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

  $$c(x) := \arg\max_{y \in \mathcal{Y}} \hat{p}(x, y) \quad \text{or} \quad c_\ell(x) := \arg\min_{y \in \mathcal{Y}} \mathbb{E}_{\bar{y} \sim \hat{p}(x, \bar{y})} \ell(\bar{y}, y).$$

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

  $$g(x) := \arg\max_{y \in \mathcal{Y}} \hat{p}(y|x) \quad \text{or} \quad c_\ell(x) := \arg\min_{y \in \mathcal{Y}} \mathbb{E}_{\bar{y} \sim \hat{p}(\bar{y}|x)} \ell(\bar{y}, y).$$

- a **decision theoretic approach**: if we use $\mathcal{D}$ to directly search 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.
Let’s use $\mathcal{D}$ to estimate a classifier $c : \mathcal{X} \rightarrow \mathcal{Y}$ directly.
Let’s use $\mathcal{D}$ to estimate a classifier $c : \mathcal{X} \rightarrow \mathcal{Y}$ 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:

- 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
\[ g(x) = \text{sign } f(x) \]
for some \( f \in \mathcal{F} \).

Given a training set \( \mathcal{D} = \{(x^1, y^1), \ldots, (x^n, y^n)\} \sim p \), what’s the best \( f \) (and induced \( g \))? 
A linear classifier, $g(x) = \text{sign}(w, x)$, with $b = 0$
A linear classifier $g(x) = \text{sign}(\langle w, x \rangle + b)$, 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 $x = (x^1, \ldots, x^d) \in \mathbb{R}^d$, use $\tilde{x} = (x^1, \ldots, x^d, 1) \in \mathbb{R}^{d+1}$
- 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 $\mathbb{R}^{d+1}$:

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

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

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

Definition (Ad hoc)
We call a classifier, \( g \), correct (for a training set \( D \)), if it predicts the correct labels for all training examples:

\[
g(x^i) = y^i \quad \text{for } i = 1, \ldots, n.
\]

Example (Perceptron)
- if the Perceptron converges, the result is an correct classifier.
- any classifier with zero training error is correct.
Linear classifiers

**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:

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

**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
Linear Classifiers

**Definition (Ad hoc)**

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

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

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, $\rho$, of a linear classifier
Definition (Margin)

Let \( f(x) = \langle w, x \rangle + b \) define a correct linear classifier. The margin of \( f \) (with respect to \( D \)) 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 \( D \) to the decision boundary. We can compute the margin of a linear classifier \( f = \langle w, x \rangle + b \) as

\[
\rho = \min_{i=1,...,n} \left| \frac{w}{\|w\|}, x^i \right| + b.
\]

Proof.

High school maths: distance between a point and a hyperplane in Hessian normal form.
Margin, $\rho$, of a linear classifier
Theorem

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

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

Proof by Picture
Proof (blackboard). For any $i = 1, \ldots, n$ 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

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

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 $\epsilon$, i.e. $|f(x^i)| \geq \|w\|\|\epsilon\|$ for $i = 1, \ldots, n$.

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

$\square$
Theorem

Let $\mathcal{D}$ be a linearly separable training set. Then the most robust, correct linear classifier (without bias term) is given by $g(x) = \text{sign}(\langle w^*, x \rangle)$ 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 called **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
   \[
   \text{sign}\langle v, x^i \rangle = y_i, \text{ i.e. } y_i\langle v, x^i \rangle \geq \gamma > 0.
   \]
   for \( \gamma = \min_i y_i\langle v, x^i \rangle \). 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

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

for $\gamma = \min_i y_i \langle v, x^i \rangle$. 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 $i = 1, \ldots, n$):

$$\min_{w \in \mathbb{R}^d} \frac{1}{2} \|w\|^2 \quad \text{sb.t.} \quad y^i \langle w, x^i \rangle \geq 1$$

$$\Leftrightarrow \max_{w \in \mathbb{R}^d} \frac{1}{\|w\|} \quad \text{sb.t.} \quad y^i \langle w, x^i \rangle \geq 1$$

$$\Leftrightarrow \max_{\{w' : \|w'\|=1\}, \rho \in \mathbb{R}} \rho \quad \text{sb.t.} \quad y^i \langle \frac{w'}{\rho}, x^i \rangle \geq 1$$

$$\Leftrightarrow \max_{\{w' : \|w'\|=1\}, \rho \in \mathbb{R}} \rho \quad \text{sb.t.} \quad y^i \langle w', x^i \rangle \geq \rho$$

$$\Leftrightarrow \max_{\{w' : \|w'\|=1\}, \rho \in \mathbb{R}} \rho \quad \text{sb.t.} \quad |\langle w', x^i \rangle| \geq \rho \quad \text{and} \quad \text{sign} \langle w', x^i \rangle = y_i$$
Proof.

1. All $w$ that fulfill the inequalities yield correct classifiers.

2. Since $\mathcal{D}$ is linearly separable, there exists some $v$ with

$$\text{sign} \langle v, x^i \rangle = y_i,$$

i.e. $y_i \langle v, x^i \rangle \geq \gamma > 0.$

for $\gamma = \min_i y_i \langle v, x^i \rangle$. 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 $i = 1, \ldots, n$):

$$\min_{w \in \mathbb{R}^d} \frac{1}{2} \|w\|^2 \quad \text{s.t.} \quad y^i \langle w, x^i \rangle \geq 1$$

$$\Leftrightarrow \max_{w \in \mathbb{R}^d} \frac{1}{\|w\|} \quad \text{s.t.} \quad y^i \langle w, x^i \rangle \geq 1$$

$$\Leftrightarrow \max_{\{w' : \|w'\| = 1\}, \rho \in \mathbb{R}} \rho \quad \text{s.t.} \quad y^i \langle \frac{w'}{\rho}, x^i \rangle \geq 1$$

$$\Leftrightarrow \max_{\{w' : \|w'\| = 1\}, \rho \in \mathbb{R}} \rho \quad \text{s.t.} \quad y^i \langle w', x^i \rangle \geq \rho$$

$$\Leftrightarrow \max_{\{w' : \|w'\| = 1\}, \rho \in \mathbb{R}} \rho \quad \text{s.t.} \quad \| \langle w', x^i \rangle \| \geq \rho \quad \text{and} \quad \text{sign} \langle w', x^i \rangle = y_i$$

maximal robustness

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

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

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

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

**Proof:** the original optimization problem is

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

We can determine the optimal values of $\xi^i$ for $i = 1, \ldots, n$:

- 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 $\xi_{i}^{\text{opt}} = \max\{0, 1 - y^i(\langle w, x^i \rangle + b)\}$

Pluggin this into the optimization yields the result.
Nonlinear Classifiers
Nonlinear Classifiers

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

![Graph showing data points in two dimensions with a linear classifier decision boundary]
Nonlinear Classifiers

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

Change the data representation, e.g. Cartesian $\rightarrow$ polar coordinates
Definition (Max-margin Generalized Linear Classifier)

Let $C > 0$. Assume a training set

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

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

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

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

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

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 $\mathbb{R}^D = \mathbb{R}^2$

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

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

$$\phi(x, y) = (\sqrt{x^2 + y^2}, \arctan \frac{y}{x}) \quad \text{(and } \phi(0, 0) = (0, 0))$$
Other popular feature mappings, $\phi$

### Example ($d$-th degree polynomials)

$\phi : (x_1, \ldots, x_n) \mapsto (1, x_1, \ldots, x_n, x_1^2, \ldots, x_n^2, x_1^2, x_1 x_2, \ldots, x_n^2, \ldots, x_n^d)$

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

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

### Example (Distance map)

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

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

Classifier: combine weights from close enough prototypes

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

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:

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 \( \phi : \mathcal{X} \rightarrow \mathcal{F} \).
\[ \mathcal{X} = \{ \text{red}, \text{green}, \text{blue} \} \]

"One-hot encoding": encode by vector of binary indicator variables, \( \phi : \mathcal{X} \rightarrow \mathbb{R}^{\mid \mathcal{X} \mid} \),

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

Caveat

Don't use:

- \( \text{red} \mapsto 1 \)
- \( \text{green} \mapsto 2 \)
- \( \text{blue} \mapsto 3 \)

That would introduce spurious relations, such as \( \text{green} + \text{red} = \text{blue} \)?!?

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).
Categorical data

\[ \mathcal{X} = \{ \text{red}, \text{green}, \text{blue} \} \]

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

- \( \phi(\text{red}) = (1, 0, 0) \),  \( \phi(\text{green}) = (0, 1, 0) \),  \( \phi(\text{blue}) = (0, 0, 1) \)

Caveat

Don't use:  \( \text{red} \mapsto 1 \)  \( \text{green} \mapsto 2 \)  \( \text{blue} \mapsto 3 \)

That would introduce spurious relations, such as

\[ \text{green} + \text{red} = \text{blue} \]  ??

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**

\[ \mathcal{X} = \{\text{poor, fair, good, very good, excellent}\} \]

Best treatment depends on the situation

- working with distances?
  \[ \phi(\text{poor}) = 1 \quad \phi(\text{fair}) = 2 \quad \ldots \quad \phi(\text{excellent}) = 5 \]
  might work well.

- in other situations, one-hot might work better.

- if values derive from a continuous quantity by quantization
  - \( \leq 60\%: \text{poor} \quad 61-70\%: \text{good} \quad \ldots \quad \geq 91-100\%: \text{excellent} \)
  it might make sense to reflect those
  \[ \phi(\text{poor}) = 0.55 \quad \phi(\text{fair}) = 0.65 \quad \ldots \quad \phi(\text{excellent}) = 0.95 \]
Example: $\mathcal{X} = \{\text{all English words}\}$, 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)

<table>
<thead>
<tr>
<th></th>
<th>tiger</th>
<th>lion</th>
<th>pion</th>
<th>quark</th>
</tr>
</thead>
<tbody>
<tr>
<td>tiger</td>
<td>-0.70</td>
<td>-0.34</td>
<td>0.44</td>
<td>-0.38</td>
</tr>
<tr>
<td>lion</td>
<td>-0.89</td>
<td>-0.56</td>
<td>-0.37</td>
<td>0.76</td>
</tr>
<tr>
<td>pion</td>
<td>-0.53</td>
<td>-0.62</td>
<td>-0.13</td>
<td>0.55</td>
</tr>
<tr>
<td>quark</td>
<td>-0.53</td>
<td>-0.55</td>
<td>0.17</td>
<td>-0.67</td>
</tr>
</tbody>
</table>

$\phi(\text{tiger}) \approx \phi(\text{lion}) \quad \phi(\text{pion}) \not\approx \phi(\text{lion})$, etc.

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

<table>
<thead>
<tr>
<th></th>
<th>tiger</th>
<th>lion</th>
<th>pion</th>
<th>quark</th>
</tr>
</thead>
<tbody>
<tr>
<td>tiger</td>
<td>0</td>
<td>2.6</td>
<td>4.6</td>
<td>4.0</td>
</tr>
<tr>
<td>lion</td>
<td>2.6</td>
<td>0</td>
<td>4.3</td>
<td>4.6</td>
</tr>
<tr>
<td>pion</td>
<td>4.6</td>
<td>4.3</td>
<td>0</td>
<td>2.8</td>
</tr>
<tr>
<td>quark</td>
<td>4.0</td>
<td>4.6</td>
<td>2.8</td>
<td>0</td>
</tr>
</tbody>
</table>

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](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 $\phi$.

- linear function of $\Phi$ is average of linear functions on $\phi$:

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

- advantage: very simple
- disadvantage: mixes words together, not really suitable for long texts
Variable size data: text and strings

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

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

- create a dictionary of all possible words, \( w_1, \ldots, w_L \)
- represent \( x \) by histogram of word occurrences

\[
x \mapsto (h_1, \ldots, h_L) \in \mathbb{R}^L \quad \text{"bag-of-words" representation}
\]

where \( h_i \) counts how often word \( w_i \) occurs in \( x \) (absolute or relative)
Variable size data: text and strings

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

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

- create a dictionary of all possible words, \( w_1, \ldots, w_L \)
- represent \( x \) by histogram of word occurrences

\[
x \mapsto (h_1, \ldots, h_L) \in \mathbb{R}^L \quad \text{"bag-of-words" representation}
\]

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**

\[
\text{tfidf}(t, d) = \text{tf}(t, d) \times \text{idf}(t)
\]

- **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), \ldots
More powerful: count not just terms but short fragments: \textit{n-grams}

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

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

data: http://storage.googleapis.com/books/ngrams/books/datasetsv2.html
Variable size data: graphs

$$x_i =$$

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

\[
\phi(x_i) = \left( \ldots, 7_{\text{a)}, 6_{\text{b), 6_{\text{c), 1}_{\text{d),}} \ldots} \right)
\]

Many more in application-dependent literature.
From Binary to Multi-class Classification
Classification problems with $M$ classes:

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

One-versus-rest construction:

- train one binary classifier $g_c : \mathcal{X} \rightarrow \mathbb{R}$ for each class $c$:
  - all samples with class label $c$ are positive examples
  - all other samples are negative examples
- classify by finding maximal response $f(x) = \text{argmax}_{c=1, \ldots, M} g_c(x)$

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

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

$$f(x) = \arg\max_{c=1,\ldots,M} g_c(x)$$

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

**Disadvantage**: with many classes, training sets become unbalanced. no explicit calibration of scores between different $g_c$
Multiclass Classification — All-versus-all reduction

Classification problems with $M$ classes:

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

All-versus-all construction:

- train one classifier, $g_{ij} : \mathcal{X} \rightarrow \mathbb{R}$, for each pair of classes $1 \leq i < j \leq M$, in total $m(m-1)/2$ prediction functions
- classify by voting $f(x) = \text{argmax}_{m=1,\ldots,M} \# \{i \in \{1, \ldots, M\} : g_{m,i}(x) > 0\}$,

(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.
Multiclass Classification – All-versus-all reduction

Classification problems with $M$ classes:

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

All-versus-all construction:

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

$$f(x) = \arg\max_{m=1,\ldots,M} \#\{i \in \{1, \ldots, M\} : g_{m,i}(x) > 0\},$$

(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.
Multiclass Classification – Hierarchical

Classification problems with $M$ classes:
- Training samples $\{x^1, \ldots, x^n\} \subset \mathcal{X}$,
- Training labels $\{y^1, \ldots, y^n\} \subset \{1, \ldots, M\}$,
- Task: learn a prediction function $f : \mathcal{X} \rightarrow \{1, \ldots, M\}$.

Hierarchical (tree) construction:
- construct binary tree with classes at leaves
- 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 $\{x^1, \ldots, x^n\} \subset \mathcal{X}$,
- Training labels $\{y^1, \ldots, y^n\} \subset \{1, \ldots, M\}$,
- Task: learn a prediction function $f : \mathcal{X} \rightarrow \{1, \ldots, M\}$.

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 options 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 options 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 your own, One-versus-Rest is most popular, since it’s the simplest