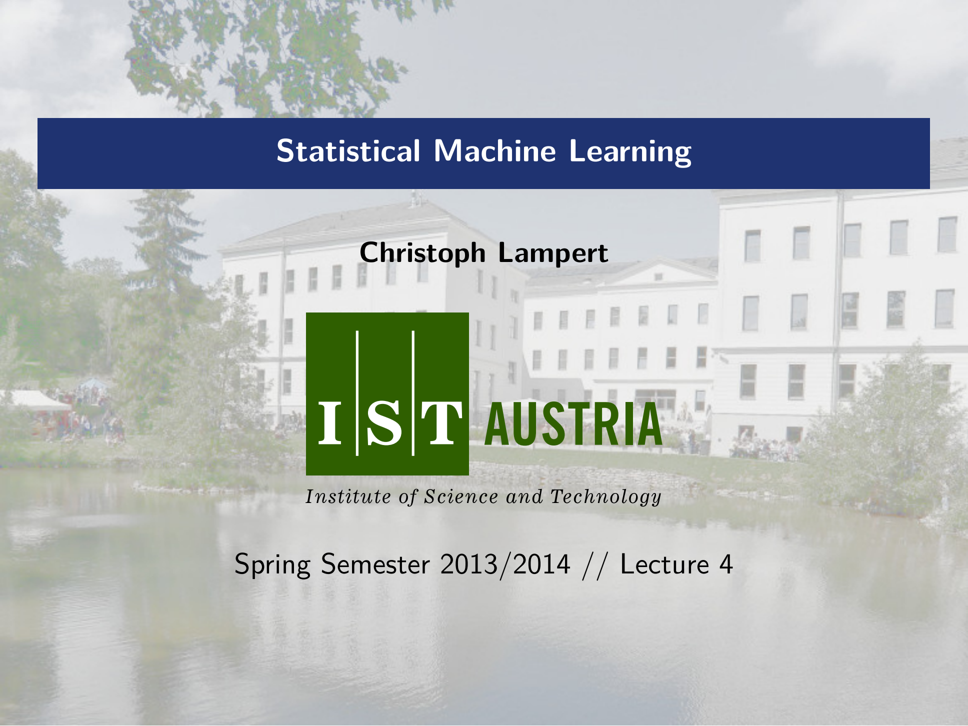


Statistical Machine Learning

Christoph Lampert

IST AUSTRIA

Institute of Science and Technology

Spring Semester 2013/2014 // Lecture 4

Constructing Kernels

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

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

Easier: construct functions that are garanteed to be kernels:

Construct explicitly:

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

Construction from other kernels:

- If k is a kernel and $\alpha \in \mathbb{R}^+$, then $k + \alpha$ and αk 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

$$\max_{\alpha^1,\dots,\alpha^n\in\mathbb{R}} \quad -\frac{1}{2}\sum_{i,j=1}^{n} \alpha^i\alpha^j y^i y^j k(x^i,x^j) + \sum_{i=1}^{n}\alpha^i$$

subject to $\sum_i \alpha_i y_i = 0$ and $0 \le \alpha_i \le C$, for $i = 1, \dots, n$.

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 α_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 $i \notin S$
4:     **for** $i = 1 \dots, n$ **do**
5:         **if** if $i \in S$ and $\alpha_i = 0$ **then** remove $i$ from $S$
6:         **if** if $i \notin S$ 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: **repeat**
3:     pick index $i$ such that $\alpha_i$ is not optimal
4:     pick index $j \neq i$ arbitrarily (usually based on some heuristic)
5:     $\alpha_i, \alpha_j \leftarrow$ solve QP for $\alpha_i, \alpha_j$ and all other $\alpha_k$ fixed
6: **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 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: $f(x) = \langle \tilde{w}, \tilde{\phi}(x) \rangle_{\tilde{\mathcal{H}}}$ with $\tilde{w} = (w, b)$, $\tilde{\phi}(x) = (\phi(x), 1)$.
- kernelize: $\tilde{k}(x, \bar{x}) = \langle \tilde{\phi}(x), \tilde{\phi}(\bar{x}) \rangle_{\tilde{\mathcal{H}}} = k(x, \bar{x}) + 1$.

SVMs Without Bias Term – Optimization

SVM without bias term – primal optimization problem

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

subject to, for $i = 1, \dots, n$,

$$y^i\langle w, x^i\rangle \geq 1 - \xi^i, \qquad \text{and} \qquad \xi^i \geq 0.$$

Difference: no b variable to optimize over

SVMs Without Bias Term – Optimization

SVM without bias term – primal optimization problem

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

subject to, for $i = 1,\dots,n$,

$$y^i\langle w, x^i\rangle \geq 1-\xi^i, \qquad \text{and} \qquad \xi^i \geq 0.$$

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, $\quad 0 \leq \alpha_i \leq C, \quad$ for $i = 1,\dots,n$.

Difference: no constraint $\sum_i y_i\alpha_i = 0$.

Linear SVM Optimization in the Dual

Stochastic Coordinate Dual Ascent

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

Properties:

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

Open Problem:

- how to make each step efficient?

SVM Optimization in the Dual

What's the complexity of the update step? Derive an explicit expression:

Original problem: $\mathbf{max}_{\alpha\in[0,C]^n} \quad -\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y^i y^j\, k(x^i,x^j) + \sum_i \alpha_i$

SVM Optimization in the Dual

What's the complexity of the update step? Derive an explicit expression:

Original problem: $\mathbf{max}_{\alpha\in[0,C]^n} \quad -\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y^i y^j\, k(x^i,x^j) + \sum_i \alpha_i$

When all α_j except α_i are fixed: $\quad \mathbf{max}_{\alpha_i\in[0,C]}\, F(\alpha_i), \quad$ with

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

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

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

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

Observation: each update has complexity $O(n)$.

(Generalized) Linear SVM Optimization in the Dual

Let $k(x,\bar{x}) = \langle \phi(x), \phi(\bar{x}) \rangle_{\mathbb{R}^d}$ for explicitly known $\phi : \mathcal{X} \to \mathbb{R}^d$.

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

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 $O(d)$)

(Generalized) Linear SVM Optimization in the Dual

SCDA for (Generalized) Linear SVMs [Hsieh, 2008]

```
initialize $\alpha \leftarrow \mathbf{0}$, $w \leftarrow \mathbf{0}$
for $t = 1, \dots, T$ 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, $g : \mathcal{X} \to \mathcal{Y}$, and you want to tell the world how good it is. How to measure this?

Reminder:

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

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

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

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

- Problem: function c must be independent of $\mathcal{D}_{tst}$, otherwise law of large numbers doesn't hold.

Classifier Training (idealized)

```
input  training data $\mathcal{D}_{trn}$
input  learning procedure $A$
    $g \leftarrow A[\mathcal{D}]$    (apply $A$ with $\mathcal{D}$ 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 $\mathcal{D}_{tst}$
    apply $g$ to $\mathcal{D}_{tst}$ and measure performance $R_{tst}$
output  performance estimate $R_{tst}$
```

Classifier Training (idealized)

```
input  training data $\mathcal{D}_{trn}$
input  learning procedure $A$
    $g \leftarrow A[\mathcal{D}]$    (apply $A$ with $\mathcal{D}$ 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 $\mathcal{D}_{tst}$
    apply $g$ to $\mathcal{D}_{tst}$ 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

```
input data $\mathcal{D}$
input learning method $A$
    split $\mathcal{D} = \mathcal{D}_{trn} \dot\cup \mathcal{D}_{tst}$ disjointly
    set aside $\mathcal{D}_{tst}$ to a safe place          // do not look at it
    $g \leftarrow A[\mathcal{D}_{trn}]$                          // learn a predictor from $\mathcal{D}_{trn}$
    apply $g$ to $\mathcal{D}_{tst}$ and measure performance $R_{tst}$
output performance estimate $R_{tst}$
```

Classifier Training and Evaluation

```
input data $\mathcal{D}$
input learning method $A$
    split $\mathcal{D} = \mathcal{D}_{trn} \dot\cup \mathcal{D}_{tst}$ disjointly
    set aside $\mathcal{D}_{tst}$ to a safe place      // do not look at it
    $g \leftarrow A[\mathcal{D}_{trn}]$                // learn a predictor from $\mathcal{D}_{trn}$
    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 $\mathcal{D}_{trn}$ and $\mathcal{D}_{tst}$ must be absolute.

- Do not use $\mathcal{D}_{tst}$ for anything except the very last step.
- Do not look at $\mathcal{D}_{tst}$! 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 $\mathcal{D}_{trn}$ and $\mathcal{D}_{tst}$ must be absolute.

- Do not use $\mathcal{D}_{tst}$ for anything except the very last step.
- Do not look at $\mathcal{D}_{tst}$! 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 $\mathcal{D}$
input set of method $\mathcal{A} = \{A_1, \ldots, A_K\}$
  split $\mathcal{D} = \mathcal{D}_{trnval} \dot\cup \mathcal{D}_{tst}$ disjointly
  set aside $\mathcal{D}_{tst}$ to a safe place (do not look at it)

  split $\mathcal{D}_{trnval} = \mathcal{D}_{trn} \dot\cup \mathcal{D}_{val}$ disjointly
  for all models $A_i \in \mathcal{A}$ do
    $g_i \leftarrow A_i[\mathcal{D}_{trn}]$
    apply $g_i$ to $\mathcal{D}_{val}$ and measure performance $E_{val}(A_i)$
  end for
  pick best performing $A_i$

  (optional) $g_i \leftarrow A_i[\mathcal{D}_{trnval}]$    // retrain on larger dataset
  apply $g_i$ to $\mathcal{D}_{tst}$ 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
    $g^{\neg i} \leftarrow A[\ \mathcal{D} \setminus \{(x^i, y^i)\}\ ]$    // $\mathcal{D}_{trn}$ is $\mathcal{D}$ with $i$-th example removed
    $r^i \leftarrow \ell(y^i, g^{\neg i}(x^i))$    // $\mathcal{D}_{val} = \{(x^i, y^i)\}$, disjoint to $\mathcal{D}_{trn}$
  end for
output $R_{loo} = \frac{1}{n}\sum_{i=1}^{n} r^i$    (average leave-one-out risk)
```

Properties.

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

K-fold Cross Validation (CV)

```
input algorithm $A$, loss function $\ell$, data $\mathcal{D}$ (trnval part)
  split $\mathcal{D} = \dot{\bigcup}_{k=1}^{K} \mathcal{D}_k$ into $K$ equal sized disjoint parts
  for $k = 1, \dots, K$ do
    $g^{\neg k} \leftarrow A[\,\mathcal{D} \setminus \mathcal{D}_k\,]$
    $r^k \leftarrow \frac{1}{|\mathcal{D}_k|} \sum_{(x,y) \in \mathcal{D}_k} \ell(y^i, g^{\neg k}(x))$
  end for
output $R_{K\text{-CV}} = \frac{1}{K} \sum_{k=1}^{n} r^k$    ($K$-fold cross-validation risk)
```

Observation.

- for $K = |\mathcal{D}|$ 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 $\mathcal{D}$ (trnval part)
  for $k = 1, \ldots, 5$ do
    Split $\mathcal{D} = \mathcal{D}_1 \dot\cup \mathcal{D}_2$
    $g_1 \leftarrow A[\mathcal{D}_1]$,
    $r_1^k \leftarrow$ evaluate $g_1$ on $\mathcal{D}_2$
    $g_2 \leftarrow A[\mathcal{D}_2]$,
    $r_2^k \leftarrow$ evaluate $g_2$ on $\mathcal{D}_1$
    $r^k \leftarrow \frac{1}{2}(r_k^1 + r_k^2)$
  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 halfs
- 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 prediction set at the same time
- positives $(y = 1)$ are important, negative $(y = -1)$ are a nuisanse
- we don't need all decisions, a few correct positives is enough

For a classifier $g(x) = \operatorname{sign} f(x)$ with $f(x) : \mathcal{X}\mathbb{R}$ (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 $g_\theta(x) = \operatorname{sign}(f(x) - \theta)$, for the right θ.

Other Ways to Evaluate Classifiers

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

Definition (Precision, Recall, F-Score)

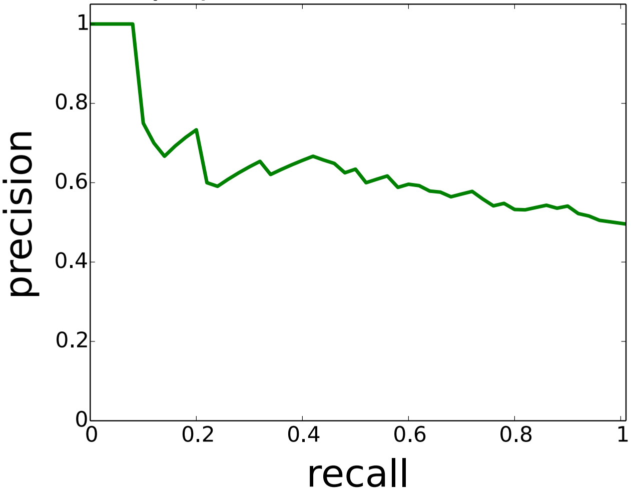
For $\mathcal{Y} = \{\pm 1\}$, let $g : \mathcal{X} \to \mathcal{Y}$ a decision function and $\mathcal{D} = \{(x^1, y^1), \dots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$ be a *database*.

Then we define

$$\begin{aligned}
\textit{precision}(g) &= \frac{\textit{number of test samples with } g(x^j) = 1 \textit{ and } y^j = 1}{\textit{number of test samples with } g(x^j) = 1} \\
\textit{recall}(g) &= \frac{\textit{number of test samples with } g(x^j) = 1 \textit{ and } y^j = 1}{\textit{number of test samples with } y^j = 1} \\
F\textit{-score}(g) &= 2\frac{\textit{precision}(g) \cdot \textit{recall}(g)}{\textit{precision}(g) + \textit{recall}(g)}
\end{aligned}$$

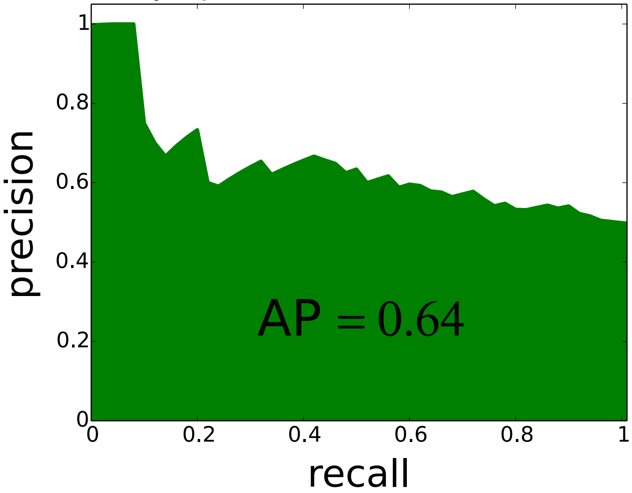
For different thresholds, θ, we obtain different precision and recall values.

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



For different thresholds, θ, we obtain different precision and recall values.

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



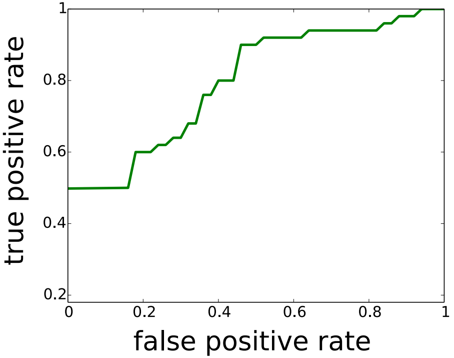
If pressured, summarize into one number: **average precision**.

A similar role in different context:

Receiver Operating Characteristic (ROC) Curve

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

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

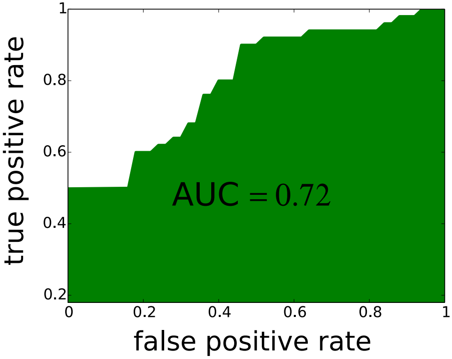


A similar role in different context:

Receiver Operating Characteristic (ROC) Curve

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

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



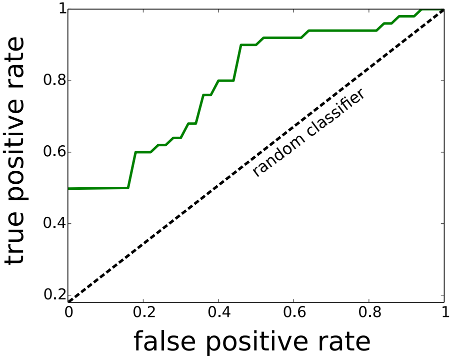
Summarize into: **area under ROC curve (AUC)**.

A similar role in different context:

Receiver Operating Characteristic (ROC) Curve

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

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



Random classifier: $AUC = 0.5$, regardless of class proportions.

Some useful inequalities for next week...

Measure Concentration Inequalities

- Z random variables, taking values $z \in \mathcal{Z} \subseteq \mathbb{R}$.
- $p(Z = z)$ probability distribution
  - $\mu = \mathbb{E}[\,Z\,]$ mean
  - $\mathrm{Var}[z] = \mathbb{E}[\,(Z-\mu)^2\,]$ variance

Lemma

Let $Z_1, \dots,$ be i.i.d. random variables with mean μ, then

$$\frac{1}{m}\sum_{i=1}^{m} Z_i \quad \xrightarrow{m\to\infty} \quad \mu \qquad \textit{with probability } 1.$$

Measure concentration inequalities quantify the deviation between the two values for finite m.

Markov's Inequality

Assumption: $\mathcal{Z} \subseteq \mathbb{R}_+$, i.e. Z takes only non-negative values.

Observation 1) We can write

Lemma (Markov's inequality)

$$\forall a \geq 0 : \quad \mathbb{P}[Z \geq a] \leq \frac{\mathbb{E}[Z]}{a}.$$

Example: Is it possible that more than half of the population have above-average salaries? No, by $a = \frac{1}{2}\mu$.

Chebyshev's Inequality

Lemma (Chebyshev's inequality)

$$\forall a \geq 0 : \quad \mathbb{P}[|Z - \mathbb{E}[Z]| \geq a] \leq \frac{\mathit{Var}[Z]}{a^2}$$

Proof. We apply Markov's Inequality to the random variable $(Z - \mathbb{E}[Z])^2$.

For any $a \geq 0$:

$$\mathbb{P}[\,|Z - \mathbb{E}[Z]| \geq a] = \mathbb{P}[(Z - \mathbb{E}[Z])^2 \geq a^2] \overset{\text{Markov}}{\leq} \frac{\mathbb{E}[\,(Z - \mathbb{E}[Z])^2\,]}{a^2} = \frac{\mathrm{Var}[Z]}{a^2}.$$

Applying Chebyshev's Inequality

Lemma

Set $Z_1, \ldots, Z_m$ be i.i.d. random variables with $\mathbb{E}[Z_i] = \mu$ and Var$[Z_i] \leq 1$. Then, for any $\delta \in (0,1)$ the following inequality holds with probability at least $1 - \delta$:

$$\left| \frac{1}{m} \sum_{i=1}^{m} Z_i - \mu \right| \leq \sqrt{\frac{1}{\delta m}}.$$

Proof. The Z_i are i.i.d., so Var $\left[\frac{1}{m} \sum_{i=1}^{m} Z_i \right] = \frac{1}{m} \sum_{i=1}^{m}$ Var$[Z_i] \leq 1$.

Chebyshev's inequality give us for any $a > 0$

$$\mathbb{P}\left[\left| \frac{1}{m} \sum_{i=1}^{m} Z_i - \mu \right| > a \right] \leq \frac{\text{Var}\left[\frac{1}{m} \sum_{i=1}^{m} Z_i \right]}{ma^2} \leq \frac{1}{ma^2}.$$

Setting $\delta = \frac{1}{ma^2}$ and solving for a yields $a = \sqrt{\frac{1}{\delta m}}$.

Chernoffs' Bound

Lemma (Multiplicative Chernoff Bound)

Let $Z_1, \ldots, Z_m$ be independent Bernoulli (i.e. $0/1$-values) variables, with $\mathbb{P}[Z_i = 1] = p_i$ and $\mathbb{P}[Z_i = 0] = 1 - p_i$ for any i. Let $p = \sum_{i=1}^m p_i$ and $Z = \sum_{i=1}^m Z_i$. Then, for any $\delta > 0$,

$$\mathbb{P}[\, Z > (1+\delta)p \,] \le e^{-h(\delta)p}$$

and

$$\mathbb{P}[\, Z < (1-\delta)p \,] \le e^{-h(-\delta)p}$$

where $h(\delta) = (1+\delta)\log(1+\delta) - \delta$.

Using $h(\delta) \ge \frac{\delta^2}{2+2\delta/3}$ one sees that

$$\mathbb{P}[\, Z > (1+\delta)p \,] \le e^{-p\frac{\delta^2}{2+2\delta/3}} \qquad \text{and} \qquad \mathbb{P}[\, Z < (1-\delta)p \,] \le e^{-p\frac{\delta^2}{2-2\delta/3}}$$

which shows that the probability drops *exponentially* in δ.

Hoeffding's Lemma and Inequality

Lemma (Hoeffding's Lemma)

Let Z be a random variable that takes values in $[a, b]$ and $\mathbb{E}[Z] = 0$. Then, for every $\lambda > 0$,

$$\mathbb{E}[e^{\lambda X}] \leq e^{\frac{\lambda^2 (b-a)^2}{8}}.$$

Lemma (Hoeffding's Inequality)

Let $Z_1, \ldots, Z_m$ be i.i.d. random variables that take values in the interval $[a, b]$. Let $\bar{Z} = \frac{1}{m}\sum_{i=1}^{m} Z_i$ and denote $\mathbb{E}[\bar{Z}] = \mu$. Then, for any $\epsilon > 0$,

$$\mathbb{P}[\left|\frac{1}{m}\sum_{i=1}^{m} Z_i - \mu\right| > \epsilon] \leq 2e^{-m\frac{\epsilon^2}{(b-a)^2}}.$$