Constructing Kernels

Checking if a given function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a kernel can be hard.

- $k(x, \bar{x}) = \tanh(1 + \langle x, \bar{x} \rangle)$?
- $k(x, \bar{x}) = \exp(-\text{edit distance between two strings } x \text{ and } \bar{x})$?
- $k(x, \bar{x}) = 1 - \|x - \bar{x}\|^2$?

Easier: construct functions that are guaranteed to be kernels:

Construct explicitly:

- any $\phi : \mathcal{X} \rightarrow \mathbb{R}^m$ induces a kernel $k(x, \bar{x}) = \langle \phi(x), \phi(\bar{x}) \rangle$.
  in particular any $f : \mathcal{X} \rightarrow \mathbb{R}$, $k(x, \bar{x}) = f(x)f(\bar{x})$

Construction from other kernels:

- If $k$ is a kernel and $\alpha \in \mathbb{R}^+$, then $k + \alpha$ and $\alpha k$ are kernels.
- if $k_1, k_2$ are kernels, then $k_1 + k_2$ and $k_1 \cdot k_2$ are kernels.
- if $k$ is a kernel, then $\exp(k)$ is a kernel.
Optimizing the SVM Dual (kernelized)

How to solve the QP

\[
\begin{align*}
\max_{\alpha_1, \ldots, \alpha_n \in \mathbb{R}} & \quad -\frac{1}{2} \sum_{i,j=1}^{n} \alpha^i \alpha^j y^i y^j k(x^i, x^j) + \sum_{i=1}^{n} \alpha^i \\
\text{subject to} & \quad \sum_i \alpha_i y_i = 0 \quad \text{and} \quad 0 \leq \alpha_i \leq C, \text{ for } i = 1, \ldots, n.
\end{align*}
\]

Observations:

- Kernel matrix \( K \) (with entries \( k_{ij} = k(x^i, x^j) \)) might be too big to fit into memory.

- In the optimum, many of the \( \alpha_i \) are 0 and do not contribute. If we knew which ones, we would save a lot of work.
Optimizing the SVM Dual (kernelized)

Working set training [Osuna 1997]

1: $S = \emptyset$
2: repeat
3: $\alpha \leftarrow$ solve QP with variables $\alpha_i$ for $i \in S$ and $\alpha_i = 0$ for $i \not\in S$
4: for $i = 1 \ldots, n$ do
5: if $i \in S$ and $\alpha_i = 0$ then remove $i$ from $S$
6: if $i \not\in S$ and $\alpha_i$ not optimal then add $i$ to $S$
7: end for
8: until convergence

Advantages:
- objective value increases monotonously
- converges to global optimum

Disadvantages:
- each step is computationally costly, since $S$ can become large
Sequential Minimal Optimization (SMO) [Platt 1998]

1: \( \alpha \leftarrow 0 \)
2: \textbf{repeat}
3: \hspace{1em} pick index \( i \) such that \( \alpha_i \) is not optimal
4: \hspace{1em} pick index \( j \neq i \) arbitrarily (usually based on some heuristic)
5: \hspace{1em} \( \alpha_i, \alpha_j \leftarrow \text{solve QP for } \alpha_i, \alpha_j \text{ and all other } \alpha_k \text{ fixed} \)
6: \textbf{until} convergence

Advantages:

- convergences monotonously to global optimum
- each step optimizes a subproblem of smallest possible size: 2 unknowns (1 doesn’t work because of constraint \( \sum_i \alpha_i y_i = 0 \))
- subproblems have a closed-form solution

Disadvantages:

- many iterations are required
- many kernel values \( k(x^i, x^j) \) are computed more than once (unless \( K \) is stored as a matrix)
SVMs Without Bias Term

For optimization, the bias term is an annoyance
- In primal optimization, it often requires a different stepsize.
- In dual optimization, it is not straight-forward to recover.
- It couples the dual variables by an equality constraint: $\sum_i \alpha_i y_i = 0$.

We can get rid of the bias by the augmentation trick.

Original:
- $f(x) = \langle w, x \rangle_{\mathbb{R}^d} + b$, with $w \in \mathbb{R}^d$, $b \in \mathbb{R}$.

New augmented:
- linear: $f(x) = \langle \tilde{w}, \tilde{x} \rangle_{\mathbb{R}^{d+1}}$, with $\tilde{w} = (w, b)$, $\tilde{x} = (x, 1)$.
- generalized: $f(x) = \langle \tilde{w}, \tilde{\phi}(x) \rangle_{\tilde{H}}$ with $\tilde{w} = (w, b)$, $\tilde{\phi}(x) = (\phi(x), 1)$.
- kernelize: $\tilde{k}(x, \bar{x}) = \langle \tilde{\phi}(x), \tilde{\phi}(\bar{x}) \rangle_{\tilde{H}} = k(x, \bar{x}) + 1$. 
SVMs Without Bias Term – Optimization

**SVM without bias term – primal optimization problem**

\[
\min_{w \in \mathbb{R}^d, \xi \in \mathbb{R}^n} \frac{1}{2} \| w \|^2 + C \sum_{i=1}^{n} \xi^i \\
\text{subject to, for } i = 1, \ldots, n,
\]

\[y^i \langle w, x^i \rangle \geq 1 - \xi^i, \quad \text{and} \quad \xi^i \geq 0.
\]

Difference: no \( b \) variable to optimize over
**SVMs Without Bias Term – Optimization**

**SVM without bias term – primal optimization problem**

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

subject to, for \( i = 1, \ldots, n, \)

\[y^i \langle w, x^i \rangle \geq 1 - \xi^i, \quad \text{and} \quad \xi^i \geq 0.\]

Difference: no \( b \) variable to optimize over

**SVM without bias term – dual optimization problem**

\[
\max_{\alpha} \quad -\frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y^i y^j k(x^i, x^j) + \sum_{i} \alpha_i
\]

subject to, \( 0 \leq \alpha_i \leq C, \) for \( i = 1, \ldots, n. \)

Difference: no constraint \( \sum_i y_i \alpha_i = 0. \)
Linear SVM Optimization in the Dual

Stochastic Coordinate Dual Ascent

\[ \alpha \leftarrow 0. \]
\[ \text{for } t = 1, \ldots, T \text{ do} \]
\[ \quad i \leftarrow \text{random index (uniformly random or in epochs)} \]
\[ \quad \text{solve QP w.r.t. } \alpha_i \text{ with all } \alpha_j \text{ for } j \neq i \text{ fixed.} \]
\[ \text{end for} \]
\[ \text{return } \alpha \]

Properties:

- converges monotonically to global optimum
- each subproblem has smallest possible size

Open Problem:

- how to make each step efficient?
What’s the complexity of the update step? Derive an explicit expression:

Original problem: $\max_{\alpha \in [0, C]^n} -\frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j k(x^i, x^j) + \sum_i \alpha_i$
What’s the complexity of the update step? Derive an explicit expression:

Original problem: \( \max_{\alpha \in [0, C]^n} \ - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j k(x^i, x^j) + \sum_i \alpha_i \)

When all \( \alpha_j \) except \( \alpha_i \) are fixed: \( \max_{\alpha_i \in [0, C]} F(\alpha_i) \), with

\[
F(\alpha_i) = - \frac{1}{2} \alpha_i^2 k(x^i, x^i) + \alpha_i \left( 1 - y^i \sum_{j \neq i} \alpha_j y^j k(x^i, x^j) \right) + \text{const.}
\]

\[
\frac{\partial}{\partial \alpha_i} F(\alpha_i) = - \alpha_i k(x^i, x^i) + \left( 1 - y^i \sum_{j \neq i} \alpha_j y^j k(x^i, x^j) \right) + \text{const.}
\]

\[
\alpha_i^{\text{opt}} = \alpha_i + \frac{1 - y^i \sum_{j=1}^{n} \alpha_j y^j k(x^i, x^j)}{k(x^i, x^i)}, \quad \alpha_i = \begin{cases} 0 & \text{if } \alpha_i^{\text{opt}} < 0, \\ C & \text{if } \alpha_i^{\text{opt}} > C, \\ \alpha_i^{\text{opt}} & \text{otherwise.} \end{cases}
\]

(except if \( k(x^i, x^i) = 0 \), but then \( k(x^i, x^j) = 0 \), so \( \alpha_i \) has no influence)

Observation: each update has complexity \( O(n) \).
Let $k(x, \bar{x}) = \langle \phi(x), \phi(\bar{x}) \rangle_{\mathbb{R}^d}$ for explicitly known $\phi : \mathcal{X} \rightarrow \mathbb{R}^d$.

$$\alpha_{i}^{\text{opt}} = \alpha_{i} + \frac{1 - y^{i} \sum \alpha_{j} y^{j} k(x^{i}, x^{j})}{k(x^{i}, x^{i})},$$

remember $w = \sum_{j} \alpha_{j} y_{j} \phi(x^{j})$

$$= \alpha_{i} + \frac{1 - y^{i} \langle w, \phi(x^{i}) \rangle}{\| \phi(x^{i}) \|^{2}},$$

- each update takes $O(d)$, independent of $n$
  - $\langle w, \phi(x^{i}) \rangle$ takes at most $O(d)$ for explicit $w \in \mathbb{R}^{d}, \phi(x^{i}) \in \mathbb{R}^{d}$
  - we must also take care that $w$ remains up to date (also at most $O(d)$)
### SCDA for (Generalized) Linear SVMs [Hsieh, 2008]

initialize $\alpha \leftarrow 0$, $w \leftarrow 0$

for $t = 1, \ldots, T$ do

   $i \leftarrow$ random index (uniformly random or in epochs)

   $\delta \leftarrow \frac{1 - y^i \langle w, \phi(x^i) \rangle}{\|\phi(x^i)\|^2}$

   $\alpha_i \leftarrow \begin{cases} 
   0, & \text{if } \alpha_i + \delta < 0, \\
   C, & \text{if } \alpha_i + \delta > C, \\
   \alpha_i + \delta, & \text{otherwise.} 
   \end{cases}$

   $w \leftarrow w + \delta y^i \phi(x^i)$

end for

return $\alpha$, $w$

### Properties:

- converges monotonically to global optimum
- complexity of each step is independent of $n$
- resembles stochastic gradient method, but **automatic step size**
Practical Interlude:
Doing Machine Learning Experiments
You’ve trained a new predictor, $g : \mathcal{X} \rightarrow \mathcal{Y}$, and you want to tell the world how good it is. How to measure this?

**Reminder:**

- The average loss on the training set, $\frac{1}{|D_{trn}|} \sum_{(x,y) \in D_{trn}} \ell(y, g(x))$ tells us (almost) nothing about the future loss. Reporting it would be misleading as best.
- The relevant quantity is the expected risk,

  $$\mathcal{R}(g) = \mathbb{E}_{(x,y) \sim p(x,y)} \ell(y, g(x))$$

  which unfortunately we cannot compute, since $p(x, y)$ is unknown.
- If we have data $D_{tst} \overset{i.i.d.}{\sim} p(x, y)$, we have,

  $$\frac{1}{|D_{tst}|} \sum_{(x,y) \in D_{tst}} \ell(y, g(x)) \xrightarrow{|D_{tst}| \rightarrow \infty} \mathbb{E}_{(x,y) \sim p(x,y)} \ell(y, g(x))$$

  - Problem: samples $\ell(y, g(x))$ must me independent, otherwise law of large numbers doesn’t hold.
  - Make sure that $g$ is independent of $D_{tst}$. 
Classifier Training (idealized)

**input**  training data $D_{trn}$

**input**  learning procedure $A$

$$g \leftarrow A[D] \quad \text{(apply $A$ with $D$ as training set)}$$

**output**  resulting classifier $g : \mathcal{X} \to \mathcal{Y}$

Classifier Evaluation

**input**  trained classifier $g : \mathcal{X} \to \mathcal{Y}$

**input**  test data $D_{tst}$

apply $g$ to $D_{tst}$ and measure performance $R_{tst}$

**output**  performance estimate $R_{tst}$

Remark: In commercial applications, this is realistic:
• given some training set one builds a single system,
• one deploys it to the customers,
• the customers use it on their own data, and complain if disappointed

In research, one typically has no customer, but only a fixed amount of data to work with, so one simulates the above protocol.
### Classifier Training (idealized)

**input** training data $D_{trn}$  
**input** learning procedure $A$  
$$g \leftarrow A[D]$$ (apply $A$ with $D$ as training set)  
**output** resulting classifier $g : \mathcal{X} \rightarrow \mathcal{Y}$

### Classifier Evaluation

**input** trained classifier $g : \mathcal{X} \rightarrow \mathcal{Y}$  
**input** test data $D_{tst}$  
apply $g$ to $D_{tst}$ and measure performance $R_{tst}$  
**output** performance estimate $R_{tst}$

**Remark:** In commercial applications, this is realistic:
- given some training set one builds a single system,  
- one deploys it to the customers,  
- the customers use it on their own data, and complain if disappointed

In research, one typically has no customer, but only a fixed amount of data to work with, so one *simulates* the above protocol.
Classifier Training and Evaluation

\textbf{input} data $\mathcal{D}$

\textbf{input} learning method $A$

split $\mathcal{D} = \mathcal{D}_{trn} \cup \mathcal{D}_{tst}$ disjointly

set aside $\mathcal{D}_{tst}$ to a safe place \hfill // do not look at it

$g \leftarrow A[\mathcal{D}_{trn}]$ \hfill // learn a predictor from $\mathcal{D}_{trn}$

apply $g$ to $\mathcal{D}_{tst}$ and measure performance $R_{tst}$

\textbf{output} performance estimate $R_{tst}$
input  data $\mathcal{D}$

input  learning method $A$

split $\mathcal{D} = \mathcal{D}_{trn} \cup \mathcal{D}_{tst}$ disjointly

set aside $\mathcal{D}_{tst}$ to a safe place  

$g \leftarrow A[\mathcal{D}_{trn}]$  

apply $g$ to $\mathcal{D}_{tst}$ and measure performance $R_{tst}$

output  performance estimate $R_{tst}$

**Remark.** $\mathcal{D}_{tst}$ should be as small as possible, to keep $\mathcal{D}_{trn}$ as big as possible, but large enough to be convincing.

- sometimes: 50%/50% for small datasets
- more often: 80% training data, 20% test data
- for large datasets: 90% training, 10% test data.
Remark: The split because $D_{trn}$ and $D_{tst}$ must be absolute.

- Do not use $D_{tst}$ for anything except the very last step.

- Do not look at $D_{tst}$! Even if the learning algorithm doesn’t see it, you looking at it can and will influence your model design or parameter selection (human overfitting).

- In particular, this applies to datasets that come with predefined set of test data, such as MNIST, PASCAL VOC, ImageNet, etc.
Remark: The split because $D_{trn}$ and $D_{tst}$ must be absolute.

- Do not use $D_{tst}$ for anything except the very last step.
- Do not look at $D_{tst}$! Even if the learning algorithm doesn’t see it, you looking at it can and will influence your model design or parameter selection (human overfitting).
- In particular, this applies to datasets that come with predefined set of test data, such as MNIST, PASCAL VOC, ImageNet, etc.

In practice we often want more: not just evaluate one classifier, but

- select the best algorithm or parameters amongst multiple ones

We simulate the classifier evaluation step during the training procedure. This needs (at least) one additional data split:
Training and Selecting between Multiple Models

**input** data $D$

**input** set of method $A = \{A_1, \ldots, A_K\}$

split $D = D_{trnval} \cup D_{tst}$ disjointly

set aside $D_{tst}$ to a safe place (do not look at it)

split $D_{trnval} = D_{trn} \cup D_{val}$ disjointly

**for all** models $A_i \in A$ **do**

$g_i \leftarrow A_i[D_{trn}]$

apply $g_i$ to $D_{val}$ and measure performance $E_{val}(A_i)$

**end for**

pick best performing $A_i$

(optional) $g_i \leftarrow A_i[D_{trnval}]$ // retrain on larger dataset

apply $g_i$ to $D_{tst}$ and measure performance $R_{tst}$

**output** performance estimate $R_{tst}$

How to split? For example $1/3–1/3–1/3$ or $70\%–10\%–20\%$. 
Discussion.

- Each algorithm is trained on $\mathcal{D}_{trn}$ and evaluated on disjoint $\mathcal{D}_{val}$ ✓
- You select a predictor based on $E_{val}$ (its performance on $\mathcal{D}_{val}$), only afterwards $\mathcal{D}_{tst}$ is used. ✓
- $\mathcal{D}_{tst}$ is used to evaluate the final predictor and nothing else. ✓
Discussion.

• Each algorithm is trained on $\mathcal{D}_{trn}$ and evaluated on disjoint $\mathcal{D}_{val}$ ✓

• You select a predictor based on $E_{val}$ (its performance on $\mathcal{D}_{val}$), only afterwards $\mathcal{D}_{tst}$ is used. ✓

• $\mathcal{D}_{tst}$ is used to evaluate the final predictor and nothing else. ✓

Problems.

• small $\mathcal{D}_{val}$ is bad: $E_{val}$ could be bad estimate of $g_A$’s true performance, and we might pick a suboptimal method.

• large $\mathcal{D}_{val}$ is bad: $\mathcal{D}_{trn}$ is much smaller than $\mathcal{D}_{trnval}$, so the classifier learned on $\mathcal{D}_{trn}$ might be much worse than necessary.

• retraining the best model on $\mathcal{D}_{trnval}$ might overcome that, but it comes at a risk: just because a model worked well when trained on $\mathcal{D}_{trn}$, this does not mean it’ll also work well when trained on $\mathcal{D}_{trnval}$. 
Leave-one-out Evaluation (for a single model/algorithm)

**input**  algorithm \( A \)

**input**  loss function \( \ell \)

**input**  data \( \mathcal{D} \)  
(trnval part only: test part set aside earlier)

**for all** \((x^i, y^i) \in \mathcal{D}\) **do**

\[
g^{-i} \leftarrow A[\mathcal{D} \setminus \{(x^i, y^i)\}]  \quad \text{// } \mathcal{D}_\text{trn} \text{ is } \mathcal{D} \text{ with } i\text{-th example removed}
\]

\[
r^i \leftarrow \ell(y^i, g^{-i}(x^i))  \quad \text{// } \mathcal{D}_\text{val} = \{(x^i, y^i)\}, \text{ disjoint to } \mathcal{D}_\text{trn}
\]

**end for**

**output**  \( R_{\text{loo}} = \frac{1}{n} \sum_{i=1}^{n} r^i \)  
(average leave-one-out risk)

**Properties.**

- Each \( r^i \) is a unbiased (but noisy) estimate of the risk \( \mathcal{R}(g^{-i}) \)
- \( \mathcal{D} \setminus \{(x^i, y^i)\} \) is almost the same as \( \mathcal{D} \), so we can hope that each \( g^{-i} \) is almost the same as \( g = A[\mathcal{D}] \).
- Therefore, \( R_{\text{loo}} \) can be expected a good estimate of \( \mathcal{R}(g) \)

**Problem:** slow, trains \( n \) times on \( n - 1 \) examples instead of once on \( n \)
Compromise: use fixed number of small $D_{val}$

**$K$-fold Cross Validation (CV)**

**input** algorithm $A$, loss function $\ell$, data $D$ (trnval part)

split $D = \bigcup_{k=1}^{K} D_k$ into $K$ equal sized disjoint parts

for $k = 1, \ldots, K$ do

$g^{-k} \leftarrow A[D \setminus D_k]$

$r^{k} \leftarrow \frac{1}{|D_k|} \sum_{(x,y) \in D_k} \ell(y^i, g^{-k}(x))$

end for

**output** $R_{K-CV} = \frac{1}{K} \sum_{k=1}^{n} r^k$ (K-fold cross-validation risk)

**Observation.**

- for $K = |D|$ same as leave-one-out error.
- approximately $k$ times increase in runtime.
- most common: $k = 10$ or $k = 5$.

**Problem:** training sets overlap, so the error estimates are correlated.

Exception: $K = 2$
**5 × 2 Cross Validation (5 × 2-CV)**

**input** algorithm $A$, loss function $\ell$, data $D$ (trnval part)

for $k = 1, \ldots, 5$ do

Split $D = D_1 \cup D_2$

$g_1 \leftarrow A[D_1]$,

$r_{1}^{k} \leftarrow$ evaluate $g_1$ on $D_2$

$g_2 \leftarrow A[D_2]$,

$r_{2}^{k} \leftarrow$ evaluate $g_2$ on $D_1$

$r^{k} \leftarrow \frac{1}{2}(r_{1}^{k} + r_{2}^{k})$

end for

**output** $E_{5 \times 2} = \frac{1}{5} \sum_{k=1}^{5} r^{k}$

**Observation.**

- 5 × 2-CV is really the average of 5 runs of 2-fold CV
- very easy to implement: shuffle the data and split into halves
- within each run the training sets are disjoint and the classifiers $g_1$ and $g_2$ are independent

**Problem:** training sets are smaller than in 5- or 10-fold CV.
Classifiers for Information Retrieval Tasks

Some classification tasks are really rather retrieval tasks, e.g.

- database lookup: is an entry \( x \) relevant \((y = 1)\) or not \((y = -1)\)?

A typical property:

- prediction is performed on a fixed database
- we have access to all elements of the test set at the same time
- positives \((y = 1)\) are important, negative \((y = -1)\) are a nuisance
- we don’t need all decisions, a few correct positives is enough

For a classifier \( g(x) = \text{sign } f(x) \) with \( f(x) : \mathcal{X} \rightarrow \mathbb{R} \) (e.g., \( f(x) = \langle w, x \rangle \)), we interpret \( f(x) \) as its confidence.

To produce \( K \) positive we return the test samples of highest confidence.

Equivalently, we decide by \( g_\theta(x) = \text{sign}(f(x) - \theta) \), for the right \( \theta \).
Other Ways to Evaluate Classifiers

Retrieval quality is often measure in terms of precision and recall:

**Definition (Precision, Recall, F-Score)**

For $\mathcal{Y} = \{\pm 1\}$, let $g : \mathcal{X} \rightarrow \mathcal{Y}$ a decision function and $D = \{(x^1, y^1), \ldots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$ be a database.

Then we define

$$\text{precision}(g) = \frac{\text{number of test samples with } g(x^j) = 1 \text{ and } y^j = 1}{\text{number of test samples with } g(x^j) = 1}$$

$$\text{recall}(g) = \frac{\text{number of test samples with } g(x^j) = 1 \text{ and } y^j = 1}{\text{number of test samples with } y^j = 1}$$

$$F\text{-score}(g) = 2 \frac{\text{precision}(g) \cdot \text{recall}(g)}{\text{precision}(g) + \text{recall}(g)}$$
For different thresholds, $\theta$, we obtain different precision and recall values. They are summarized by a **precision-recall curve**: 

<table>
<thead>
<tr>
<th>precision</th>
<th>recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>0.8</td>
<td>0.4</td>
</tr>
<tr>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>0.4</td>
<td>0.8</td>
</tr>
<tr>
<td>0.2</td>
<td>1</td>
</tr>
</tbody>
</table>

- If pressured, summarize into one number: average precision.
- Curve/value depends on class ratio: higher values for more positives.
For different thresholds, $\theta$, we obtain different precision and recall values.

They are summarized by a **precision-recall curve**:

- If pressured, summarize into one number: **average precision**.
- Curve/value depends on class ratio: higher values for more positives.
A similar role in different context:

**Receiver Operating Characteristic (ROC) Curve**

\[
\text{true-positive-rate}(g) = \frac{\text{number of samples with } g(x^j) = 1 \text{ and } y^j = 1}{\text{number of samples with } y^j = 1}
\]

\[
\text{false-positive-rate}(g) = \frac{\text{number of samples with } g(x^j) = 1 \text{ and } y^j = -1}{\text{number of samples with } y^j = -1}
\]
A similar role in different context:

### Receiver Operating Characteristic (ROC) Curve

<table>
<thead>
<tr>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true-positive-rate( (g) )</td>
<td>[ \frac{\text{number of samples with } g(x^j) = 1 \text{ and } y^j = 1}{\text{number of samples with } y^j = 1} ]</td>
</tr>
<tr>
<td>false-positive-rate( (g) )</td>
<td>[ \frac{\text{number of samples with } g(x^j) = 1 \text{ and } y^j = -1}{\text{number of samples with } y^j = -1} ]</td>
</tr>
</tbody>
</table>

Summarize into: **area under ROC curve (AUC).**
A similar role in different context:

**Receiver Operating Characteristic (ROC) Curve**

\[
\text{true-positive-rate}(g) = \frac{\text{number of samples with } g(x^j) = 1 \text{ and } y^j = 1}{\text{number of samples with } y^j = 1}
\]

\[
\text{false-positive-rate}(g) = \frac{\text{number of samples with } g(x^j) = 1 \text{ and } y^j = -1}{\text{number of samples with } y^j = -1}
\]

Random classifier: \( AUC = 0.5 \), regardless of class proportions.