Statistical Machine Learning
https://cvml.ist.ac.at/courses/SML_W20

Christoph Lampert

I S T AUSTRIA
Institute of Science and Technology

Fall Semester 2020/2021
Lecture 4

Overview (tentative)

Date		no.	Topic
Oct 05	Mon	1	A Hands-On Introduction
Oct 07	Wed	2	Bayesian Decision Theory, Generative Probabilistic Models
Oct 12	Mon	3	Discriminative Probabilistic Models
Oct 14	Wed	4	Maximum Margin Classifiers, Generalized Linear Models
Oct 19	Mon	5	Estimators; Overfitting/Underfitting, Regularization, Model Selection
Oct 21	Wed	6	Bias/Fairness, Domain Adaptation
Oct 26	Mon	-	no lecture (public holiday)
Oct 28	Wed	7	Learning Theory I
Nov 02	Mon	8	Learning Theory II
Nov 04	Wed	9	Deep Learning I
Nov 09	Mon	10	Deep Learning II
Nov 11	Wed	11	Unsupervised Learning
Nov 16	Mon	12	project presentations
Nov 18	Wed	13	buffer

In the real world, $p(x, y)$ is unknown, but we have a training set $\mathcal{D}$. At least 3 approaches:

Definition

Given a training set $\mathcal{D}$, we call it

- a **generative probabilistic approach**:
  if we use $\mathcal{D}$ to build a model $\hat{p}(x, y)$ of $p(x, y)$, and then define

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

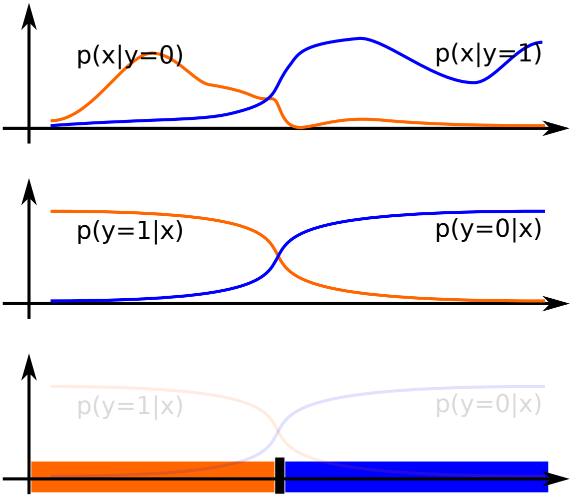
- a **discriminative probabilistic approach**:
  if we use $\mathcal{D}$ to build a model $\hat{p}(y|x)$ of $p(y|x)$ and define

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

- a **decision theoretic approach**: if we use $\mathcal{D}$ to directly seach for a classifier c.

Observation

Even easier than estimating $p(y|x)$ or $p(x,y)$ should be to just estimate the decision boundary between classes.



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

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

For a start, we fix

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

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

- Perceptron
- Generative classifiers for Gaussian class-conditional densities with shared covariance matrix
- Logistic Regression

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

Maximum Margin Classifiers

Definition

Let
$$\mathcal{F} = \{\ f : \mathbb{R}^d \to \mathbb{R} \text{ with } f(x) = b + w_1 x_1 + \cdots + w_d x_d = b + \langle w, x \rangle\ \}$$
be the set of linear (affine) function from $\mathbb{R}^d \to \mathbb{R}$. For any $f \in \mathcal{F}$,

- w is called weight vector,
- b is called bias term.

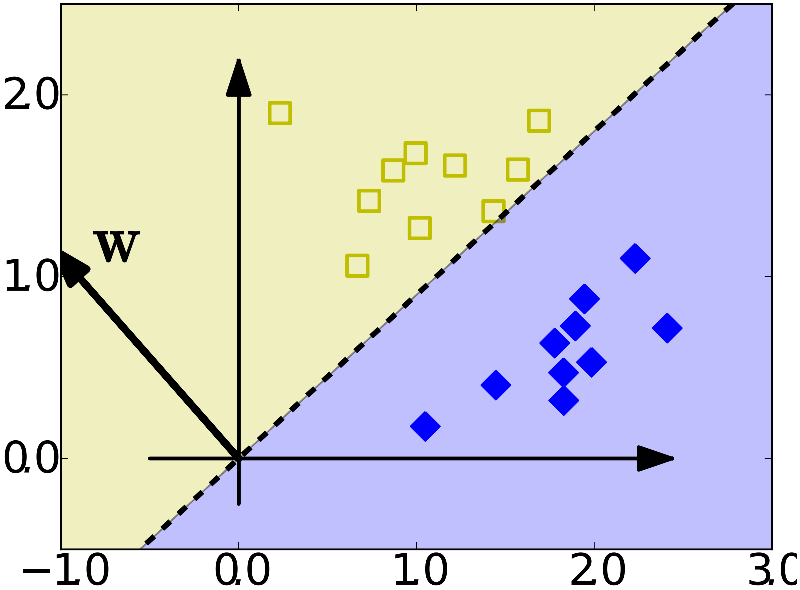
A classifier $g : \mathcal{X} \to \mathcal{Y}$ is called **linear**, if it can be written as

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

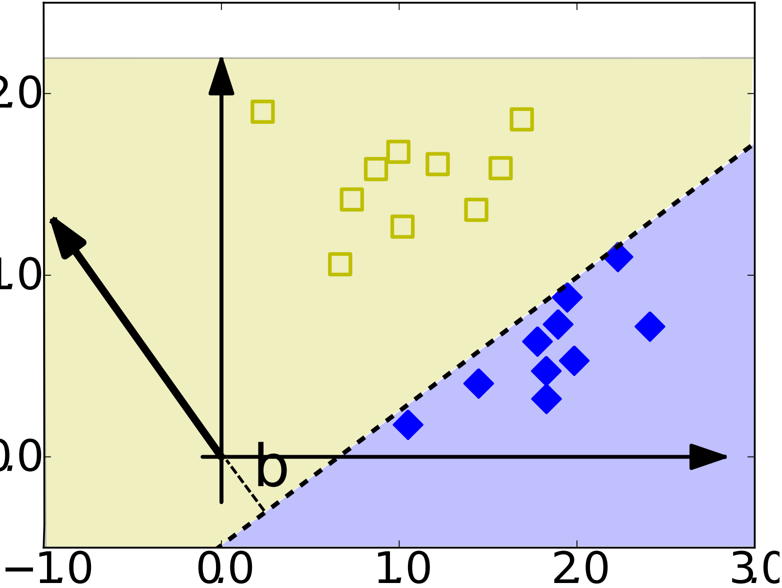
for some $f \in \mathcal{F}$.

Given a training set $\mathcal{D} = \{(x^1, y^1), \ldots, (x^n, y^n)\} \overset{i.i.d.}{\sim} p$, what's the best f (and induced g)?

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



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



Feature augmentation

The bias term is good for intuition, but annoying in analysis:

Useful trick: feature augmentation

Adding a constant feature allows us to avoid models with explicit bias term:

- instead of $x = (x^1, \dots, x^d) \in \mathbb{R}^d$, use $\tilde{x} = (x^1, \dots, x^d, 1) \in \mathbb{R}^{d+1}$
- for any $\tilde{w} \in \mathbb{R}^{d+1}$, think $\tilde{w} = (w, b)$ with $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$

Linear function in $\mathbb{R}^{d+1}$:

$$f(\tilde{x}) = \langle \tilde{w}, \tilde{x} \rangle = \sum_{i=1}^{d+1} \tilde{w}_i \tilde{x}_i = \sum_{i=1}^{d} \tilde{w}_i \tilde{x}_i + \tilde{w}_{d+1} \tilde{x}_{d+1} = \langle w, x \rangle + b$$

$$\boxed{\text{Linear classifier with bias in } \mathbb{R}^d \equiv \text{linear classifier with no bias in } \mathbb{R}^{d+1}}$$

Augmenting with other (larger) values than 1 can make sense, see later...

Definition (Ad hoc)

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

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

Example (Perceptron)

- if the *Perceptron* converges, the result is an *correct* classifier.
- any classifier with zero training error is *correct*.

Definition (Ad hoc)

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

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

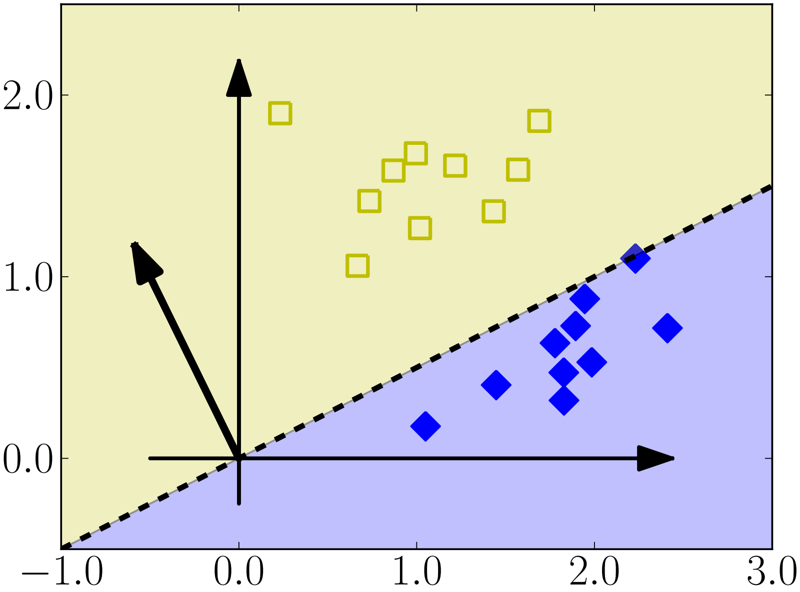
Example (Perceptron)

- if the *Perceptron* converges, the result is an *correct* classifier.
- any classifier with zero training error is *correct*.

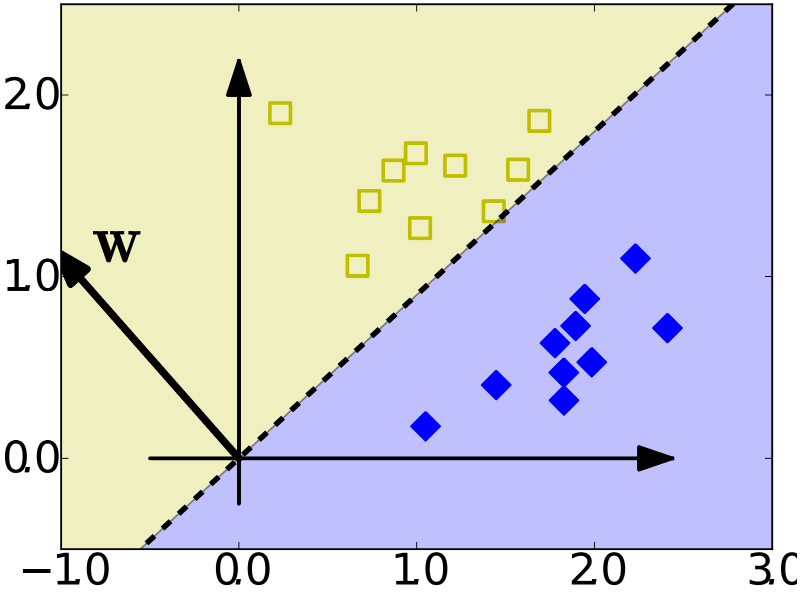
Definition (Linear Separability)

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).

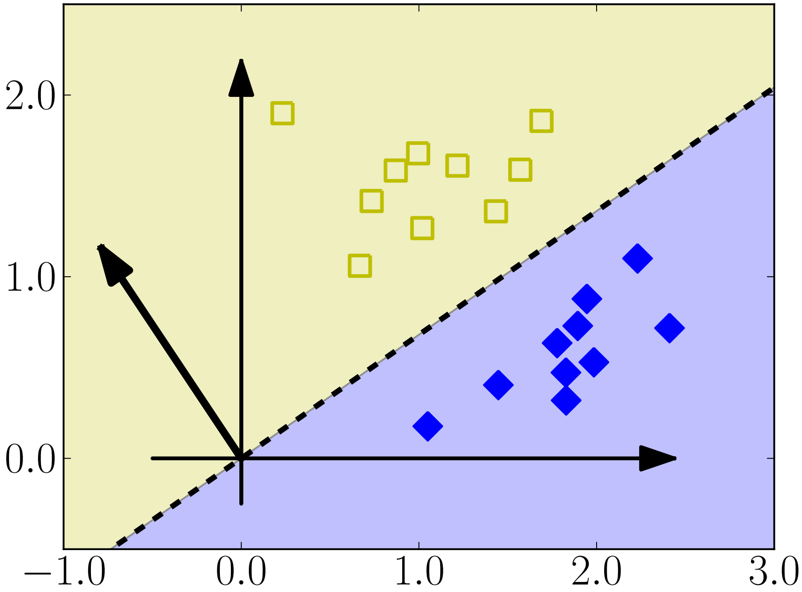
A linearly separable dataset and a correct classifier



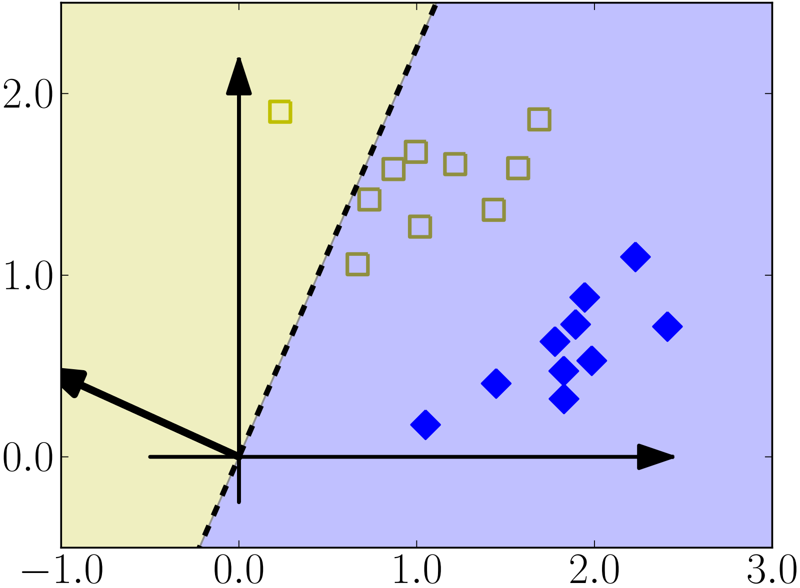
A linearly separable dataset and a correct classifier



A linearly separable dataset and a correct classifier



An incorrect classifier



Definition (Ad hoc)

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.

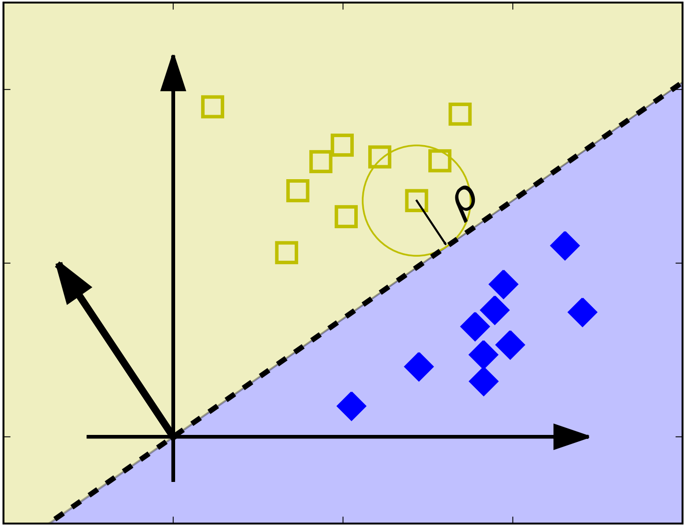
$$g(x^i + \epsilon) = g(x^i), \qquad \text{for all } i = 1, \dots, n.$$

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.
The **margin** of f (with despect to $\mathcal{D}$) is the largest amount by which the decision hyperplane in the direction of the weight vector or its negative without making the classifier incorrect.

Lemma

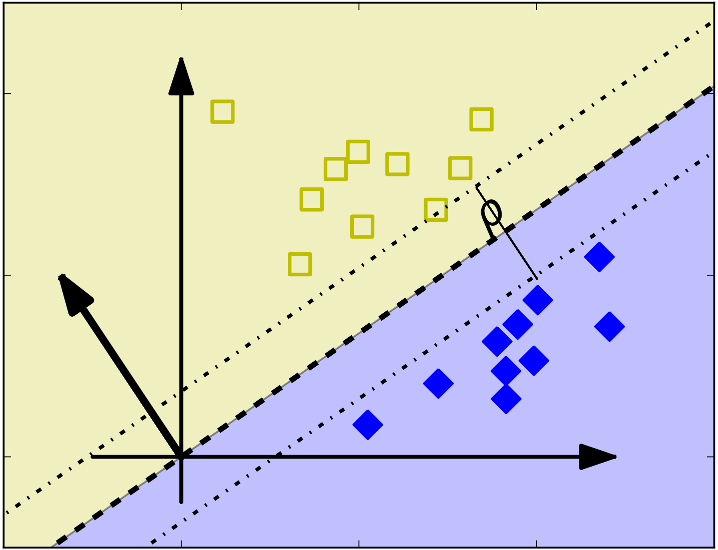
The margin of f is identical to the smallest distance of any point in $\mathcal{D}$ to the decision boundary. We can compute the margin of a linear classifier $f = \langle w, x\rangle + b$ as

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

Proof.

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

Margin, ρ, of a linear classifier



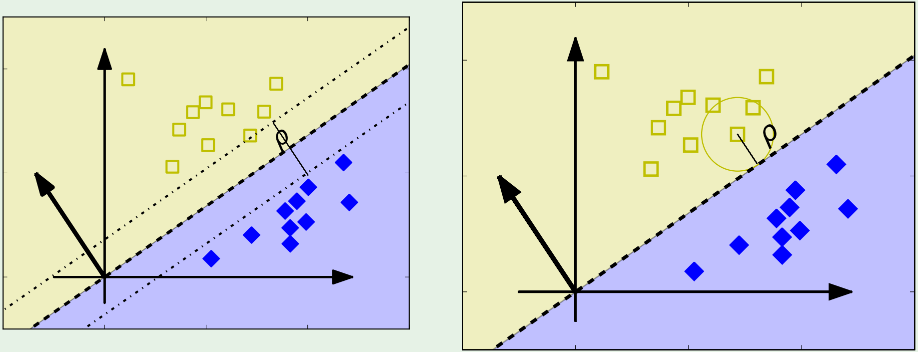
Theorem

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

Theorem

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

Proof by Picture



Proof (blackboard). For any $i = 1, \dots, n$ and any $\epsilon \in \mathbb{R}^d$

$$f(x^i + \epsilon) = \langle w, x^i + \epsilon \rangle = \langle w, x^i \rangle + \langle w, \epsilon \rangle = f(x^i) + \langle w, \epsilon \rangle,$$

so it follows (Cauchy-Schwarz inequality) that

$$f(x^i) - \|w\|\|\epsilon\| \quad \leq \quad f(x^i + \epsilon) \quad \leq \quad f(x^i) + \|w\|\|\epsilon\|.$$

Checking the cases $\epsilon = \pm \frac{\|\epsilon\|}{\|w\|} w$, we see that these inequalities are sharp.

To ensure $g(x^i + \epsilon) = g(x^i)$ for all training samples, $f(x^i)$ and $f(x^i + \epsilon)$ have the same sign for all ϵ, i.e. $|f(x^i)| \geq \|w\|\|\epsilon\|$ for $i = 1, \dots, n$.

This inequality holds for all samples, so in particular it holds for the one of minimal score, and $\mathbf{min}_i \, |f(x^i)| = \mathbf{min}_i \, |\langle w, x^i \rangle| = \rho$.

$\square$

Theorem

Let $\mathcal{D}$ be a linearly separable training set. Then the ***most robust, correct linear classifier*** *(without bias term) is given by* $g(x) = \mathrm{sign}\langle w^*, x\rangle$ *where* w^* *are the solution to*

$$\min_{w \in \mathbb{R}^d} \frac{1}{2}\|w\|^2$$

subject to

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

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).

Proof.

1. All w that fulfill the inequalities yield *correct* classifiers.
2. Since $\mathcal{D}$ is linearly separable, there exists some v with
$$\operatorname{sign}\langle v, x^i\rangle = y_i, \quad \text{i.e.} \quad y_i\langle v, x^i\rangle \geq \gamma > 0.$$
for $\gamma = \mathbf{min}_i\, y_i\langle v, x^i\rangle$. So $\tilde{v} = v/\gamma$, fulfills the inequalities and we see that the constraint set is at least not empty.

Proof.

1. All w that fulfill the inequalities yield *correct* classifiers.
2. Since $\mathcal{D}$ is linearly separable, there exists some v with
$$\operatorname{sign}\langle v, x^i\rangle = y_i, \quad \text{i.e.} \quad y_i\langle v, x^i\rangle \geq \gamma > 0.$$
for $\gamma = \mathbf{min}_i\, y_i\langle v, x^i\rangle$. So $\tilde{v} = v/\gamma$, fulfills the inequalities and we see that the constraint set is at least not empty.
3. Now we check (with $i = 1, \dots, n$):
$$\begin{aligned}
& \min_{w\in\mathbb{R}^d} \frac{1}{2}\|w\|^2 \text{ sb.t. } y^i\langle w, x^i\rangle \geq 1 \\
\Leftrightarrow \quad & \max_{w\in\mathbb{R}^d} \frac{1}{\|w\|} \quad \text{sb.t. } y^i\langle w, x^i\rangle \geq 1 \\
\Leftrightarrow \quad & \max_{\{w':|w'\|=1\},\rho\in\mathbb{R}} \rho \quad \text{sb.t. } y^i\langle \frac{w'}{\rho}, x^i\rangle \geq 1 \\
\Leftrightarrow \quad & \max_{\{w':|w'\|=1\},\rho\in\mathbb{R}} \rho \quad \text{sb.t. } y^i\langle w', x^i\rangle \geq \rho \\
\Leftrightarrow \quad & \max_{\{w':|w'\|=1\},\rho\in\mathbb{R}} \rho \quad \text{sb.t. } |\langle w', x^i\rangle| \geq \rho \text{ and } \operatorname{sign}\langle w', x^i\rangle = y_i
\end{aligned}$$

Proof.

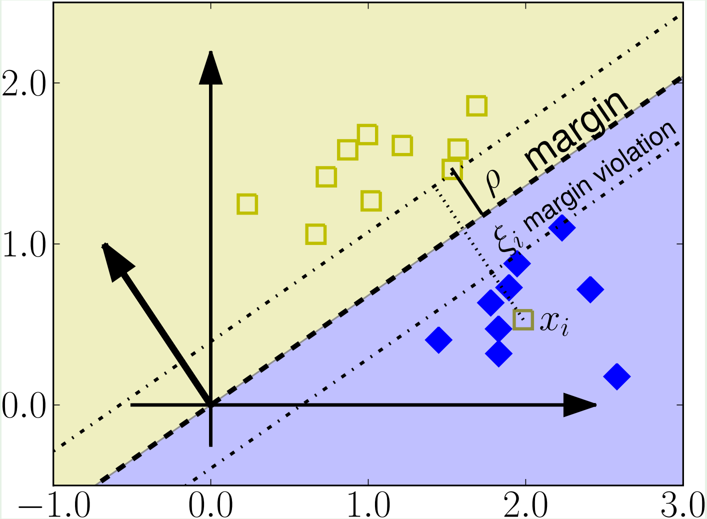
1. All w that fulfill the inequalities yield *correct* classifiers.
2. Since $\mathcal{D}$ is linearly separable, there exists some v with

$$\operatorname{sign}\langle v, x^i\rangle = y_i, \quad \text{i.e.} \quad y_i\langle v, x^i\rangle \geq \gamma > 0.$$

   for $\gamma = \mathbf{min}_i\, y_i\langle v, x^i\rangle$. So $\tilde{v} = v/\gamma$, fulfills the inequalities and we see that the constraint set is at least not empty.
3. Now we check (with $i = 1, \ldots, n$):

$$\begin{aligned}
&\min_{w\in\mathbb{R}^d} \frac{1}{2}\|w\|^2 \text{ sb.t. } y^i\langle w, x^i\rangle \geq 1 \\
\Leftrightarrow\quad &\max_{w\in\mathbb{R}^d} \frac{1}{\|w\|} \quad \text{sb.t. } y^i\langle w, x^i\rangle \geq 1 \\
\Leftrightarrow\quad &\max_{\{w':|w'\|=1\},\rho\in\mathbb{R}} \rho \quad \text{sb.t. } y^i\langle \frac{w'}{\rho}, x^i\rangle \geq 1 \\
\Leftrightarrow\quad &\max_{\{w':|w'\|=1\},\rho\in\mathbb{R}} \rho \quad \text{sb.t. } y^i\langle w', x^i\rangle \geq \rho \\
\Leftrightarrow\quad &\underbrace{\max_{\{w':|w'\|=1\},\rho\in\mathbb{R}} \rho \quad \text{sb.t. } |\langle w', x^i\rangle| \geq \rho}_{\text{maximal robustness}} \text{ and } \underbrace{\operatorname{sign}\langle w', x^i\rangle = y_i}_{\text{and correct}}
\end{aligned}$$

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 $g(x) = \operatorname{sign}\langle w^*, x\rangle + b)$ where (w^*, b^*) are the solution to

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

subject to

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

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

The variables $\xi_1, \ldots, \xi_n$ are called *slack* variables.

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.

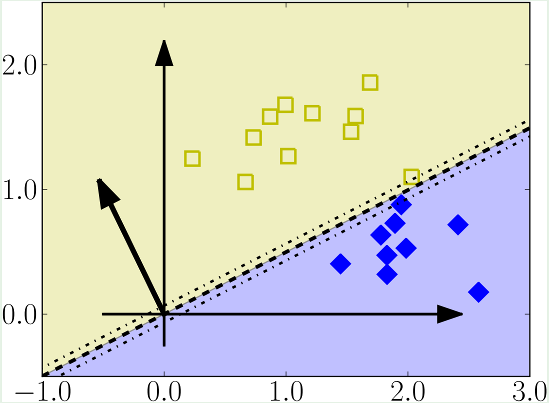
It balances the wishes for robustness and for correctness

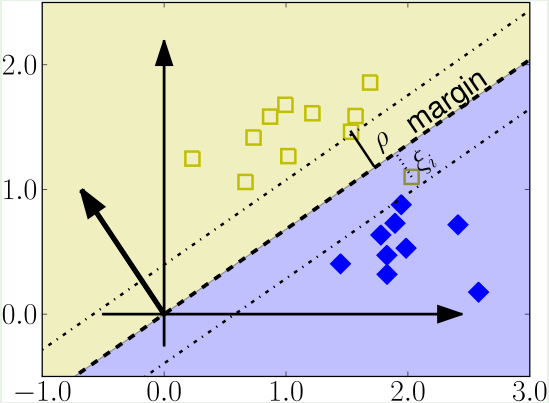
- $C \to 0$: mistakes don't matter much, emphasis on short w
- $C \to \infty$: as few errors as possible, might not be robust

We'll see more about this in the next lecture.

Remark

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





Left: small margin, no errors

Right: large margin, but 1 error

Maximum Soft-Margin Classifier

Lemma

Let $\mathcal{D}$ be a training set, not necessarily linearly separable. Let $C > 0$. Then the maximum soft-margin classifier (=linear SVM) can also be computed as

$$\min_{w\in\mathbb{R}^d, b\in\mathbb{R}} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \mathbf{max}\{0, 1 - y^i(\langle w, x^i\rangle + b)\}$$

Proof: the original optimization problem is

$$\min_{w,b,\xi} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi^i \quad \text{sb.t.} \quad y^i(\langle w, x^i\rangle + b) \geq 1 - \xi^i, \quad \xi^i \geq 0, \quad \text{for } i = 1,\dots,n.$$

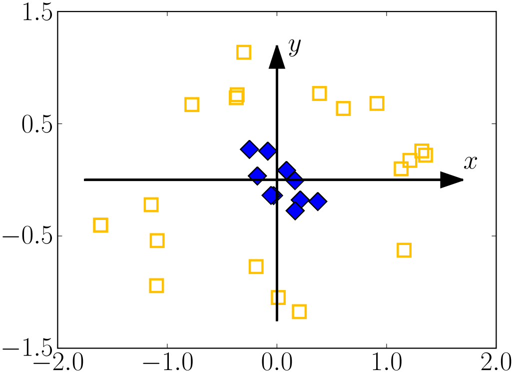
We can determine the optimal values of ξ_i for $i = 1,\dots,n$:

- they should be bigger or equal to 0 and to $1 - y^i(\langle w, x^i\rangle + b)$ (from the constraints)
- they should be as small as possible (because of the objective)
- in combination, we obtain $\quad \xi_i^{\text{opt}} = \mathbf{max}\{0, 1 - y^i(\langle w, x^i\rangle + b)\}$

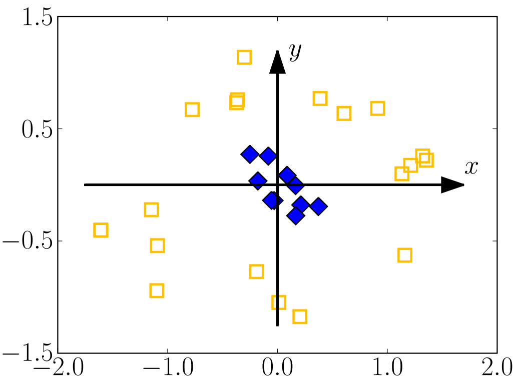
Pluggin this into the optimization yields the result.

Nonlinear Classifiers

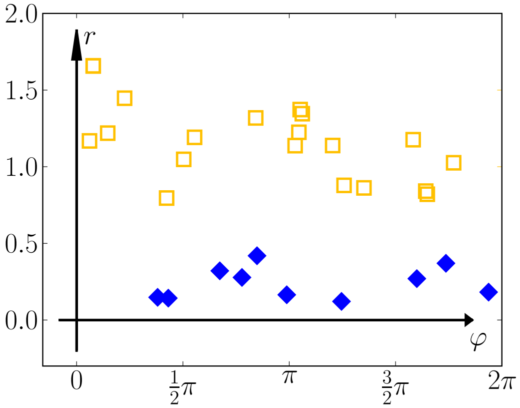
What, if a linear classifier is really not a good choice?



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



→



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

Definition (Max-margin Generalized Linear Classifier)

Let $C > 0$. Assume a training set

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

Let $\phi : \mathcal{X} \to \mathbb{R}^D$ be a feature map from $\mathcal{X}$ into a feature space $\mathbb{R}^D$.

Then we can form a new training set

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

The maximum-(soft)-margin linear classifier in $\mathbb{R}^D$,

$$g(x) = \operatorname{sign}[\langle w, \phi(x)\rangle_{\mathbb{R}^D} + b]$$

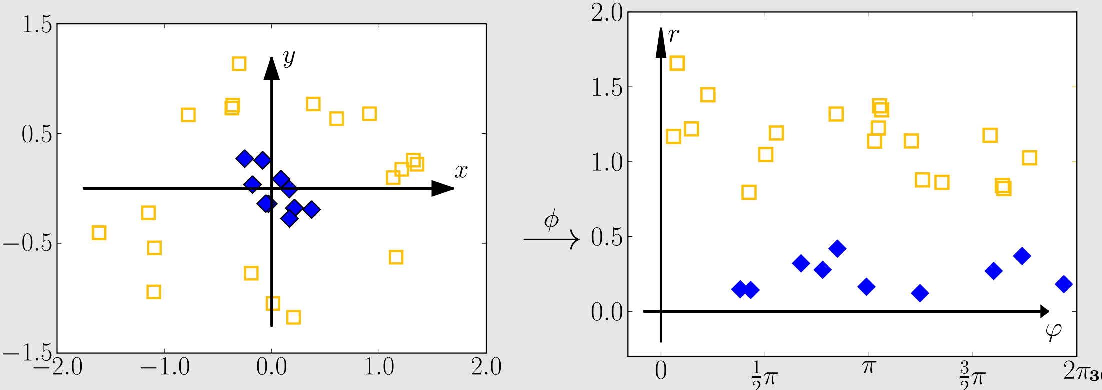
for $w \in \mathbb{R}^D$ and $b \in \mathbb{R}$ is called **max-margin generalized linear classifier**.

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 $\phi : \mathcal{X} \to \mathbb{R}^D$ with $\mathcal{X} = \mathbb{R}^2$ and $\mathbb{R}^D = \mathbb{R}^2$

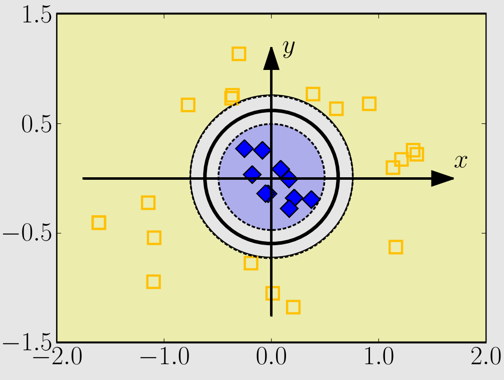
$$\phi(x,y) = \left(\sqrt{x^2+y^2}, \arctan\frac{y}{x}\right) \qquad \text{(and } \phi(0,0) = (0,0)\text{)}$$



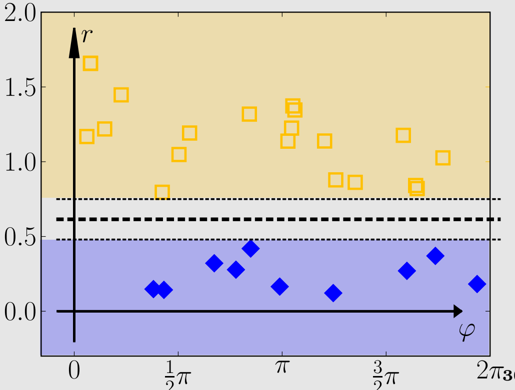
Example (Polar coordinates)

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

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



$\xrightarrow{\phi}$



Example (d-th degree polynomials)

$$\phi : \left(x_1, \dots, x_n\right) \mapsto \left(1, x_1, \dots, x_n, x_1^2, \dots, x_n^2, x_1^2, x_1 x_2, \dots, x_n^2, \dots, x_n^d\right)$$

Resulting classifier: d-th degree polynomial in x. $g(x) = \operatorname{sign} f(x)$ with

$$f(x) = \langle w, \phi(x) \rangle = \sum_j w_j \phi(x)_j = a + \sum_i b_i x_i + \sum_{ij} c_{ij} x_i x_j + \dots$$

Example (Distance map)

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

$$\phi : \vec{x} \mapsto \left(e^{-\|\vec{x} - \vec{p}_1\|^2}, \dots, e^{-\|\vec{x} - \vec{p}_N\|^2}\right)$$

Classifier: combine weights from close enough prototypes

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

Example (Pre-trained deep network)

The internet is full of already trained (deep) neural networks that one can download, e.g. trained on ImageNet for image classification.

Idea: use initial segment of network as feature extractor for other data:

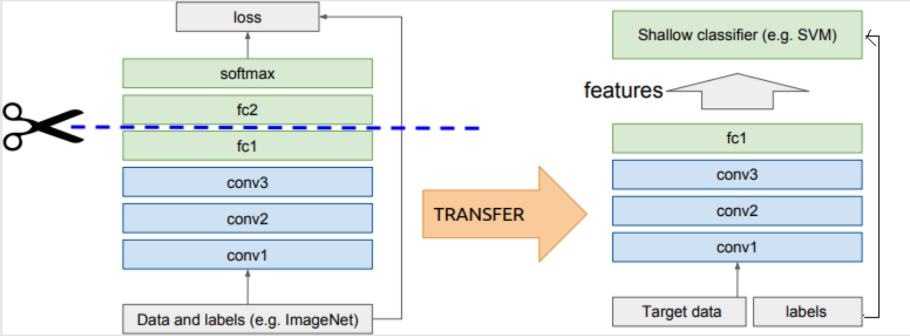


Image: Steven Schmatz,
https://www.quora.com/What-is-the-difference-between-transfer-learning-domain-adaptation-and-multitask-learning-in-machine-learning

Beyond Vectors as Inputs

Beyond Vectors as Inputs

Linear models, such as

$$f(x) = \langle w, x \rangle + b$$

only makes sense if data $x \in \mathcal{X}$ are vectors of equal dimension, $x \in \mathbb{R}^d$.

Real data

- can be categorical,
- can be structured,
- can be of variable size.

Beyond Vectors as Inputs

Linear models, such as

$$f(x) = \langle w, x \rangle + b$$

only makes sense if data $x \in \mathcal{X}$ are vectors of equal dimension, $x \in \mathbb{R}^d$.

Real data

- can be categorical,
- can be structured,
- can be of variable size.

Generalized linear models,

$$f(x) = \langle w, \phi(x) \rangle + b$$

can make sense for other input sets $\mathcal{X}$, if we define a suitable feature map $\phi : \mathcal{X} \to \mathcal{F}$.

Categorical data

$$\mathcal{X} = \{\texttt{red}, \texttt{green}, \texttt{blue}\}$$

"One-hot encoding": encode by vector of binary indicator variables, $\phi : \mathcal{X} \to \mathbb{R}^{|\mathcal{X}|}$,

- $\phi(\texttt{red}) = (1, 0, 0)$, $\quad \phi(\texttt{green}) = (0, 1, 0)$, $\quad \phi(\texttt{blue}) = (0, 0, 1)$

$$\mathcal{X} = \{\texttt{red}, \texttt{green}, \texttt{blue}\}$$

"One-hot encoding": encode by vector of binary indicator variables, $\phi : \mathcal{X} \to \mathbb{R}^{|\mathcal{X}|}$,

- $\phi(\texttt{red}) = (1, 0, 0)$, $\quad \phi(\texttt{green}) = (0, 1, 0)$, $\quad \phi(\texttt{blue}) = (0, 0, 1)$

Caveat

Don't use: $\quad \texttt{red} \mapsto 1 \quad \texttt{green} \mapsto 2 \quad \texttt{blue} \mapsto 3$

That would introduce spurious relations, such as

$$\texttt{green} + \texttt{red} = \texttt{blue} \qquad ?!?$$

One-hot encoding works well even for large $\mathcal{X}$, e.g. all English words, when using the right data structures (e.g. sparse vectors/matrices).

Ordinal data

$$\mathcal{X} = \{\texttt{poor}, \texttt{fair}, \texttt{good}, \texttt{very good}, \texttt{excellent}\}$$

Best treatment depends on the situation

- working with distances?

$$\phi(\texttt{poor}) = 1 \qquad \phi(\texttt{fair}) = 2 \qquad \ldots \qquad \phi(\texttt{excellent}) = 5$$

  might work well.

- in other situations, one-hot might work better.
- if values derive from a continuous quantity by quantization
  - $\leq 60\%$: `poor`   61–70%: `good` $\ldots$ $\geq$ 91-100%: `excellent`

  it might make sense to reflect those

$$\phi(\texttt{poor}) = 0.55 \qquad \phi(\texttt{fair}) = 0.65 \qquad \ldots \qquad \phi(\texttt{excellent}) = 0.95$$

Example: $\mathcal{X} = \{$ all English words $\}$, task-specific encoding: "word vectors"

- represent each word w by a vector $\phi(w) \in \mathbb{R}^d$ (e.g. $25 \leq d \leq 300$)
- similar vectors encode words of similar meaning (more or less)

`tiger`	-0.70	-0.34	0.44	-0.38	-0.55	0.29	0.79	0.01	0.56	...
`lion`	-0.89	-0.56	-0.37	0.76	-0.78	0.56	0.80	-0.05	0.80	...
`pion`	-0.53	-0.62	-0.13	0.55	-0.55	-0.43	-1.12	-0.39	0.67	...
`quark`	-0.53	-0.55	0.17	-0.67	-0.51	-0.32	-0.90	-1.41	0.74	...

- $\phi(\texttt{tiger}) \approx \phi(\texttt{lion})$ $\phi(\texttt{pion}) \not\approx \phi(\texttt{lion})$, etc.

Euclidean distances, $\|\phi(w_i) - \phi(w_j)\|$:

	`tiger`	`lion`	`pion`	`quark`
`tiger`	0	2.6	4.6	4.0
`lion`	2.6	0	4.3	4.6
`pion`	4.6	4.3	0	2.8
`quark`	4.0	4.6	2.8	0

Vectors that have been learned automatically (unsupervised) from large corpora (e.g. Wikipedia) are available for download, e.g. `https://github.com/3Top/word2vec-api#where-to-get-a-pretrained-models`

Variable size data: text and strings

Given: a text fragment or short sentence $W =$ "$w_1\ w_2 \ldots\ w_k$".

Easiest option: average individual representations

$$\Phi(W) = \frac{1}{k}\sum_{i=1}^{k}\phi(w_i)$$

for a word representation ϕ.

- linear function of Φ is average of linear functions on ϕ:

$$w^\top\Phi(W) = w^\top\Big(\frac{1}{k}\sum_i\phi(w_i)\Big) = \frac{1}{k}\sum_i w^\top\phi(w_i)$$

- advantage: very simple
- disadvantage: mixes words together, not really suitable for long texts

Example: $\mathcal{X} = \{$ arbitrary lengths text documents $\}$

Task-specific encoding, $x \mapsto \phi(x)$, e.g.,

- create a dictionary of all possible words, $w_1, \dots, w_L$
- represent x by histogram of word occurrences

$$x \mapsto (h_1, \dots, h_L) \in \mathbb{R}^L$$ **"bag-of-words"** representation

where h_i counts how often word w_i occurs in x (absolute or relative)

Example: $\mathcal{X} = \{$ arbitrary lengths text documents $\}$

Task-specific encoding, $x \mapsto \phi(x)$, e.g.,

- create a dictionary of all possible words, $w_1, \dots, w_L$
- represent x by histogram of word occurrences

$$x \mapsto (h_1, \dots, h_L) \in \mathbb{R}^L$$ **"bag-of-words"** representation

where h_i counts how often word w_i occurs in x (absolute or relative)

Include domain-knowledge if possible, e.g. stop-words

- ignore words a priori known not to be useful for the task at hand:

```
a an as at be ... the ... you
```

Variable size data: text and strings

Given: a set $D = \{d_1, d_2, \ldots, d_N\}$ of variable length documents.

tf-idf: term frequency – inverse document frequency

$$\text{tfidf}(t,d) = \text{tf}(t,d) \cdot \text{idf}(t)$$

- **term frequency** $\text{tf}(t,d)$: how frequent is term t in document d?

$$\text{tf}(t,d) = \text{ raw count of how often } t \text{ occurs in } d$$

- **inverse document frequency** $\text{idf}(t)$: in how many documents does the term occur?

$$\text{idf}(t,d) = \log \frac{N}{1+n_t} \quad \text{for } n_t = |\{d \in D : t \in d\}| \text{ and } N = |D|.$$

Many variants: normalization, boolean or logarithmic tf, constant idf (unweighted), ...

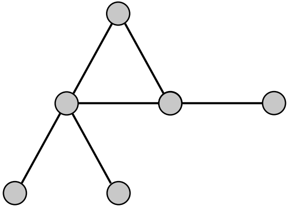
Variable size data: text and strings

More powerful: count not just terms but short fragments: n-**grams**

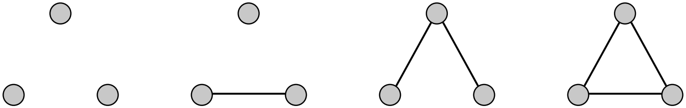
- $x_i =$ `CTCCTGACTTTCCTCGCTTGGTGGTTTGAGTGGACCTCCCAGGCCAGTGCCGGGCCCCTCATAGGAGAGG`
- count `A`,`C`,`G`,`T`: $\phi_1(x_i) = (9, 22, 22, 17) \in \mathbb{R}^4$
- count `AA`,`AC`,...,`TT`: $\phi_2(x_i) = (0, 2, 6, 1, 3, \dots, 4, 1, 5, 6, 3) \in \mathbb{R}^{16}$
- count `AAA`,...,`TTT`: $\phi_3(x_i) = (0, 0, 0, 0, 0, 1, 0, 1, \dots, 1, 2, 2) \in \mathbb{R}^{64}$
- etc.

fun demo: `https://books.google.com/ngrams`

data: `http://storage.googleapis.com/books/ngrams/books/datasetsv2.html`

$x_i =$ 

Possible feature map: count characteristic patterns, e.g. subgraphs

...  ...

a) b) c) d)

$$\phi(x_i) = \Big(\dots, \underbrace{7}_{a)}, \underbrace{6}_{b)}, \underbrace{6}_{c)}, \underbrace{1}_{d)}, \dots\Big)$$

Many more in application-dependent literature.

From Binary to Multi-class Classification

Classification problems with M classes:

- Training samples $\{x^1, \dots, x^n\} \subset \mathcal{X}$,
- Training labels $\{y^1, \dots, y^n\} \subset \{1, \dots, M\}$,
- Task: learn a prediction function $f : \mathcal{X} \to \{1, \dots, M\}$.

Classification problems with M classes:

- Training samples $\{x^1, \dots, x^n\} \subset \mathcal{X}$,
- Training labels $\{y^1, \dots, y^n\} \subset \{1, \dots, M\}$,
- Task: learn a prediction function $f : \mathcal{X} \to \{1, \dots, M\}$.

One-versus-rest construction:

- train one binary classifier $g_c : \mathcal{X} \to \mathbb{R}$ for each class c:
  - all samples with class label c are positive examples
  - all other samples are negative examples
- classify by finding maximal response

$$f(x) = \operatorname*{\mathbf{argmax}}_{c=1,\dots,M} g_c(x)$$

Advantage: easy to implement, parallel, works well in practice

Disadvantage: with many classes, training sets become unbalanced.
no explicit *calibration* of scores between different g_c

Classification problems with M classes:

- Training samples $\{x^1, \dots, x^n\} \subset \mathcal{X}$,
- Training labels $\{y^1, \dots, y^n\} \subset \{1, \dots, M\}$,
- Task: learn a prediction function $f : \mathcal{X} \to \{1, \dots, M\}$.

Classification problems with M classes:

- Training samples $\{x^1, \dots, x^n\} \subset \mathcal{X}$,
- Training labels $\{y^1, \dots, y^n\} \subset \{1, \dots, M\}$,
- Task: learn a prediction function $f : \mathcal{X} \to \{1, \dots, M\}$.

All-versus-all construction:

- train one classifier, $g_{ij} : \mathcal{X} \to \mathbb{R}$, for each pair of classes $1 \leq i < j \leq M$, in total $\frac{m(m-1)}{2}$ prediction functions
- classify by voting

$$f(x) = \operatorname*{\mathbf{argmax}}_{m=1,\dots,M} \#\{i \in \{1, \dots, M\} : g_{m,i}(x) > 0\},$$

(writing $g_{j,i} = -g_{i,j}$ for $j > i$ and $g_{j,j} = 0$)

Advantage: small and balanced training problems, parallel, works well.

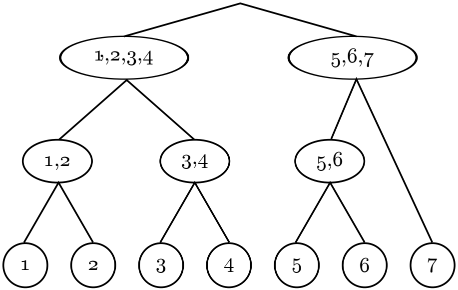
Disadvantage: number of classifiers grows quadratically in classes.

Classification problems with M classes:

- Training samples $\{x^1, \dots, x^n\} \subset \mathcal{X}$,
- Training labels $\{y^1, \dots, y^n\} \subset \{1, \dots, M\}$,
- Task: learn a prediction function $f : \mathcal{X} \to \{1, \dots, M\}$.

Hierarchical (tree) construction:

- construct binary tree with classes at leafs
- learn one classifier for each decision



Advantage: at most $\lceil \log_2 M \rceil$ classifier evaluation at test time

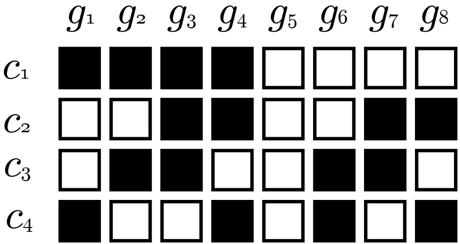
Disadvantage: not parallel, not robust to mistakes at any stage

Classification problems with M classes:

- Training samples $\{x^1, \dots, x^n\} \subset \mathcal{X}$,
- Training labels $\{y^1, \dots, y^n\} \subset \{1, \dots, M\}$,
- Task: learn a prediction function $f : \mathcal{X} \to \{1, \dots, M\}$.

Define a binary codeword for each class

- one classifier for codeword entry
- classify by comparing predictions to code words (exact or in some norm)



Advantage: parallel, trade off between speed and robustness

Disadvantage: optimal code design is NP-hard

Many different option for multi-class to binary reduction:

- One-versus-Rest
- One-versus-One
- Hierarchical (fixed or learned)
- Error-correcting output codes (ECOC)
- ...

Hot topic in the 2000s: which is the best one?

Many different option for multi-class to binary reduction:

- One-versus-Rest
- One-versus-One
- Hierarchical (fixed or learned)
- Error-correcting output codes (ECOC)
- ...

Hot topic in the 2000s: which is the best one?

Answer: None (or all of them)!

- there's dozens of studies, they all disagree
- use whatever is available, or best fits the target application
- to implement own yourself, One-versus-Rest is most popular, since it's the simplest