$$

is called **Maximum (Soft-)Margin Classifier** or **Linear Support Vector Machine**.

Maximum Soft-Margin Classifier

Theorem

The maximum soft-margin classifier exists and is unique for any $C > 0$.

Proof. optimization problem is strictly convex

Remark

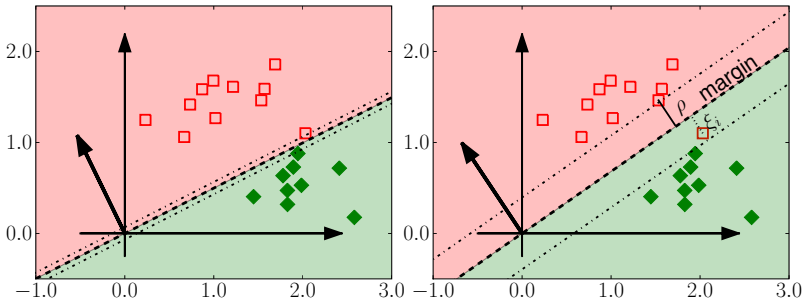
The constant $C > 0$ is called **regularization** parameter.

It balances the wishes for robustness and for correctness

- $C \to 0$: mistakes don't matter much, emphasis on short w
- $C \to \infty$: as few errors as possible, might not be robust

Remark

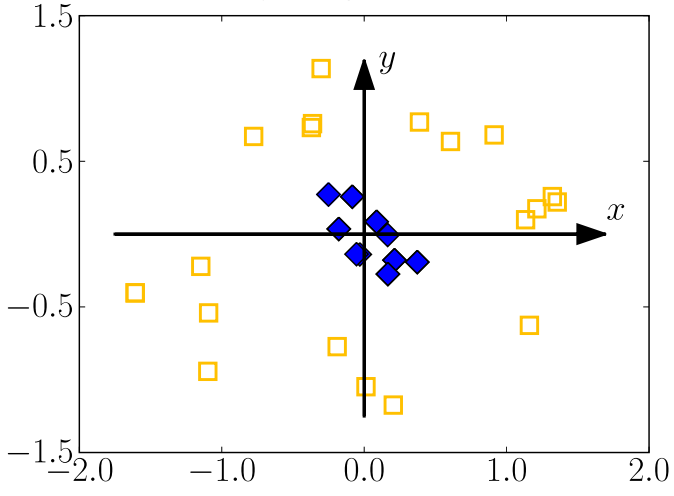
Sometimes, a soft margin is better even for linearly separable datasets!



Left: small margin, no errors) Right: large margin, but 1 error

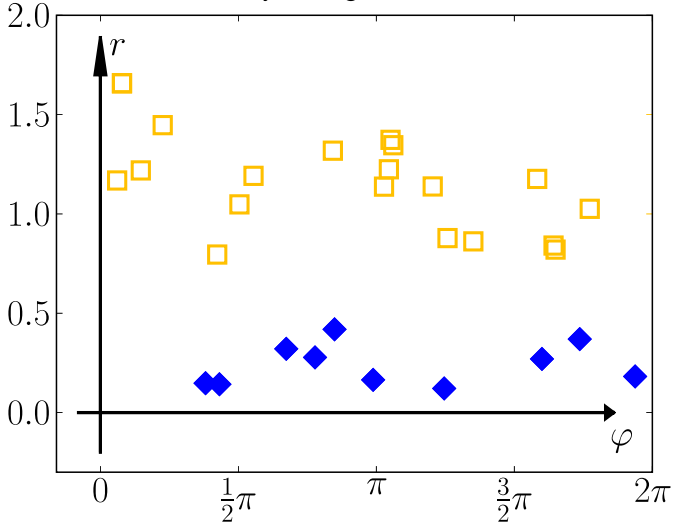
Nonlinear Classifiers

What, if a linear classifier is really not a good choice?



Nonlinear Classifiers

What, if a linear classifier is really not a good choice?



Change the data representation, e.g. Cartesian → polar coordinates

Nonlinear Classifiers

Definition (Max-margin Generalized Linear Classifier)

Let $C > 0$. Assume a necessarily linearly separable training set

$$\mathcal{D} = \{(x^1, y^1), \dots x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}.$$

Let $\phi : \mathcal{X} \to \mathcal{H}$ be a feature map from $\mathcal{X}$ into a Hilbert space $\mathcal{H}$.

Then we can form a new training set

$$\mathcal{D}^\phi = \{\ (\phi(x^1), y^1),\ \dots,\ (\phi(x^n), y^n)\ \} \subset \mathcal{H} \times \mathcal{Y}.$$

The maximum-(soft)-margin linear classifier in $\mathcal{H}$,

$$g(x) = \operatorname{sign}\langle w, \phi(x)\rangle_{\mathcal{H}} + b,$$

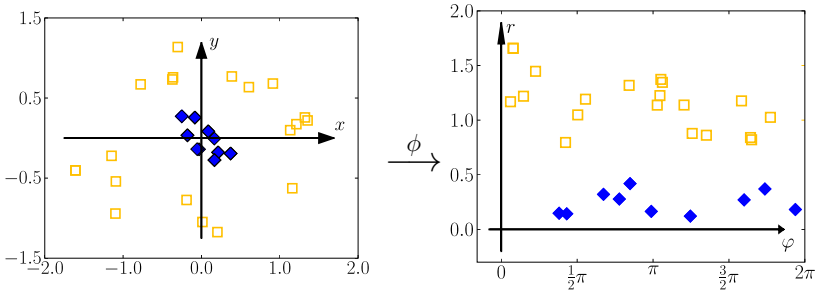
for $w \in \mathcal{H}$ and $b \in \mathbb{R}$ is called **max-margin generalized linear classifier**.

It is still *linear* w.r.t w, but (in general) nonlinear with respect to x.

Example (Polar coordinates)

Left: dataset $\mathcal{D}$ for which no good linear classifier exists.
Right: dataset $\mathcal{D}^\phi$ for $\phi : \mathcal{X} \to \mathcal{H}$ with $\mathcal{X} = \mathbb{R}^2$ and $\mathcal{H} = \mathbb{R}^2$

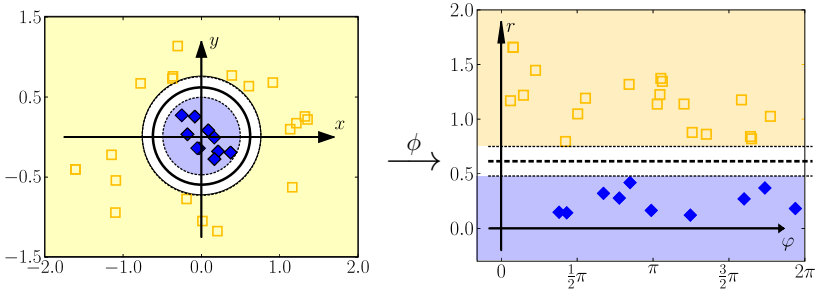
$$\phi(x, y) = (\sqrt{x^2 + y^2},\ \arctan\frac{y}{x}) \qquad (\text{and } \phi(0,0) = (0,0))$$



Example (Polar coordinates)

Left: dataset $\mathcal{D}$ for which no good linear classifier exists.
Right: dataset $\mathcal{D}^\phi$ for $\phi : \mathcal{X} \to \mathcal{H}$ with $\mathcal{X} = \mathbb{R}^2$ and $\mathcal{H} = \mathbb{R}^2$

$$\phi(x, y) = (\sqrt{x^2 + y^2},\ \arctan\frac{y}{x}) \qquad (\text{and } \phi(0,0) = (0,0))$$



Any classifier in $\mathcal{H}$ induces a classifier in $\mathcal{X}$.

Other popular feature mappings, ϕ

Example (d-th degree polynomials)

$$\phi : (x_1, \dots, x_n) \mapsto (1, x_1, \dots, x_n, x_1^2, \dots, x_n^2, \dots, x_1^d, \dots, x_n^d)$$

Resulting classifier: d-th degree polynomial in x. $g(x) = \operatorname{sign} f(x)$ with
$f(x) = \langle w, \phi(x)\rangle = \sum_j w_j \phi(x)_j = \sum_i a_i x_i + \sum_{ij} b_{ij} x_i x_j + \dots$

Example (Distance map)

For a set of prototype $p_1, \dots, p_N \in \mathcal{H}$:

$$\phi : \vec{x} \mapsto \left(e^{-\|\vec{x} - \vec{p}_i\|^2}, \dots, e^{-\|\vec{x} - \vec{p}_N\|^2}\right)$$

Classifier: combine weights from close enough prototypes

$$g(x) = \operatorname{sign}\langle w, \phi(x)\rangle = \operatorname{sign} \sum_{i=1}^{n} a_i e^{-\|\vec{x} - \vec{p}_i\|^2}.$$

Finding the Maximum Margin Classifier numerically – Optimization II

$$\min_{w\in\mathbb{R}^d, b\in\mathbb{R}\xi\in\mathbb{R}^n} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi^i$$

subject to

$$y^i\langle w, \phi(x^i)\rangle \geq 1-\xi^i, \quad \text{for } i=1,\ldots,n,$$
$$\xi^i \geq 0. \quad \text{for } i=1,\ldots,n.$$

How to solve numerically?

- off-the-shelf Quadratic Program (QP) solver
  only for small dimensions and training sets (a few hundred),
- variants of gradient descent,
  high dimensional data, large training sets (millions)
- by convex duality,
  for very high dimensional data and not so many examples ($d \gg n$)