Advanced course in machine learning
582744
Lecture 8

Arto Klami

Problem 1

Easier to write using $z_{nk} = [0, ..., 0, 1, 0, ..., 0]$

Complete data log-likelihood is

$$\log p(X, Z|\pi, \mu) = \sum_n \sum_k z_{nk} \log (\pi_k p(x_n|\mu_k))$$

$$= \sum_n \sum_k z_{nk} \left(\log \pi_k + \sum_d x_{ni} \log \mu_{ki} + (1 - x_{ni}) \log(1 - \mu_{ni}) \right)$$

and the expected complete data log-likelihood just takes expectation over z_n

The E-step is the same for all mixture models: Just normalize the joint likelihoods

$$P(z_{nk} \mid x_n, \pi, \mu) = \frac{P(x_n, z_{nk} \mid \pi, \mu)}{p(x_n \mid \pi, \mu)}$$

$$= \frac{\pi_k \prod_{d=1}^D \mu_{kd}^{x_{nd}} (1 - \mu_{kd})^{1 - x_{nd}}}{\sum_{k=1}^K \pi_k \prod_{d=1}^D \mu_{kd}^{x_{nd}} (1 - \mu_{kd})^{1 - x_{nd}}} \equiv r_{nk}$$

Problem 1

During the M-step we assume r_{nk} to be constant, plugging it in place of $\mathbb{E}[z_{nk}]$

μ is quite simple:

$$\nabla_{\mu_i} \mathbb{E}_z[\log p(X \mid \pi, \mu)] = \sum_{n=1}^{N} r_{nk} \left(\frac{x_{ni}}{\mu_{ki}} - \frac{1 - x_{ni}}{1 - \mu_{ki}} \right)$$

and setting it to zero gives

$$\mu_{ki} = \frac{\sum_{n=1}^{N} x_{ni} r_{nk}}{\sum_{n=1}^{N} r_{nk}}$$

Problem 1

For π we also need to take into account the constraint:
$\nabla_\pi \mathbb{E}_z[\log p(X \mid \pi, \mu)] - \lambda(\sum_k \pi_k - 1) = 0$ gives

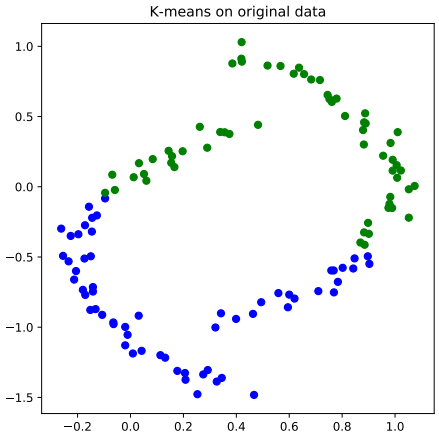
$$\frac{\sum_n r_{nk}}{\pi_k} = \lambda \rightarrow \pi_k = \frac{\sum_n r_{nk}}{\lambda}$$

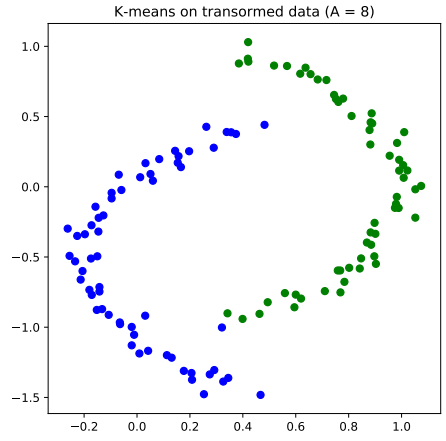
and plugging in the constraint $\sum_k \pi_k = 1$ simplifies it into

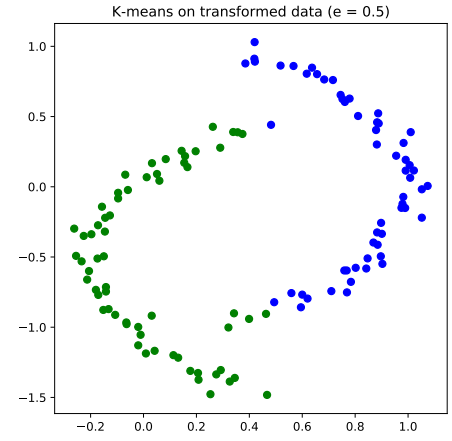
$$\pi_k = \frac{\sum_n r_{nk}}{\sum_j \sum_n r_{nj}}$$

This is identical for every mixture model and quite obvious anyway

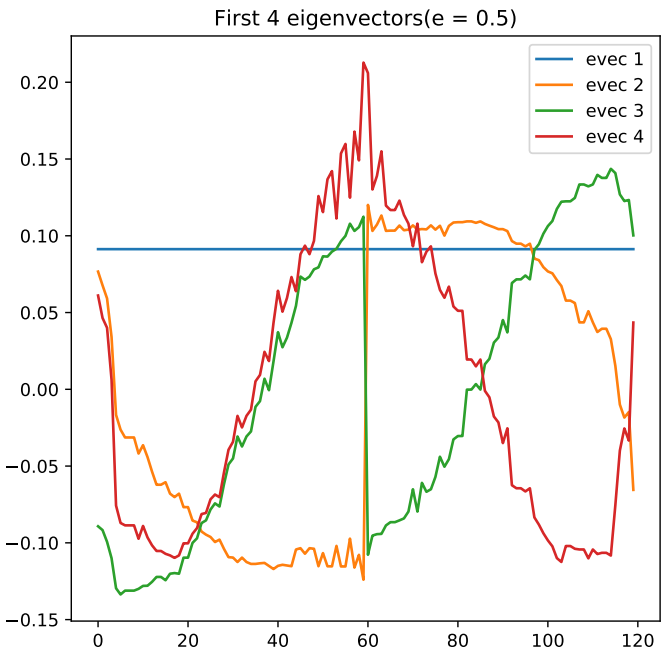
Problem 2

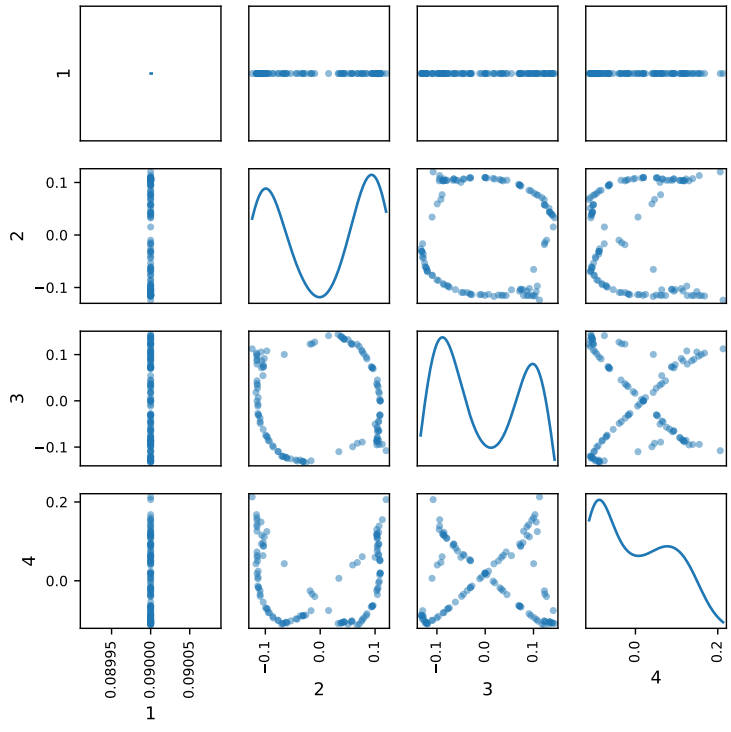

K-means on original data
1.0
0.5
0.0
−0.5
−1.0
−1.5
−0.2
0.0
0.2
0.4
0.6
0.8
1.0


K-means on transormed data (A = 8)
1.0
0.5
0.0
−0.5
−1.0
−1.5
−0.2
0.0
0.2
0.4
0.6
0.8
1.0


K-means on transformed data (e = 0.5)
1.0
0.5
0.0
−0.5
−1.0
−1.5
−0.2
0.0
0.2
0.4
0.6
0.8
1.0

Problem 2





Why the first eigenvector is constant?

Problem 2

What happens with small ϵ or A?

How about too large?

Why did we use $M = 4$?

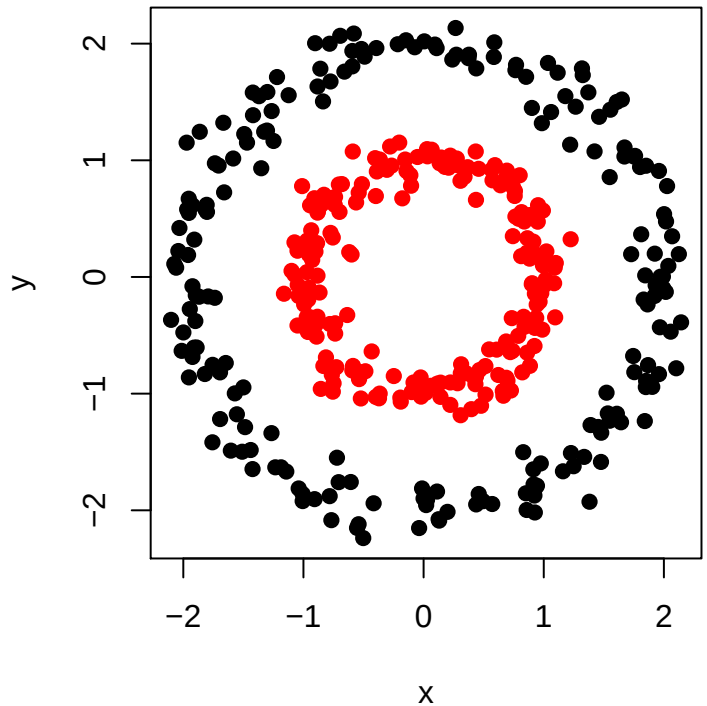
Problem 3

Just plug in the definitions. The latter step trivially results in unit norm, so it is enough to check that the solution is orthogonal wrt any previous $\mathbf{w}_l$

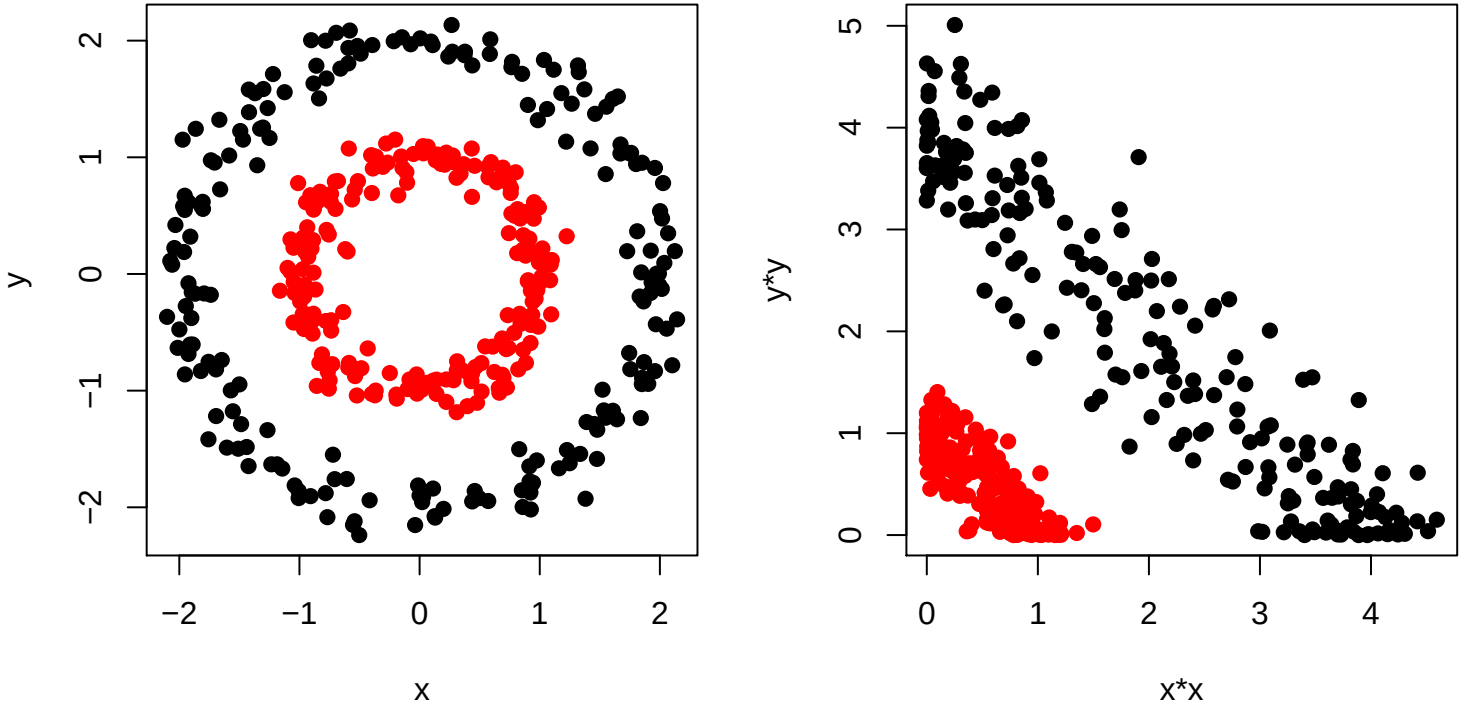
$$\begin{aligned}\mathbf{w}_l^T \mathbf{v}_i &= \mathbf{w}_l^T (\mathbf{w}_i - \sum_{j \leq k} (\mathbf{w}_i^T \mathbf{w}_j) \mathbf{w}_j) \\ &= \mathbf{w}_l^T \mathbf{w}_i - \sum_{j \leq k} (\mathbf{w}_i^T \mathbf{w}_j)(\mathbf{w}_l^T \mathbf{w}_j)\end{aligned}$$

Since $\mathbf{w}_l^T \mathbf{w}_j = 0$ for $l \neq j$ and $\mathbf{w}_l^T \mathbf{w}_l = 1$ for all previously found components, the only non-zero term in the sum is $\mathbf{w}_i^T \mathbf{w}_l$

Are linear classifiers enough?



Are linear classifiers enough?



Non-linear classifiers by preprocessing the inputs

Cover's theorem: Higher-dimensional representations are more often linearly separable

...so let's simply make some such transformation, for example, $\phi(\mathbf{x}) = [\mathbf{x}, \mathbf{x}^2, \mathbf{x}^3]$ and then use $y_n = \mathbf{w}^T \phi(\mathbf{x}_n)$

Nasty side-effect: The dimensionality of the representation grows

Strong regularization and sparsity can help to avoid overfitting, but the computational cost is still high (unless we get real sparsity)

Can we get around the high dimensionality?

Think of a simple nearest-neighbor classifier that computes distances between samples

$$(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_i + \mathbf{x}_j^T\mathbf{x}_j - 2\mathbf{x}_i^T\mathbf{x}_j$$

We see they depend only on inner products of the vectors

The *kernel trick* is a magic trick that allows computing inner products corresponding to some feature maps $\phi(\mathbf{x})$ quickly, independent of the dimensionality

We collect the inner products into *kernel matrix*: $k_{ij} = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$

Simple kernels

- $\phi(\mathbf{x}) = \mathbf{x}$: $k_{ij} = \mathbf{x}_i^T \mathbf{x}_j$ (linear kernel)
- $\phi(\mathbf{x}) = [1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2\sqrt{2}x_1x_2]$: $k_{ij} = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2$ (polynomial kernel)
- $\phi(\mathbf{x}) =$???: $k_{ij} = e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_h\|^2}{2\sigma^2}}$ (Gaussian/RBF kernel)

Take a close look at the polynomial kernel: The kernel computation is based on an inner product in the original two-dimensional space whereas the innner product of the feature maps operates in six-dimensional space

Mercer kernels

A kernel should be positive definite (positive eigenvalues) to correspond to inner products between some feature maps

$$k = U^T \Lambda U \rightarrow k_{ij} = (\Lambda^{1/2} U_{:,i})^T (\Lambda^{1/2} U_{:,j}) \rightarrow \phi(\mathbf{x}_i) = \Lambda^{1/2} U_{:,i}$$

...but it is not necessarily easy to find the mapping that corresponds to the kernel; the above expression only tells what the output is for a given sample and it depends on the set of samples – for the exact definition of $\phi(\mathbf{x}_i)$ we would need the eigenfunctions of the kernel

For Gaussian kernel $\phi(\mathbf{x})$ is actually infinite-dimensional

More advanced kernels

Very often the linear kernel or the Gaussian kernel are the default choices

However, we can also design dedicated kernels for specific types of data. The book lists some in Section 14.

1. TF-IDF for comparing text documents
2. String kernels that compute how many substrings two string have in common
3. Pyramid match kernels for images
4. Fisher kernel that converts a generative model into kernels

Computing with kernels

If your loss function is directly expressed as inner products, then just plug in an arbitrary kernel

If not, spend some time thinking if it could be represented that way – it is not always easy, and often we end up studying the solution of the problem instead of the loss

The basic idea is to use *dual variables* $\boldsymbol{\alpha}$ such that the *primal variables* $\mathbf{w}$ can be expressed as

$$\mathbf{w} = \mathbf{X}\boldsymbol{\alpha} = \sum_n \alpha_n \mathbf{x}_n$$

Kernelized ridge regression

Remember the solution

$$\mathbf{w} = (\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_D)^{-1}\mathbf{X}\mathbf{y}$$

for the ridge regression

Matrix inversion lemma tells this is equivalent to

$$\mathbf{w} = \mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_N)^{-1}\mathbf{y}$$

and now we can define

$$\boldsymbol{\alpha} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_n)^{-1}\mathbf{y}$$

as the dual variables, making the solution of the form $\mathbf{w} = \mathbf{X}\boldsymbol{\alpha}$

Predictions with the model are then

$$\mathbf{w}^T\mathbf{x} = \sum_n \alpha_n \mathbf{x}_n^T\mathbf{x} = \sum_n \alpha_n k(\mathbf{x}_n, \mathbf{x})$$

and hence only depend on inner product between the test sample and the training samples

Kernelized representations

Not happy with the matrix inversion lemma?

Start by directly assuming $\mathbf{w} = \sum_j \alpha_j \mathbf{x}_j$ and re-write the loss as

$$\sum_n (y_n - \sum_j \alpha_j \mathbf{x}_j^T \mathbf{x}_n)^2$$

to see that it only depends on the inner products

The *representer theorem* states that for *reproducing kernel Hilbert spaces* all functions $f(\mathbf{x})$ that minimize some regularized loss can be represented as

$$f(\mathbf{x}) = \sum_j \alpha_j k(\mathbf{x}_j, \mathbf{x})$$

where k is the reproducing kernel of the space. Hence what we assumed was that we operate in a RKHS

Kernelized ridge regression

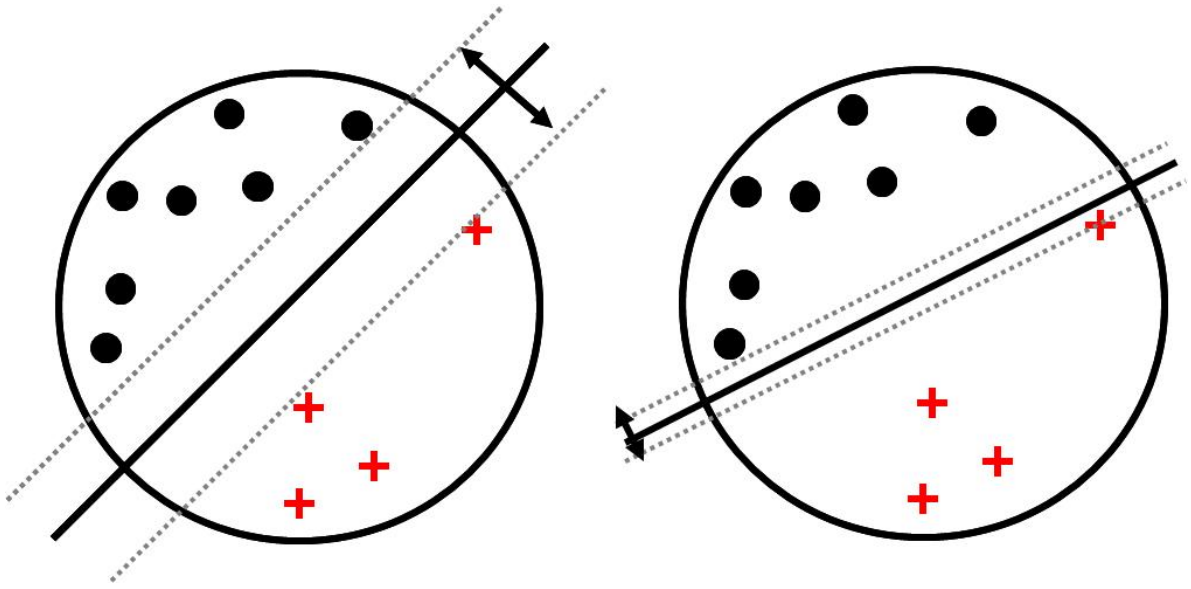
Computing $\boldsymbol{\alpha}$ requires inverting a matrix that is of size $N \times N$, compared to $D \times D$ for the primal problem

This is slow for large training data, whereas the primal problem is slow for high dimensionality – with high-dimensional implicit feature maps the dual problem is obviously better, but it can still be slow

For $D \gg N$ this is a faster way for solving even linear problems, but the predictions are slower since we need inner products between the test sample and all training samples (ND vs just D for $\mathbf{w}^T\mathbf{x}$)

With linear models we could get away without regularization, but with infinite-dimensional kernels we definitely need regularization

Large-margin principle



Linear classifier with hard decision: If the class labels are $+1$ and -1 then the decision rule "Sample belongs to class $+1$ iff $\mathbf{w}^T\mathbf{x} + b \geq 0$" can be re-written as $(\mathbf{w}^T\mathbf{x} + b)y \geq 0$

The *margin* is the distance to the separating hyperplane, given by $\gamma_n = (\mathbf{w}^T\mathbf{x}_n + b)y_n$

Maximum margin classifier

For linearly separable data we can maximize the smallest margin $\gamma = \min_n \gamma_n$, normalized by $\|\mathbf{w}\|$ since otherwise we get infinite margins by increasing the norm

The optimization problem is hence

$$\begin{aligned} &\max \frac{\gamma}{\|w\|} \\ \text{s.t.} \quad &(\mathbf{w}^T \mathbf{x}_n + b) y_n \geq \gamma \end{aligned}$$

which is equivalent to

$$\begin{aligned} &\min \frac{1}{2} \|w\|^2 \\ \text{s.t.} \quad &(\mathbf{w}^T \mathbf{x}_n + b) y_n \geq 1 \end{aligned}$$

Maximizing the margin is equivalent to minimizing the norm of the projection vector under constraint of at least unit margin

Kernelizing the maximum margin classifier

The constrained optimization problem

$$\min \frac{1}{2}\|w\|^2$$
$$\text{s.t.} \quad (\mathbf{w}^T\mathbf{x}_n + b)\mathbf{y}_n \geq 1$$

can be solved with lagrange multipliers using

$$\min \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_n \alpha_n \left[(\mathbf{w}^T\mathbf{x}_n + b)y_n - 1\right]$$

Taking derivatives wrt to $\mathbf{w}$ and b gives

$$\mathbf{w} = \sum_n \alpha_n \mathbf{x}_n y_n$$
$$\sum_n \alpha_n y_n = 0$$

and shows that the weights are indeed a weighted sum of the samples

Kernelizing the maximum margin classifier

Plugging the above in to the loss function gives

$$\min \sum_i \boldsymbol{\alpha}_i - \frac{1}{2} \sum_{i,j} \boldsymbol{\alpha}_i \boldsymbol{\alpha}_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{s.t.} \quad \boldsymbol{\alpha}_i \geq 0 \quad \forall i$$

$$\sum_i \boldsymbol{\alpha}_i y_i = 0$$

The optimization problem only involves inner products so it can be kernelized, and we are minimizing $\boldsymbol{\alpha}_i$ that are all non-negative (because they are Lagrange multipliers) – intuitively this tries to make them zero

The predictions are given by

$$\mathbf{w}^T \mathbf{x} + b \geq 0 \equiv \sum_i \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b \geq 0$$

where we only need to sum over the *support vectors*, the samples with $\alpha_i > 0$

Non-separable data

What if the data is not separable? Then we cannot satisfy the constraints

The solution is to add *slack variables* $\xi_i \geq 0$ that measure the violation and minimize over them

$$\min \frac{1}{2}\|w\|^2 + C\sum_n \xi_n$$

$$\text{s.t.} \quad (\mathbf{w}^T\mathbf{x}_n + b)\mathbf{y}_n \geq 1 - \xi_n$$

$$\xi_n \geq 0$$

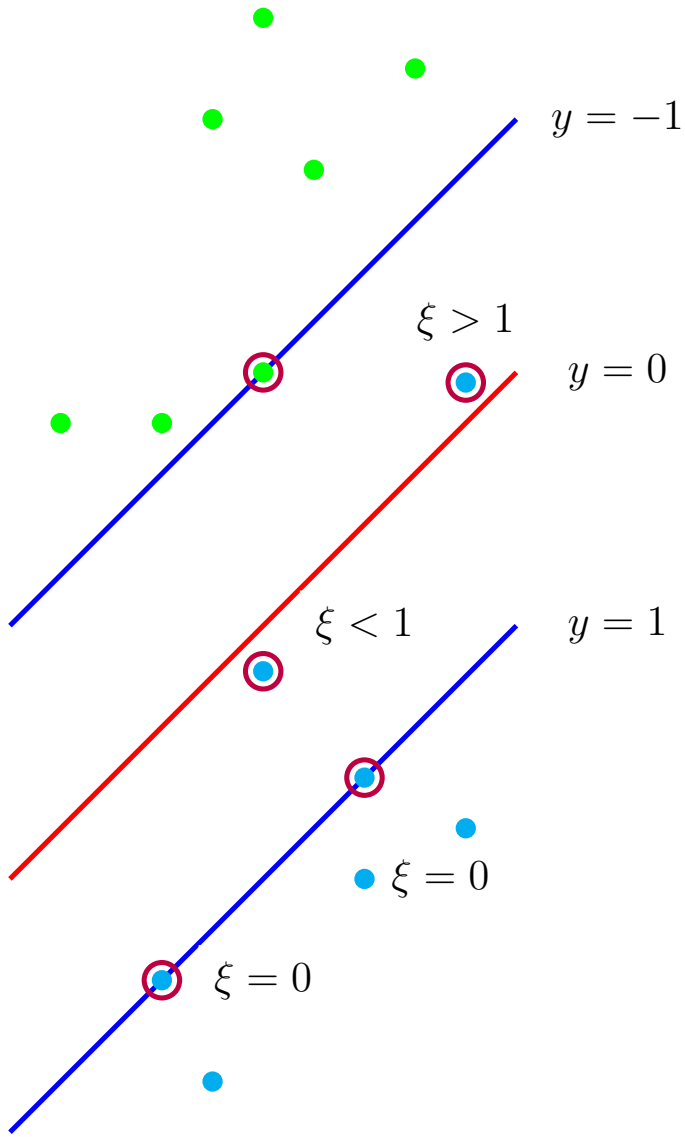
The corresponding dual formulation becomes

$$\min \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C \quad \forall i$$

$$\sum_i \alpha_i y_i = 0$$

Support vector machine



Support vector machine

The optimization problem presented on the previous slide corresponds to *support vector machine*, which is rather powerful classifier

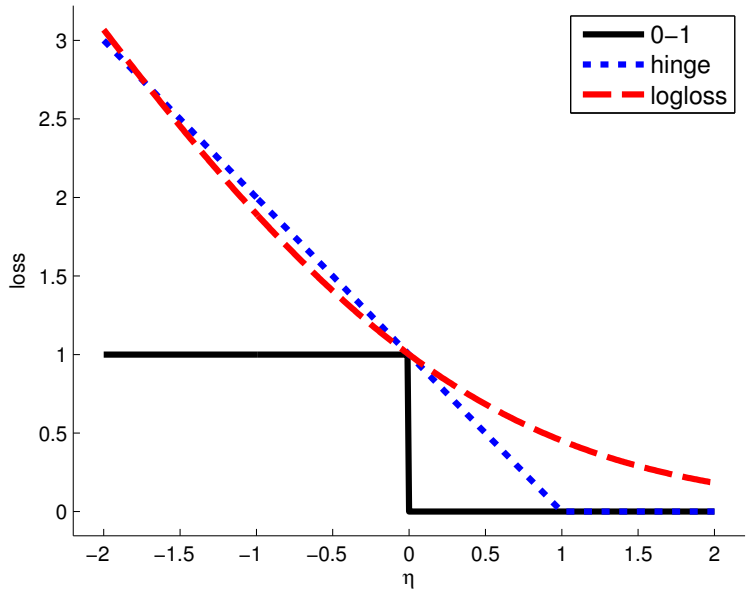
The parameter C can be interepreted as regularization, even though it was defined as the weight for margin violations.

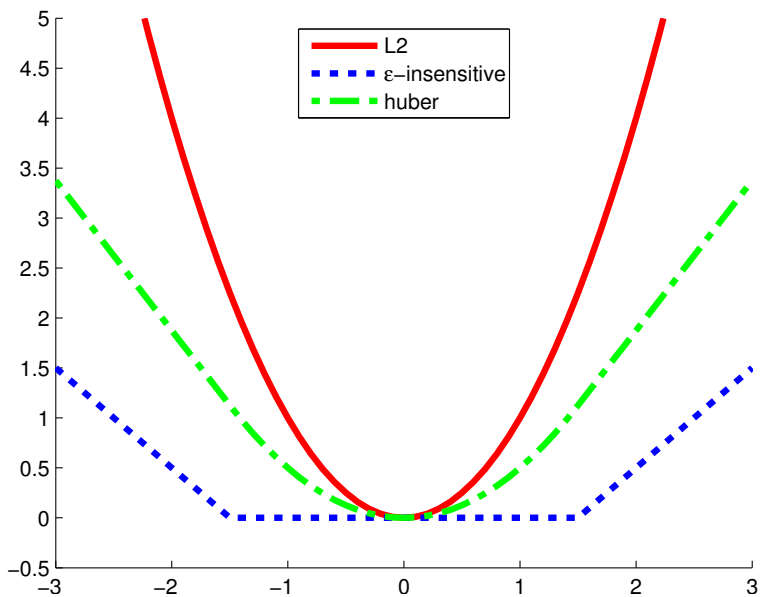
The weights $\boldsymbol{\alpha}$ are zero for all samples on the correct side of the margin, and non-zero only for samples lying on the margin or that have non-zero slack variables

Alternative viewpoint

SVM can alternatively be interepreted as mimizing the *hinge loss* that penalizes already for violations of the marging

Similarly, support vector regression is obtained by minimizing ϵ-insensitive loss that does not penalize at all for small error





Solving the SVM problem

We could use generic constrained optimization packages, but dedicated algorithms are often faster. Coordinate ascent is a good algorithm for problems with constraints for individual variables

However, we cannot change any of the $\boldsymbol{\alpha}$ alone since $\sum_n \boldsymbol{\alpha}_n y_n = 0$

Sequential minimal optimization (SMO) updates pairs $(\boldsymbol{\alpha}_i, \boldsymbol{\alpha}_j)$ at a time, keeping all others fixed:

$$\boldsymbol{\alpha}_i y_i + \boldsymbol{\alpha}_j y_j = -\sum_{k \neq i,j} \boldsymbol{\alpha}_k y_k = \eta$$

and hence their relationship is linear

$$\boldsymbol{\alpha}_i = (\eta - \boldsymbol{\alpha}_j y_j) y_i$$

Now we optimize wrt to $\boldsymbol{\alpha}_j$, which is a quadratic function, remembering that both $\boldsymbol{\alpha}_j$ and $\boldsymbol{\alpha}_i$ have to stay within $[0, C]$

SVM in practice

- Pick a suitable kernel. Perhaps use the Gaussian kernel as the default choice, but remember that you need to pick σ^2 as well
- Use SMO for learning and choose the regularization parameter C by cross-validation
- Note that C and σ^2 depend on each other – narrow kernels require more regularization
- Hence, you might want to choose both C and σ^2 with CV
- Multiclass classification with one-vs-rest or one-vs-one

Probabilistic interpretation?

The outputs of SVM are not probabilities, but we do get some certainty estimates by looking at the margin of invididual samples

One could attempt converting them into numbers that look like probabilities, but the results are not properly calibrated

Section 14.5.5 sketches a probabilistic interepretation for SVM, by interpreting the hinge loss as a Gaussian scale mixture, making EM a feasible algorithm for solving SVMs

Gaussian processes (Section 15; not covered on the course) are a more reasonable probabilistic alternative for kernel-based regression and classification – for example, we can then choose the kernel parameters based on marginal likelihood