Kernels, SVMs
Ensembling (Bagging, Boosting)

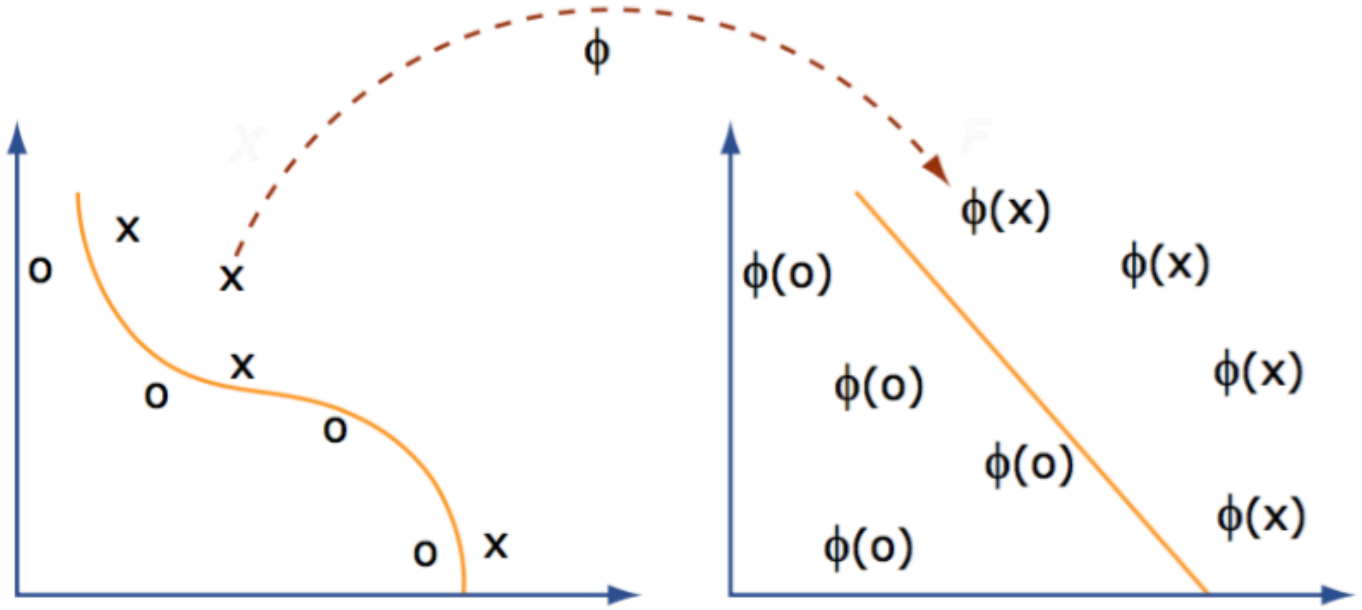
M. Vazirgiannis & N. Tziortziotis

November 2018

Outline

- **Introduction to Kernels**
- SVMs
- Ensembling
  - Bagging – Random forests
  - Boosting – Adaboost

Mapping



- Map data points into an inner product space H with some function ϕ: $\phi : x \rightarrow \phi(x) \in H$
- The map ϕ aims to convert the nonlinear relations into linear ones.

Constructing Features

Problems

- Need to be an expert in the domain
- Features may not be adequate
- Extracting features can sometimes be computationally expensive
  - Example: second order features in 1000 dimensions.

Solutions

- Calculate a similarity measure in the feature space instead of the coordinates of the vectors there,
- apply algorithms that only need the value of this measure

Kernels

- A kernel is a function $k : X \times X \rightarrow R$ for which the following property holds

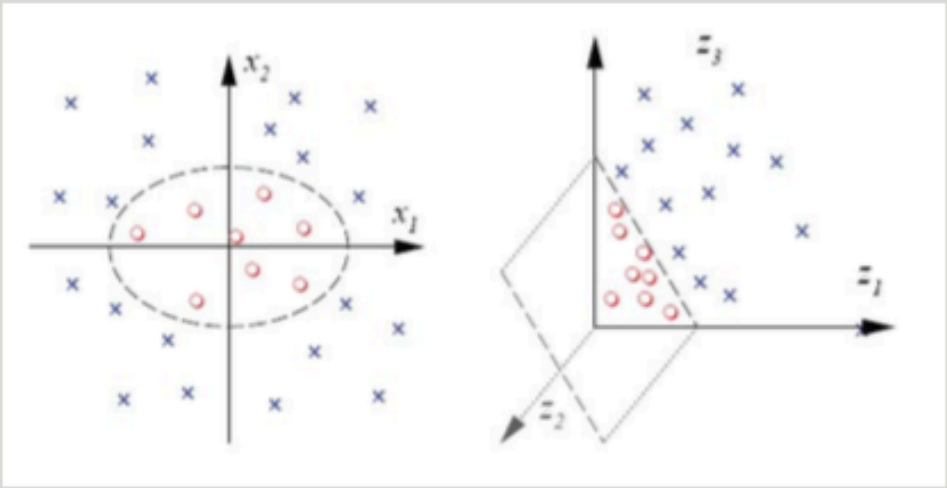
$$k(x, x') = \langle \phi(x), \phi(x') \rangle ,$$

where ϕ is a mapping from X to a Hilbert (inner product) space H

$$\phi : x \rightarrow \phi(x) \in H$$

Kernel Example

Quadratic Features in $\mathbb{R}^2$



$$\phi : x = (x_1, x_2) \rightarrow \phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

Inner product in the feature space

$$\begin{aligned}\langle \phi(x), \phi(z) \rangle &= \left\langle (x_1^2, x_2^2, \sqrt{2}x_1x_2), (z_1^2, z_2^2, \sqrt{2}z_1z_2) \right\rangle \\ &= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 = \langle x, z \rangle^2\end{aligned}$$

Kernel trick

- enable operation in a high-dimensional, implicit feature space
- without computing the coordinates of the data in that space - $\phi(x)$
- simply computing inner products between the images of all pairs of data in the feature space
- need a function: $k(x, x') = <\phi(x), \phi(x')>$
  - computationally cheaper than the explicit computation of the coordinates.
  - introduced for sequence data, graphs, text, images, as well as vectors.

Properties of Kernels

Distance in Feature Space

The distance between points in feature space is given by

$$\begin{aligned}\|\phi(x) - \phi(x')\|^2 &= \langle \phi(x), \phi(x) \rangle - 2\langle \phi(x), \phi(x') \rangle + \langle \phi(x'), \phi(x') \rangle \\ &= k(x, x) - 2k(x, x') + k(x', x')\end{aligned}$$

Symmetry

Kernel is symmetric due to the symmetry of the dot product:

$$k(x, x') = \langle \phi(x), \phi(x') \rangle = \langle \phi(x'), \phi(x) \rangle = k(x', x)$$

Cauchy-Schwarz inequality

$$\begin{aligned}k(x, x')^2 &= \langle \phi(x), \phi(x') \rangle^2 \\ &\leq \|\phi(x)\|^2 \|\phi(x')\|^2 = k(x, x)k(x', x')\end{aligned}$$

Properties of a Kernel Matrix

K is Positive Semidefinite

$a^\top K a \geq 0$ for all $a \in \mathbb{R}^n$ and all kernel matrices $K \in \mathbb{R}^{n \times n}$.

Proof:

$$\sum_{i,j}^{n} a_i a_j K_{ij} = \sum_{i,j}^{n} a_i a_j \langle \phi(x_i), \phi(x_j) \rangle$$

$$= \left\langle \sum_{i}^{n} a_i \phi(x_i), \sum_{j}^{n} a_j \phi(x_j) \right\rangle = \left\| \sum_{i}^{n} a_i \phi(x_i) \right\|^2 \geq 0$$

Symmetry

K is symmetric due to the symmetry of the dot product:

$$K_{ij} = K_{ji} \text{ as } \langle \phi(x), \phi(x') \rangle = \langle \phi(x'), \phi(x) \rangle .$$

Mercer's theorem

Theorem

For any symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which is square integrable in $\mathcal{X} \times \mathcal{X}$ and satisfies

$$\int_{\mathcal{X} \times \mathcal{X}} k(x, x') f(x) f(x') dx dx' \geq 0 \text{ for all } f \in L_2(\mathcal{X}) \tag{0.1}$$

exist $\phi : \mathcal{X} \to \mathbb{R}$ and $\lambda \geq 0$ where

$$k(x, x') = \sum_i \lambda_i \phi(x) \phi(x') \forall x, x' \in \mathcal{X} \tag{0.2}$$

Interpretation

Mercel's condition tells us whether or not a prospective kernel is actually a dot product in the *Hilbert* space, $\mathcal{H}$.

Mercer Kernels

- Let $X = \{x_1, \ldots x_n\}$ finite set of samples from X, The Gram matrix $K(X;\varkappa)$, such that for $(K)_{ij} = \langle x_i, x_j \rangle$
- The Gram matrix positive definite:
  - Thus eigenvector decomposition: $K = U \Lambda U^T$
- An element of K expressed as a dot product among 2 vectors: $K_{ij} = (\Lambda^{1/2} U_{:,i})^T (\Lambda^{1/2} U_{:,j})$
- Thus if we define $\phi(x_i) = (\Lambda^{1/2} U_{:,i})$
- Then $K_{ij} = \phi(x_i)^T \phi(x_j)$

Mercer Kernels

$K_{ij} = \phi(x_i)^T \phi(x_j)$

- each element of the kernel can be described as the inner product of a function φ applied to objects x .

- Therefore, if a kernel is a Mercer kernel, then there exists a function
- $\phi : X \rightarrow D$
- such that $\varkappa(\boldsymbol{x}, \boldsymbol{x'}) = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x'})$, ϕ:basis function.

Constructing Kernels from Kernels

Assuming valid kernels $k_1(x, z)$ and $k_2(x, z)$, the following are also valid kernels:

- $k(x, z) = ck_1(x, z)$, where $c \in \mathcal{R}^+$
- $k(x, z) = k_1(x, z) + k_2(x, z)$
- $k(x, z) = k_1(x, z)k_2(x, z)$
- $k(x, z) = \exp(k_1(x, z))$
- $k(x, x') = k_a(x_a, x'_a) + k_b(x_b, x'_b)$, where $x = (x_a, x_b)$
- $k(x, x') = k_a(x_a, x'_a)k_b(x_b, x'_b)$, where $x = (x_a, x_b)$

Typical Kernels

- **Linear**

$$k(x, x') = \langle x, x' \rangle$$

- **Laplacian RBF**

$$k(x, x') = \exp(-\lambda \|x - x'\|)$$

- **Gaussian RBF**

$$k(x, x') = \exp(-\lambda \|x - x'\|^2)$$

- **Polynomial**

$$k(x, x') = \left(\langle x, x' \rangle + c\right)^d, c \geq 0, d \in \mathbb{N}$$

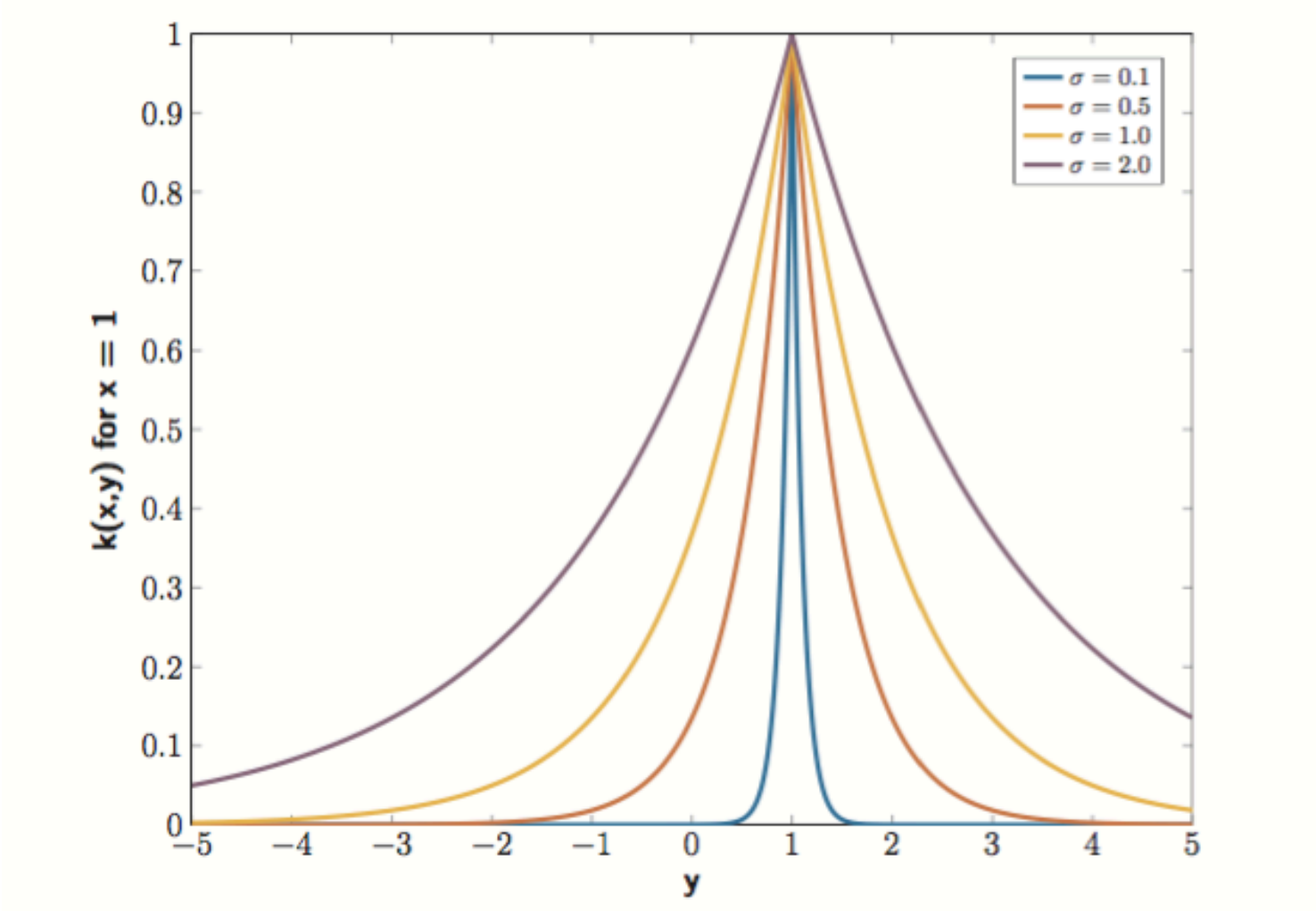
- **Sigmoid**

$$k(x, x') = \tanh(\eta \langle x, x' \rangle + b)$$

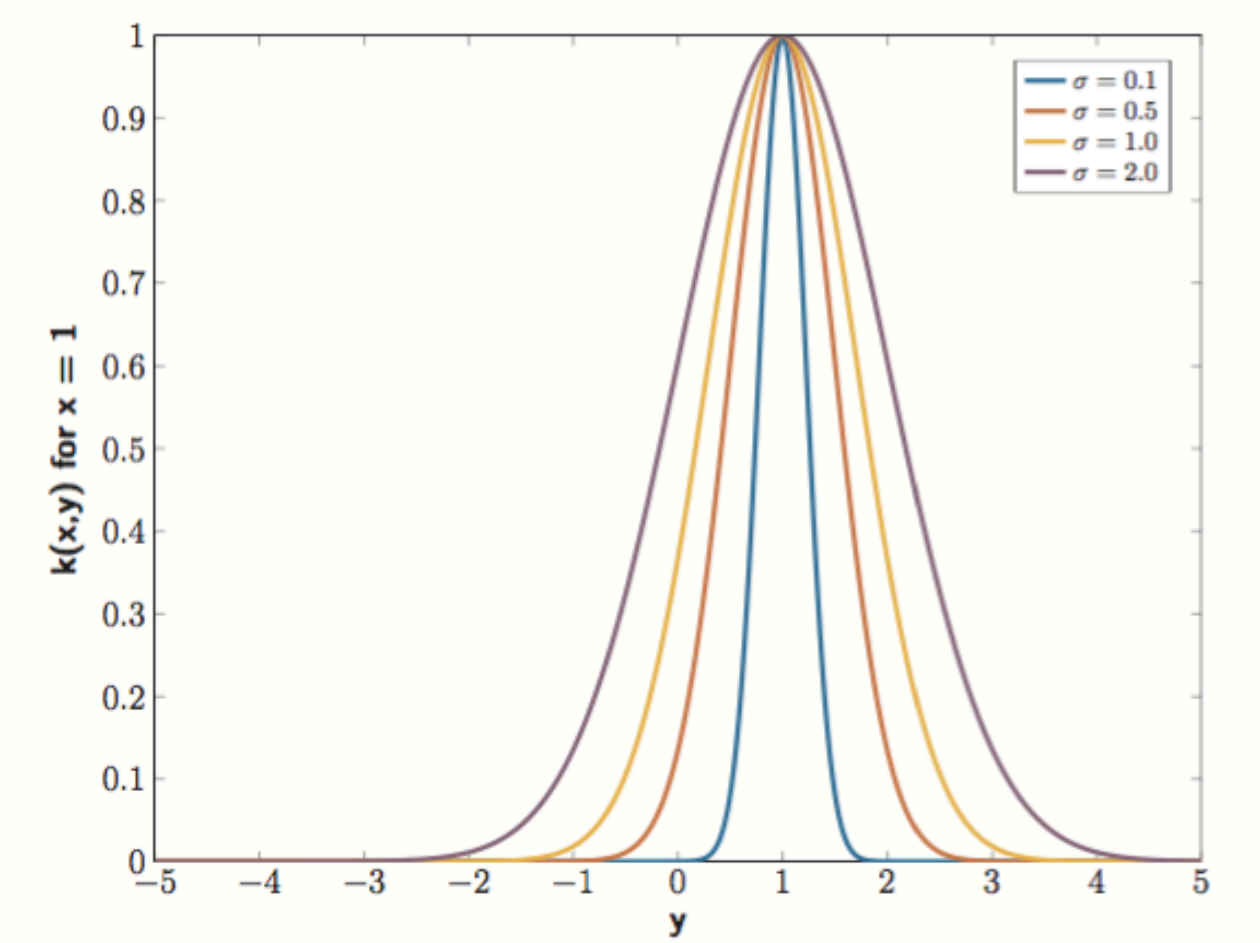
- **B-Spline**

$$B_{2n+1}(x - x')$$

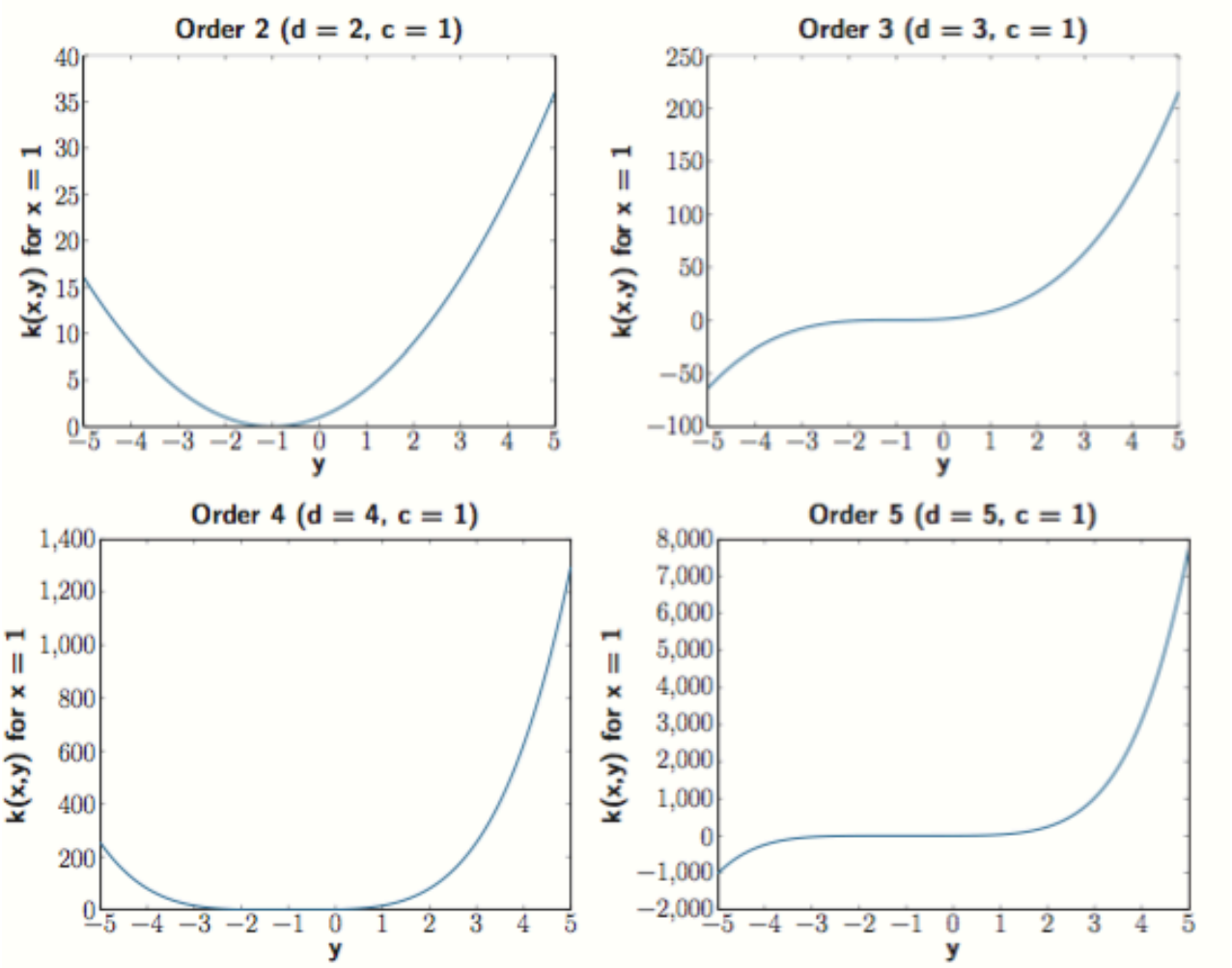
Laplacian Kernel



Gaussian Kernel



Polynomial Kernel



$$k(x, x') = (< x, x' > +c)^d, c \geq 0, d \in N$$

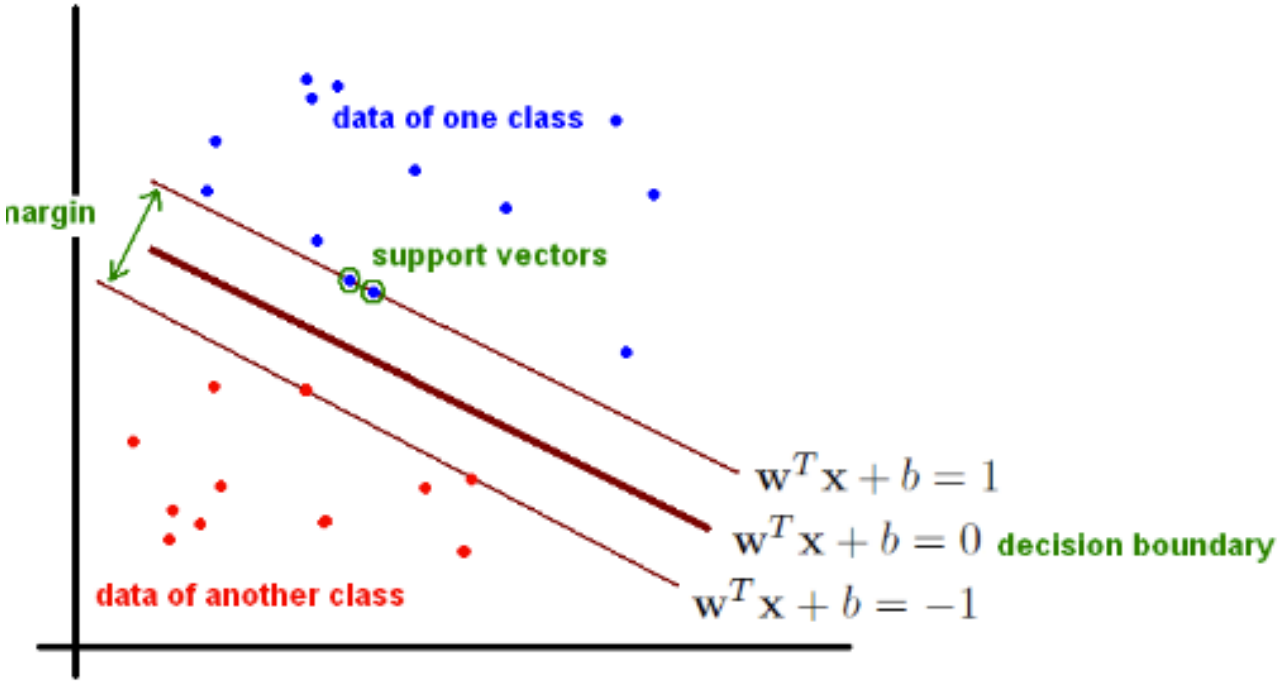
Outline

- Introduction to Kernels
- **SVMs**
- Ensembling
  - Bias Variance -tradeoff
  - Bagging – Random forests
  - Boosting – Adaboost

SVMs

- Issues that motivated SVMS:
  - bias variance tradeoff
  - capacity control
  - Over fitting
- For a given learning task, a finite amount of training data, the best generalization performance is achieved by jointly optimizing
  - accuracy attained on a training set,
  - “capacity” : ability to learn from any training set without error

SVMs



Goal: find a a hyperplane (i.e. decision boundary) linearly separating our classes.

Boundary equation: $\mathbf{w}^T\mathbf{x} + b = 0$

If $x_i : \mathbf{w}^T\mathbf{x} + b > 0$ then $y_i = 1$

if $x_i : \mathbf{w}^T\mathbf{x} + b < 0$ then $y_i = -1$

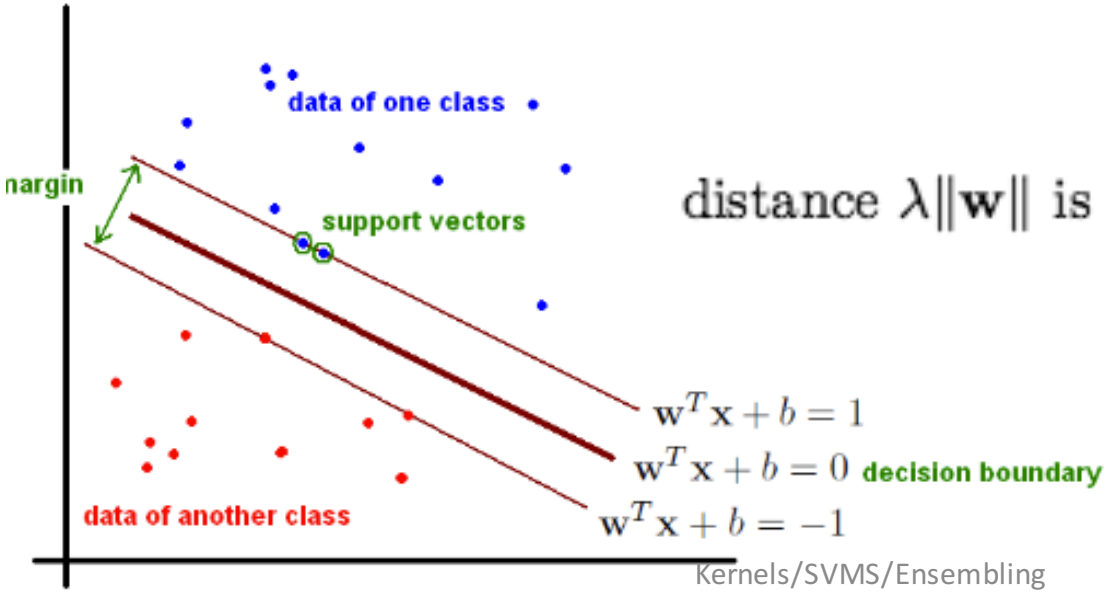
equivalent: $y(\mathbf{w}^T\mathbf{x} + b) >= 1$

SVMs – distance between the boundaries

$$\mathbf{w}^T\mathbf{x} + b = -1$$
$$\mathbf{w}^T\mathbf{x} + b = 1$$

- lines are parallel, with same parameters $\boldsymbol{w}, b$
- Assume x_1 on $\boldsymbol{w}^T\boldsymbol{x}+b=-1$, *the closest point of* x_2 *on line* $\boldsymbol{w}^T\boldsymbol{x}+b=-1$. Thus $x_2 = \boldsymbol{w}^T\boldsymbol{x}+b=-1$ and $\lambda\boldsymbol{w}$ the distance (x_1,x_2).

Solving for λ: $\mathbf{w}^T\mathbf{x_2} + b = 1$ where $\mathbf{x_2} = \mathbf{x_1} + \lambda\mathbf{w}$ => $\lambda = \frac{2}{\mathbf{w}^T\mathbf{w}} = \frac{2}{\|\mathbf{w}\|^2}$

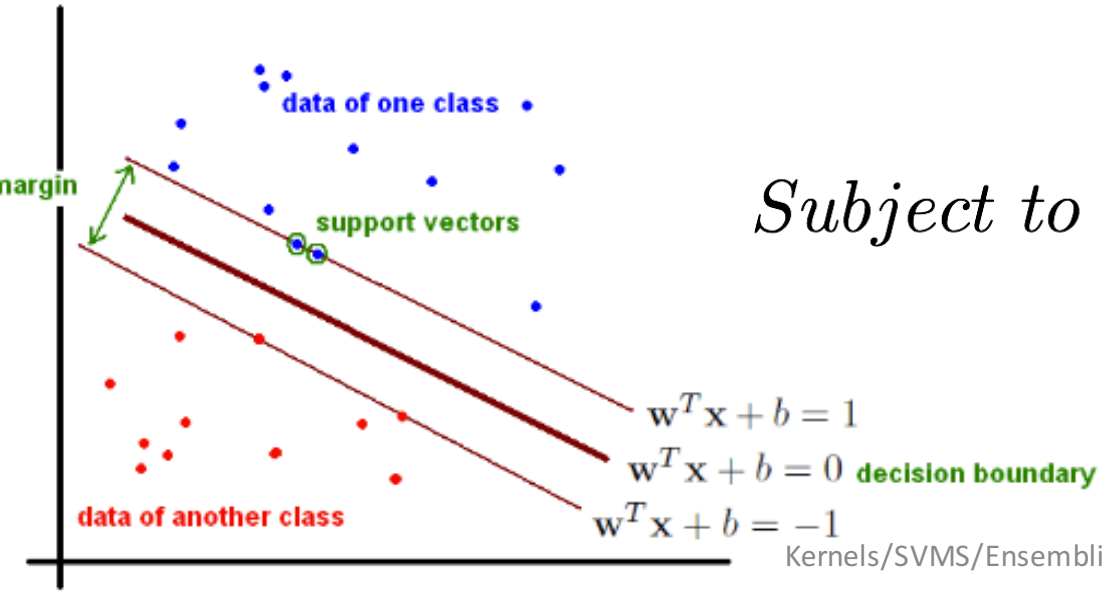


distance $\lambda\|\mathbf{w}\|$ is $\frac{2}{\|\mathbf{w}\|^2}\|\mathbf{w}\| = \frac{2}{\|\mathbf{w}\|} = \frac{2}{\sqrt{\mathbf{w}^T\mathbf{w}}}$

SVMs – optimization formulation

- maximize the distance between the two boundaries defining the classes – to avoid mis-classifications: *maximal margin*
- Objective: $$max \frac{2}{\sqrt{\mathbf{w}^T\mathbf{w}}} \approx min \frac{\sqrt{\mathbf{w}^T\mathbf{w}}}{2} \approx min \frac{\mathbf{w}^T\mathbf{w}}{2}$$
- Quadratic formulation problem:

$$min_{\mathbf{w},b} \frac{\mathbf{w}^T\mathbf{w}}{2}$$



$$Subject\ to: y_i(\mathbf{w}^T x_i + b) >= 1, \forall x_i$$

Soft Margin extension

- We allow some miss-classification: some data points on the other side of the boundary (slack variables: $\varepsilon_i > 0$ for each point x_i).
- The problems becomes:

$$min_{\mathbf{w},b,C} \frac{\mathbf{w}^T\mathbf{w}}{2} + C\sum_I \epsilon_i$$

$$Subject\ to: y_i(\mathbf{w}^T\mathbf{x}_i + b) >= 1 - \epsilon_i, \epsilon_i >= 0 (\forall \mathbf{x}_i)$$

SVMs – Non Linear Decision Boundary

- If data are not linearly separable we consider a mapping to a higher dimensional space via a function $\phi(\chi)$. Then the optimization becomes:

$$min_{\mathbf{w},b,C} \frac{\mathbf{w}^T\mathbf{w}}{2} + C\sum_I \epsilon_i$$

$$Subject\ to: y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) >= 1 - \epsilon_i, \epsilon_i >= 0(\forall \mathbf{x}_i)$$

SVMs – reformulation as a Lagrancian

- Introduce Langrancian multipliers to represent the condition
- $y_i(\mathbf{w}^T\phi(\mathbf{x_i})+b)$ should be as close to 1 as possible :
- This condition is captured by: $max_{\alpha_i \geq 0}\alpha_i[1-y_i(\mathbf{w}^T\phi(\mathbf{x_i})+b)]$
  - When $y_i(\mathbf{w}^T\phi(\mathbf{x_i})+b) \geq 1$ the expressions is maximal when a_i=0 *
  - Otherwise $y_i(\mathbf{w}^T\phi(\mathbf{x_i})+b) < 1$, so $[1-y_i(\mathbf{w}^T\phi(\mathbf{x_i})+b)]$ is a positive value and the expression is maximal when $\alpha_i \to \infty$
- This results in penalizing (large α_i)misclassified data points, while 0 penalty to properly classified ones
- Thus we have the following formulation:

$$min_{\mathbf{w},b}[\frac{\mathbf{w}^T\mathbf{w}}{2}+\sum_i max_{a_i \geq 0} a_i[1-y_i(\mathbf{w}^T\phi(\mathbf{x}_i)+b)]]$$

SVMs – reformulation as a Lagrancian

$$min_{\mathbf{w},b}[\frac{\mathbf{w}^T\mathbf{w}}{2} + \sum_i max_{a_i \geq 0} a_i[1 - y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b)]]$$

- To preventing α variables to ∞
- we impose constraints on Lagrange multipliers $0 <= a_i <= C$

- We define the dual problem interchanging the max, min:

$max_{\alpha \geq 0}[min_{\mathbf{w},b} J(\mathbf{w}, b; \alpha)]$ where $J(\mathbf{w}, b; \alpha) = \frac{\mathbf{w}^T\mathbf{w}}{2} + \sum_i \alpha_i[1 - y_i(\mathbf{w}^T\phi(\mathbf{x_i}) + b)]$

- To solve the optimization problem:
- $\frac{\partial J}{\partial \mathbf{w}} = 0$ hence **w**: $\sum_i \alpha_i y_i \phi(x_i)$

  $\frac{\partial J}{\partial b} = 0$ hence $\sum_i \alpha_i y_i = 0.$

Substitute and simplify: $min_{\mathbf{w},b} J(\mathbf{w}, b; \alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \phi(\mathbf{x_i})^T \phi(\mathbf{x_j})$

The the dual problem is: $max_{\alpha \geq 0}[\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \phi(\mathbf{x_i})^T \phi(\mathbf{x_j})]$

$$\text{Subject to: } \sum_i \alpha_i y_i = 0 \text{ and } 0 \leq \alpha_i \leq C$$

SVM- Kernel trick

- As dimensionality may be infinite computation of $\phi(\mathbf{x}_i)^T, \phi(\mathbf{x}_j)$

may be intractable

Kernel Trick: $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T, \mathbf{x}_j)^2 = \phi(\mathbf{x}_i)^T, \phi(\mathbf{x}_j)$

Thus our computation is simplified with rewriting the dual in terms of the kernel:

$$max_{\alpha \geq 0}[\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)]$$

SVM – Decision function

- To classify a novel instance x, having learned the optimal α_i parameters:

$$f(\mathbf{x}) = sign(\mathbf{w}^T\mathbf{x} + b) = \sum_i \alpha_i y_i K(\mathbf{x_i}, \mathbf{x}) + b$$

- Setting $\mathbf{w} = \sum_i \alpha_i y_i \phi(\mathbf{x_i})$ and using the kernel trick

- α_i are non zero for $\phi(x_i)$ on or close to the boundary – support vectors

Outline

- Introduction to Kernels
- SVMs
- **Ensembling**
  - Bagging – Random forests
  - Boosting – Adaboost

Ensembling in supervised learning

- Supervised learning algorithms may suffer by
  - bias/variance tradeoff
  - overfitting
- Ensembles: methods that generate *multiple hypotheses (predictors)* using the same base learner.
- ensemble typically requires more computation
- Can be considered as a way to compensate for poor learning algorithms by performing a lot of extra computation.
- i.e. *Random Forest* capitalize on decision trees

Ensemble types:

- Bagging
- Boosting

Generalization

- Empirical error $E = \frac{1}{n} \sum_{i=1}^{N} \delta(f(x), y)$
- Expected error $E_x = \int_{X,Y} \delta(f(x), y) p(x, y) dx dy$

Generalization

- Generalization error: $G = E(f) - E_x(f)$
  - difference between the *training set* and the underlying *joint probability distribution* error
  - An algorithm generalizes well if

$$\lim_{n \to \infty} E(f) - E_x(f) = 0$$

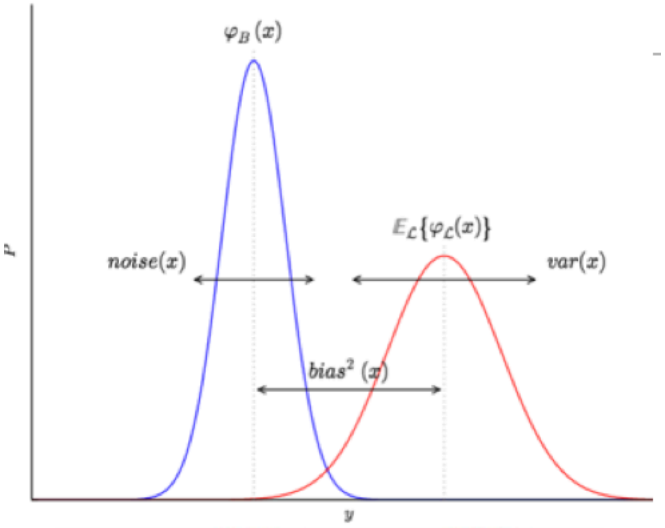
$p(x,y)$ unknown probability distribution => impossible to compute

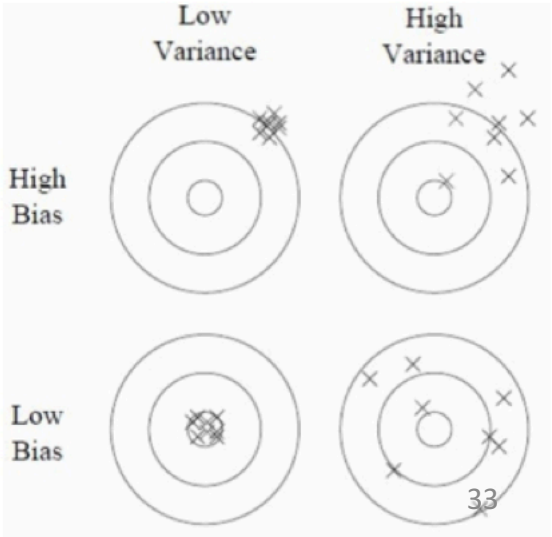
Error as Bias, Variance

- Let $y = f(x) + \varepsilon$ learning function, ε: noise, mean = 0, variance σ^2
- Let $\hat{f}(x)$ approximation of $f(x)$
- Expected Error

$$\mathrm{E}\left[(y - \hat{f}(x))^2\right] = \left(\mathrm{Bias}\left[\hat{f}(x)\right]\right)^2 + \mathrm{Var}\left[\hat{f}(x)\right] + \sigma^2$$

- where: $\mathrm{Bias}\left[\hat{f}(x)\right] = \mathrm{E}\left[\hat{f}(x) - f(x)\right]$
- var: $\mathrm{Var}\left[\hat{f}(x)\right] = \mathrm{E}[\hat{f}(x)^2] - \left(\mathrm{E}[\hat{f}(x)]\right)^2$





Understanding Random Forests, From Theory to Practice , Gilles Louppe
https://www.lri.fr/~kegl/Polytechnique/louppe.pdf

Bootstrap Aggregating - Bagging

- Assume training set $D = \{(x_i, y_i)\}$
- Objective: predict label for an unknown x
  - Sample B data sets – each size n, randomly with replacement from D: $\{D_1, \ldots D_B\}$
  - For each D_b train a tree f_b and make a prediction => obtain a set of B predictions fb on $D_b(X_b, Y_b)$
  - Assume unseen samples x' the final prediction obtained either by
    - averaging (regression) $\hat{f} = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(x')$
    - majority voting (classification)

Bootstrap Aggregating - Bagging

- *Decreases variance in the predictions - without increasing bias*
- predictions of a single tree may sensitive to noise in the training set
- average of many trees is not, as long as the trees *are not correlated*.
  - training many trees on a single training set would give strongly correlated trees
  - bootstrap sampling de-correlates the trees
- B: free parameter:
- - few hundred to several thousand trees are used, depending on the size and nature of the training set.
- An optimal number of trees *B* using
  - cross-validation
  - by observing the out-of-bag error
    - mean prediction error on each training sample x_i, using only trees not having x_i in their bootstrap sample.
  - The training and test error tend to stabilize after some number of trees have been fit.

Random forests

- use a modified decision tree learning algorithm
- selects, at each candidate split in the learning process, a *random subset of the features - "feature bagging"*.
- Reason - correlation of the trees:
  - if some features are very correlated to the class label will be selected in many of the trees,
  - Resulting trees correlated.
- # of features selected:
  - classification problem with p features, $\sqrt{p}$ features are used in each split .
  - Regression problems have different defaults

Random forest convergence

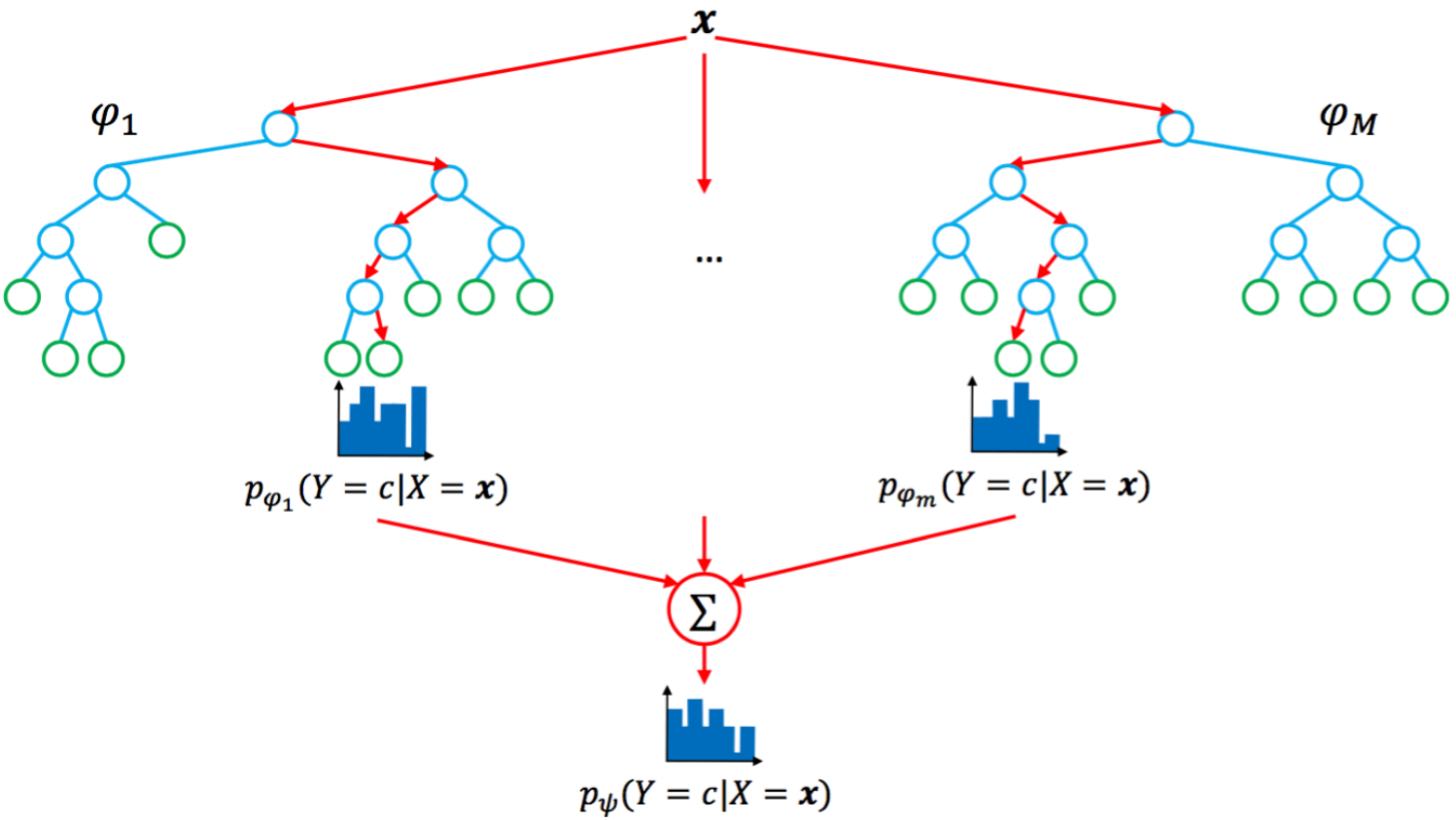
- Assume an ensemble of classifiers $h_1(\mathbf{x}), h_2 1(\mathbf{x}), ..., h_K(\mathbf{x})$ with training set drawn from the distribution of $Y, \boldsymbol{X}$
- Margin function:

$$mg(\mathbf{X},Y) = av_k I(h_k(\mathbf{X}) = Y) - max_{j \neq Y} av_k I(h_K(\mathbf{X}) = j)$$

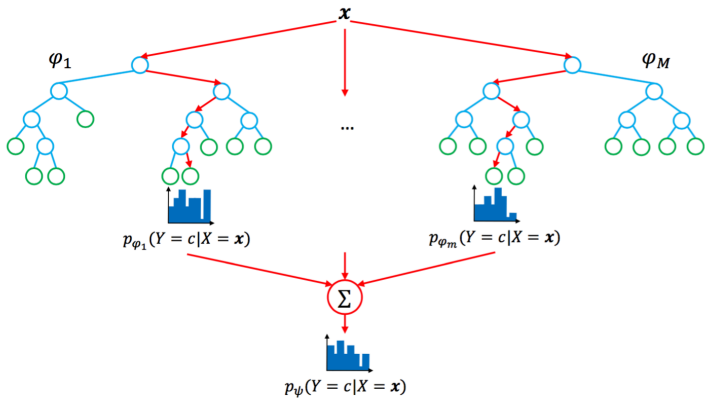
  - Measures average # votes for the correct class is larger than the average vote for any other class
  - Larger margin: larger confidence to the classifier
- As # of trees increases for all sequences Θi (random vectors)

$$P_{\mathbf{X},Y}(P_\Theta(h(\mathbf{X},\Theta) = Y) - max_{j \neq Y}(P_\Theta(h(\mathbf{X},\theta) = j) < 0)$$

Random forests



Random forests



Randomization cases

- Bootstrap samples
- Random selection of $K<=p$ split variables
- *Random selection of the threshold (extra trees)*

Expected generalization error

Theorem. For the squared error loss, the bias-variance decomposition of the expected generalization error $\mathbb{E}_{\mathcal{L}}\{Err(\psi_{\mathcal{L},\theta_1,\ldots,\theta_M}(\mathbf{x}))\}$ at $X = \mathbf{x}$ of an ensemble of M randomized models $\varphi_{\mathcal{L},\theta_m}$ is

$$\mathbb{E}_{\mathcal{L}}\{Err(\psi_{\mathcal{L},\theta_1,\ldots,\theta_M}(\mathbf{x}))\} = \text{noise}(\mathbf{x}) + \text{bias}^2(\mathbf{x}) + \text{var}(\mathbf{x}),$$

where

$$\begin{aligned}
\text{noise}(\mathbf{x}) &= Err(\varphi_B(\mathbf{x})), \\
\text{bias}^2(\mathbf{x}) &= (\varphi_B(\mathbf{x}) - \mathbb{E}_{\mathcal{L},\theta}\{\varphi_{\mathcal{L},\theta}(\mathbf{x})\})^2, \\
\text{var}(\mathbf{x}) &= \rho(\mathbf{x})\sigma^2_{\mathcal{L},\theta}(\mathbf{x}) + \frac{1-\rho(\mathbf{x})}{M}\sigma^2_{\mathcal{L},\theta}(\mathbf{x}).
\end{aligned}$$

and where $\rho(\mathbf{x})$ is the Pearson correlation coefficient between the predictions of two randomized trees built on the same learning set.

Understanding Random Forests, From Theory to Practice, Gilles Louppe
https://www.lri.fr/~kegl/Polytechnique/louppe.pdf

Generalization error of random forests

- Bias : Identical to the bias of a single randomized tree.
- Variance : $\mathrm{var}(\mathbf{x}) = \rho(\mathbf{x})\sigma^2_{\mathcal{L},\theta}(\mathbf{x}) + \frac{1-\rho(\mathbf{x})}{M}\sigma^2_{\mathcal{L},\theta}(\mathbf{x})$
  As $M \to \infty$, $\mathrm{var}(\mathbf{x}) \to \rho(\mathbf{x})\sigma^2_{\mathcal{L},\theta}(\mathbf{x})$
  - The stronger the randomization, $\rho(\mathbf{x}) \to 0$, $\mathrm{var}(\mathbf{x}) \to 0$.
  - The weaker the randomization, $\rho(\mathbf{x}) \to 1$, $\mathrm{var}(\mathbf{x}) \to \sigma^2_{\mathcal{L},\theta}(\mathbf{x})$

Bias-variance trade-off. Randomization increases bias but makes it possible to reduce the variance of the corresponding ensemble model. The crux of the problem is to find the right trade-off.

Understanding Random Forests, From Theory to Practice, Gilles Louppe
https://www.lri.fr/~kegl/Polytechnique/louppe.pdf

Extremely randomized trees

- Adding one further step of randomization
- random subspace method @ training
  - Instead of computing the locally *optimal* feature/split combination
  - randomized top-down splitting in the tree learner
- for each feature considered, a random value selected for the split.
  - value is selected from the feature's empirical range (in the tree's training set, i.e., the bootstrap sample)

Extra trees - splitting procedure for numerical attributes

Table 1 Extra-Trees splitting algorithm (for numerical attributes)

Split_a_node(S)
Input: the local learning subset S corresponding to the node we want to split
Output: a split $[a < a_c]$ or nothing
– If **Stop_split**(S) is TRUE then return nothing.
– Otherwise select K attributes $\{a_1, \ldots, a_K\}$ among all non constant (in S) candidate attributes;
– Draw K splits $\{s_1, \ldots, s_K\}$, where $s_i = \textbf{Pick\_a\_random\_split}(S, a_i), \forall i = 1, \ldots, K$;
– Return a split s_* such that $\text{Score}(s_*, S) = \max_{i=1,\ldots,K} \text{Score}(s_i, S)$.

Pick_a_random_split(S,a)
Inputs: a subset S and an attribute a
Output: a split
– Let $a^S_{\max}$ and $a^S_{\min}$ denote the maximal and minimal value of a in S;
– Draw a random cut-point a_c uniformly in $[a^S_{\min}, a^S_{\max}]$;
– Return the split $[a < a_c]$.

Stop_split(S)
Input: a subset S
Output: a boolean
– If $|S| < n_{\min}$, then return TRUE;
– If all attributes are constant in S, then return TRUE;
– If the output is constant in S, then return TRUE;
– Otherwise, return FALSE.

Extremely randomized treesPierre Geurts·Damien Ernst·Louis Wehenkel, Mach Learning Journal, DOI 10.1007/s10994-006-6226-1

Extra trees

Splitting procedure parameters:

- K, the number of attributes randomly selected at each node
- n_{min} ,the minimum sample size for splitting a node.
- M number of trees

Bias-variance point of view, rationale behind the Extra-Trees

- explicit randomization of the cut-point and attribute combined with ensemble averaging reduces variance strongly
- usage of full original learning sample instead of bootstrap replicas minimizes bias.

Computational cost

- tree growing procedure complexity - assuming balanced trees – $O(N\ Log\ N)$ with respect to learning sample size N.
- The parameters K, n_{min}, M have different effects:
  - K determines the strength of the attribute selection process,
  - n_{min} the strength of averaging output noise, and
  - M the strength of the variance reduction of the ensemble model aggregation.
- The final prediction is by majority vote in classification problems and arithmetic average in regression problems.

Outline

- Introduction to Kernels
- Ensembling
  - Bagging – Random forests
  - **Boosting – Adaboost**

Boosting

- Boosting: creating a highly accurate prediction combining many relatively weak learners
- Weak Learning Algorithm (Weak Learner)
  - a classifier only slightly correlated with the true classification
  - it can label examples better than random guessing - i.e. precision slightly >50%.
- boosting to generate a single weighted classifier with very high precision

Boosting

- iteratively learning weak classifiers with respect to a training set distribution and adding them to a final strong classifier.
- typically weighted in some way related to the weak learners' accuracy.
  - After a weak learner is added, data are reweighted
  - classified examples gain weight
  - examples that are classified correctly lose weight
- Thus, future weak learners focus more on the examples that previous weak learners misclassified.

Adaboost

- Data points weighting:
  - Focus on problematic data points: those misclassified most by the previous weak classifier.
- weak learners composition
  - Use an optimally weighted majority vote of weak classifier.

Adaboost

- Assume a training set (x_i,y_i) , y_i in $\{-1,+1\}$
- Let probability of each data point to be in training set: $D_1(i)=1/m$
- For each round $t=1..T$
  - Train weak learner using distribution D_t
  - Weak learner h_t: $X->\{-1,+1\}$ with error ε_t
  - $\alpha = \frac{1}{2} ln(\frac{1-\epsilon}{\epsilon})$
  - Update

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$$

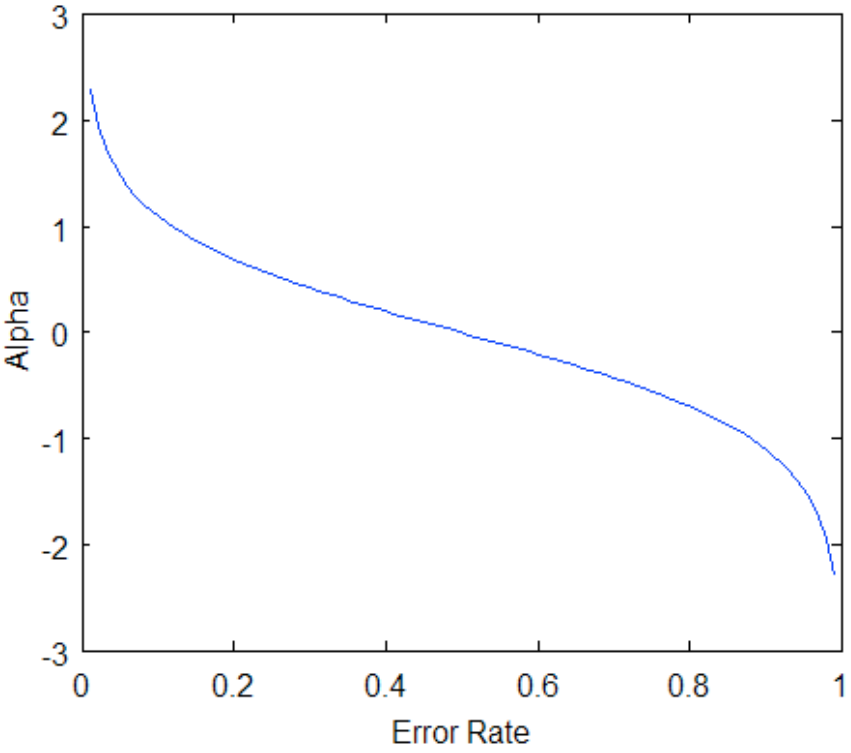
$$= \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

- Final Hypothesis: $H(x) = sign(\sum_{t=1}^{T} a_t h_t(x))$

Adaboost – weights intuition

- Error => 0: classifier weight grows exponentially
  - Better classifiers are given exponentially more weight.
- error rate ~ 0.5.
  - A classifier with 50% accuracy is random guessing: ignore it.
- error => 1: classifier weight grows exponentially negative
  - negative weight to classifiers with worse than 50% accuracy.

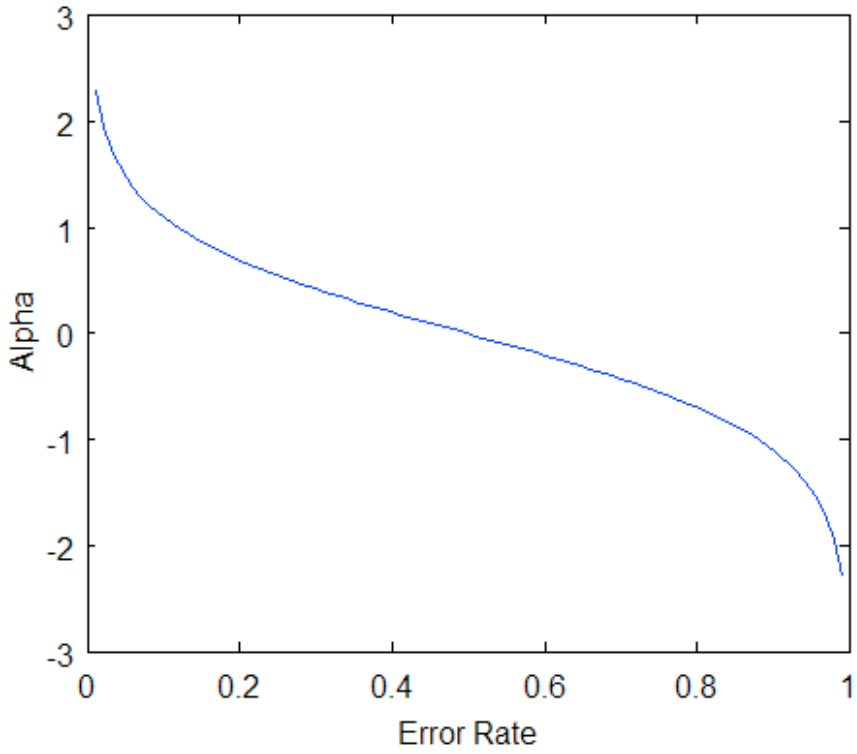
$$\alpha_t = \frac{1}{2} ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$



Adaboost – Training set selection

- Each weak classifier trained on a random subset of total training set.
- AdaBoost assigns a "weight" to each training example that determines the probability to appear in the training set.
- Examples with higher weights are more likely to be included in the training set

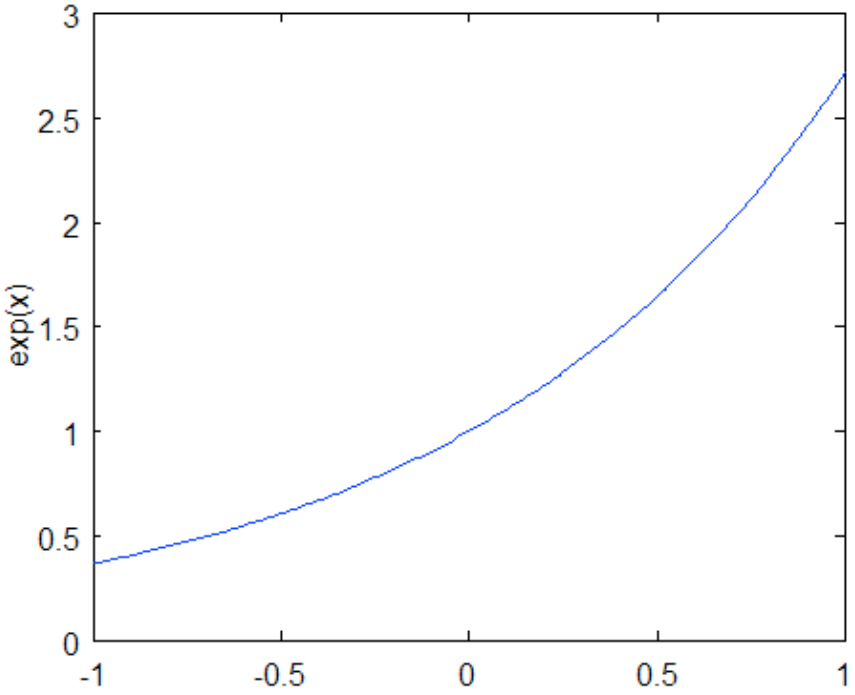
$$\alpha_t = \frac{1}{2} ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$



Adaboost - Classifier Output Weights

After training a classifier

- Increase weight of misclassified examples.
- Those will have larger probability to be in next classifiers training set
- next classifier trained will perform better on them.

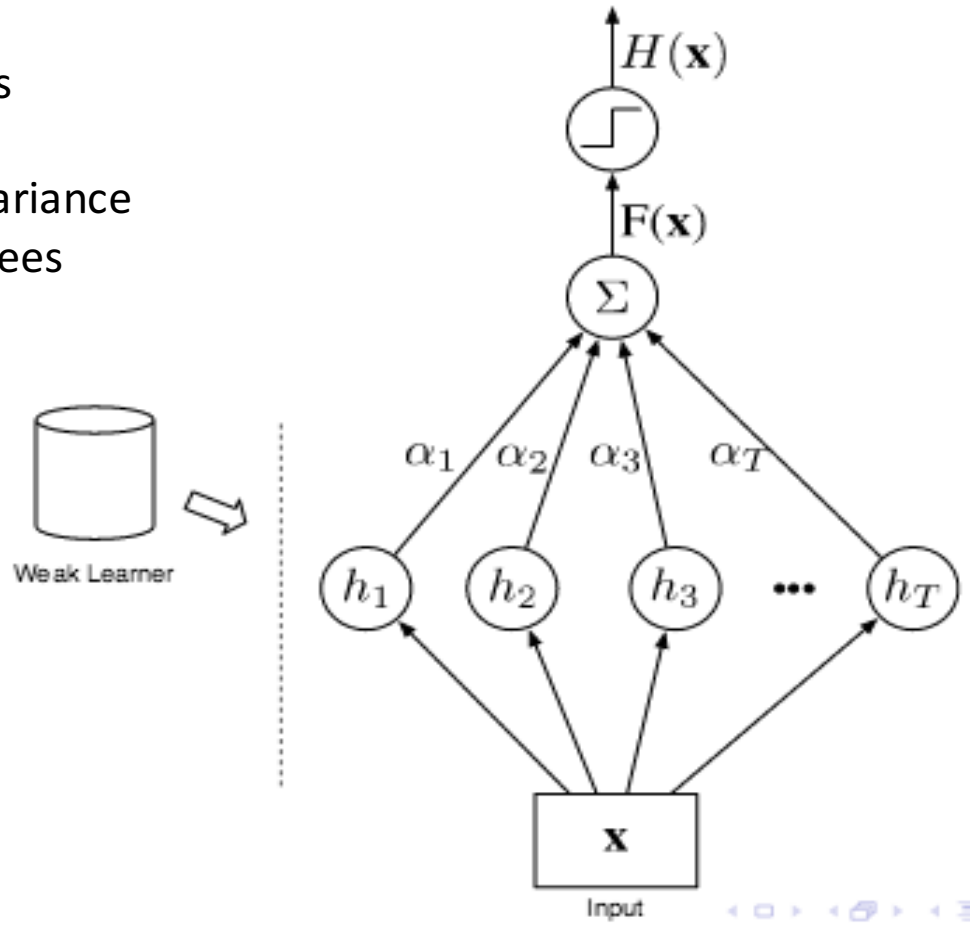


$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$$

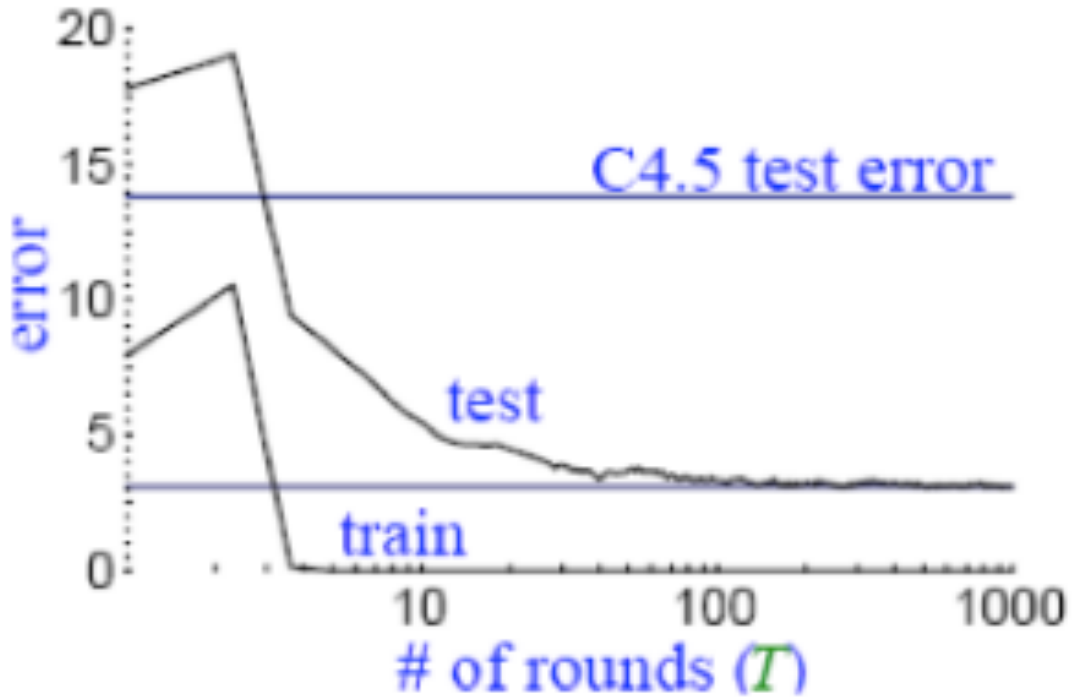
$$= \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Graphical representation

Choice of classifiers
- Easy to train
- Low bias high variance
  - Decision trees



error empirical results



- Data Set: OCR data set
- Weak learner: C.4.5 decision tree
- Test error does not increase even after 1000 rounds.
- Test error continues to drop after training error reaches zero

Adaboost

- Advantages
  - Simple to program
  - No parameters (except T)
  - No prior knowledge needed for weak learner
  - versatile
- Disadvantages
  - Complex Weak classifiers lead to over fitting
  - Weak classifiers too weak can lead to low margins, and can also lead to over fitting
  - empirical evidence: AdaBoost particularly vulnerable to uniform noise.

References

- Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. Bernhard Scholkopf and Alexander J. Smola. 2001.
- Kernels, Course notes , Synthia Rudin, link
- Pattern Recognition and Machine Learning. Christopher M. Bishop. 2006
- Explaining AdaBoost, Robert E. Schapire, http://rob.schapire.net/papers/explaining-adaboost.pdf
- https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf