KERNEL TRICK AND RVMS

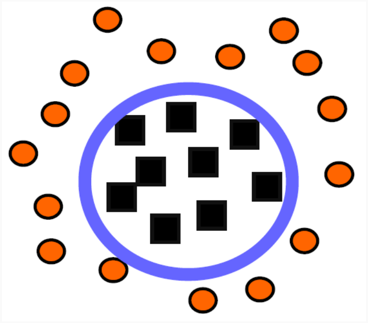
Sergey Nikolenko

Harbour Space University, Barcelona, Spain
March 22, 2017

SVM AND NONLINEAR FUNCTIONS

NONLINEAR FUNCTIONS

- Often we have to use nonlinear functions for separation.
- What do we do?



NONLINEAR FUNCTIONS

- We already know the answer: use linear classification in a space of larger dimension (feature space), which we obtain by adding nonlinear features.
- E.g., to get a polynomial surface we introduce a new variable for each monomial of the corresponding degree.

EXAMPLE

- E.g., to get quadratic functions in two-dimensional space $[r, s]$, we pass to a five-dimensional space:

$$[r, s] \longrightarrow [r, s, rs, r^2, s^2].$$

- Formally, we define $\theta : \mathbb{R}^2 \to \mathbb{R}^5$: $\theta(r, s) = (r, s, rs, r^2, s^2)$. The classification function is now

$$f(\vec{x}) = \mathrm{sign}(\theta(\vec{w}) \cdot \theta(\vec{x}) - b).$$

- Linear separation in this new space corresponds to quadratic separation in the original space.

PROBLEMS WITH THIS APPROACH

- First, the number of variables grows exponentially.
- Second, overfitting becomes a problem again.
- But note that *in essence* we are done. Only *technical* problems remain: how do we handle the huge dimension?

MAIN IDEA OF THE KERNEL TRICK

- The original scheme of SVM operation is as follows:
    - input vector $\vec{x}$ is transformed by θ to an input vector in the (very high dimensional) feature space;
    - in this large space we compute support vectors and solve the linear separation problem;
    - then classify the input vector with this problem.
- This is impossible to do directly: the dimension is too large.

MAIN IDEA OF THE KERNEL TRICK

- But it turns out that certain steps here can be swapped:
    - compute support vectors in the original low-dimensional space;
    - multiply them there (we'll see what it means shortly);
    - and only then make a linear transformation of the result to classify a new input vector.
- Wtf? :)

PROBLEM SETTING

- We remind that the problem is

$$\min_{\alpha}\left\{\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m}y_iy_j\alpha_i\alpha_j\left(\vec{x}_i\cdot\vec{x}_j\right)-\sum_{i=1}^{m}\alpha_i,\right.$$

$$\left.\text{where }\sum_{i=1}^{m}y_i\alpha_i=0,\quad 0\leq\alpha_i\leq C.\right\}$$

PROBLEM SETTING

- We now want to introduce a mapping $\theta : \mathbb{R}^n \to \mathbb{R}^N$, $N > n$. We get:

$$\min_{\alpha} \left\{ \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \left(\theta(\vec{x}_i) \cdot \theta(\vec{x}_j) \right) - \sum_{i=1}^{m} \alpha_i, \right.$$

$$\left. \text{where } \sum_{i=1}^{m} y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C. \right\}$$

HILBERT--SCHMIDT--MERCER'S THEORY

- Let's recall a bit of functional analysis.
- We want to generalize the notion of a *scalar product*, introduce a new function that will compute the scalar product of vectors in the feature space directly:

$$k(\vec{u}, \vec{v}) := \theta(\vec{u}) \cdot \theta(\vec{v}).$$

HILBERT--SCHMIDT--MERCER'S THEORY

- First result: any symmetric function $k(\vec{u}, \vec{v}) \in L_2$ can be represented as

$$k(\vec{u}, \vec{v}) = \sum_{i=1}^{\infty} \lambda_i \theta_i(\vec{u}) \cdot \theta_i(\vec{v}),$$

  where $\lambda_i \in \mathbb{R}$ are eigenvalues, and θ_i are eigenvectors of the integral operator with kernel k, i.e.,

$$\int k(\vec{u}, \vec{v}) \theta_i(\vec{u}) \mathrm{d}\vec{u} = \lambda_i \theta_i(\vec{v}).$$

HILBERT--SCHMIDT--MERCER'S THEORY

- In order for k to define a scalar product, it suffices that its eigenvalues are all positive.
- Eigenvalues are positive iff (*Mercer's theorem*)

$$\int\int k(\vec{u},\vec{v})g(\vec{u})g(\vec{v})\mathrm{d}\vec{u}\mathrm{d}\vec{v} > 0$$

  for all g such that $\int g^2(\vec{u})\mathrm{d}\vec{u} < \infty$.
- And that's all. Now we can instead of computing $\theta(\vec{u}) \cdot \theta(\vec{v})$ simply use a suitable *kernel* $k(\vec{u},\vec{v})$ in the quadratic programming problem.

HILBERT--SCHMIDT--MERCER'S THEORY

- Thus, the problem now looks as follows:

$$\min_{\alpha}\left\{\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j k(\vec{x}_i, \vec{x}_j) - \sum_{i=1}^{m}\alpha_i, \right.$$
$$\left. \text{where } \sum_{i=1}^{m} y_i\alpha_i = 0, \quad 0 \le \alpha_i \le C. \right\}$$

- By simply changing the kernel k, we can compute very different separating surfaces.
- Conditions for k to be a suitable kernel are given by Mercer's theorem.

SAMPLE KERNELS

- Consider the kernel

$$k(\vec{u}, \vec{v}) = (\vec{u} \cdot \vec{v})^2.$$

- Which feature space does it correspond to?

SAMPLE KERNELS

- We get that

$$k(\vec{u},\vec{v}) = (\vec{u}\cdot\vec{v})^2 =$$
$$= \left(u_1^2, u_2^2, \sqrt{2}u_1u_2\right)\cdot\left(v_1^2, v_2^2, \sqrt{2}v_1v_2\right).$$

- That is, a linear surface in the new feature space corresponds to a quadratic surface in the original (e.g., an ellipse).

SAMPLE KERNELS

- A natural generalization: kernel $k(\vec{u}, \vec{v}) = (\vec{u} \cdot \vec{v})^d$ defines a space whose axes correspond to all *uniform* monomials of degree d.
- How can we make a space corresponding to an arbitrary polynomial surface, not necessarily uniform?

SAMPLE KERNELS

- Easy:
$$k(\vec{u}, \vec{v}) = (\vec{u} \cdot \vec{v} + 1)^d.$$
- Now linear separation in the feature space exactly corresponds to polynomial separation in the base space.

SAMPLE KERNELS

- Radial basis functions:

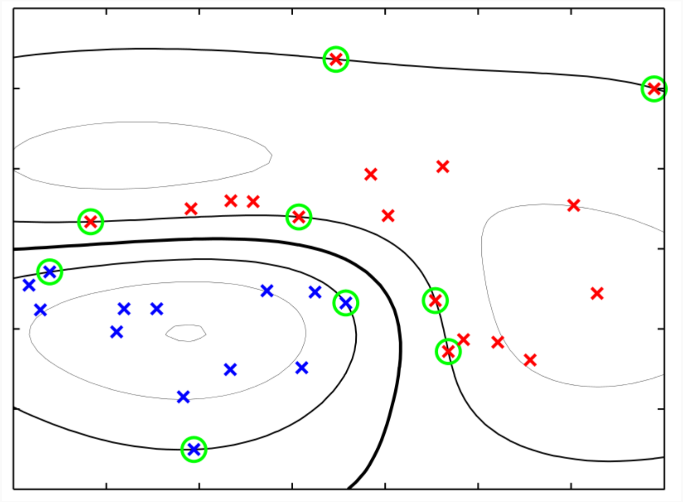
$$k(\vec{u}, \vec{v}) = e^{-\frac{||\vec{u}-\vec{v}||^2}{2\sigma}}.$$

- Two-level perceptron:

$$k(\vec{u}, \vec{v}) = o(\eta \vec{u} \cdot \vec{v} + c),$$

where o is a sigmoid function.

EXAMPLE



RESUME

- Here is the algorithm we get in the end.
    1. Choose parameter C, which shows the tradeoff between minimizing error and maximizing margin.
    2. Choose a kernel and its parameters if it has any.
    3. Solve the quadratic programming problem.
    4. By the resulting values of support vectors find w_0 (how?).
    5. Classify new points as

$$f(\vec{x}) = \mathrm{sign}(\sum_i y_i \alpha_i k(\vec{x}, \vec{x}_i) - w_0).$$

IN PRACTICE

- In practice:
    - small C – simpler separating surface, few support vectors;
    - large C – more complex separating surface, lots of support vectors.
- For the RBF kernel:
    - small γ – support vectors have far-reaching influence, the model is simpler;
    - large γ – support vectors influence only near, the model is more complex.

ν-SVM

- Another variant for inseparable data – ν-SVM [Schölkopf et al., 2000].
- We maximize

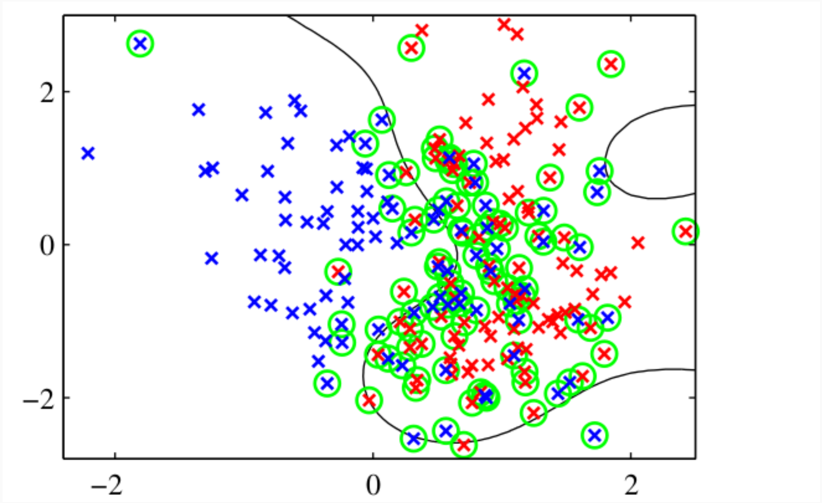
$$L(\mathbf{a}) = -\frac{1}{2}\sum_n\sum_m a_n a_m t_n t_m k\left(\mathbf{x}_n, \mathbf{x}_m\right)$$

  under constraints

$$0 \leq a_n \leq \frac{1}{N},\ \sum_n a_n t_n = 0,\ \sum_n a_n \geq \nu.$$

- Parameter ν can be interpreted as an upper bound on the fraction of errors.

SVM FOR CLASSIFICATION



SVM WITH SEVERAL CLASSES

- How can we generalize SVM to several classes?
- Possibilities (all of them far from perfect):
    - train one against all and classify $y(\mathbf{x}) = \max_k y_k(\mathbf{x})$ (the problem becomes imbalanced, and $y_k(\mathbf{x})$ are actually incomparable);
    - try to construct a single function for all K SVMs, but then training slows down significantly;
    - train $K(K-1)/2$ pairwise classifiers and then count their votes;
    - DAGSVM: organize pairwise classifiers into a graph and classify by walking along paths in this graph;
    - and so on; unfortunately, there is no one true way to get an SVM with several classes.

ONE-CLASS SVM

- On the other hand, SVM can be used with *one* class.
- How and why?

ONE-CLASS SVM

- On the other hand, SVM can be used with *one* class.
- How and why?
- We can encircle a high density region, find the boundary with an SVM.
- And this is how we can find outliers in the data.
- The problem would be to find the smallest surface (e.g., a sphere) that contains all points except fraction ν.

SVM FOR REGRESSION

- SVM can be used for regression, and it will preserve sparsity (i.e., the fact that SVM depends only on support vectors).
- In our common linear regression we minimized

$$\frac{1}{2}\sum_{n=1}^{N}(y_n - t_n)^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2.$$

- In SVM we say that if we are in an ϵ-neighborhood of the correct answer, then the error is zero.

SVM FOR REGRESSION

- ϵ-insensitive error function:

$$E_\epsilon(y(\mathbf{x}) - t) = \begin{cases} 0, & |y(\mathbf{x}) - t| < \epsilon, \\ |y(\mathbf{x}) - t| - \epsilon & \text{otherwise.} \end{cases}$$

- The problem is now to minimize

$$C \sum_{n=1}^{N} E_\epsilon \left(y(\mathbf{x}_n) - t_n \right) + \frac{\lambda}{2} \|\mathbf{w}\|^2.$$

SVM FOR REGRESSION

- To reformulate, we need two slack variables for both sides of the "tube":

$$y(\mathbf{x}_n) - \epsilon \leq t_n \leq y(\mathbf{x}_n) + \epsilon$$

turns into

$$\begin{aligned} t_n &\leq y(\mathbf{x}_n) + \epsilon + \xi_n, \\ t_n &\geq y(\mathbf{x}_n) - \epsilon - \hat{\xi}_n, \end{aligned}$$

and we optimize

$$C \sum_{n=1}^{N} E_\epsilon \left(\xi_n + \hat{\xi}_n \right) + \frac{\lambda}{2} \|\mathbf{w}\|^2.$$

SVM FOR REGRESSION

- The dual problem is now

$$L(\mathbf{a}, \hat{\mathbf{a}}) = -\frac{1}{2} \sum_{n} \sum_{m} (a_n - \hat{a}_n)(a_m - \hat{a}_m) k(\mathbf{x}_n, \mathbf{x}_m) - \\ - \epsilon \sum_{n=1}^{n} (a_n + \hat{a}_n) + \sum_{n=1}^{N} (a_n - \hat{a}_n) t_n,$$

and we minimize it over $a_n, \hat{a}_n$ with conditions

$$0 \le a_n \le C,$$
$$0 \le \hat{a}_n \le C,$$
$$\sum_{n=1}^{N} (a_n - \hat{a}_n) = 0.$$

SVM FOR REGRESSION

- When we solve this problem, we will be able to predict new values as

$$y(\mathbf{x}) = \sum_{n=1}^{N} (a_n - \hat{a}_n)\, k(\mathbf{x}, \mathbf{x}_n) + b,$$

where b can be found as

$$\begin{aligned} b = t_n - \epsilon - \mathbf{w}^\top \phi(\mathbf{x}_n) = \\ = t_n - \epsilon - \sum_{m=1}^{N} (a_m - \hat{a}_m)\, k(\mathbf{x}_n, \mathbf{x}_m). \end{aligned}$$

SVM FOR REGRESSION

- And KKT conditions turn into

$$\begin{aligned} a_n\left(\epsilon + \xi_n + y(\mathbf{x}_n) - t_n\right) &= 0, \\ \hat{a}_n\left(\epsilon + \hat{\xi}_n - y(\mathbf{x}_n) + t_n\right) &= 0, \\ (C - a_n)\xi_n &= 0, \\ (C - \hat{a}_n)\hat{\xi}_n &= 0. \end{aligned}$$

- This implies that either a_n or $\hat{a}_n$ are always 0, and at least one of them is not zero only if the point lies at or beyond the boundary of the “tube”.
- We’ve got a solution that depends only on “support vectors” again.

SVM FOR REGRESSION

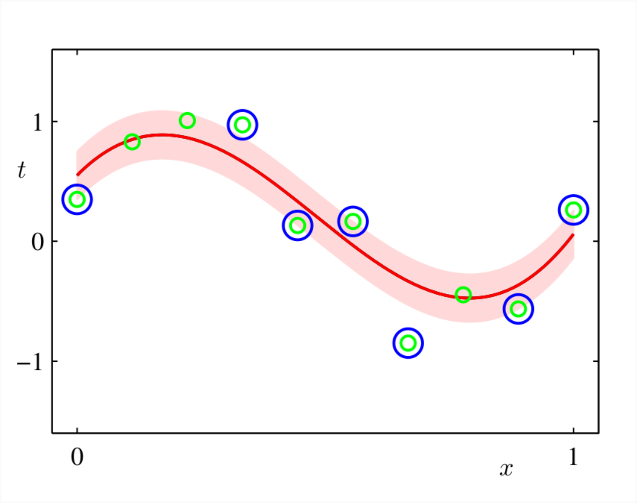
- Again, we can reformulate as ν-SVM, where the parameter is more intuitively clear: instead of the tube width ϵ we consider ν, the fraction of points outside the tube; then we minimize

$$L(\mathbf{a}) = -\frac{1}{2}\sum_{n}\sum_{m}(a_n - \hat{a}_n)(a_m - \hat{a}_m)\,k(\mathbf{x}_n, \mathbf{x}_m) + \sum_{n=1}^{N}(a_n - \hat{a}_n)\,t_n$$

  under constraints

$$\begin{aligned} 0 \le a_n \le \tfrac{C}{N}, &\qquad \textstyle\sum_{n=1}^{N}(a_n - \hat{a}_n) = 0, \\ 0 \le \hat{a}_n \le \tfrac{C}{N}, &\qquad \textstyle\sum_{n=1}^{N}(a_n + \hat{a}_n) \le \nu C. \end{aligned}$$

SVM FOR REGRESSION



RVM

PROBLEM SETTING

- SVM is great. But there still are drawbacks:
    - SVM outputs are solutions, and posterior probabilities are hard to get;
    - SVM works for two classes, hard to generalize;
    - parameter C (and ν, and/or ϵ) has to be tuned, no general answer;
    - kernels have to satisfy the conditions of Mercer's theorem.
- Now we will (briefly) consider the Bayesian counterpart of SVM: *relevance vector machines* (RVM).

RVM FOR REGRESSION

- It is more convenient to formulate RVM for regression.
- Recall the usual linear model:

$$p(t \mid \mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t \mid y(\mathbf{x}), \beta^{-1}), \text{ where}$$

$$y(\mathbf{x}) = \sum_{i=1}^{M} w_i \phi_i(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}).$$

RVM FOR REGRESSION

- RVM is a variation of such a model, which tries to work as an SVM.
- Consider
$$y(\mathbf{x}) = \sum_{n=1}^{N} w_n k(\mathbf{x}, \mathbf{x}_n) + b.$$
- That is, we look for the solution as a linear combination of kernels from the very beginning (recall “equivalent kernel” for linear regression), but unlike SVM there are no restrictions on the kernel now.

RVM FOR REGRESSION

- For N observations of vector $\mathbf{x}$ (we denote them by $\mathbf{X}$) with values $\mathbf{t}$ we get the likelihood

$$p(\mathbf{t} \mid \mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} p(t_n \mid \mathbf{x}_n, \mathbf{w}, \beta^{-1}).$$

- The prior distribution is normal too, but instead of a single hyperparameter for all weights we consider a separate hyperparameter for every one:

$$p(\mathbf{w} \mid \alpha) = \prod_{i=1}^{M} \mathcal{N}(w_i \mid 0, \alpha_i^{-1}).$$

RVM FOR REGRESSION

- Separate hyperparameters:

$$p(\mathbf{w} \mid \alpha) = \prod_{i=1}^{M} \mathcal{N}(w_i \mid 0, \alpha_i^{-1}).$$

- The idea is that when we maximize the posterior, most α_i will simply tend to infinity, and the corresponding weights will be zero.

RVM for regression

- We know the posterior:

$$p(\mathbf{w} \mid \mathbf{t}, \mathbf{X}, \alpha, \beta) = \mathcal{N}(\mathbf{w} \mid \mathbf{m}, \Sigma), \text{ where}$$

$$\begin{aligned} \mathbf{m} &= \beta \Sigma \Phi^\top \mathbf{t}, \\ \Sigma &= \left(\mathbf{A} + \beta \Phi^\top \Phi\right)^{-1}, \end{aligned}$$

  where $\mathbf{A} = \mathrm{diag}(\alpha_1, \dots, \alpha_M)$, and Φ in our case is $\mathbf{K}$, a symmetric matrix with elements $k(\mathbf{x}_n, \mathbf{x}_m)$.

RVM FOR REGRESSION

- How do we find α and β? We need to maximize the marginal likelihood of the dataset

$$p(\mathbf{t} \mid \mathbf{X}, \alpha, \beta) = \int p(\mathbf{t} \mid \mathbf{X}, \mathbf{w}, \beta) p(\mathbf{w} \mid \alpha) d\mathbf{w}.$$

- This is a convolution of two Gaussians:

$$\ln p(\mathbf{t} \mid \mathbf{X}, \alpha, \beta) = \ln \mathcal{N}(\mathbf{t} \mid 0, \mathbf{C}) =$$

$$= -\frac{1}{2}\left[N \ln(2\pi) + \ln|\mathbf{C}| + \mathbf{t}^\top \mathbf{C}^{-1}\mathbf{t}\right], \text{ where } \mathbf{C} = \beta^{-1}\mathbf{I} + \Phi\mathbf{A}^{-1}\Phi^\top.$$

- How do we optimize this?

RVM FOR REGRESSION

- Computing the derivatives, we get

$$\alpha_i = \frac{\gamma_i}{m_i^2},$$
$$\beta^{-1} = \frac{\|\mathbf{t} - \Phi\mathbf{m}\|^2}{N - \sum_i \gamma_i},$$

  where $\gamma_i = 1 - \alpha_i \Sigma_{ii}$.
- Now we can simply iteratively recompute α, β from $\mathbf{m}, \Sigma$ and vice versa, until convergence.

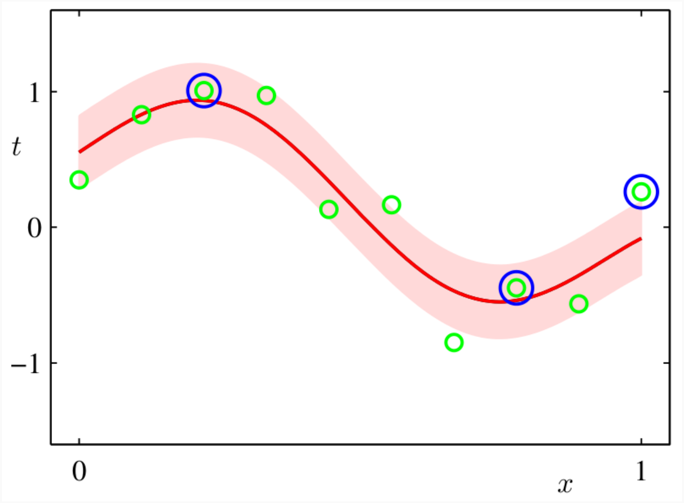
RVM FOR REGRESSION

- As a result, most α_i usually grow unboundedly, and the corresponding weights can be assumed to be zero.
- The rest are called *relevance vectors*, usually very few of those.
- If we now find α^*, β^*, we can predict in new points as

$$p(t \mid \mathbf{x}, \mathbf{X}, \mathbf{t}, \alpha^*, \beta^*) = \int p(t \mid \mathbf{x}, \mathbf{w}, \beta^*) p(\mathbf{w} \mid \mathbf{X}, \mathbf{t}, \alpha^*, \beta^*) d\mathbf{w} =$$
$$= \mathcal{N}(t \mid \mathbf{m}^\top \phi(\mathbf{x}), \sigma^2(\mathbf{x})),$$

where $\sigma^2(\mathbf{x}) = (\beta^*)^{-1} + \phi(\mathbf{x})^\top \Sigma \phi(\mathbf{x})$.

RVM FOR REGRESSION



RVM FOR CLASSIFICATION

- We can do the same for classification. Consider binary classification, $t \in \{0, 1\}$:

$$y(\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^\top \phi(\mathbf{x})).$$

- We add here, again, a prior distribution with different α_i for each weight:

$$p(\mathbf{w} \mid \alpha) = \prod_{i=1}^{M} \mathcal{N}(w_i \mid 0, \alpha_i^{-1}).$$

- Idea: initialize α, compute Laplace approximation to the posterior, maximize, get new α, and so on.

RVM FOR CLASSIFICATION

- Posterior:

$$\ln p(\mathbf{w} \mid \mathbf{t}, \alpha) = \ln \left(p(\mathbf{t} \mid \mathbf{w}) p(\mathbf{w} \mid \alpha)\right) - \ln p(\mathbf{t} \mid \alpha) =$$

$$= \sum_{n=1}^{N} \left[t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\right] - \frac{1}{2} \mathbf{w}^\top \mathbf{A} \mathbf{w} + \text{const.}$$

- We can maximize it with IRLS:

$$\nabla \ln p(\mathbf{w} \mid \mathbf{t}, \alpha) = \Phi^\top (\mathbf{t} - \mathbf{y}) - \mathbf{A}\mathbf{w},$$
$$\nabla\nabla \ln p(\mathbf{w} \mid \mathbf{t}, \alpha) = -\left(\Phi^\top \mathbf{B} \Phi + \mathbf{A}\right),$$

where $\mathbf{B}$ is a diagonal matrix with elements $b_n = y_n(1 - y_n)$.

RVM FOR CLASSIFICATION

- Laplace approximation results from $\nabla \ln p(\mathbf{w} \mid \mathbf{t}, \alpha)$, and we get

$$\mathbf{w}^* = \mathbf{A}^{-1}\Phi^\top (\mathbf{t} - \mathbf{y}),$$
$$\Sigma = \left(\Phi^\top \mathbf{B} \Phi + \mathbf{A}\right)^{-1},$$

and the predictive distribution is

$$p(\mathbf{t} \mid \alpha) = \int p(\mathbf{t} \mid \mathbf{w}) p(\mathbf{w} \mid \alpha) d\mathbf{w} \approx$$
$$\approx p(\mathbf{t} \mid \mathbf{w}^*) p(\mathbf{w}^* \mid \alpha) (2\pi)^{M/2} |\Sigma|^{1/2}.$$

RVM FOR CLASSIFICATION

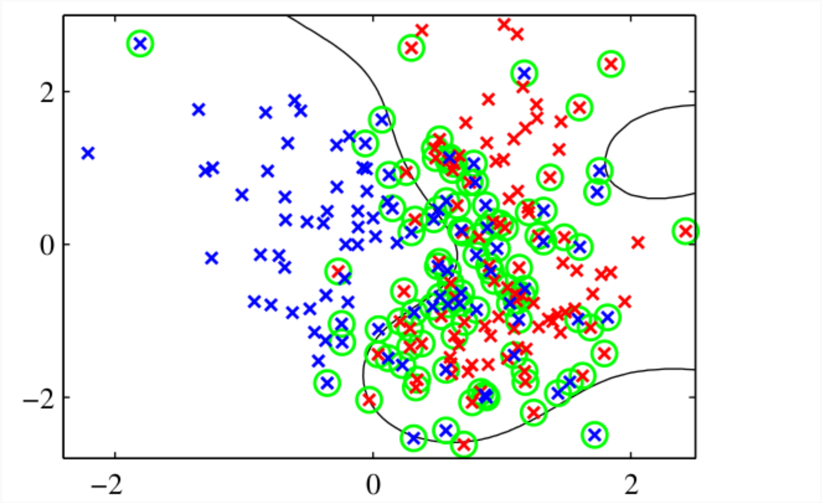
- $p(\mathbf{t} \mid \alpha) = \int p(\mathbf{t} \mid \mathbf{w})p(\mathbf{w} \mid \alpha)d\mathbf{w} \approx p(\mathbf{t} \mid \mathbf{w}^*)p(\mathbf{w}^* \mid \alpha)(2\pi)^{M/2}|\Sigma|^{1/2}$.
- We now optimize it w.r.t. α: take the derivative and get

$$-\frac{1}{2}(w_i^*)^2 + \frac{1}{2\alpha_i} - \frac{1}{2}\Sigma_{ii} = 0, \text{ i.e.,}$$

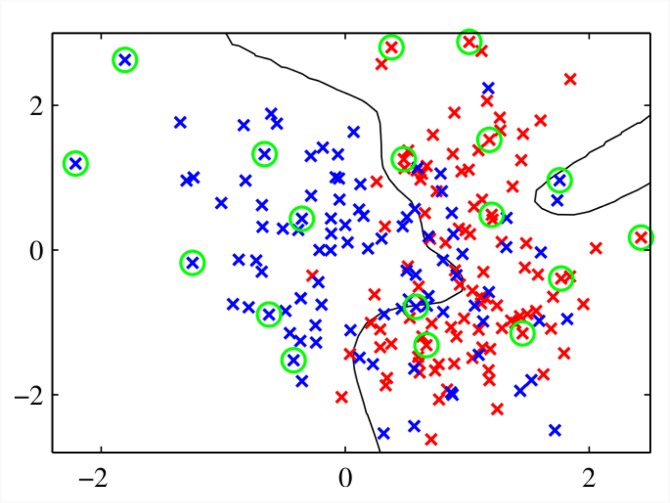
$$\alpha_i = \frac{\gamma_i}{(w_i^*)^2}, \ \gamma_i = 1 - \alpha_i\Sigma_{ii}.$$

- i.e., we get the same formula as for regression.

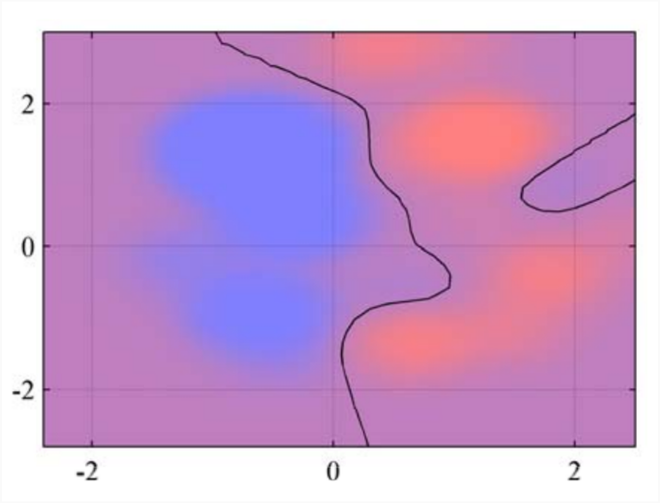
BEFORE: SVM



AFTER: RVM



AFTER: RVM



RVM FOR SEVERAL CLASSES

- And we get a very natural generalization to several classes:

$$a_k = \mathbf{w}_k^\top \mathbf{x}, \quad y_k(\mathbf{x}) = \frac{e^{a_k}}{\sum_j e^{a_j}}.$$

- And then nothing changes.

COMPARING SVM AND RVM

- RVM looks better, and usually is.
- Main drawback: RVM training is much longer (even though SVM training is long enough by itself).
- But even this is not really a drawback because SVM needs cross-validation to tune parameters, and RVM is much faster to apply to new points because there are usually fewer support vectors.

THANK YOU!

Thank you for your attention!