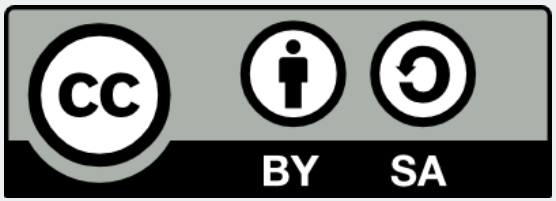
Support vector machines

2019-05-09

Daniel Oberski
Dept. of Methodology & Statistics
Focus Area Applied Data Science
d.l.oberski@uu.nl



Cortes & Vapnik, 1995, *Machine Learning*

> *The machine conceptually implements the following idea: input vectors are non-linearly mapped to a very high-dimension feature space. In this feature space a linear decision surface is constructed. Special properties of the decision surface ensures [sic] high generalization ability of the learning machine.*

Building blocks (statistical terminology):

Cortes & Vapnik, 1995, *Machine Learning*

The machine conceptually implements the following idea: input vectors are ***non-linearly mapped to a very high-dimension feature space****. In this feature space a linear decision surface is constructed. Special properties of the decision surface ensures [sic] high generalization ability of the learning machine.*

Building blocks (statistical terminology):

1. **The ”kernel trick”**

Cortes & Vapnik, 1995, *Machine Learning*

The machine conceptually implements the following idea: input vectors are non-linearly mapped to a very high-dimension feature space. In this feature space ***a linear decision surface is constructed****. Special properties of the decision surface ensures [sic] high generalization ability of the learning machine.*

Building blocks (statistical terminology):

1. **The "kernel trick"**
2. **Linear classifier** / Linear predictor

Cortes & Vapnik, 1995, *Machine Learning*

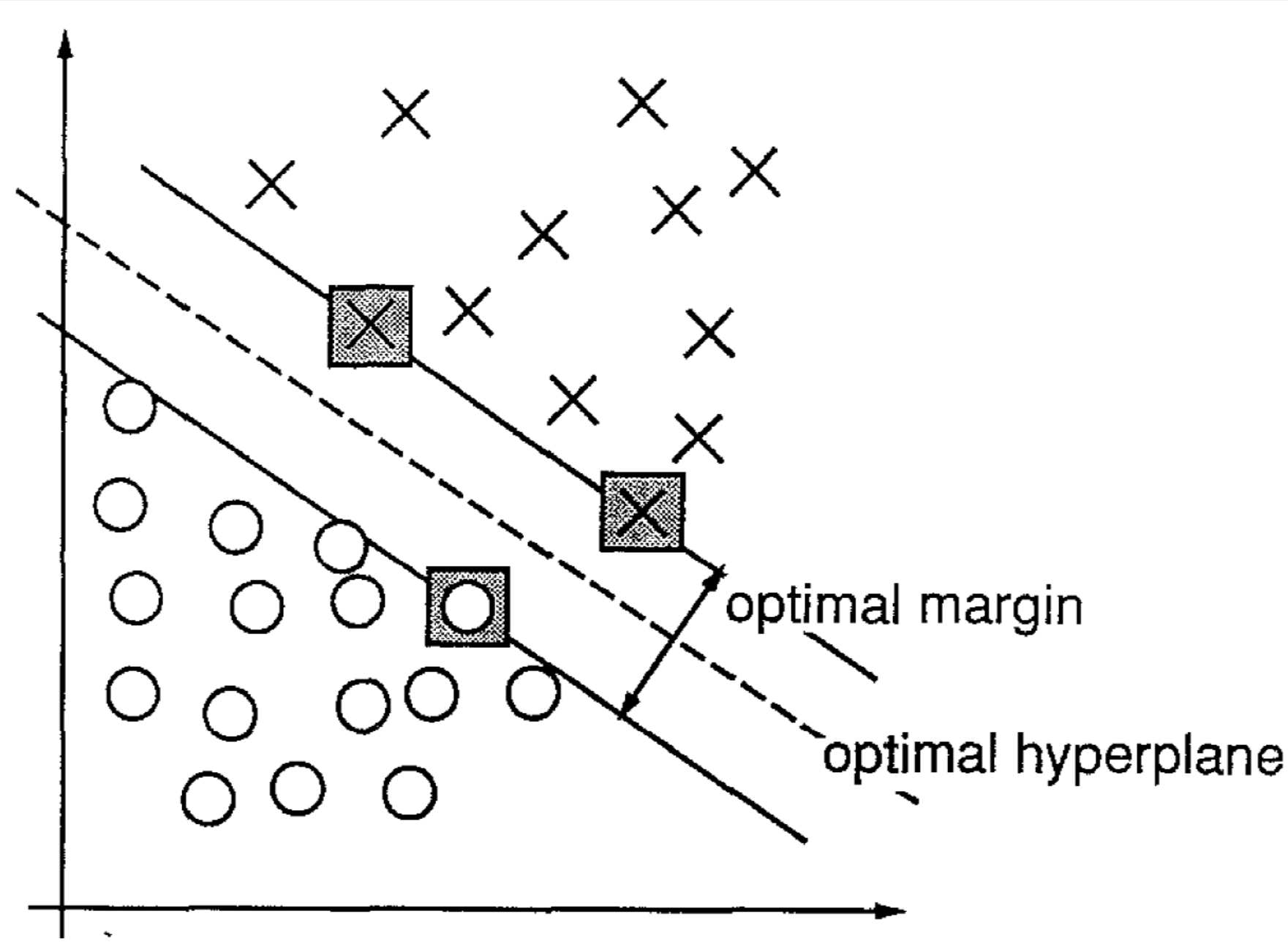
The machine conceptually implements the following idea: input vectors are non-linearly mapped to a very high-dimension feature space. In this feature space a linear decision surface is constructed. ***Special properties of the decision surface ensures [sic] high generalization ability*** *of the learning machine.*

Building blocks (statistical terminology):

1. **The "kernel trick"**
2. **Linear classifier** / Linear predictor
3. **Maximum-margin** / Hinge loss with ridge penalty

Why (not) study SVMs?

- ✗ The absolute overall best
- ✗ Unique in using kernels to be nonlinear
- ✗ Unique in using maximum-margin principle
- ✔ Often used
- ✔ Sometimes useful
- ✔ Interesting history connecting ML and stats

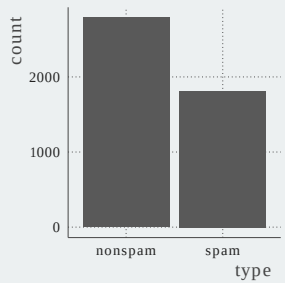


[Cortes & Vapnik, 1998]

Software

The spam dataset

	x_1 make	x_1 address	x_3 all	... ...	x_{58} capitalTotal	y type
1	0.00	0.64	0.64	...	278.00	spam
2	0.21	0.28	0.50	...	1028.00	spam
3	0.06	0.00	0.71	...	2259.00	spam
4	0.00	0.00	0.00	...	191.00	spam
5	0.00	0.00	0.00	...	191.00	nonspam
6	0.00	0.00	0.00	...	54.00	nonspam
...	...	...	...	...	...	...
4601						



Support vector machine in R with e1071

```
library(e1071)
data(spam)

idx_train <- sample(1:nrow(spam), size = 4000)

spam_train <- spam[idx_train, ]
spam_test <- spam[-idx_train, ]

fit_spam <- svm(type ~., data = spam_train,
                cost = 63, gamma = 0.005, kernel = "radial")
```

Support vector machine in R with kernlab

```
library(kernlab)

fit_spam <- ksvm(type ~., data = spam_train,
           cost = 63, sigma = 1/(2*0.005), kernel = "rbfdot")
```

Support vector machine in python with sklearn

```
>>> from sklearn import svm
>>> X = [[0, 0], [1, 1]]
>>> y = [0, 1]
>>> clf = svm.SVC(gamma='scale')
>>> clf.fit(X, y)
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale',
    kernel='rbf', max_iter=-1, probability=False,
    random_state=None, shrinking=True, tol=0.001,
    verbose=False)
```

Practical SVM

- A **black-box** classification method
  - Input: features, target to classify
  - Output: predicted label
    (Optionally: estimated probability output)
- **Tuning** parameters:
  - `kernel` type $\in$ {`linear`, `radial`, `poly`, ...} (default `radial`)
  - `cost` $\in (0, \infty)$ (default 1) or nu $\in (0, 1)$ (defaults 0.2, 0.5)
  - kernel parameters, e.g. for `radial` kernel:
    gamma $\in (0, \infty)$ (default 0.2 or heuristic) or `sigma`, $\sigma = \frac{1}{2\gamma} > 0$.

Practical SVM: tunability

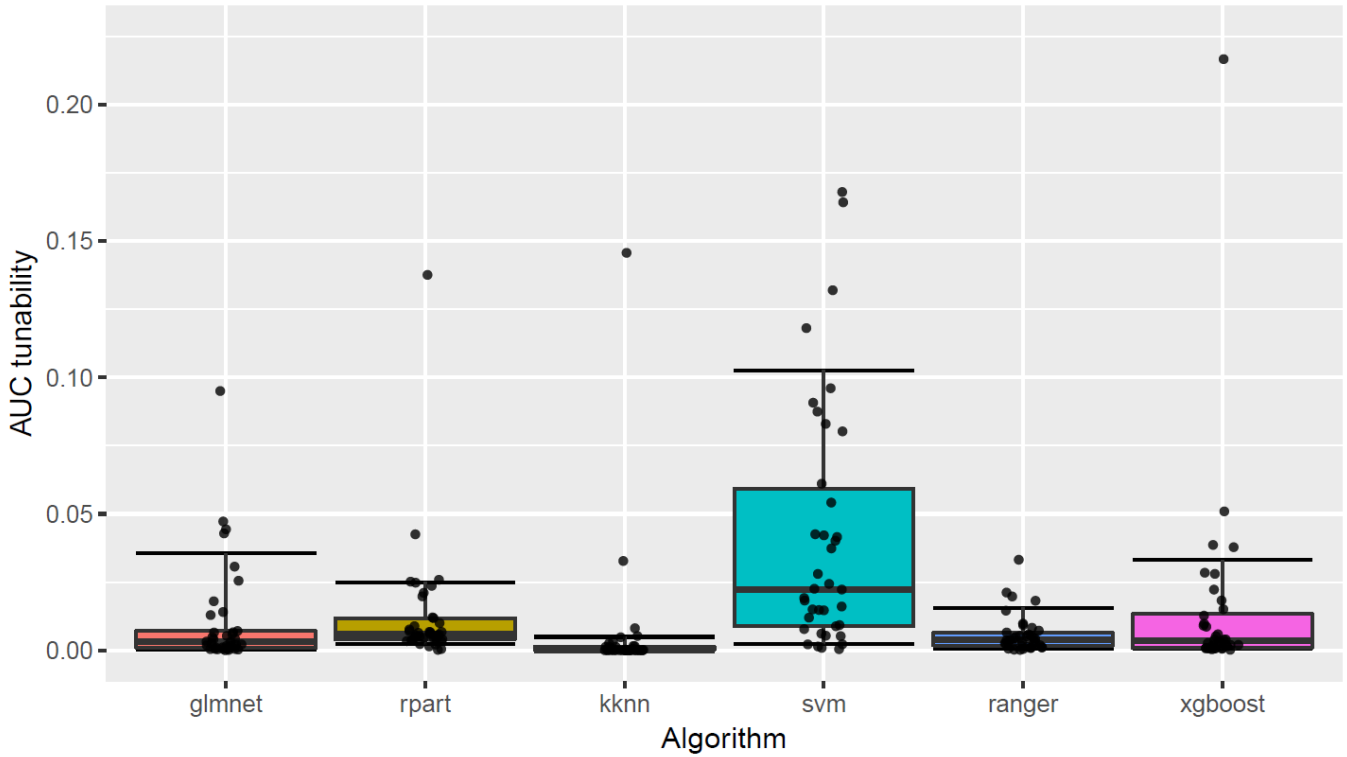


Figure 2: Boxplots of the tunabilities (AUC) of the different algorithms with respect to optimal defaults. The upper and lower whiskers (upper and lower line of the boxplot rectangle) are in our case defined as the 0.1 and 0.9 quantiles of the tunability scores. The 0.9 quantile indicates how much performance improvement can be expected on at least 10% of datasets. One outlier of `glmnet` (value 0.5) is not shown.

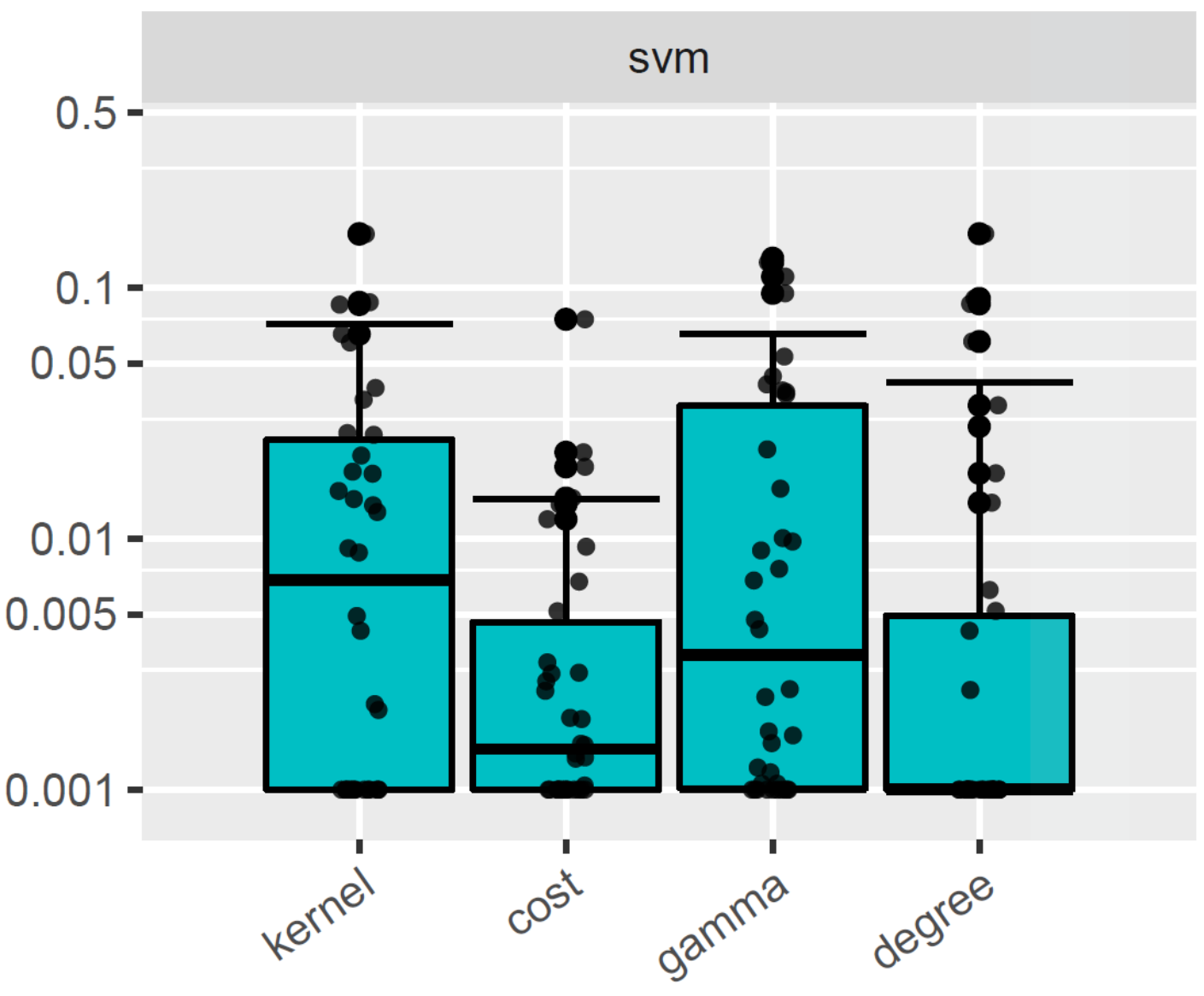
[Probst et al. 2019, *JMLR*]

Practical SVM: tunability

Parameter	Def.P	Def.O	Tun.P	Tun.O	$q_{0.05}$	$q_{0.95}$
svm			0.056	0.042		
kernel	radial	radial	0.030	0.024		
cost	1	682.478	0.016	0.006	0.002	920.582
gamma	$1/p$	0.005	0.030	0.022	0.003	18.195
degree	3	3	0.008	0.014	2	4

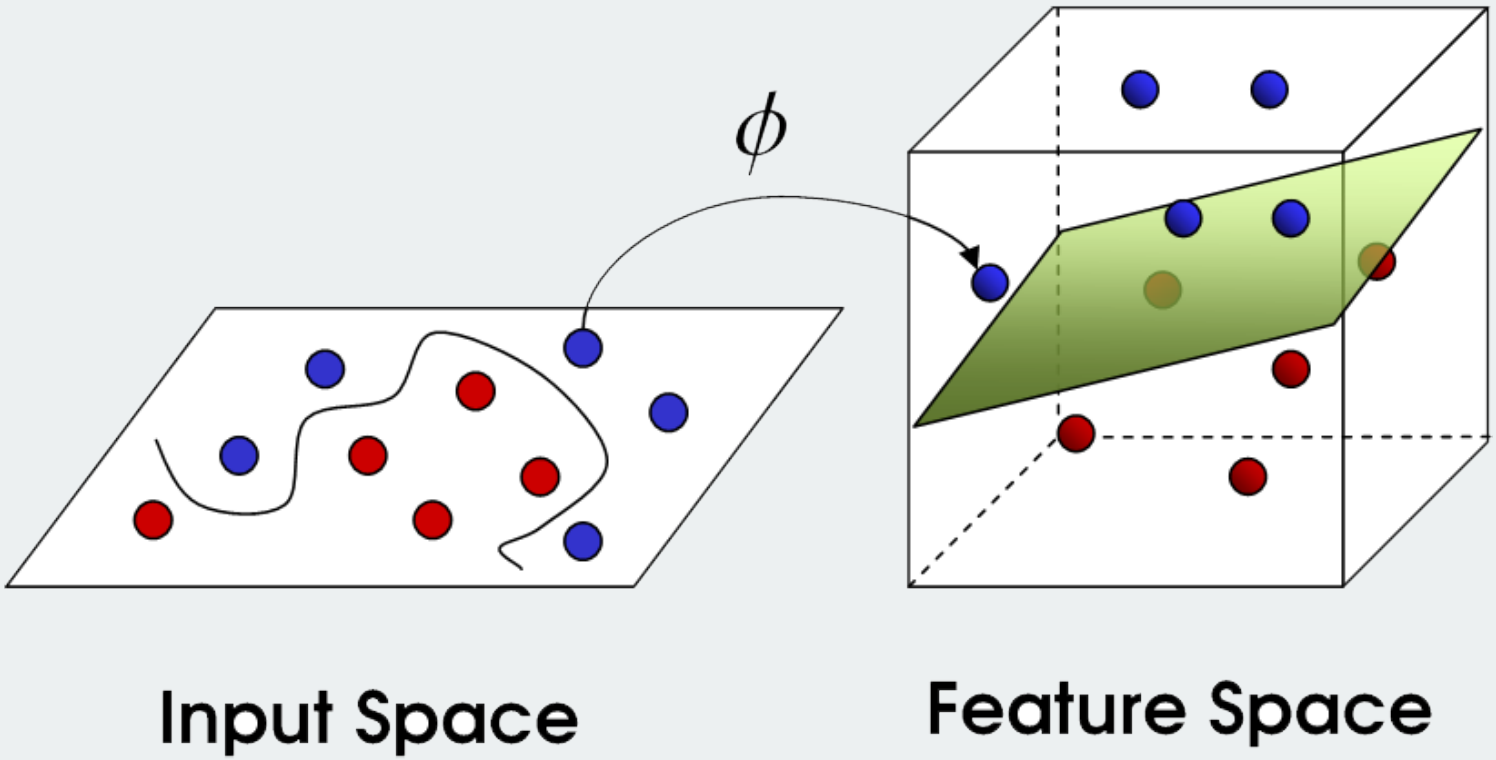
[Probst et al. 2019, *JMLR*]

Practical SVM: tunability



[Probst et al. 2019, *JMLR*]

Kernels



I kissed a kernel and I liked it

A **kernel** is a function of two arguments $\kappa(\mathbf{x}, \mathbf{x}') \in \mathbb{R}$, with both $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$
We'll use kernels that are "similarities":

- $\kappa(\mathbf{x}, \mathbf{x}') = \kappa(\mathbf{x}', \mathbf{x})$
- $\kappa(\mathbf{x}, \mathbf{x}') \geq 0$

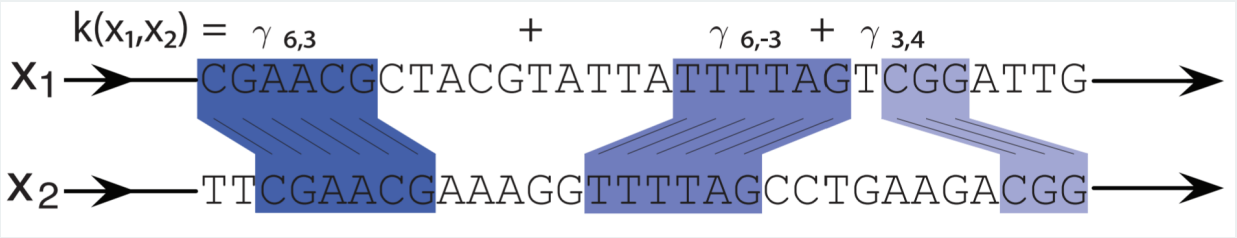
For example:

Name	$\kappa(\mathbf{x}, \mathbf{x}')$	Parameter
Linear	$\mathbf{x}^T\mathbf{x}'$	-
Polynomial	$(1 + \mathbf{x}^T\mathbf{x}')^d$	degree, d
Radial basis function (RBF)	$\exp\left(-\|\|\mathbf{x} - \mathbf{x}'\|\|^2\gamma\right)$	concentration, γ
...		

[Murphy 2012]

Bonus kernels for your enjoyment

- String kernel:



  E.g. comparing DNA sequences

- Fisher kernel: $\mathbf{g}(\mathbf{x})^T \mathbf{I}^{-1} \mathbf{g}(\mathbf{x}')$ with $\mathbf{g}$ the score function and $\mathbf{I}$ the Fisher information of any likelihood.

  e.g. size & shape of atomic structures/objects, ...[Le et al. 2018, *NIPS*]

- Matérn kernel:

$$\kappa(\mathbf{x}, \mathbf{x}') = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu} \frac{||\mathbf{x}-\mathbf{x}'||}{\rho} \right)^{\nu} K_\nu \left(\sqrt{2\nu} \frac{||\mathbf{x}-\mathbf{x}'||}{\rho} \right),$$

  Bessel function K_ν, "degrees of freedom" $\nu > 0$, dispersion$\rho > 0$.

  (important in **gaussian processes** and geostats "**kriging**")

[Murphy 2012]

Mercer kernels and the kernel trick

Now we now how to compute a "distance" $\kappa(\mathbf{x}, \mathbf{x}')$ for any pair of observations, we can construct a matrix of pairwise distances, like this:

$$\text{"Gram matrix"} = \mathbf{K} = \begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \dots & \kappa(\mathbf{x}_1, \mathbf{x}_n) \\ & \vdots & \\ \kappa(\mathbf{x}_n, \mathbf{x}_1) & \dots & \kappa(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}$$

A Mercer kernel is any kernel that gives a symmetric positive definite Gram matrix. This allows the famous **kernel trick**:

$$\mathbf{K} = \mathbf{U}^T \mathbf{L} \mathbf{U},$$

where $\mathbf{L}$ diagonal matrix of positive eigenvalues. So any element of $\mathbf{K}$ is:

$$\mathbf{K}_{i,j} = (\mathbf{L}^{\frac{1}{2}} \mathbf{U}_{:,i})^T (\mathbf{L}^{\frac{1}{2}} \mathbf{U}_{:,j}) := \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Note that $\phi(\mathbf{x})$ can be as crazy and nonlinear as we like. For RBF it's even infinite-dimensional!

$$\mathbf{K}_{i,j} = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Kernel trick

The **K** matrix (which is easy to compute) turns out to be the pairwise inner product (similarity) among observations of a high-dimensional nonlinear transformation $\phi(\mathbf{x})$ of the original data (which is difficult or impossible to compute).

Famous transformation functions $\phi(\mathbf{x})$:

Name	$\kappa(\mathbf{x}, \mathbf{x}')$	Implicit transformation
Linear	$\mathbf{x}^T \mathbf{x}'$	Nothing!
Polynomial	$(1 + \mathbf{x}^T \mathbf{x}')^d$	Polynomial of order d
RBF	$\exp\left(-\|\|\mathbf{x} - \mathbf{x}'\|\|^2 \gamma\right)$	Weighted average over n Gaussian densities

[Murphy 2012]

Why is the kernel trick important?

- Whenever you see the data $\mathbf{x}$ enter into the algorithm only through their inner products, you can replace these super-easily with inner products of complex functions, without having to compute those functions (just $\mathbf{K}$).
- Examples:
  - SVM (obviously)
  - k-NN $\longrightarrow$ kernel nearest neighbors
  - Linear regression $\longrightarrow$ kernel regression
  - Logistic regression; Any GLM $\longrightarrow$ kernel logistic regression
  - PCA $\longrightarrow$ kernel PCA
  - k-medioids clustering $\longrightarrow$ kernel k-medioids
  - ...

- **Conclusion**: many models can be rewritten as a function of similarities among observations ("dual" formulation).
- All these models can be kernelized.

[Murphy 2012]

Artificial data example

We'll create a highly nonlinear decision surface and see how we do.

$$f(\mathbf{x}) = w \sin(x_1 x_2)$$

$$y \sim \text{Bernoulli}\left(\frac{1}{1 + \exp\left(f(\mathbf{x})\right)}\right)$$

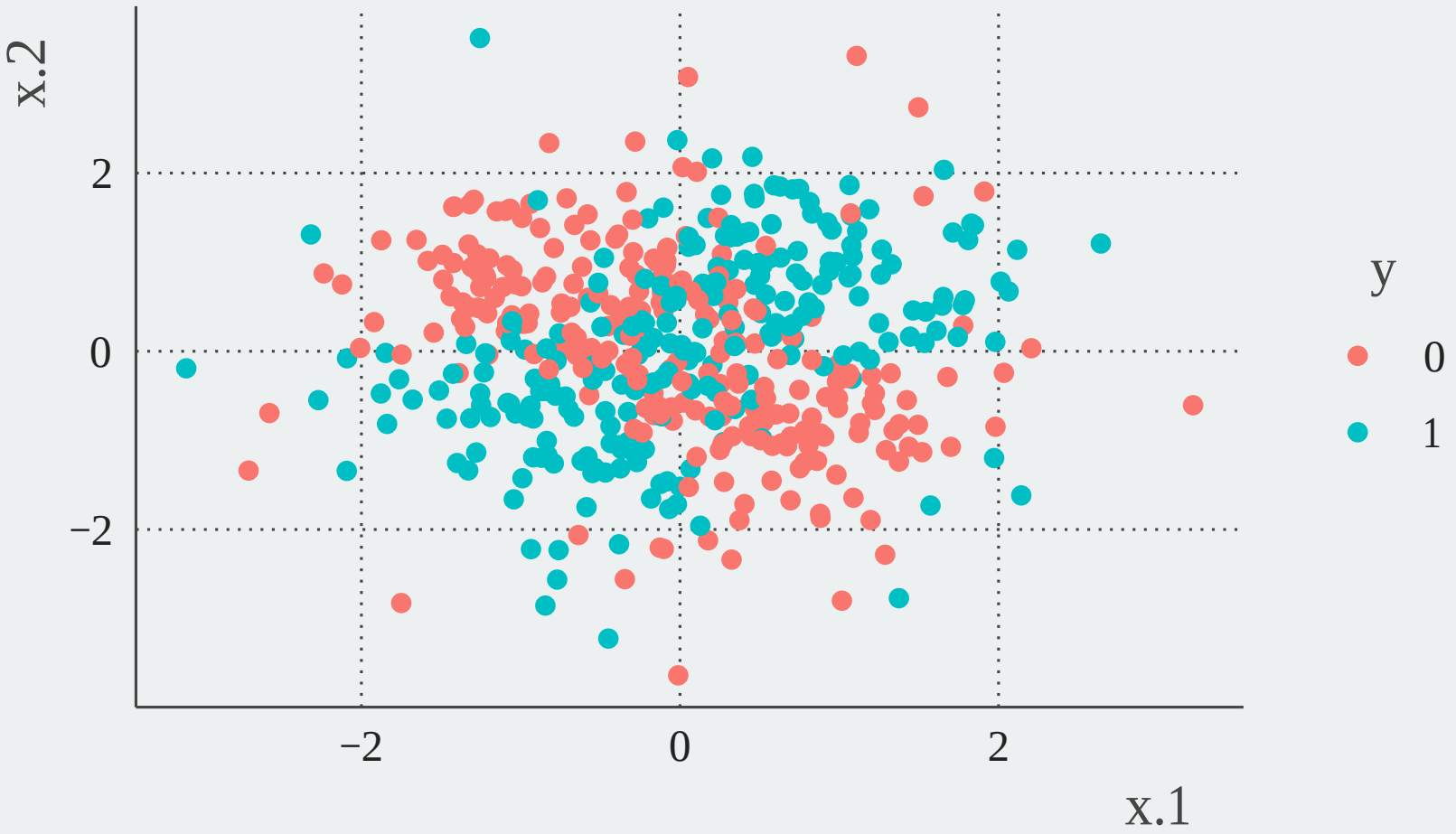
First 10 observations in our data

	x_1	x_2	y
1	0.856	0.885	1
2	-0.313	-0.435	0
3	-0.878	-0.105	0
4	0.344	-0.777	0
5	0.382	0.874	1
6	0.494	0.599	1
7	0.854	0.777	1
8	0.764	-1.085	0
9	-1.276	-1.502	1
10	-2.218	-0.173	1
...	...	...	...
500			

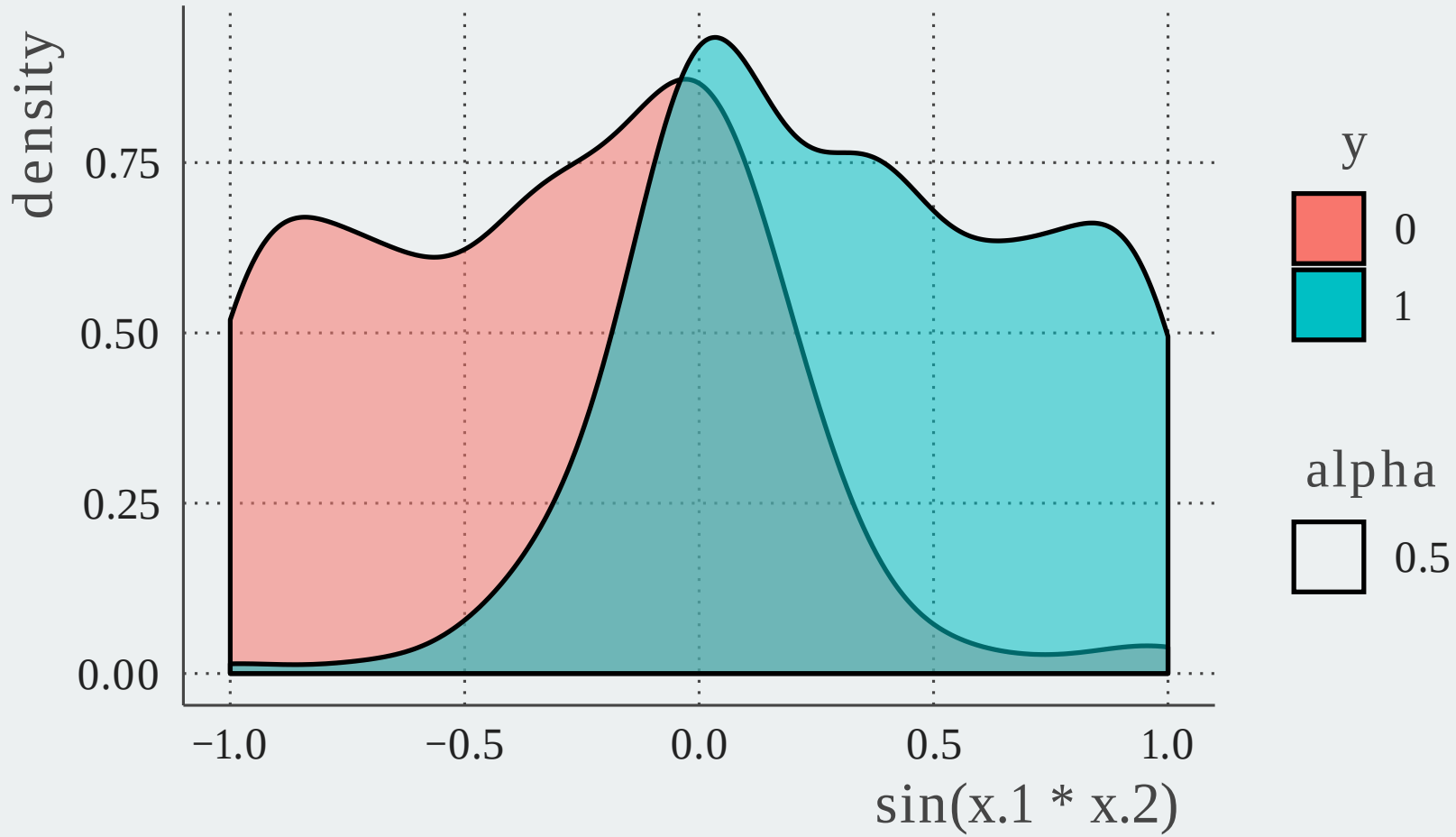
Gram matrix K, `radial` basis function kernel (RBF)

	1	2	3	4	5	6	7	8	9	10
1	1.000									
2	0.537	1.000								
3	0.451	0.918	1.000							
4	0.546	0.896	0.678	1.000						
5	0.956	0.644	0.601	0.580	1.000					
6	0.958	0.709	0.621	0.682	0.983	1.000				
7	0.998	0.568	0.470	0.586	0.955	0.968	1.000			
8	0.459	0.729	0.481	0.947	0.451	0.559	0.499	1.000		
9	0.129	0.662	0.656	0.533	0.187	0.221	0.143	0.420	1.000	
10	0.121	0.478	0.698	0.250	0.208	0.204	0.127	0.143	0.588	1.000
...	...	...	...	...	...	...	...	...	...	...

Original data

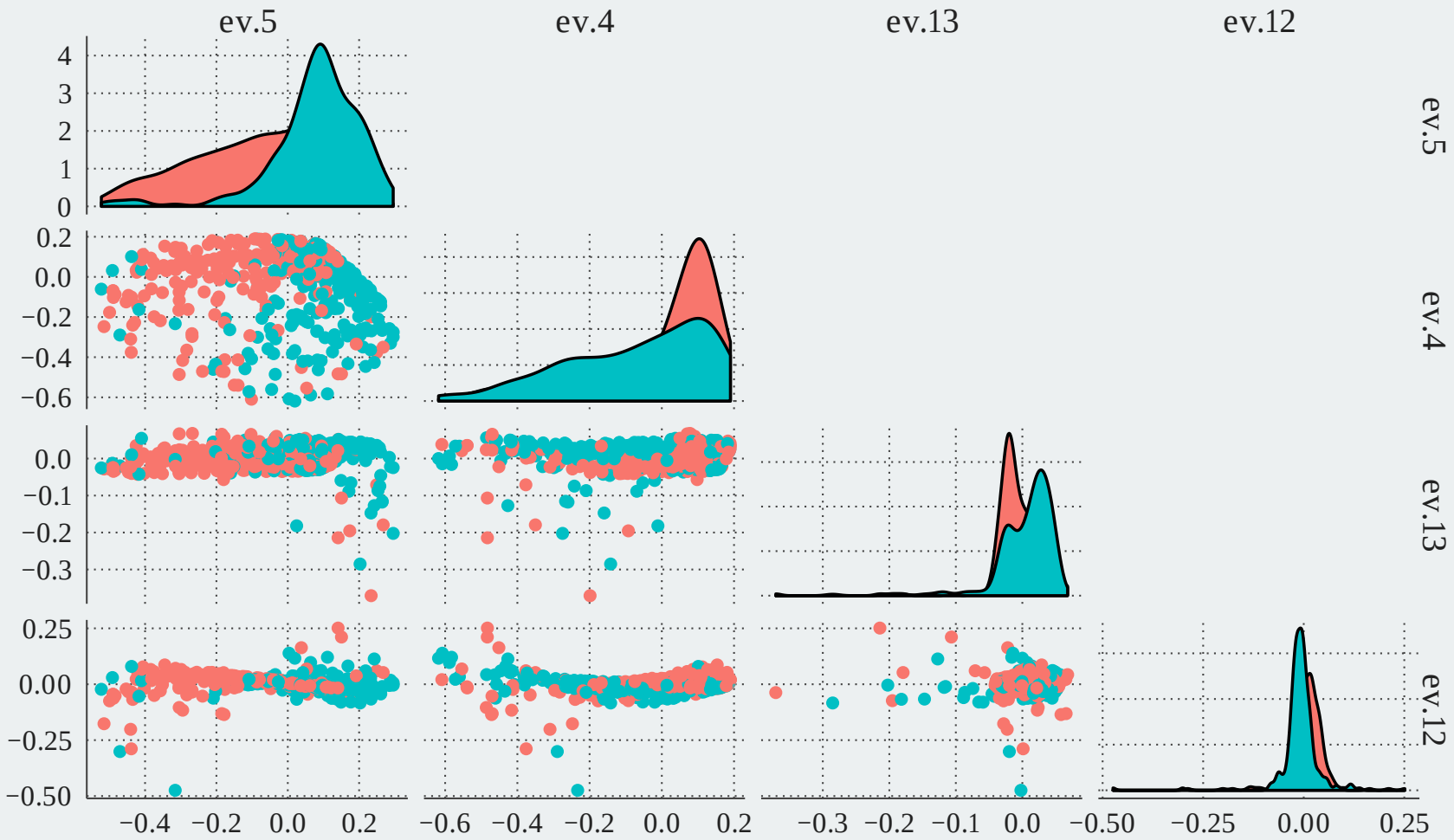


Best possible linear decision rule after transformation (unknown truth)



Bayes accuracy: 0.774

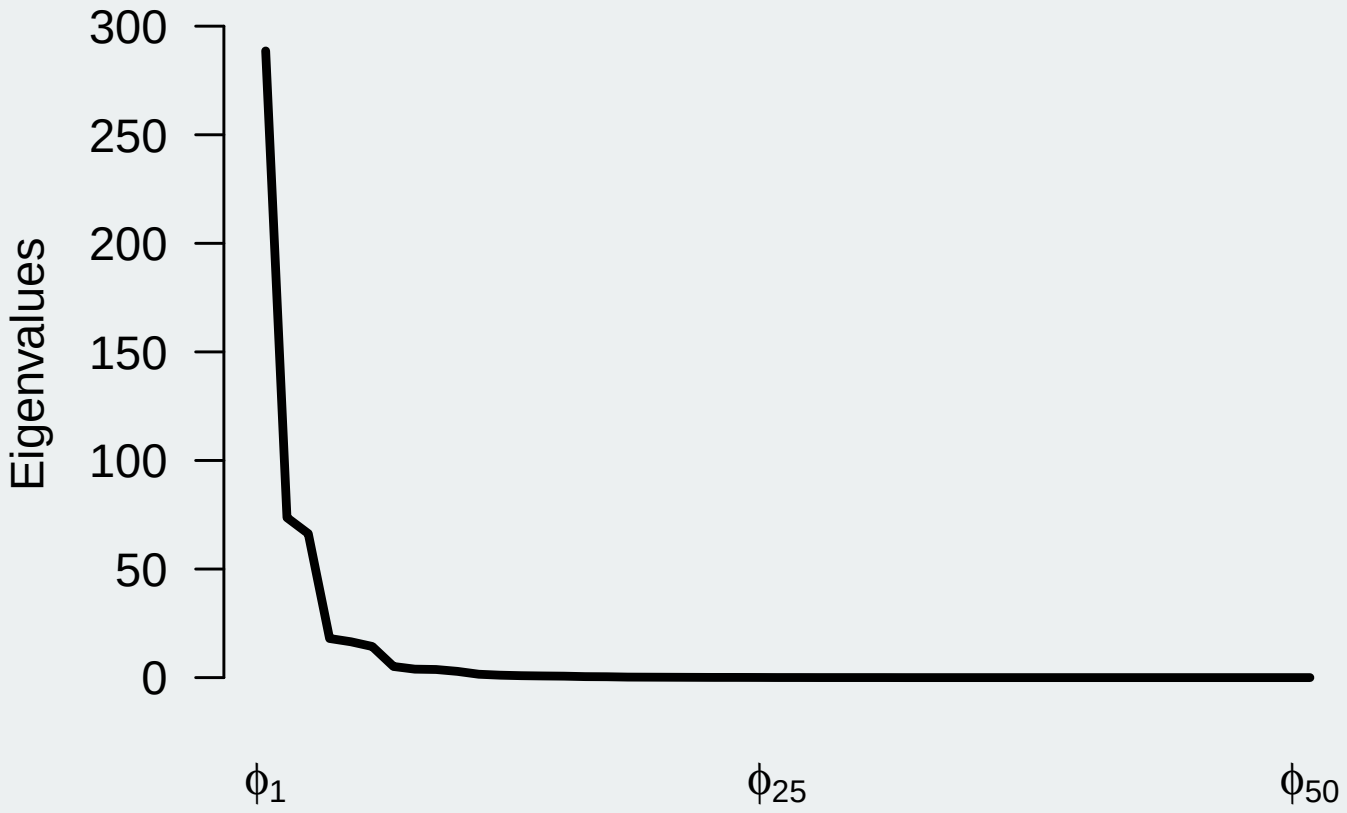
Projection using `radial` kernel



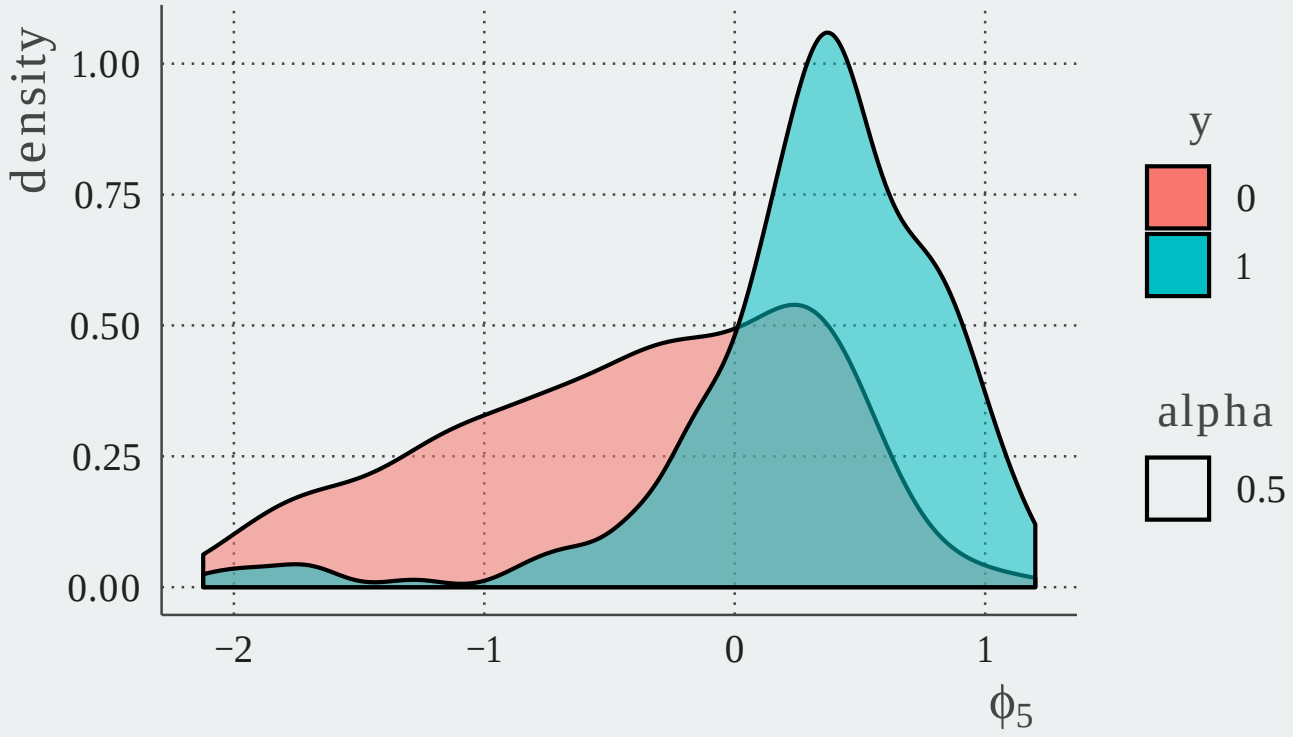
SVM radial basis kernel accuracy: 0.754 (train $n = 1000$)

Projection using `radial` kernel

- "Features" are scaled eigenvectors of Gram (similarity) matrix $\mathbf{K}_{n \times n}$
- With n observations, there are n "features" after applying the kernel trick $\longrightarrow$ potentially *infinite-dimensional* feature space...
- But their eigenvalues drop off faster than n increases $\longrightarrow$ reduced *effective* number of dimensions



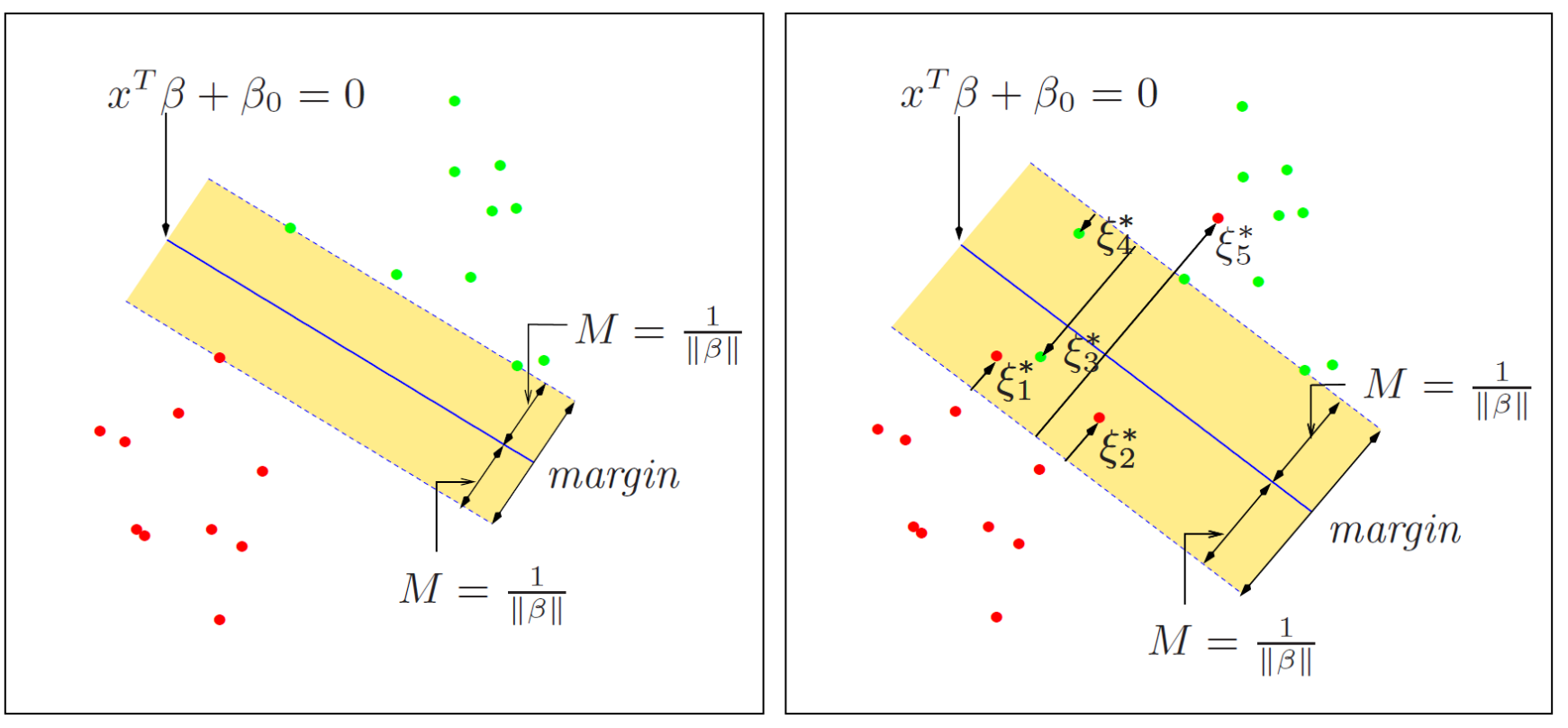
Projection using `radial` kernel



- In example, ϕ_5 correlates 0.8 with true linear predictor, $\sin(x_1 x_2)$.
- Just the decision rule "$\text{sign}(\phi_5)$" gives accuracy **0.73**.

$\rightarrow$ implicit projection of `radial` kernel closely approximates true nonlinear decision function!

Linear classifiers



[Hastie et al. 2017]

Classification

Predict $y \in \{\text{spam, not spam}\} \to \{-1, +1\}$ from p-vector of features $\mathbf{x}$. Both are observed from unknown data-generating process $D(\mathbf{x}, y)$.

Let's choose some "hypothesis set" $\mathcal{F}$ of possible prediction functions and a **loss** indicating how badly members of this set deviate from y.

Under this loss, the best possible choice from the hypothesis set is

$$\hat{f}^*(\mathbf{x}) = \min_{f \in \mathcal{F}} E_{(\mathbf{x},y)\sim D}\left[\text{loss}(y, f(\mathbf{x}))\right]$$

For regression, the loss is a function of the **residual** $y - f(\mathbf{x})$. For classification, we can measure the incorrectness of the prediction as $yf(\mathbf{x})$.

Classification, continued

In practice we can't observe the true "risk", $Q := E_{(\mathbf{x},y)\sim D}\left[\text{loss}(y, f(\mathbf{x}))\right]$.
We only observe a **sample**, $S = \mathcal{D}^n$.
We then work with the **empirical risk**,

$$Q_{\text{train}} := \sum_{i \in S} \text{loss}(y_i, f(\mathbf{x}_i))$$

A key problem of ML is that $E_{\mathcal{D}}(Q_{\text{train}}) \leq Q$, i.e. the training loss is optimistic.
This means minimizing Q_{train} to estimate f would lead to **overfitting**.
A common solution is to **regularize** the objective:

$$\hat{f}_S(\mathbf{x}) \quad = \quad \min_{f \in \mathcal{F}} \left[Q_{\text{train}} + \lambda \|f\|_{\mathcal{F}} \right],$$

where the second term penalizes the "capacity" of the function f.
Support vector machines also take this "loss + penalty" form.

Linear decision boundary

$$f(\mathbf{x}) = \mathbf{x}^T\beta + \beta_0$$

Remember: $yf(\mathbf{x})$ is the ”residual” incorrectness of classifcations

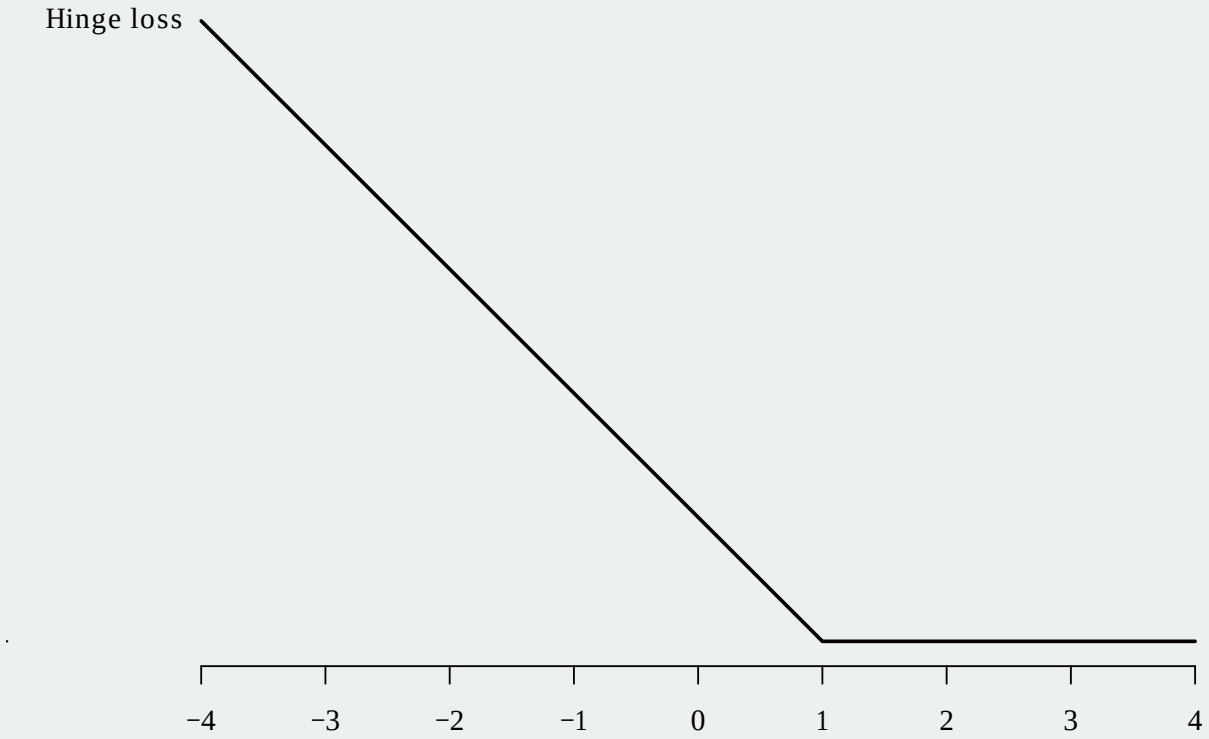
SVCs, loss + penalty formulation

$$\hat{f}_S(\mathbf{x}) = \hat{\beta}_0 + \mathbf{x}^T\hat{\beta}$$

with

$$\hat{\beta_0}, \hat{\beta} = \min_{\beta_0,\beta} \left[\sum_{i \in S} \left(1 - y_i f(\mathbf{x}_i)\right)_+ + \frac{\lambda}{2}||\beta||^2 \right],$$

where $(.)_+$ gives positive part of input ("relu"). Loss $(1 - yf)$ is "hinge loss":



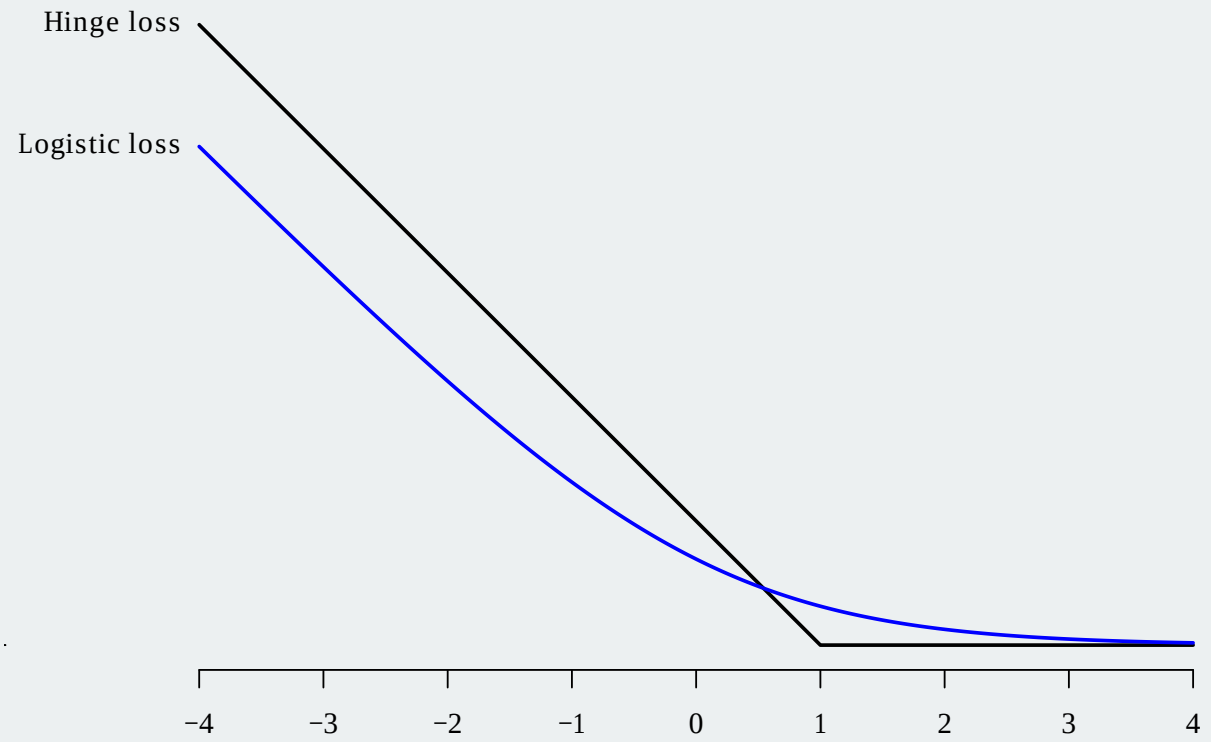
SVC and logistic regression are similar

$$\hat{f}_S(\mathbf{x}) = \hat{\beta}_0 + \mathbf{x}^T\hat{\beta}$$

with

$$\hat{\beta_0}, \hat{\beta} = \min_{\beta_0,\beta} \left[\sum_{i \in S} \left(1 + \exp[-y_i f(\mathbf{x}_i)]\right) + \frac{\lambda}{2}||\beta||^2 \right],$$

Loss $(1 + \exp(yf))$ is "logistic deviance" loss:



SVC vs. logistic regression

- **Hinge loss** gives exact zeroes for all training observations that are correctly classified by the model;

  $\rightarrow$ only need to remember observations on or beyond margin, the "**support vectors**".

  Some robustness to "inliers"

- **Logistic loss** is optimized when f gives the **log-odds ratio**, which can be converted easily into a probability $\rightarrow$ "self-calibrating"

  $\hat{f}_S(\mathbf{x}) \xrightarrow{p} \ln\left[P(y=+1|\mathbf{x})/P(y=-1|\mathbf{x})\right]$

  ("proper scoring rule")

- All observations play a role, less computationally efficient prediction, less sensitive to shifts in wrong classifications than SVM

[Hastie et al. 2017]

Kernelized SVC = SVM

$$f(\mathbf{x}_i) = \beta_0 + \mathbf{K}_{i,:}(\alpha \odot \mathbf{y}),$$

where $\odot$ is the elementwise ("Hadamard") product. Objective is then

$$\hat{\beta_0}, \hat{\alpha} = \min_{\beta_0, \alpha} \left[\sum_{i \in S} \left(1 - y_i f(\mathbf{x}_i)\right)_+ + \frac{\lambda}{2} \alpha^T \mathbf{K} \alpha \right],$$

where α is an n-vector of weights. Notice we only need the Gram matrix $\mathbf{K}$ and the training outcomes $\mathbf{y}$.

Due to the hard break in the hinge loss, most elements of $\hat{\alpha}$ will equal zero. Nonzero elt's of $\hat{\alpha}$ correspond to training observations that are **support vectors**.

Kernel logistic regression: why not

$$f(\mathbf{x}_i) = \beta_0 + \mathbf{K}_{i,:}(\alpha \odot \mathbf{y}),$$

$$\hat{\beta_0}, \hat{\alpha} = \min_{\beta_0, \alpha} \left[\sum_{i \in S} \left(1 + \exp[-y_i f(\mathbf{x}_i)]\right) + \frac{\lambda}{2} \alpha^T \mathbf{K} \alpha \right],$$

Support vector machines, classical formulation (SVC)

$$\min_{\beta,\beta_0} ||\beta||$$

subject to $y_i f(\mathbf{x}) = y_i(\mathbf{x}_i^T \beta + \beta_0) \geq 1, \ i \in S.$

This is the same as maximizing the "dual":

$$\max_{\alpha} \left[\sum_{i \in S} \alpha_i - \frac{1}{2} \sum_{i \in S} \sum_{i' \in S} \alpha_i \alpha_{i'} y_i y_{i'} \mathbf{x}_i^T \mathbf{x}_{i'} \right],$$

subject to $0 < \alpha_1 < 1$ and $\sum_i \alpha_i y_i = 0$, where the α_i are n Lagrange multipliers. This problem can be solved with standard quadratic programming software. We'll then get

$$\beta = \mathbf{x} \sum_{i \in S} \alpha_i y_i \mathbf{x}_i$$

The important thing to note is that x enters the algorithm only through the casewise **inner product**, $\mathbf{x}_i^T \mathbf{x}$.

Conclusion

- SVM is still alive and kicking
- SVM needs careful tuning
- Kernels are interesting, especially with structure
- Lots of things besides the hinge loss linear classifier can be kernelized