Statistical Machine Learning

Christoph Lampert

IST AUSTRIA

Institute of Science and Technology

Spring Semester 2015/2016 // Lecture 5

Kernelization

Definition (Positive Definite Kernel Function)

Let $\mathcal{X}$ be a non-empty set. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called **positive definite kernel function**, if the following conditions hold:

- k is symmetric, i.e. $k(x, x') = k(x', x)$ for all $x, x' \in \mathcal{X}$.
- For any finite set of points $x_1, \dots, x_n \in \mathcal{X}$, the *kernel matrix*

  $$K_{ij} = \left(k(x_i, x_j)\right)_{i,j} \tag{1}$$

  is positive semidefinite, i.e. for all vectors $t \in \mathbb{R}^n$

  $$\sum_{i,j=1}^{n} t_i K_{ij} t_j \geq 0. \tag{2}$$

Kernelization

Lemma (Kernel function)

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

$$k(x, \bar{x}) = \langle\ \phi(x), \phi(\bar{x})\ \rangle_{\mathcal{H}}$$

is a positive definite kernel function.

Proof.

- symmetry: $k(x,\bar{x}) = \langle \phi(x), \phi(\bar{x}) \rangle_{\mathcal{H}} = \langle \phi(\bar{x}), \phi(x) \rangle_{\mathcal{H}} = k(\bar{x}, x)$
- positive definiteness: $x_1, \ldots, x_n \in \mathcal{X}$, and arbitrary $t \in \mathbb{R}^n$, then

$$\begin{aligned}\sum_{i,j=1}^{n} t_i k(x_i, x_j) t_j &= \sum_{i,j=1}^{n} t_i t_j \langle \phi(x^i), \phi(x^j) \rangle_{\mathcal{H}} \\ &= \Big\langle \sum_i t_i \phi(x^i), \sum_j t_j \phi(x^j) \Big\rangle_{\mathcal{H}} = \Big\| \sum_i t_i \phi(x^i) \Big\|_{\mathcal{H}}^2 \geq 0.\end{aligned}$$

$\square$

Theorem (Mercer's Condition)

Let $\mathcal{X}$ be non-empty set. For any positive definite kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there exists a Hilbert space $\mathcal{H}$ with inner product $\langle \cdot\,, \cdot \rangle_{\mathcal{H}}$, and a feature map $\phi : \mathcal{X} \to \mathcal{H}$ such that

$$k(x, \bar{x}) = \langle\ \phi(x), \phi(\bar{x})\ \rangle_{\mathcal{H}}.$$

Proof. later, in more refined form

Note: $\mathcal{H}$ and ϕ are not unique, e.g.

$$k(x, \bar{x}) = 2x\bar{x}$$

- $\mathcal{H}_1 = \mathbb{R}$, $\phi_1(x) = \sqrt{2}x$, $\quad \langle \phi_1(x), \phi_1(\bar{x}) \rangle_{\mathcal{H}_1} = 2x\bar{x}$
- $\mathcal{H}_2 = \mathbb{R}^2$, $\phi_2(x) = \begin{pmatrix} x \\ -x \end{pmatrix}$, $\quad \langle \phi_1(x), \phi_2(\bar{x}) \rangle_{\mathcal{H}_2} = 2x\bar{x}$
- $\mathcal{H}_3 = \mathbb{R}^3$, $\phi_3(x) = \begin{pmatrix} x \\ 0 \\ x \end{pmatrix}$, $\quad \langle \phi_3(x), \phi_3(\bar{x}) \rangle_{\mathcal{H}_3} = 2x\bar{x}$, etc.

Definition (Reproducing Kernel Hilbert Space)

Let $\mathcal{H}$ be a Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$. A kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called **reproducing kernel**, if

$$f(x) = \langle k(x, \cdot),\ f(\cdot) \rangle_{\mathcal{H}} \qquad \text{for all } f \in \mathcal{H}.$$

$\mathcal{H}$ is then called a **reproducing kernel Hilbert space (RKHS).**

Theorem (Moore-Aronszajn Theorem)

Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a positive definite kernel on $\mathcal{X}$. Then there is a unique Hilbert space of functions, $f : \mathcal{X} \to \mathbb{R}$, for which k is a reproducing kernel.

Proof sketch. One can construct the space explicitly: Set

$$\mathcal{H}^{\mathsf{pre}} = \mathsf{span}\{\ k(\cdot, x) \text{ for } x \in \mathcal{X}\ \},$$

i.e., for every $f \in \mathcal{H}^{\mathsf{pre}}$ exist $x^1, \dots, x^m \in \mathcal{X}$ and $\alpha^1, \dots, \alpha^m \in \mathbb{R}$, with

$$f(\cdot) = \sum_{i=1}^{m} \alpha^i k(\cdot, x^i).$$

We define an inner product as

$$\langle f, g \rangle = \Big\langle \sum_i \alpha^i k(\cdot, x^i), \sum_j \bar{\alpha}^j k(\cdot, \bar{x}^j) \Big\rangle := \sum_{i,j} \alpha^i \bar{\alpha}^j k(x^i, \bar{x}^j).$$

Make $\mathcal{H}^{\mathsf{pre}}$ into Hilbert space $\mathcal{H}$ by enforcing *completeness*.

Complete proof: [B. Schölkopf, A. Smola, *"Learning with Kernels"*, 2001].

Let

- $\mathcal{D} = \{(x^1, y^1), \dots, (x^n, y^n)\} \subset \mathcal{X} \times \{\pm 1\}$ training set
- $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a pos.def. kernel with feature map $\phi : \mathcal{X} \to \mathcal{H}$.

Support Vector Machine in Kernelized Form

For any $C > 0$, the max-margin classifier for the feature map ϕ is

$$g(x) = \operatorname{sign} f(x) \qquad \text{with} \quad f(x) = \sum_i \alpha_i k(x^i, x) + b,$$

for coefficients $\alpha_1, \dots, \alpha_n$ obtained by solving

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

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

Note: we don't need to know ϕ or $\mathcal{H}$, explicitly. Knowing k is enough.

Useful and Popular Kernel Functions

For $x, \bar{x} \in \mathbb{R}^d$:

- $k(x, \bar{x}) = (1 + \langle x, x' \rangle)^p$ for $p \in \mathbb{N}$ *(polynomial kernel)*

  $f(x) = \sum_i \alpha_i y^i k(x^i, x) =$ polynomial of degree d

- $k(x, \bar{x}) = \exp(-\lambda \|x - \bar{x}\|^2)$ for $\lambda > 0$ *(Gaussian or RBF kernel)*

  $f(x) = \sum_i \alpha_i y^i \exp(-\lambda \|x^i - x\|^2) =$ weighted/soft nearest neighbor

For $x, \bar{x}$ histograms with d bins:

- $k(x, \bar{x}) = \sum_{j=1}^d \mathbf{min}(x_j, \bar{x}_j)$ histogram intersection kernel
- $k(x, \bar{x}) = \sum_{j=1}^d \frac{x_j \bar{x}_j}{x_j + \bar{x}_j}$ χ^2 kernel
- $k(x, \bar{x}) = \exp\big(-\lambda \sum_{j=1}^d \frac{(x_j - \bar{x}_j)^2}{x_j + \bar{x}_j}\big)$ exponentiated χ^2 kernel

Generally: interpret kernel function as a **similarly measure**.

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 $0 \leq \alpha_i \leq C$, 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** $i = 1 \ldots, n$ **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
- we can get away without storing complete kernel matrix

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, sometimes 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: 1-dimensional

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: $\quad \mathbf{max}_{\alpha_i\in[0,C]} F(\alpha_i)$, 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 $O(n)$ kernel evaluations

(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,

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

  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: samples $\ell(y, g(x))$ must be independent, otherwise law of large numbers doesn't hold.
- Make sure that g is independent of $\mathcal{D}_{tst}$.

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 $R_{tst}$
```

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 $g : \mathcal{X} \to \mathcal{Y}$
```

Classifier Evaluation

```
input trained classifier $g : \mathcal{X} \to \mathcal{Y}$
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 $R_{tst}$
```

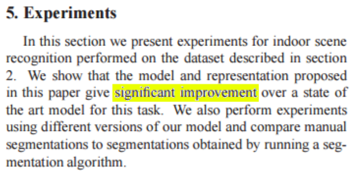
Classifier Training and Evaluation

```
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 $R_{tst}$
```

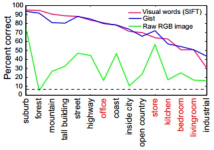
Remark. $\mathcal{D}_{tst}$ should be as small as possible, to keep $\mathcal{D}_{trn}$ as big as possible, but large enough to be convincing.

- sometimes: 50%/50% for small datasets
- more often: 80% training data, 20% test data
- for large datasets: 90% training, 10% test data.

Significance of Results


5. Experiments

In this section we present experiments for indoor scene recognition performed on the dataset described in section 2. We show that the model and representation proposed in this paper give significant improvement over a state of the art model for this task. We also perform experiments using different versions of our model and compare manual segmentations to segmentations obtained by running a segmentation algorithm.



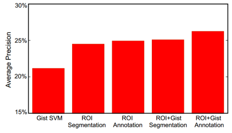
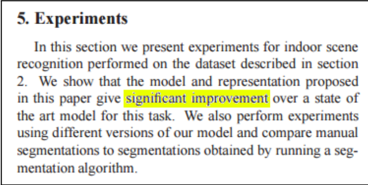


Figure 6. Multiclass average precision performance for the baseline and four different versions of our model.

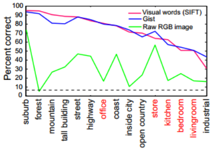
How to tell if reported differences are due to chance?

Significance of Results



5. Experiments

In this section we present experiments for indoor scene recognition performed on the dataset described in section 2. We show that the model and representation proposed in this paper give significant improvement over a state of the art model for this task. We also perform experiments using different versions of our model and compare manual segmentations to segmentations obtained by running a segmentation algorithm.



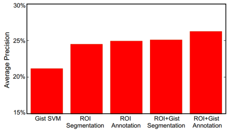


Figure 6. Multiclass average precision performance for the baseline and four different versions of our model.

How to tell if reported differences are due to chance?

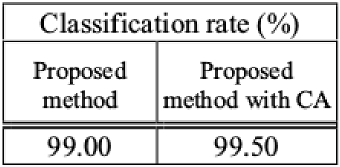
Accuracy on a Single Dataset

- two-sample significance tests
- paired significance tests

Multiple Datasets

- non-parametric paired significance tests

Test on a Single Dataset



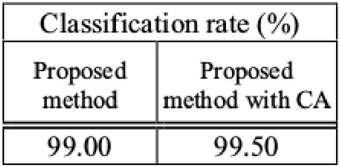
Classification rate (%)	
Proposed method	Proposed method with CA
99.00	99.50

Two classifiers are evaluated on the same test set.

- classifier 1 has error rate $e_1 \in [0, 1]$
- classifier 2 has error rate $e_2 \in [0, 1]$

Are these significantly different, or due to chance?

Test on a Single Dataset



Classification rate (%)	
Proposed method	Proposed method with CA
99.00	99.50

Two classifiers are evaluated on the same test set.

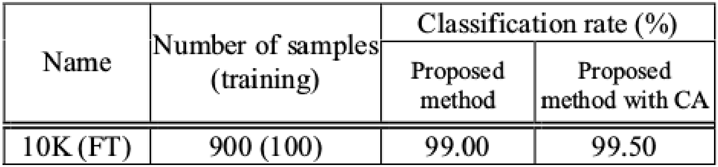
- classifier 1 has error rate $e_1 \in [0, 1]$
- classifier 2 has error rate $e_2 \in [0, 1]$

Are these significantly different, or due to chance?

Impossible to tell, unless we know how many test samples!

How many examples do you guess?

Test on a Single Dataset



Name	Number of samples (training)	Classification rate (%)	
		Proposed method	Proposed method with CA
10K (FT)	900 (100)	99.00	99.50

Two classifiers are evaluated on the same test set.

- classifier 1 has error rate $e_1 \in [0, 1]$
- classifier 2 has error rate $e_2 \in [0, 1]$

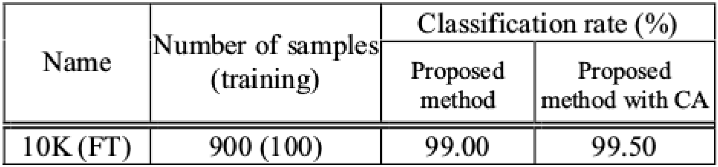
Are these significantly different, or due to chance?

Impossible to tell, unless we know how many test samples!

How many examples do you guess? **Okay, that's a start...**

Error bars

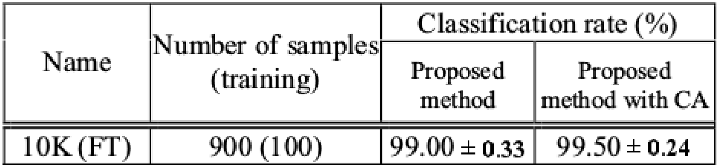
- true error rate of classifier f is $p \in [0, 1]$ $\quad \rightarrow$ Bernoulli
- estimate from m test samples: $\hat{p} = \frac{1}{m} \sum_i [\![f(x_i) \neq y_i]\!]$
- variance of estimate from m test samples: $V = \frac{1}{m} \hat{p}(1 - \hat{p})$
- report mean $\pm$ standard error of the mean: $\hat{p} \pm \sqrt{\frac{\hat{p}(1-\hat{p})}{m}}$



Name	Number of samples (training)	Classification rate (%)	
		Proposed method	Proposed method with CA
10K (FT)	900 (100)	99.00	99.50

Error bars

- true error rate of classifier f is $p \in [0, 1]$ $\quad \rightarrow$ Bernoulli
- estimate from m test samples: $\hat{p} = \frac{1}{m} \sum_i [\![f(x_i) \neq y_i]\!]$
- variance of estimate from m test samples: $V = \frac{1}{m} \hat{p}(1 - \hat{p})$
- report mean $\pm$ standard error of the mean: $\hat{p} \pm \sqrt{\frac{\hat{p}(1-\hat{p})}{m}}$

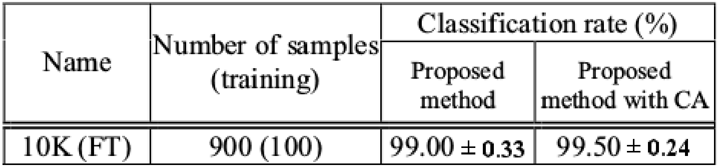


Name	Number of samples (training)	Classification rate (%)	
		Proposed method	Proposed method with CA
10K (FT)	900 (100)	99.00 ± 0.33	99.50 ± 0.24

Not particularly convincing... but also not a proper test.

Error bars

- true error rate of classifier f is $p \in [0,1]$ $\quad \to$ Bernoulli
- estimate from m test samples: $\hat{p} = \frac{1}{m}\sum_i [\![f(x_i) \neq y_i]\!]$
- variance of estimate from m test samples: $V = \frac{1}{m}\hat{p}(1-\hat{p})$
- report mean $\pm$ standard error of the mean: $\hat{p} \pm \sqrt{\frac{\hat{p}(1-\hat{p})}{m}}$



Name	Number of samples (training)	Classification rate (%)	
		Proposed method	Proposed method with CA
10K (FT)	900 (100)	99.00 ± 0.33	99.50 ± 0.24

Not particularly convincing... but also not a proper test.

Could be formalized to...

Two-sample test

We observe two sets of samples S_1, S_2 (the losses of each method). Are both sampled from the same underlying distribution?

...but ignores that that we evalute two systems *on the same test set.*

Paired test

For a sequence of experiments we always observe two sets of outcomes A, B. Are the differences between them due to chance?

2×2 contingency table:

	g is right	g is wrong
f is right	a	b
f is wrong	c	d

binomial test: ignore a and d, analyze b and c.

- null hypothesis: f and g are equally good. we'd expect $b \approx c$
- probability of seeing (b, c) split or more extreme in $b + c$ differences:

$$p\text{-value} \;=\; 2\frac{1}{2^{b+c}} \sum_{i=0}^{\mathbf{min}(b,c)} \binom{b+c}{i}$$

- `scipy.stats.binom_test( min(b,c), n=b+c, p=0.5 )`

Paired test

For a sequence of experiments we always observe two sets of outcomes A, B. Are the differences between them due to chance?

2×2 contingency table:

	g is right	g is wrong
f is right	a	b
f is wrong	c	d

binomial test: ignore a and d, analyze b and c.

- null hypothesis: f and g are equally good. we'd expect $b \approx c$
- probability of seeing (b, c) split or more extreme in $b + c$ differences:

$$p\text{-value} \;=\; 2\frac{1}{2^{b+c}} \sum_{i=0}^{\mathbf{min}(b,c)} \binom{b+c}{i}$$

- `scipy.stats.binom_test( min(b,c), n=b+c, p=0.5 )`

Example:

891	0
4	5

$p \approx 0.25$

887	5
9	0

$p \approx 0.85$

8910	0
45	45

$p \approx 10^{-13}$

8865	45
90	0

$p \approx 0.0003$

Caveats

Remark: $\mathcal{D}_{tst}$ and $\mathcal{D}_{trn}$ must be truly independent

- No overlapping data between $\mathcal{D}_{trn}$ and $\mathcal{D}_{tst}$
- No hidden dependence (e.g. time series, same patient/animals, . . .)
- 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.

Caveats

Remark: $\mathcal{D}_{tst}$ and $\mathcal{D}_{trn}$ must be truly independent

- No overlapping data between $\mathcal{D}_{trn}$ and $\mathcal{D}_{tst}$
- No hidden dependence (e.g. time series, same patient/animals, ...)
- 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 classifiers, 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: