

Generative and discriminative classification techniques

Machine Learning and Object Recognition 2015-2016

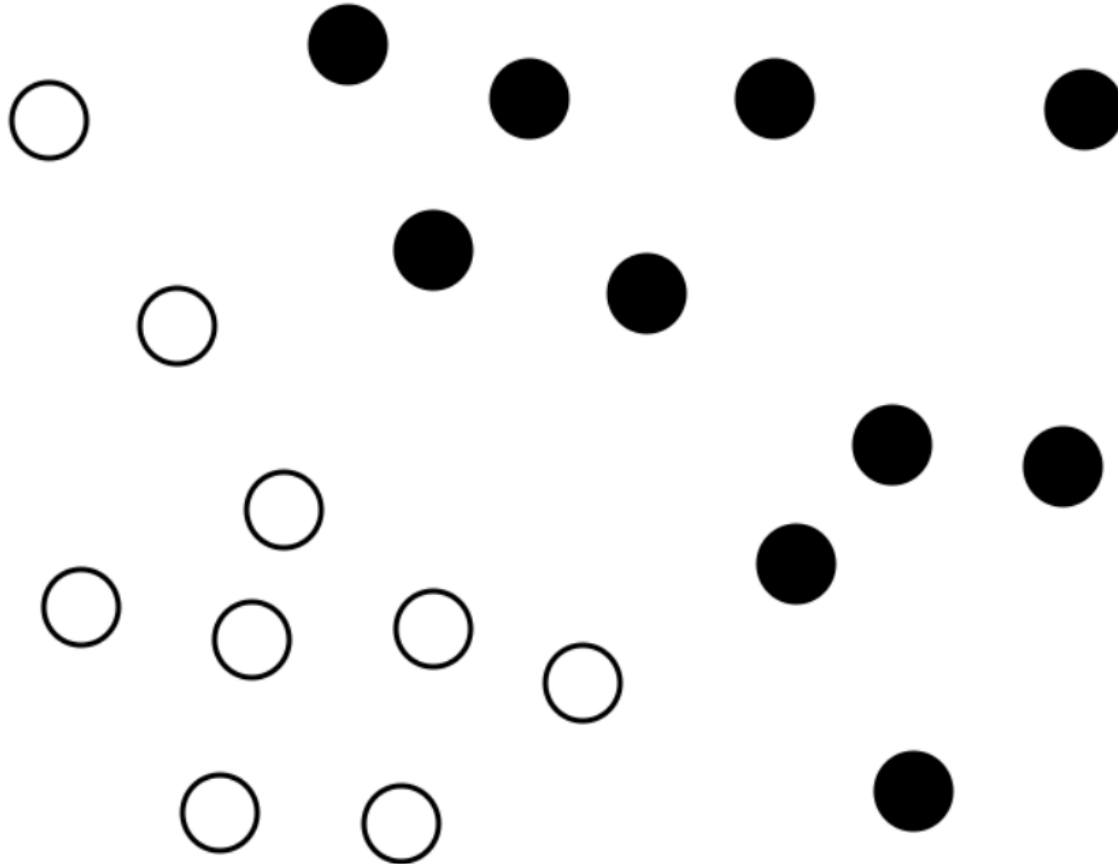
Jakob Verbeek, December 11, 2015

Course website:

<http://lear.inrialpes.fr/~verbeek/MLOR.15.16>

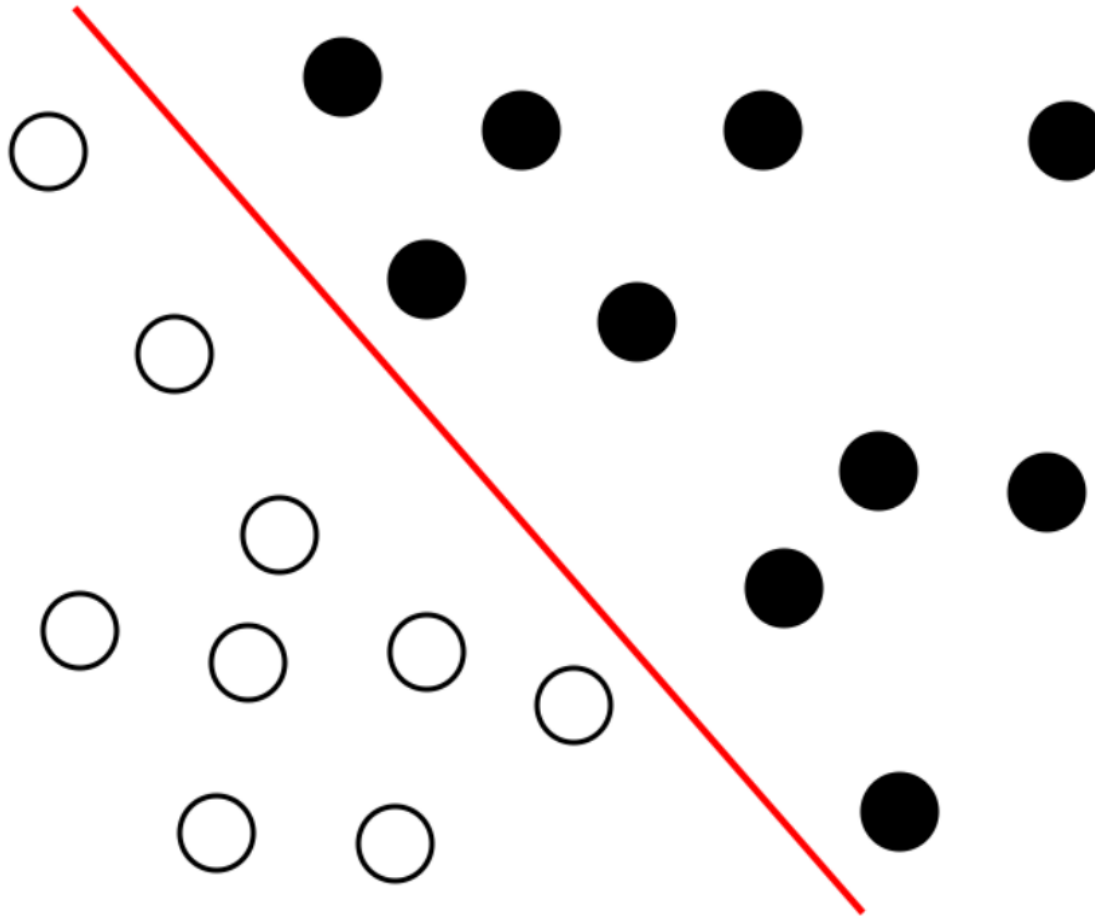
Classification

- Given training data labeled for two or more classes



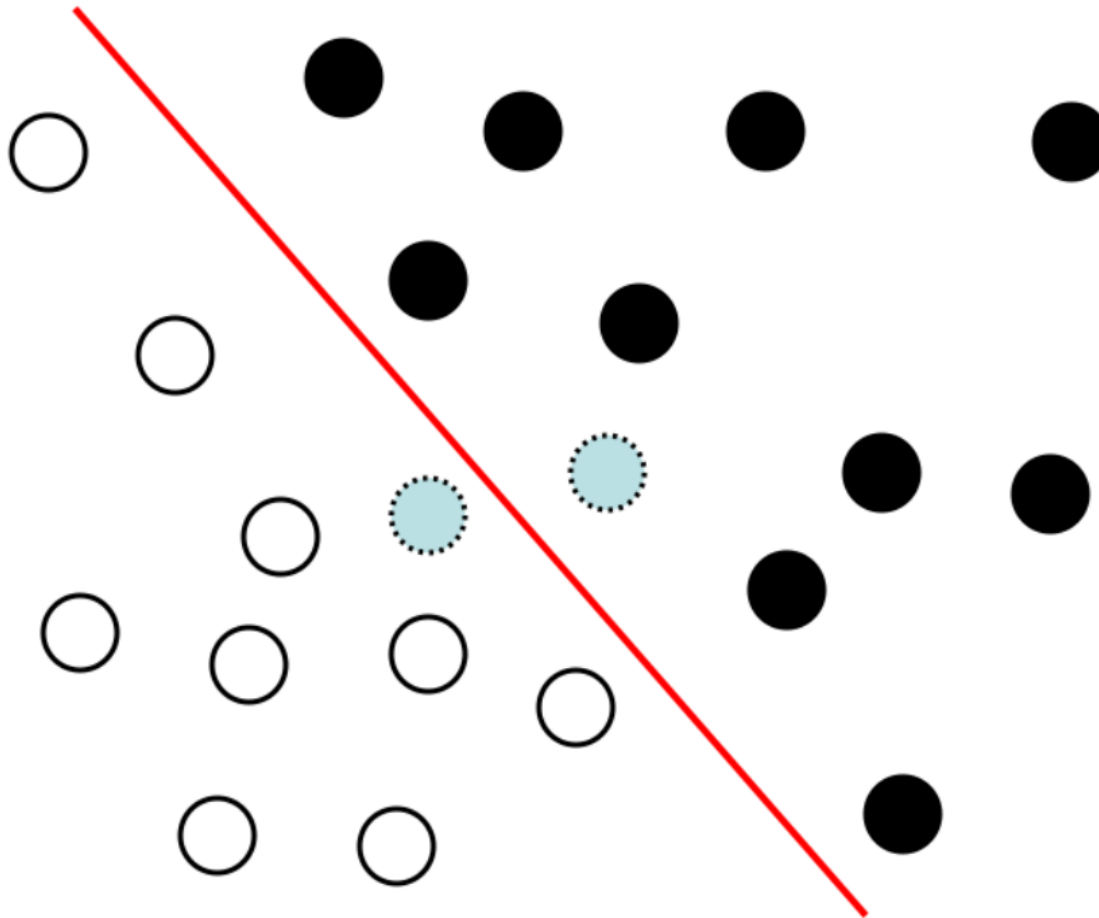
Classification

- Given training data labeled for two or more classes
- Determine a surface that separates those classes



Classification

- Given training data labeled for two or more classes
- Determine a surface that separates those classes
- Use that surface to predict the class membership of new data



Classification examples in category-level recognition

- Image classification: for each of a set of labels, predict if it is relevant or not for a given image.
- For example: Person = yes, TV = yes, car = no, ...



Classification examples in category-level recognition

- Category localization: predict bounding box coordinates.
- Classify each possible bounding box as containing the category or not.
- Report most confidently classified box.



Classification examples in category-level recognition

- Semantic segmentation: classify pixels to categories (multi-class)
- Impose spatial smoothness by Markov random field models.



Classification

- Goal is to predict for a test data input the corresponding class label.
 - **Data input x** , e.g. image but could be anything, format may be vector or other
 - **Class label y** , can take one out of at least 2 discrete values, can be more
- ▶ In binary classification we often refer to one class as “positive”, and the other as “negative”
- Classifier: function $f(x)$ that assigns a class to x , or probabilities over the classes.
- Training data: pairs (x,y) of inputs x , and corresponding class label y .
- Learning a classifier: determine function $f(x)$ from some family of functions based on the available training data.
- Classifier partitions the input space into regions where data is assigned to a given class
 - Specific form of these boundaries will depend on the family of classifiers used

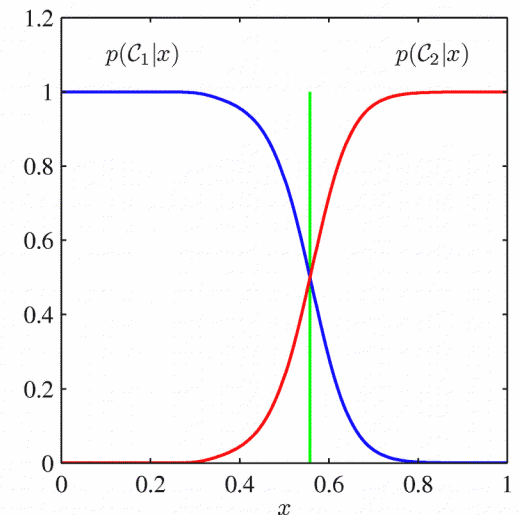
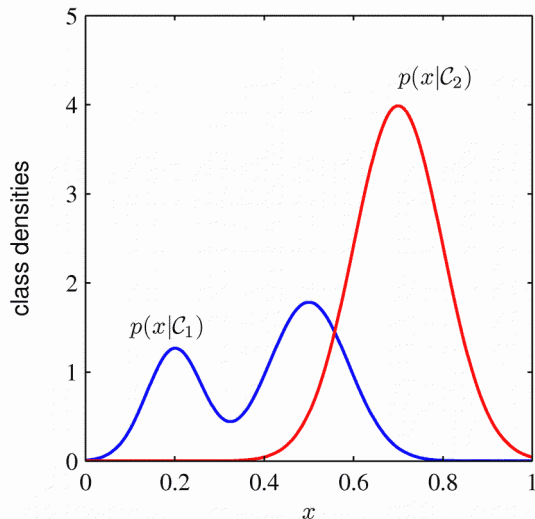
Generative classification: principle

- Model the class conditional distribution over data x for each class y : $p(x|y)$
 - ▶ Data of the class can be sampled (generated) from this distribution
- Estimate the a-priori probability that a class will appear $p(y)$
- Infer the probability over classes using Bayes' rule of conditional probability

$$p(y|x) = \frac{p(y) p(x|y)}{p(x)}$$

- Marginal distribution on x is obtained by marginalizing the class label y

$$p(x) = \sum_y p(y) p(x|y)$$



Generative classification: practice

- In order to apply Bayes' rule, we need to estimate two distributions.
- A-priori class distribution
 - ▶ In some cases the class prior probabilities are known in advance.
 - ▶ If the frequencies in the training data set are representative for the true class probabilities, then estimate the prior by these frequencies.
- Class conditional data distributions
 - ▶ Select a class of density models
 - Parametric model, e.g. Gaussian, Bernoulli, ...
 - Semi-parametric models: mixtures of Gaussian, Bernoulli, ...
 - Non-parametric models: histograms, nearest-neighbor method, ...
 - Or more structured models taking problem knowledge into account.
 - ▶ Estimate the parameters of the model using the data in the training set associated with that class.

Estimation of the class conditional model

- Given a set of n samples from a certain class, and a family of distributions

$$X = \{x_1, \dots, x_n\}$$

$$P = \{p_\theta(x); \theta \in \Theta\}$$

- How do we quantify the fit of a certain model to the data, and how do we find the best model defined in this sense?

- Maximum a-posteriori (MAP) estimation: use Bayes' rule again as follows:

- ▶ Assume a prior distribution over the parameters of the model $p(\theta)$

- ▶ Then the posterior likelihood of the model given the data is

$$p(\theta|X) = p(X|\theta)p(\theta)/p(X)$$

- ▶ Find the most likely model given the observed data

$$\hat{\theta} = \operatorname{argmax}_\theta p(\theta|X) = \operatorname{argmax}_\theta \{\ln p(\theta) + \ln p(X|\theta)\}$$

- Maximum likelihood parameter estimation: assume prior over parameters is uniform (for bounded parameter spaces), or “near uniform” so that its effect is negligible for the posterior on the parameters.

- ▶ In this case the MAP estimator is given by $\hat{\theta} = \operatorname{argmax}_\theta p(X|\theta)$

- ▶ For i.i.d. samples:

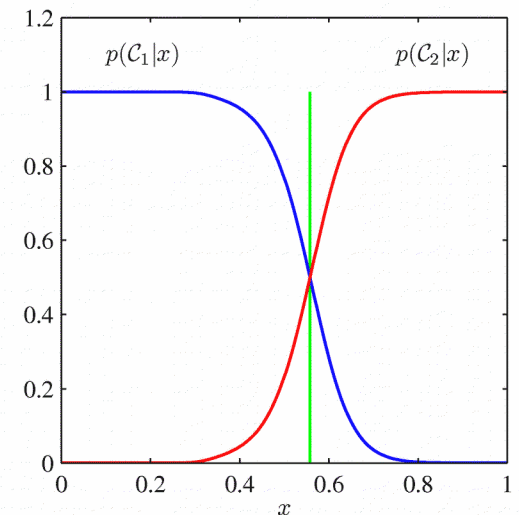
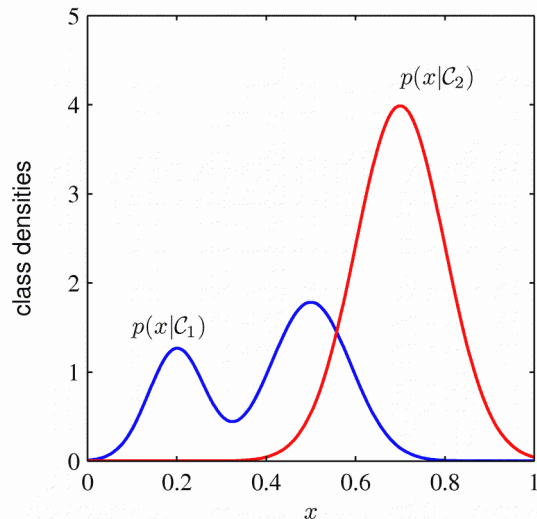
$$\hat{\theta} = \operatorname{argmax}_\theta \prod_{i=1}^n p(x_i|\theta) = \operatorname{argmax}_\theta \sum_{i=1}^n \ln p(x_i|\theta)$$

Generative classification methods

- Generative probabilistic methods use Bayes' rule for prediction
 - ▶ Problem is reformulated as one of parameter/density estimation

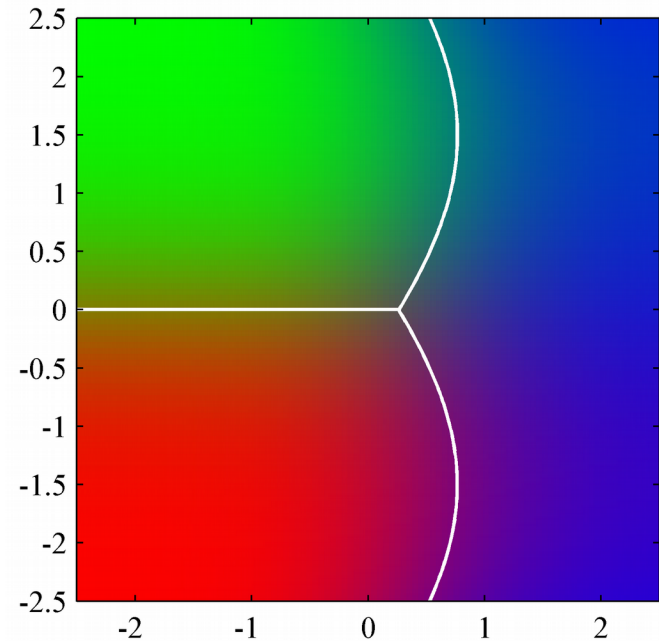
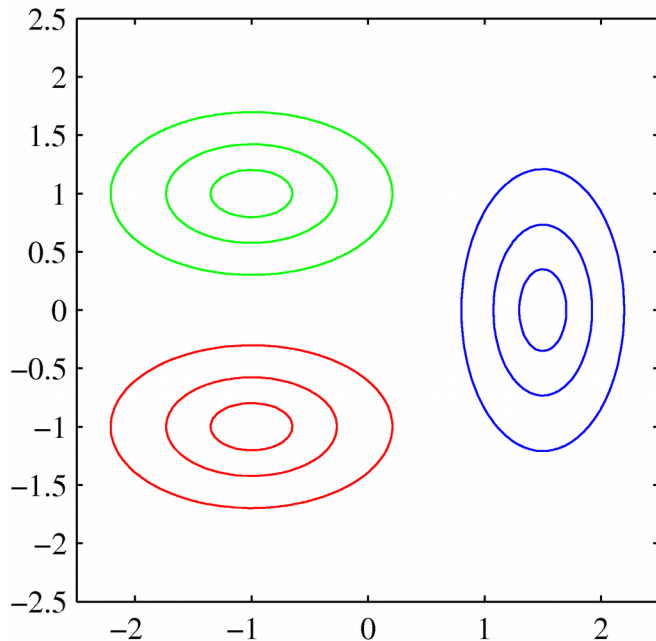
$$p(y|x) = \frac{p(y) p(x|y)}{p(x)} \quad p(x) = \sum_y p(y) p(x|y)$$

- Adding new classes to the model is easy:
 - ▶ Existing class conditional models stay as they are
 - ▶ Estimate $p(x|\text{new class})$ from training examples of new class
 - ▶ Re-estimate class prior probabilities



Example of generative classification

- Three-class example in 2D with parametric model
 - Single Gaussian model per class, uniform class prior
 - Exercise 1: how is this model related to the Gaussian mixture model we looked at before for clustering ?
 - Exercise 2: characterize surface of equal class probability when the covariance matrices are the same for all classes



$$p(y|x) = \frac{p(y)p(x|y)}{p(x)}$$

Density estimation, e.g. for class-conditional models

- Any type of data distribution may be used, preferably one that is modeling the data well, so that we can hope for accurate classification results.
- If we do not have a clear understanding of the data generating process, we can use a generic approach,
 - ▶ Gaussian distribution, or other reasonable parametric model
 - Estimation often in closed form or relatively simple process
 - ▶ Mixtures of parametric models
 - Estimation using EM algorithm, not more complicated than single parametric model
 - ▶ Non-parametric models can adapt to any data distribution given enough data for estimation. Examples: (multi-dimensional) histograms, and nearest neighbors.
 - Estimation often trivial, given a single smoothing parameter.

Histogram density estimation

- Suppose we have N data points use a histogram with C cells
- Consider maximum likelihood estimator

$$\hat{\theta} = \operatorname{argmax}_{\theta} \sum_{i=1}^n \ln p_{\theta}(x_i) = \operatorname{argmax}_{\theta} \sum_{c=1}^C n_c \ln \theta_c$$

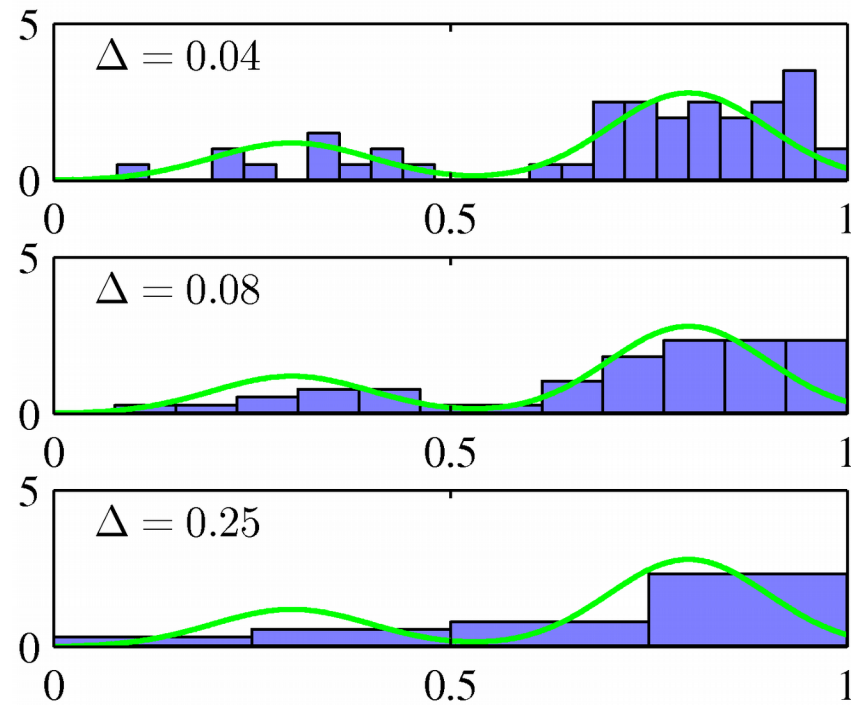
- Take into account constraint that density should integrate to one

$$\theta_C := 1 - \left(\sum_{k=1}^{C-1} v_k \theta_k \right) / v_C$$

- Exercise: derive maximum likelihood estimator

- Some observations:

- ▶ Discontinuous density estimate
- ▶ Cell size determines smoothness
- ▶ Number of cells scales exponentially with the dimension of the data



Histogram density estimation

- Suppose we have N data points use a histogram with C cells
- Data log-likelihood

$$L(\theta) = \sum_{i=1}^N \ln p_{\theta}(x_i) = \sum_{c=1}^C n_c \ln \theta_c$$

- Take into account constraint that density should integrate to one

$$\theta_C := 1 - \left(\sum_{k=1}^{C-1} v_k \theta_k \right) / v_C$$

- Compute derivative, and set to zero for $i=1, \dots, C-1$

$$\frac{\partial L(\theta)}{\partial \theta_i} = \frac{n_i}{\theta_i} - \frac{n_c}{\theta_c} \frac{v_i}{v_c}$$

$$\theta_i v_i = \frac{\theta_c v_c}{n_c} n_i$$

- Use fact that probability mass should integrate to one, and substitute

$$\sum_{i=1}^C \theta_i v_i = \frac{\theta_c v_c}{n_c} \sum_{i=1}^C n_i = \frac{\theta_c v_c}{n_c} N = 1$$

$$\theta_i = \frac{n_i}{v_i N}$$

The Naive Bayes model

- Histogram estimation, and other methods, scale poorly with data dimension
 - ▶ Fine division of each dimension: many empty bins
 - ▶ Rough division of each dimension: poor density model
 - Even for one cut per dimension: 2^D cells, eg. a million cells in 20 dims.
- The number of parameters can be made linear in the data dimension by assuming independence between the dimensions

$$p(x) = \prod_{d=1}^D p(x(d))$$

- For example, for histogram model: we estimate a histogram per dimension
 - ▶ Still C^D cells, but only $D \times C$ parameters to estimate, instead of C^D
- Independence assumption can be unrealistic for high dimensional data
 - ▶ But classification performance may still be good using the derived $p(y|x)$
 - ▶ Partial independence, e.g. using graphical models, relaxes this problem.
- Principle can be applied to estimation with any type of density estimate

Example of a naïve Bayes model

- Hand-written digit classification

- Input: binary 28x28 scanned digit images, collect in 784 long bit string



- Desired output: class label of image

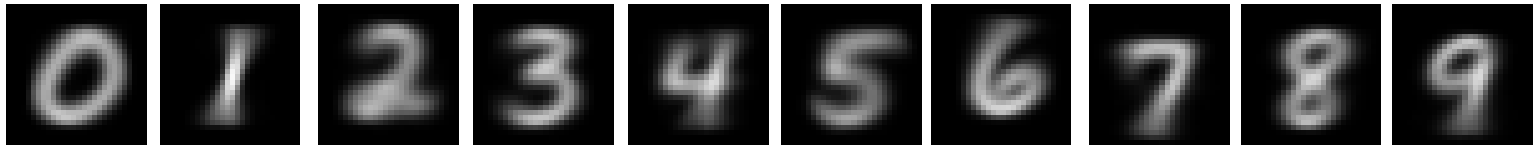
- Generative model over 28 x 28 pixel images: 2^{784} possible images

- Independent Bernoulli model for each class
- Probability per pixel per class
- Maximum likelihood estimator is average value per pixel/bit per class

$$p(x|y=c) = \prod_d p(x^d|y=c)$$

$$p(x^d=1|y=c) = \theta_{cd}$$

$$p(x^d=0|y=c) = 1 - \theta_{cd}$$



- Classify using Bayes' rule: $p(y|x) = \frac{p(y) p(x|y)}{p(x)}$

k-nearest-neighbor density estimation: principle

- **Instead of having fixed cells** as in histogram method,
 - ▶ **Center cell on the test sample** for which we evaluate the density.
 - ▶ Fix number of samples in the cell, find the corresponding **cell size**.

- Probability to find a point in a sphere **A** centered on x_0 with volume **v** is

$$P(x \in A) = \int_A p(x) dx$$

- A smooth density is approximately constant in small region, and thus

$$P(x \in A) = \int_A p(x) dx \approx \int_A p(x_0) dx = p(x_0) v_A$$

- Alternatively: estimate **P** from the fraction of training data in **A**: $P(x \in A) \approx \frac{k}{N}$
 - Total N data points, k in the sphere **A**

- Combine the above to obtain estimate $p(x_0) \approx \frac{k}{N v_A}$
 - ▶ Same per-cell density estimate as in histogram estimator
- Note: density estimates not guaranteed to integrate to one!

k-nearest-neighbor density estimation: practice

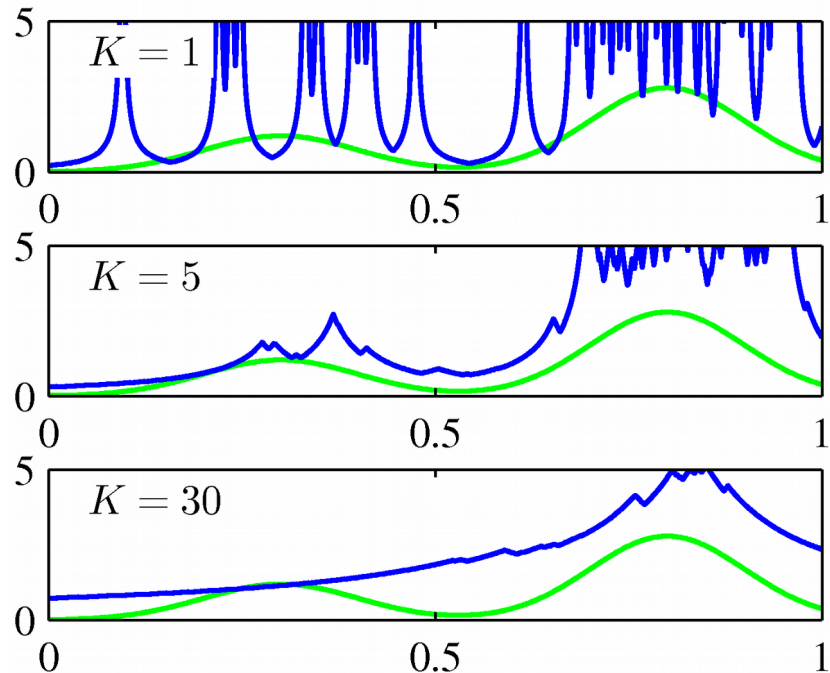
- Procedure in practice:

- ▶ Choose *k*
- ▶ For given *x*, compute the volume *v* which contain *k* samples.
- ▶ Estimate density with $p(x) \approx \frac{k}{Nv}$

- Volume of a sphere with radius *r* in *d* dimensions is
$$v(r, d) = \frac{2r^d \pi^{d/2}}{\Gamma(d/2 + 1)}$$

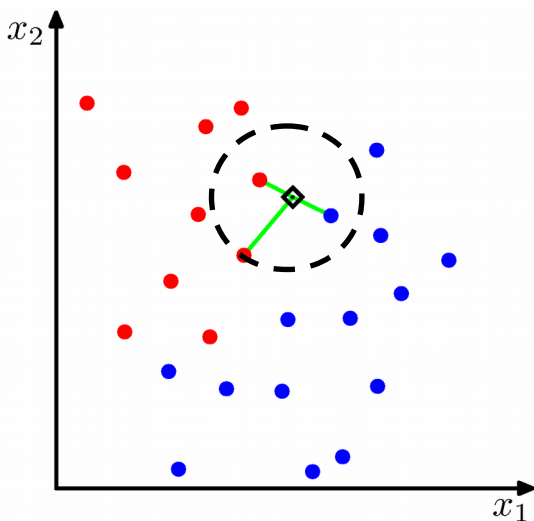
- What effect does *k* have?

- ▶ Data sampled from mixture of Gaussians plotted in green
- ▶ Larger *k*, larger region, smoother estimate
- ▶ Similar role as cell size for histogram estimation

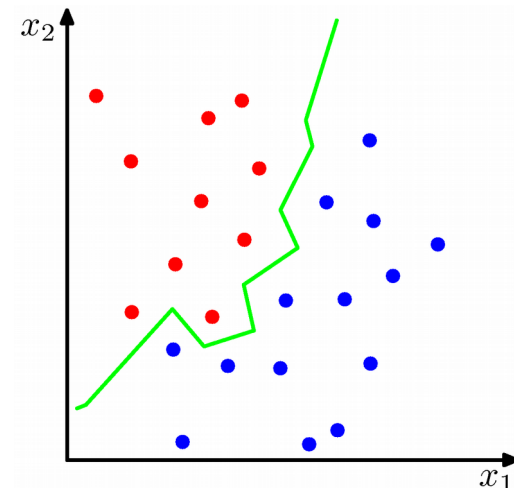


K-nearest-neighbors for classification

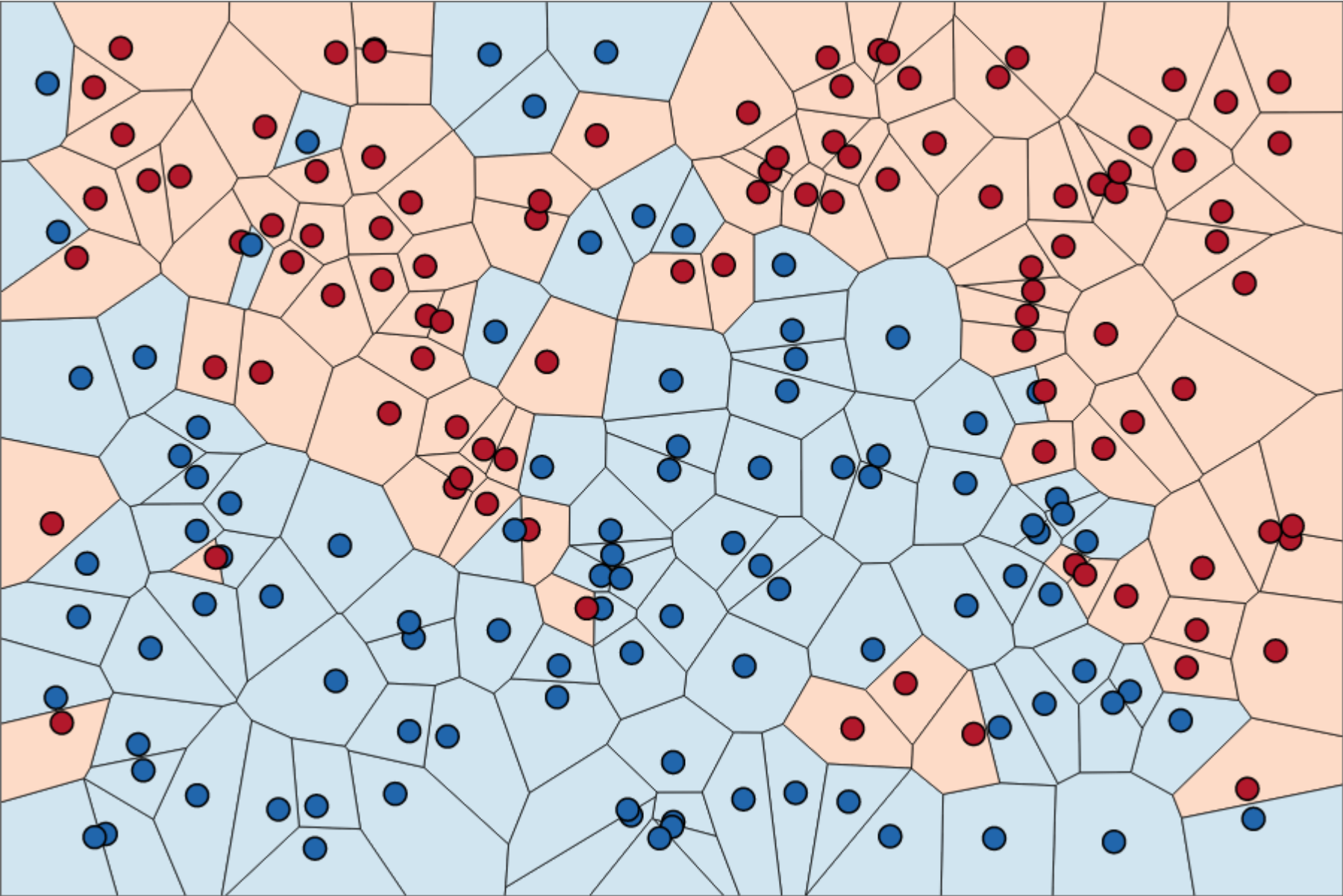
- Use Bayes' rule with kNN density estimation for $p(x|y)$
 - ▶ Find sphere volume v to capture k data points for estimate $p(x) = \frac{k}{Nv}$
 - ▶ Use the same sphere for each class for estimates $p(x|y=c) = \frac{k_c}{N_c v}$
 - ▶ Estimate class prior probabilities $p(y=c) = \frac{N_c}{N}$
 - ▶ Calculate class posterior distribution as fraction of k neighbors in class c



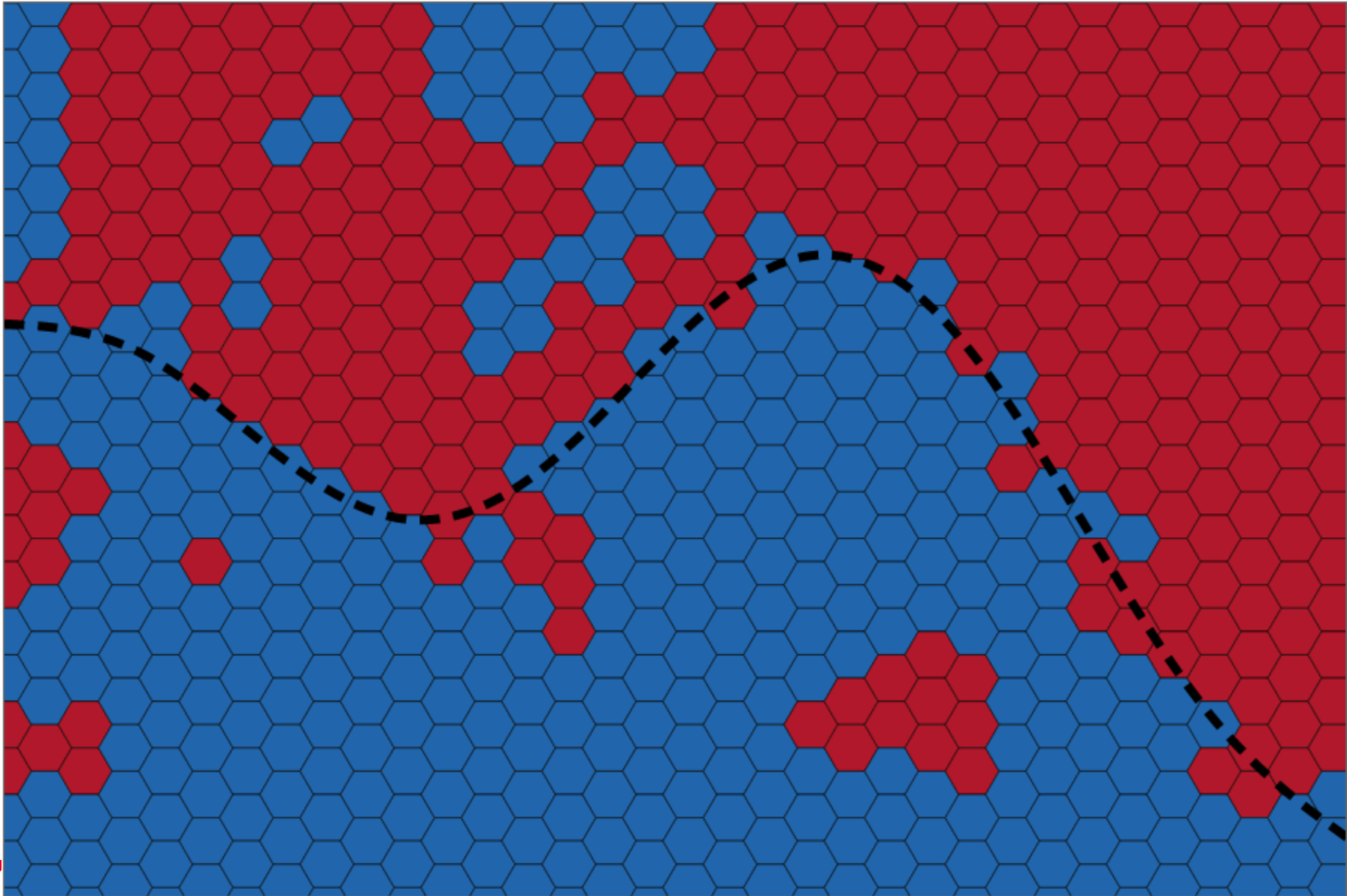
$$\begin{aligned} p(y=c|x) &= \frac{p(y=c) p(x|y=c)}{p(x)} \\ &= \frac{1}{p(x)} \frac{k_c}{Nv} \\ &= \frac{k_c}{k} \end{aligned}$$



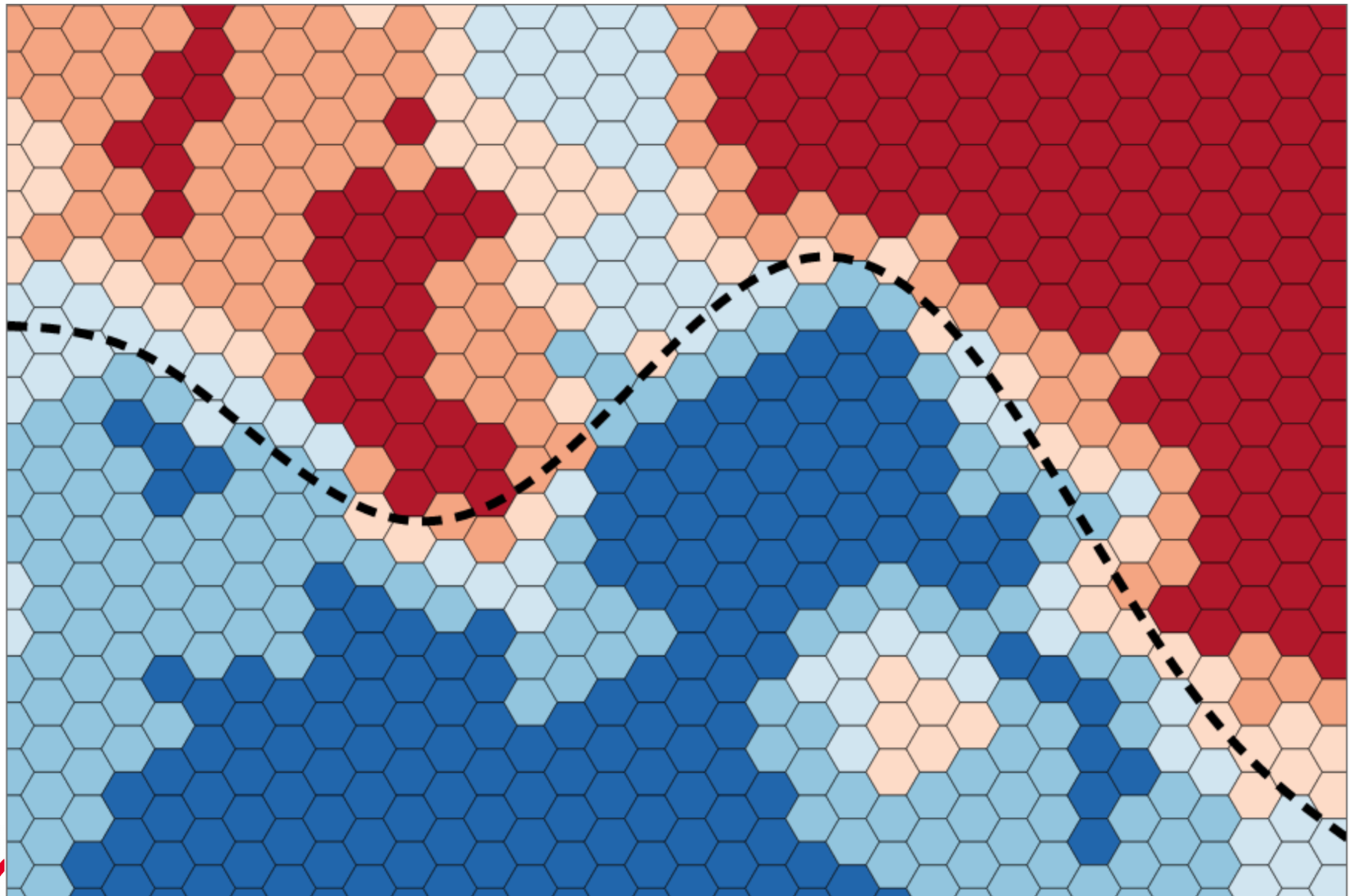
Smoothing effects for large values of k: data set



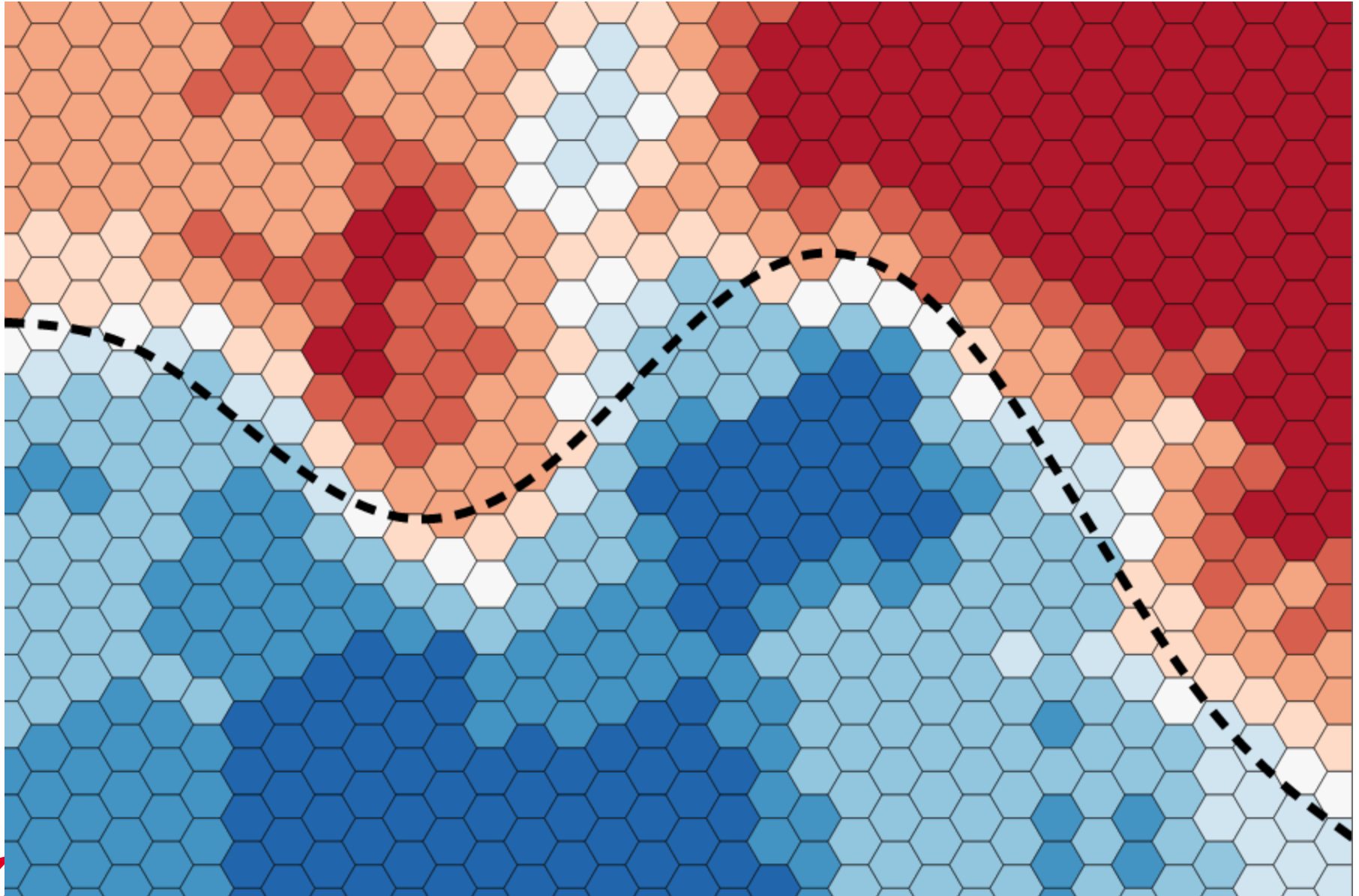
Smoothing effects for large values of k , $k=1$



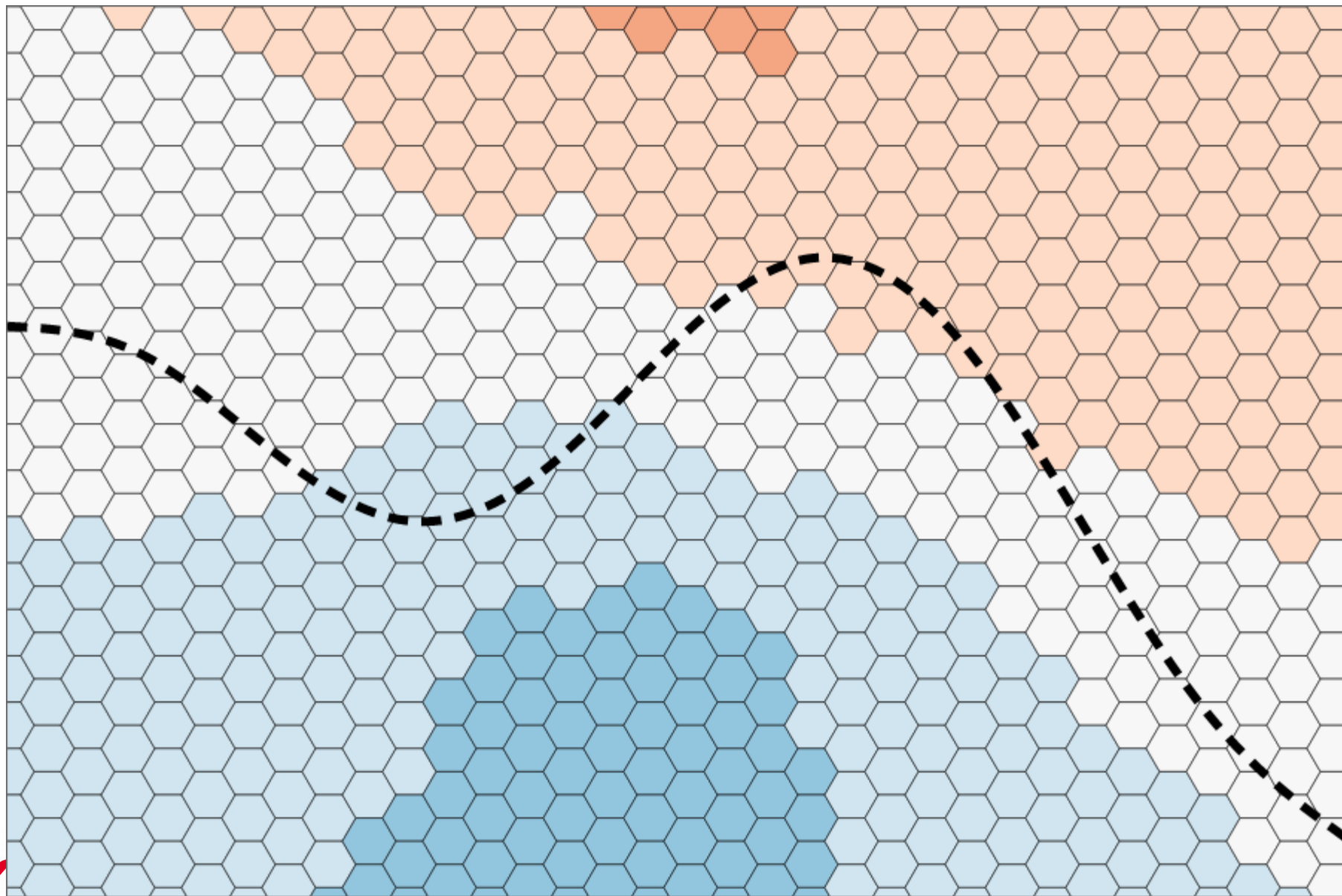
Smoothing effects for large values of k , $k=5$



Smoothing effects for large values of k , $k=10$



Smoothing effects for large values of k , $k=100$

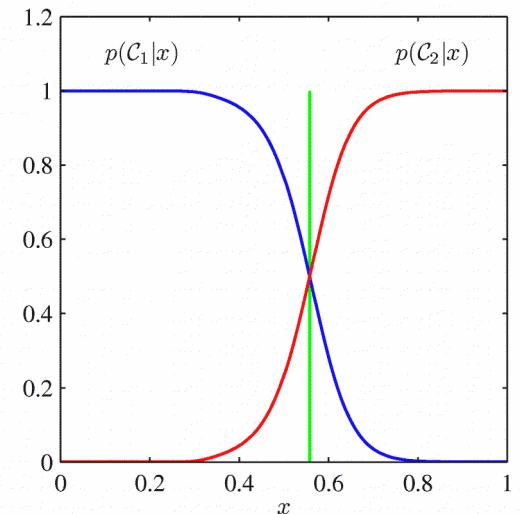
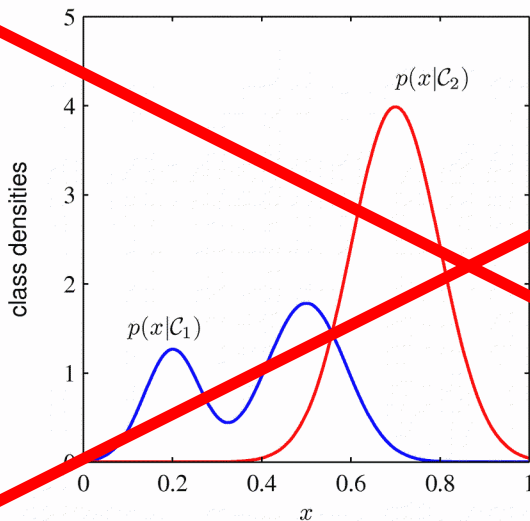


Summary generative classification methods

- (Semi-) Parametric models, e.g. $p(x|y)$ is Gaussian, or mixture of ...
 - ▶ Pros: no need to store training data, just the class conditional models
 - ▶ Cons: may fit the data poorly, and might therefore lead to poor classification result
- Non-parametric models:
 - ▶ Pros: flexibility, no assumptions distribution shape, “learning” is trivial. KNN can be used for anything that comes with a distance.
 - ▶ Cons of histograms:
 - Only practical in low dimensional data (<5 or so), application in high dimensional data leads to exponentially many and mostly empty cells
 - Naïve Bayes modeling in higher dimensional cases
 - Cons of k-nearest neighbors
 - Need to store all training data (memory cost)
 - Computing nearest neighbors (computational cost)

Discriminative classification methods

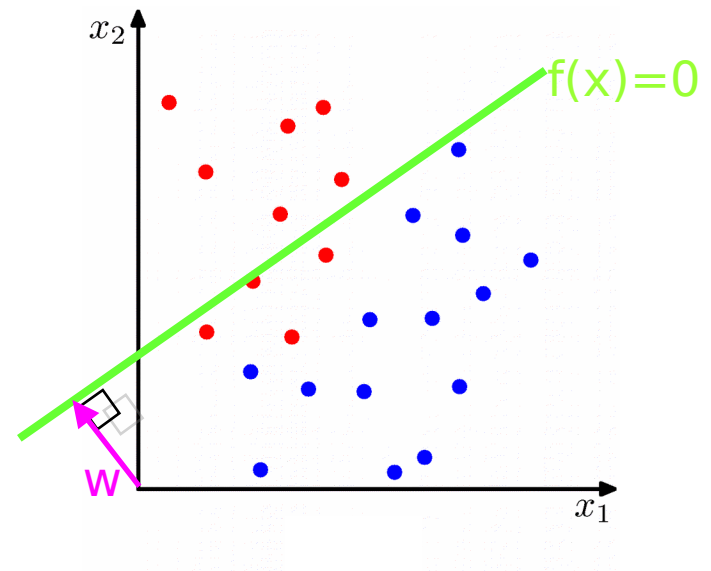
- Generative classification models
 - Model the density of inputs x from each class $p(x|y)$
 - Estimate class prior probability $p(y)$
 - Use Bayes' rule to infer distribution over class given input
- In discriminative classification methods we directly estimate class probability given input: $p(y|x)$
 - ▶ Choose class of decision functions in feature space
 - ▶ Estimate function that maximizes performance on the training set
 - ▶ Classify a new pattern on the basis of this decision rule.



Binary linear classifier

- Decision function is linear in the features:

$$f(x) = w^T x + b = b + \sum_{i=1}^d w_i x_i$$



- Classification based on the sign of $f(x)$
- Orientation is determined by w
 - ▶ w is the surface normal
- Offset from origin is determined by b
- Decision surface is $(d-1)$ dimensional hyper-plane orthogonal to w , given by

$$f(x) = w^T x + b = 0$$

- Exercise: What happens in 3d with $w=(1,0,0)$ and $b = -1$?

Binary linear classifier

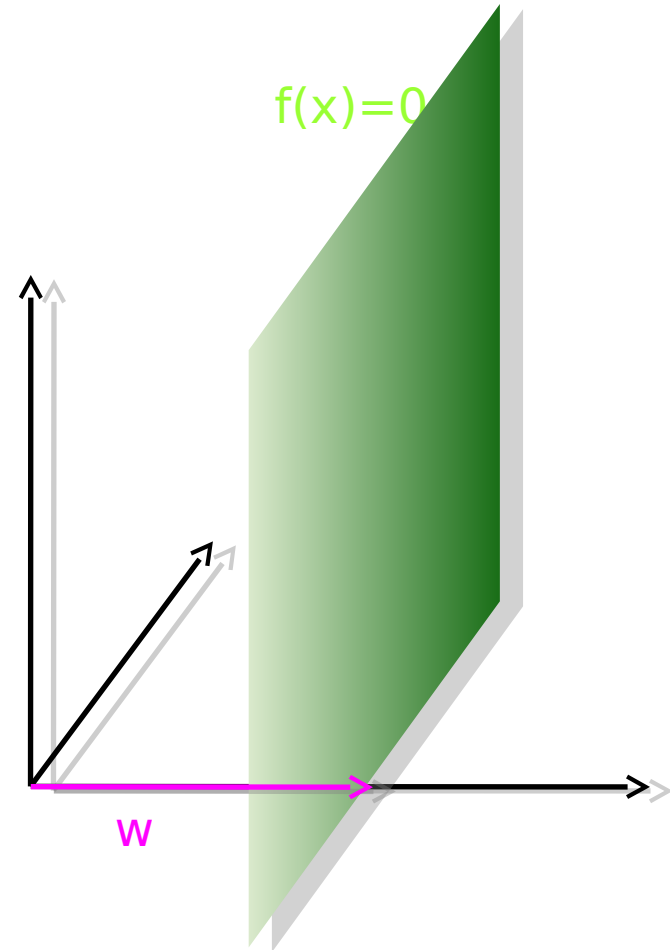
- Decision surface for $w=(1,0,0)$ and $b = -1$

$$f(x) = w^T x + b = 0$$

$$b + \sum_{i=1}^d w_i x_i = 0$$

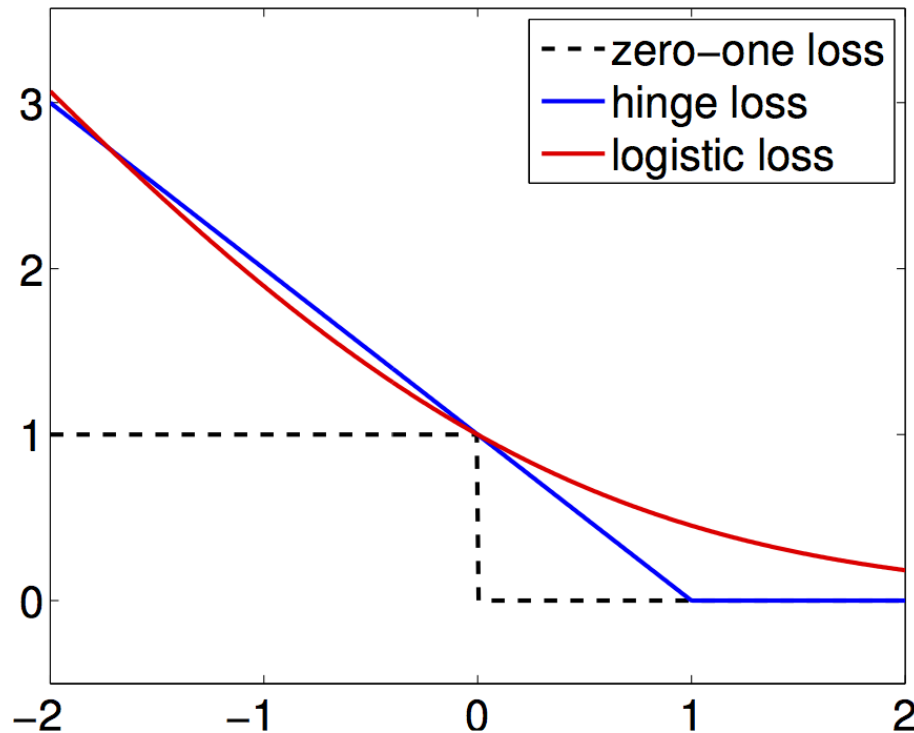
$$x_1 - 1 = 0$$

$$x_1 = 1$$



Common loss functions for classification

- Assign class label using $y = \text{sign}(f(x))$
- Measure how model quality on a test sample using loss function
 - ▶ Zero-One loss: $L(y_i, f(x_i)) = [y_i f(x_i) \leq 0]$
 - ▶ Hinge loss: $L(y_i, f(x_i)) = \max(0, 1 - y_i f(x_i))$
 - ▶ Logistic loss: $L(y_i, f(x_i)) = \log_2(1 + e^{-y_i f(x_i)})$



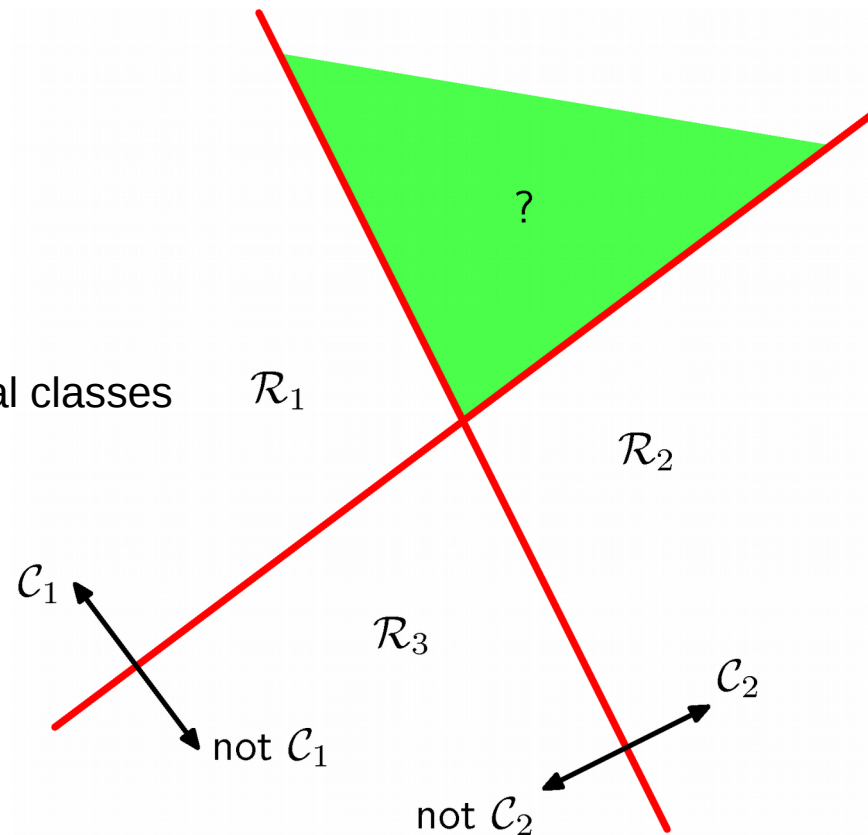
Common loss functions for classification

- Assign class label using $y = \text{sign}(f(x))$
 - ▶ Zero-One loss: $L(y_i, f(x_i)) = [y_i f(x_i) \leq 0]$
 - ▶ Hinge loss: $L(y_i, f(x_i)) = \max(0, 1 - y_i f(x_i))$
 - ▶ Logistic loss: $L(y_i, f(x_i)) = \log_2(1 + e^{-y_i f(x_i)})$
- The zero-one loss counts the number of misclassifications, which is the “ideal” empirical loss
 - ▶ Discontinuity at zero makes optimization intractable
- Hinge and logistic loss provide continuous and convex upperbounds, which can be optimized with continuous methods

Dealing with more than two classes

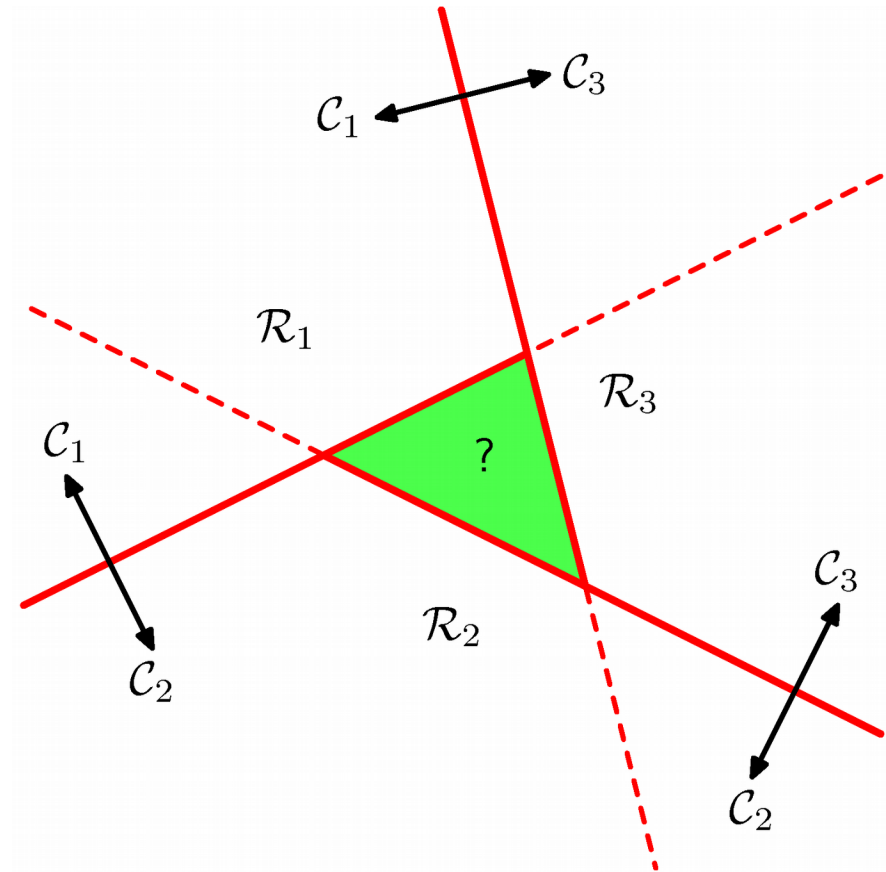
- First idea: construction from multiple binary classifiers
 - ▶ Learn binary “base” classifiers independently
- One vs rest approach:
 - ▶ 1 vs (2 & 3)
 - ▶ 2 vs (1 & 3)
 - ▶ 3 vs (1 & 2)

- Problem: Region claimed by several classes



Dealing with more than two classes

- First idea: construction from multiple binary classifiers
 - ▶ Learn binary “base” classifiers independently
- One vs one approach:
 - ▶ 1 vs 2
 - ▶ 1 vs 3
 - ▶ 2 vs 3
- Problem: conflicts in some regions



Dealing with more than two classes

- Instead: define a separate linear score function for each class

$$f_k(x) = w_k^T x + b_k$$

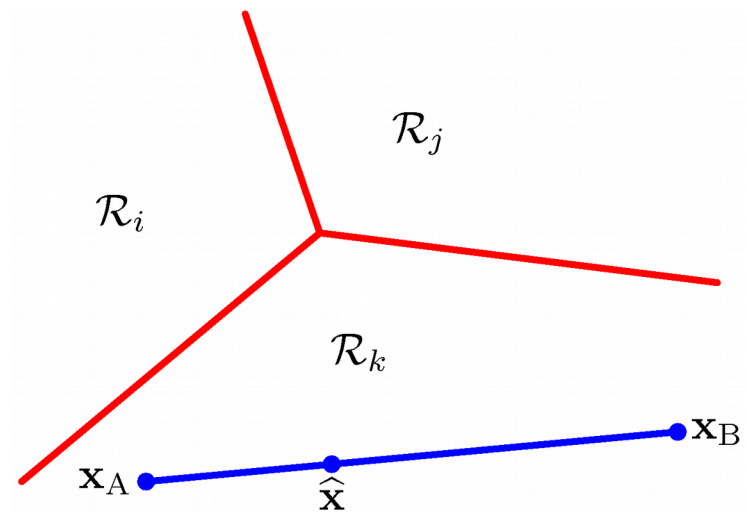
- Assign sample to the class of the function with maximum value

$$y = \arg \max_k f_k(x)$$

- Exercise 1: give the expression for points where two classes have equal score

- Exercise 2: show that the set of points assigned to a class is convex

- ▶ If two points fall in the region, then also all points on connecting line



Logistic discriminant for two classes

- Map linear score function to class probabilities with sigmoid function

$$p(y=+1|x) = \sigma(w^T x + b)$$

- ▶ For binary classification problem, we have by definition

$$p(y=-1|x) = 1 - p(y=+1|x)$$

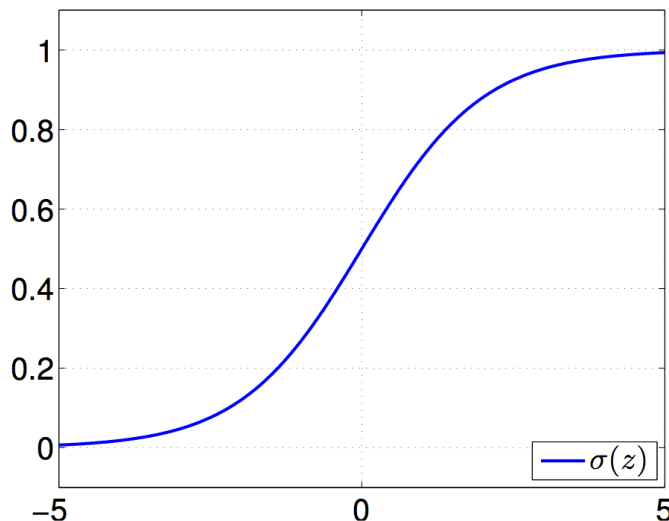
- ▶ Exercise: show that

$$p(y=-1|x) = \sigma(-(w^T x + b))$$

- ▶ Therefore:

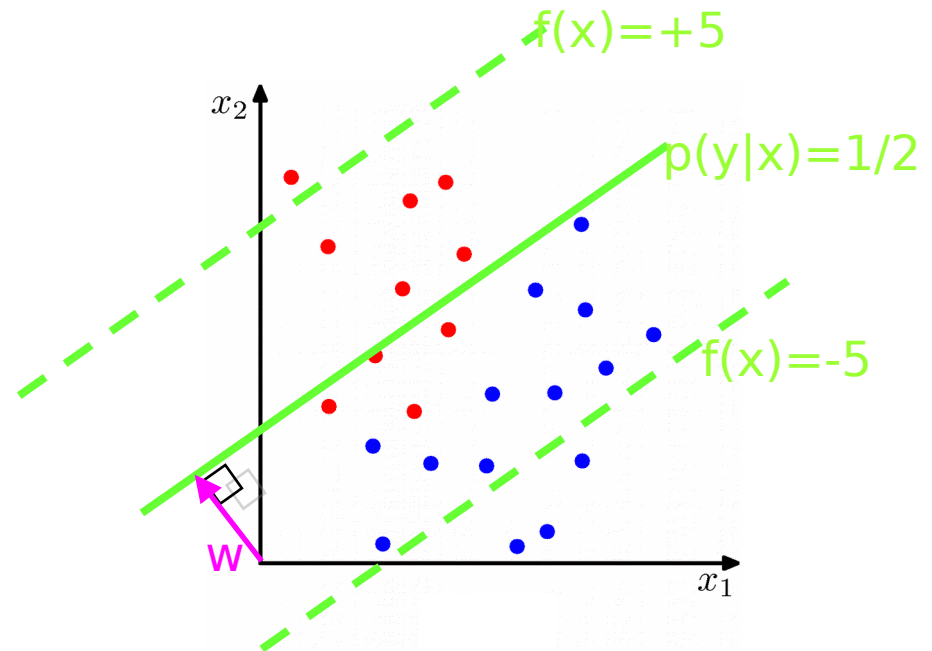
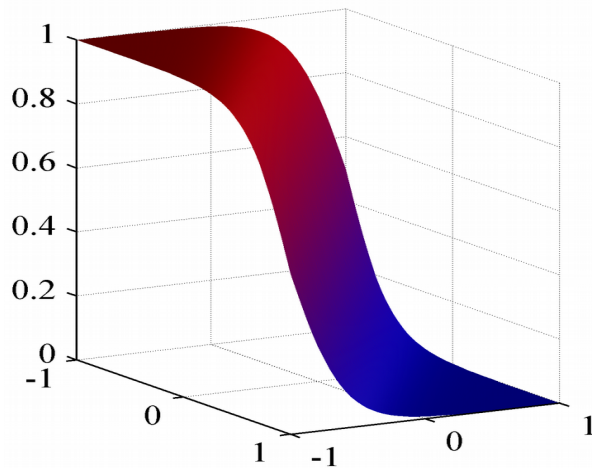
$$p(y|x) = \sigma(y(w^T x + b))$$

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



Logistic discriminant for two classes

- Map linear score function to class probabilities with sigmoid function
- The class boundary is obtained for $p(y|x)=1/2$, thus by setting linear function in exponent to zero



Multi-class logistic discriminant

- Map score function of each class to class probabilities with “soft-max” function
 - ▶ Absorb bias into w and x

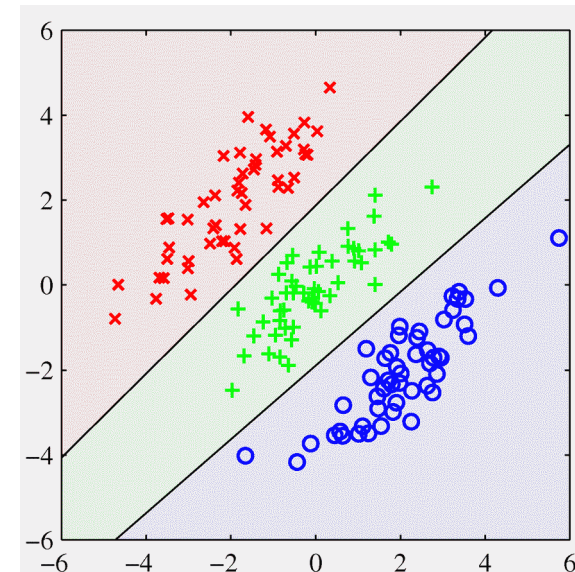
$$f_k(x) = w_k^T x$$

$$p(y=c|x) = \frac{\exp(f_c(x))}{\sum_{k=1}^K \exp(f_k(x))}$$

- ▶ The class probability estimates are non-negative, and sum to one.
- ▶ Relative probability of most likely class increases exponentially with the difference in the linear score functions

$$\frac{p(y=c|x)}{p(y=k|x)} = \frac{\exp(f_c(x))}{\exp(f_k(x))} = \exp(f_c(x) - f_k(x))$$

- ▶ For any given pair of classes we find that they are equally likely on a hyperplane in the feature space



Maximum likelihood parameter estimation

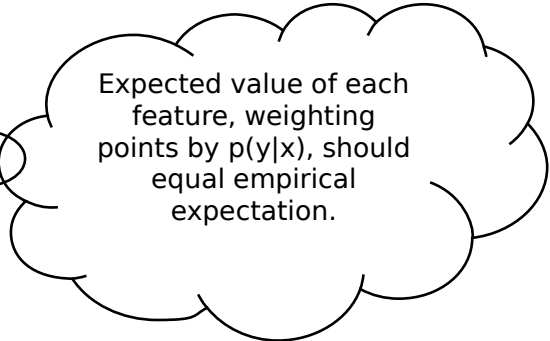
- Maximize the log-likelihood of predicting the correct class label for training data
 - ▶ Predictions are made independently, so sum log-likelihood of all training data

$$L = \sum_{n=1}^N \log p(y_n | x_n)$$

- Derivative of log-likelihood as intuitive interpretation

$$\frac{\partial L}{\partial w_k} = \sum_{n=1}^N ([y_n = k] - p(y = k | x_n)) x_n = \sum_{n=1}^N \alpha_n x_n$$

Indicator function
1 if $y_n = k$, else 0



Expected value of each feature, weighting points by $p(y|x)$, should equal empirical expectation.

- No closed-form solution, but log-likelihood is concave in parameters
 - ▶ no local optima, use general purpose convex optimization methods
 - ▶ For example: gradient started from $w=0$
 - w is linear combination of data points
 - Sign of coefficients depends on class labels

Maximum a-posteriori (MAP) parameter estimation

- Let us assume a zero-mean Gaussian prior distribution on w
 - ▶ We expect weight vectors with a small norm

- Find w that maximizes posterior likelihood

$$\hat{w} = \operatorname{argmax}_w \sum_{n=1}^N \ln p(y_n | x_n, w) + \ln p(w)$$

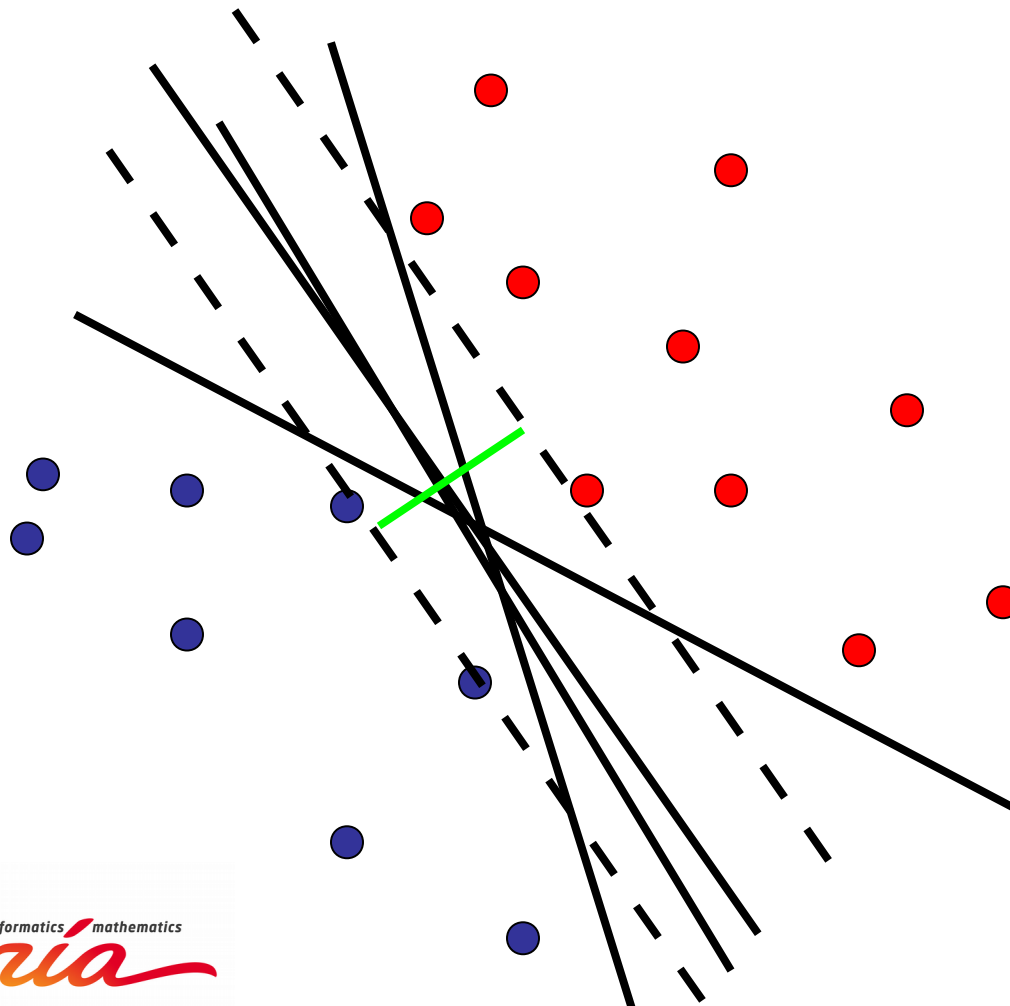
- Can be rewritten as following “penalized” maximum likelihood estimator:

$$\hat{w} = \operatorname{argmax}_w \sum_{n=1}^N \ln p(y_n | x_n, w) - \lambda \|w\|_2^2$$

- ▶ With lambda non-negative
- Penalty for “large” w , bounds the scale of w in case of separable data
- Exercise: show that for separable data the norm of the optimal w 's would be infinite without using the penalty term.

Support Vector Machines

- Find linear function to separate positive and negative examples
- Which function best separates the samples ?
 - ▶ Function inducing the largest **margin**



$$y_i = +1 : w^T x_i + b > 0$$
$$y_i = -1 : w^T x_i + b < 0$$

Support vector machines

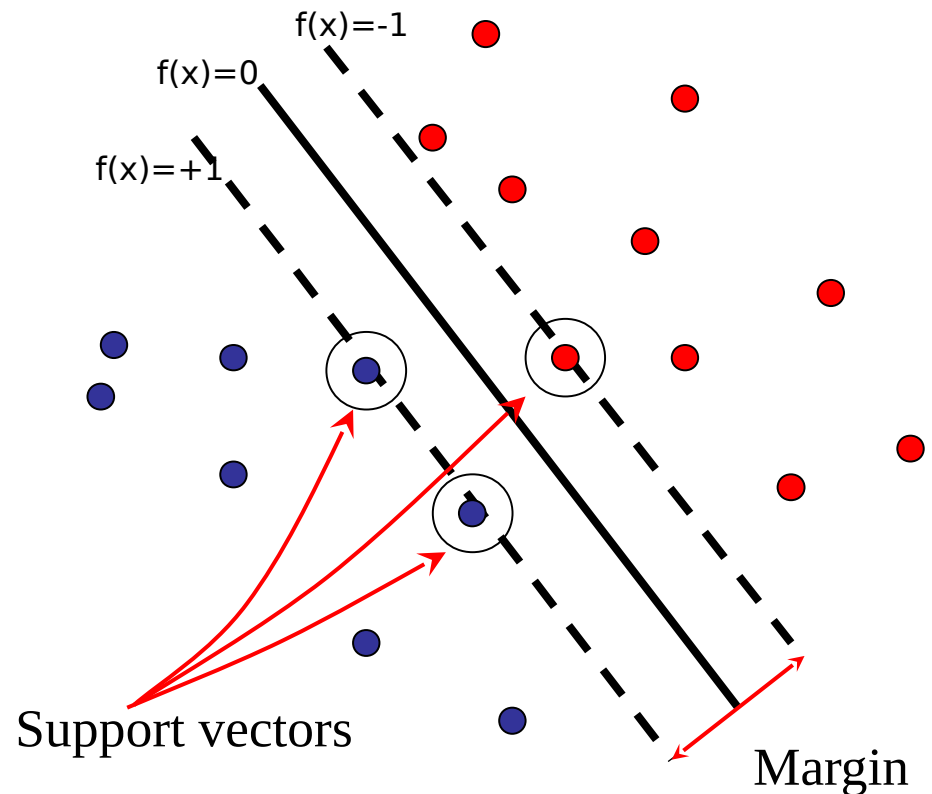
- Without loss of generality, let function value at the margin be +/- 1
- Now constrain w to that all points fall on correct side of the margin:

$$y_i(w^T x_i + b) \geq 1$$

- By construction we have that the “support vectors”, the ones that define the margin, have function values

$$w^T x_i + b = y_i$$

- Express the size of the margin in terms of w .



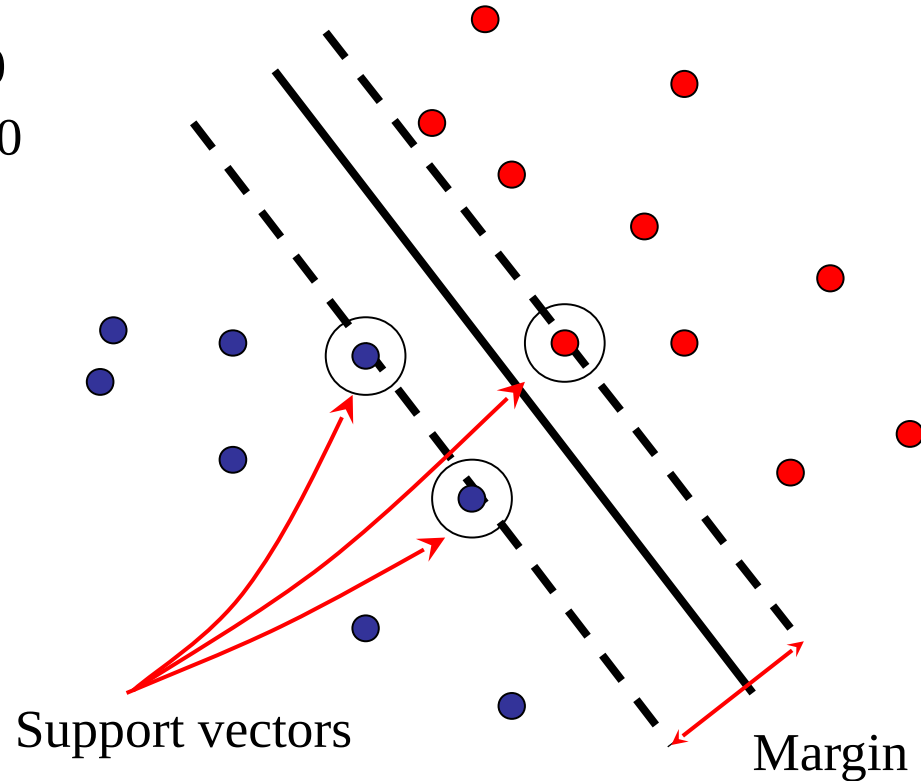
Support vector machines

- Let's consider a support vector x from the positive class $f(x) = w^T x + b = 1$
- Let z be its projection on the decision plane
 - ▶ Since w is normal vector to the decision plane, we have $z = x - \alpha w$
 - ▶ and since z is on the decision plane $f(z) = w^T (x - \alpha w) + b = 0$

- Solve for alpha
$$w^T (x - \alpha w) + b = 0$$
$$w^T x + b - \alpha w^T w = 0$$
$$\alpha w^T w = 1$$
$$\alpha = \frac{1}{\|w\|_2^2}$$

- Margin is twice distance from x to z

$$\|x - z\|_2 = \|x - (x - \alpha w)\|_2$$
$$\|\alpha w\|_2 = \alpha \|w\|_2$$
$$\frac{\|w\|_2}{\|w\|_2^2} = \frac{1}{\|w\|_2}$$

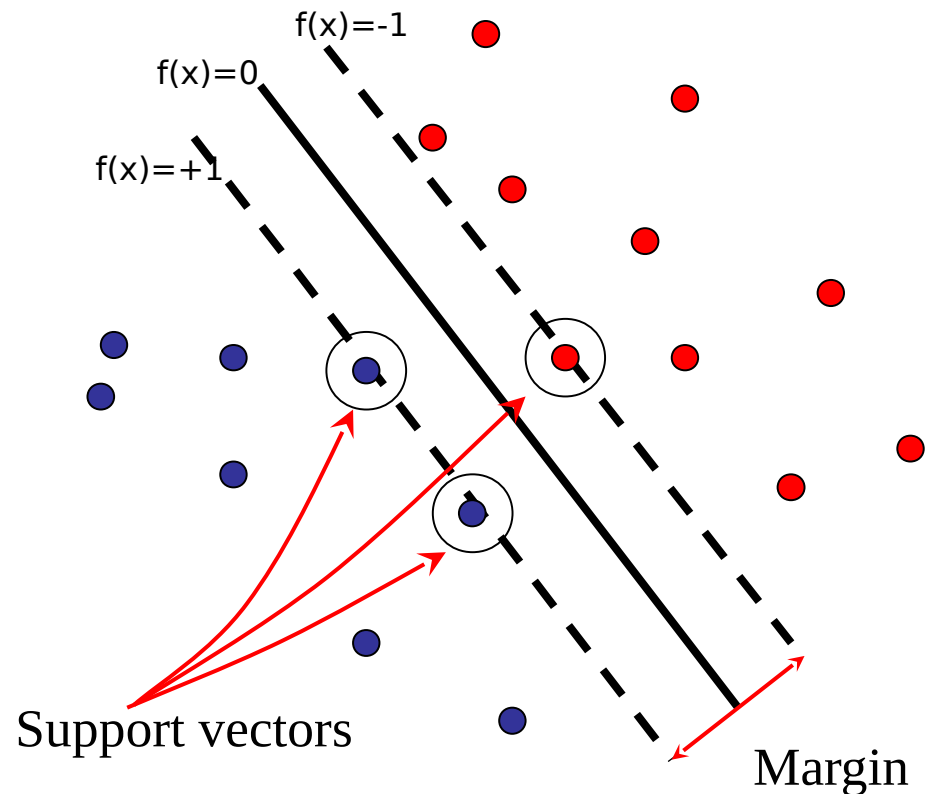


Support vector machines

- To find the maximum-margin separating hyperplane, we
 - ▶ Maximize the margin, while ensuring correct classification
 - ▶ Minimize the norm of w , s.t. $\forall_i: y_i(w^T x_i + b) \geq 1$
- Solve using quadratic program with linear inequality constraints over $p+1$ variables

$$\operatorname{argmin}_{w,b} \frac{1}{2} w^T w$$

subject to $y_i(w^T x_i + b) \geq 1$

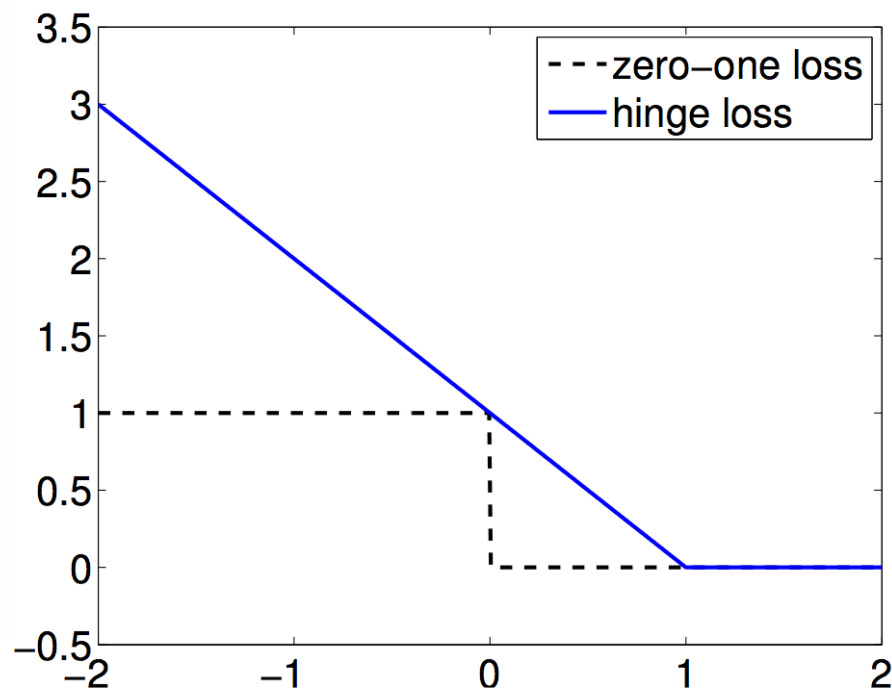
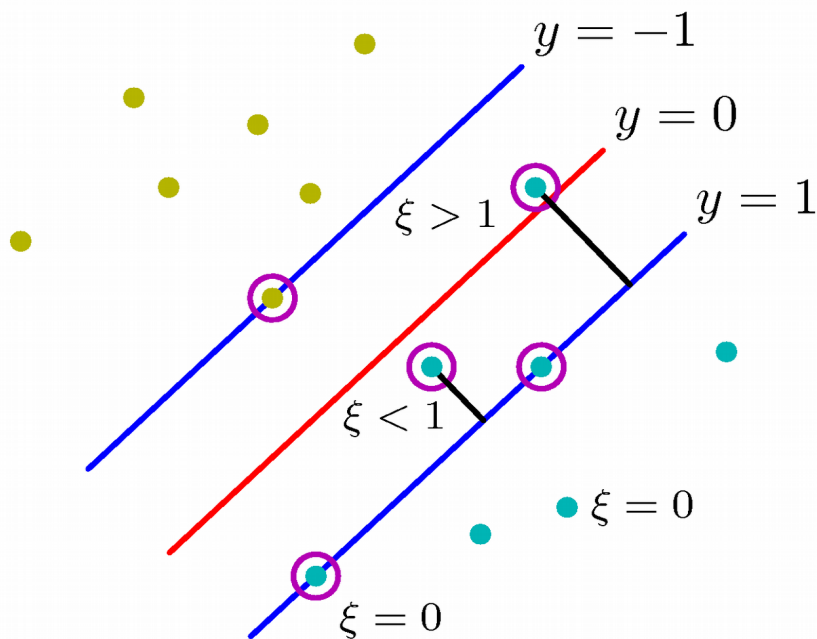


Support vector machines: inseperable classes

- For non-separable classes we incorporate hinge-loss

$$L(y_i, f(x_i)) = \max(0, 1 - y_i f(x_i))$$

- Recall: convex and piece-wise linear upper bound on zero/one loss.
 - ▶ Zero if point on the correct side of the margin
 - ▶ Otherwise given by absolute difference from score at margin



Support vector machines: inseperable classes

- Minimize penalized loss function

$$\min_{w,b} \lambda \frac{1}{2} w^T w + \sum_i \max(0, 1 - y_i(w^T x_i + b))$$

- ▶ Quadratic function, plus **piece-wise linear functions**.
- Transformation into a quadratic program
 - ▶ Define “slack variables” that measure the loss for each data point
 - ▶ Should be non-negative, and at least as large as the loss

$$\min_{w,b,\{\xi_i\}} \lambda \frac{1}{2} w^T w + \sum_i \xi_i$$

subject to $\forall_i: \xi_i \geq 0$ and $\xi_i \geq 1 - y_i(w^T x_i + b)$

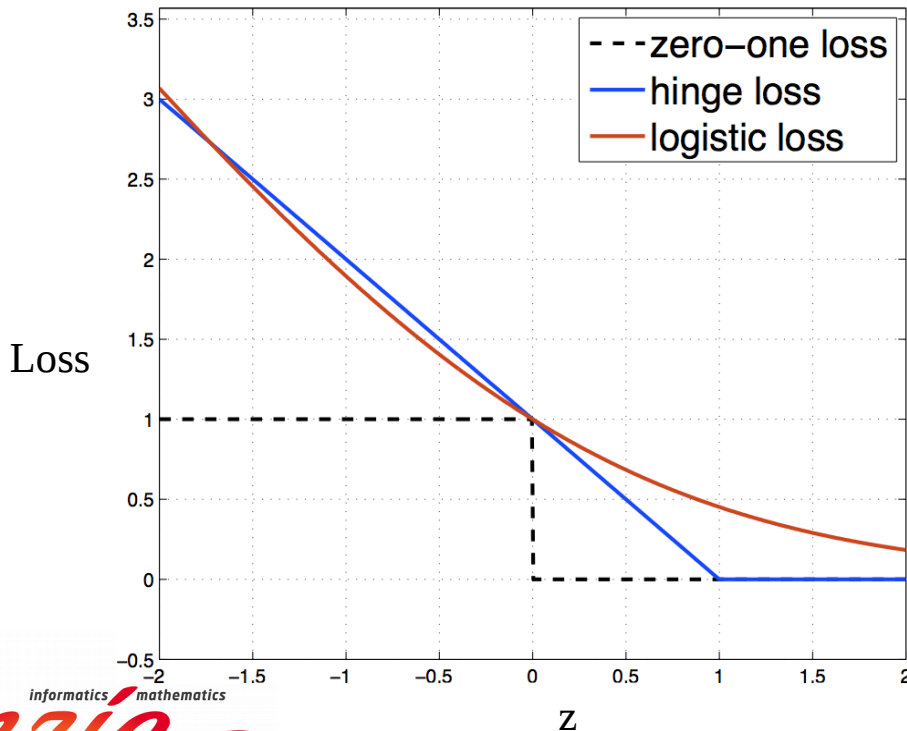
- Solution of the quadratic program has the property that w is a linear combination of the data points.

SVM solution properties

- Optimal w is a linear combination of data points $w = \sum_{n=1}^N \alpha_n y_n x_n$
 - ▶ Exercise: why is this the case ?
- Alpha weights are zero for all points on the correct side of the margin
- Points on the margin, or on the wrong side, have non-zero weight
 - ▶ Called support vectors
- Classification function thus has form $f(x) = w^T x + b = \sum_{n=1}^N \alpha_n y_n x_n^T x + b$
 - ▶ relies only on inner products between the test point x and data points with non-zero alpha's
- Solving the optimization problem also requires access to the data only in terms of inner products between pairs of training points

Relation SVM and logistic regression

- A classification error occurs when sign of the function does not match the sign of the class label: the zero-one loss $z = y_i f(x_i) \leq 0$
- Consider error minimized when training classifier:
 - Hinge loss: $\xi_i = \max(0, 1 - y_i f(x_i)) = \max(0, 1 - z)$
 - Logistic loss: $-\log p(y_i | x_i) = -\log \sigma(y_i f(x_i)) = \log(1 + \exp(-z))$



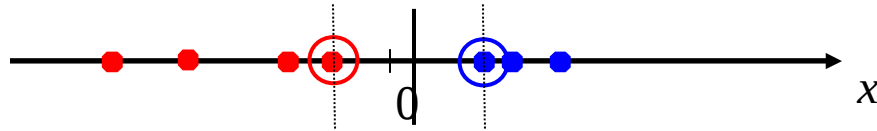
- L2 penalty for SVM motivated by margin between the classes
- For Logistic discriminant we find it via MAP estimation with a Gaussian prior
- Both lead to efficient optimization
 - ▶ Hinge-loss is piece-wise linear: quadratic programming
 - ▶ Logistic loss is smooth : smooth convex optimization methods

Summary of discriminative linear classification

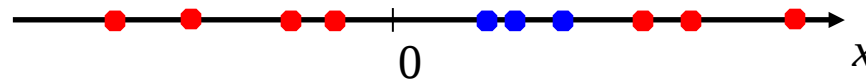
- Two most widely used linear classifiers in practice:
 - ▶ Logistic discriminant (supports more than 2 classes directly)
 - ▶ Support vector machines (multi-class extensions possible)
- For both, in the case of binary classification
 - ▶ Criterion that is minimized is a convex bound on zero-one loss
 - ▶ weight vector \mathbf{w} is a linear combination of the data points $\mathbf{w} = \sum_{n=1}^N \alpha_n \mathbf{x}_n$
- This means that we only need the inner-products between data points to calculate the linear functions
$$\begin{aligned} f(\mathbf{x}) &= \mathbf{w}^T \mathbf{x} + b \\ &= \sum_{n=1}^N \alpha_n \mathbf{x}_n^T \mathbf{x} + b \\ &= \sum_{n=1}^N \alpha_n k(\mathbf{x}_n, \mathbf{x}) + b \end{aligned}$$
 - ▶ The “kernel” function $k(\cdot, \cdot)$ computes the inner products

Nonlinear Classification

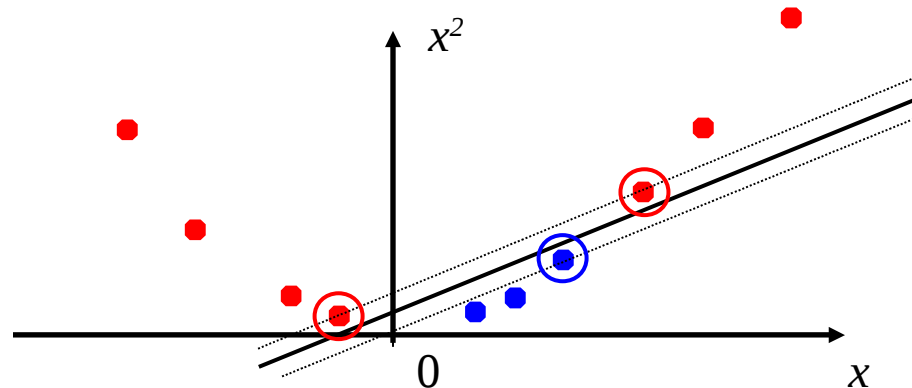
- 1 dimensional data that is linearly separable



- But what if the data is not linearly separable?



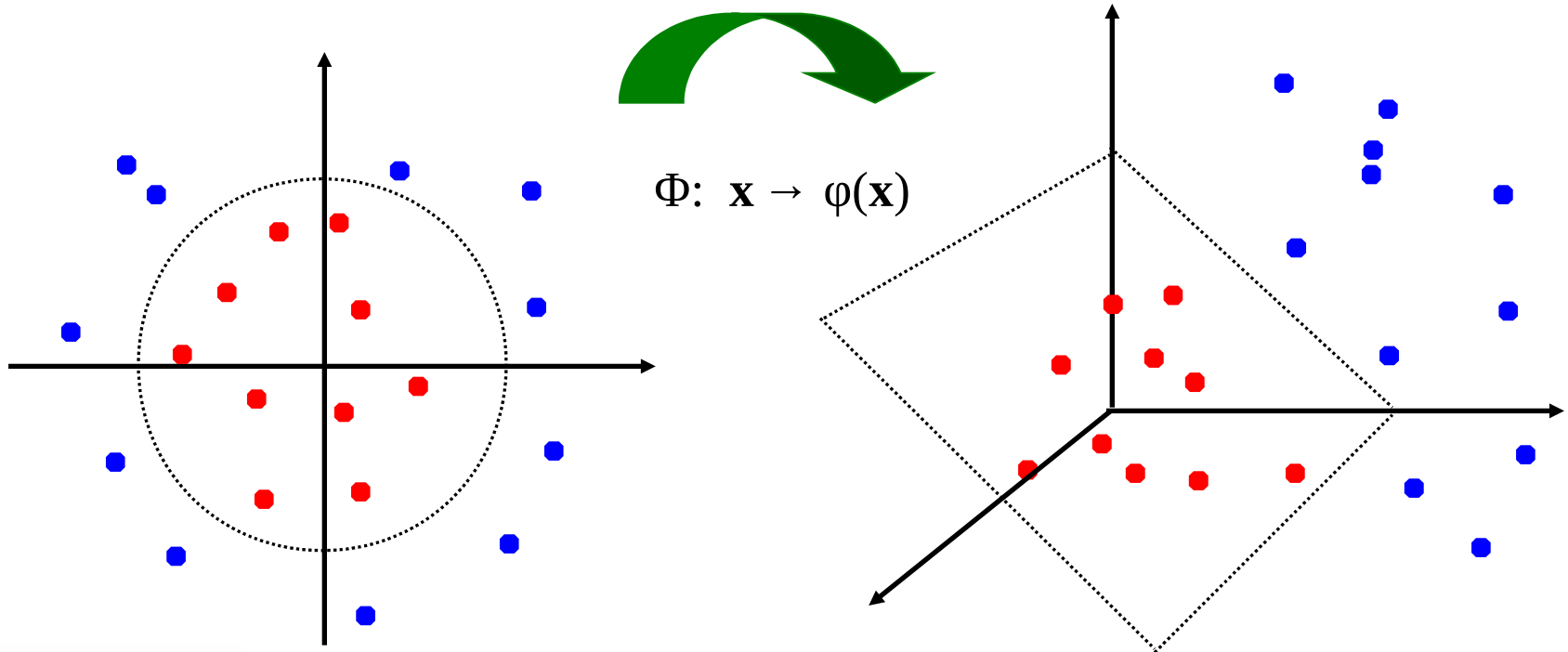
- We can map it to a higher-dimensional space:



Slide credit: Andrew Moore

Kernels for non-linear classification

- General idea: map the original input space to some higher-dimensional feature space where the training set is separable
- Exercise: find features that could separate the 2d data linearly



Nonlinear classification with kernels

- *The kernel trick*: instead of explicitly computing the feature transformation $\varphi(\mathbf{x})$, define a kernel function K such that

$$K(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}_j)$$

- Conversely, if a kernel satisfies Mercer's condition then it computes an inner product in some feature space, possibly with large or infinite number of dimensions
 - ▶ *Mercer's Condition: The square $N \times N$ matrix with kernel evaluations for any arbitrary N data points should always be a positive definite matrix.*
- This gives a **nonlinear decision boundary** in the original space:

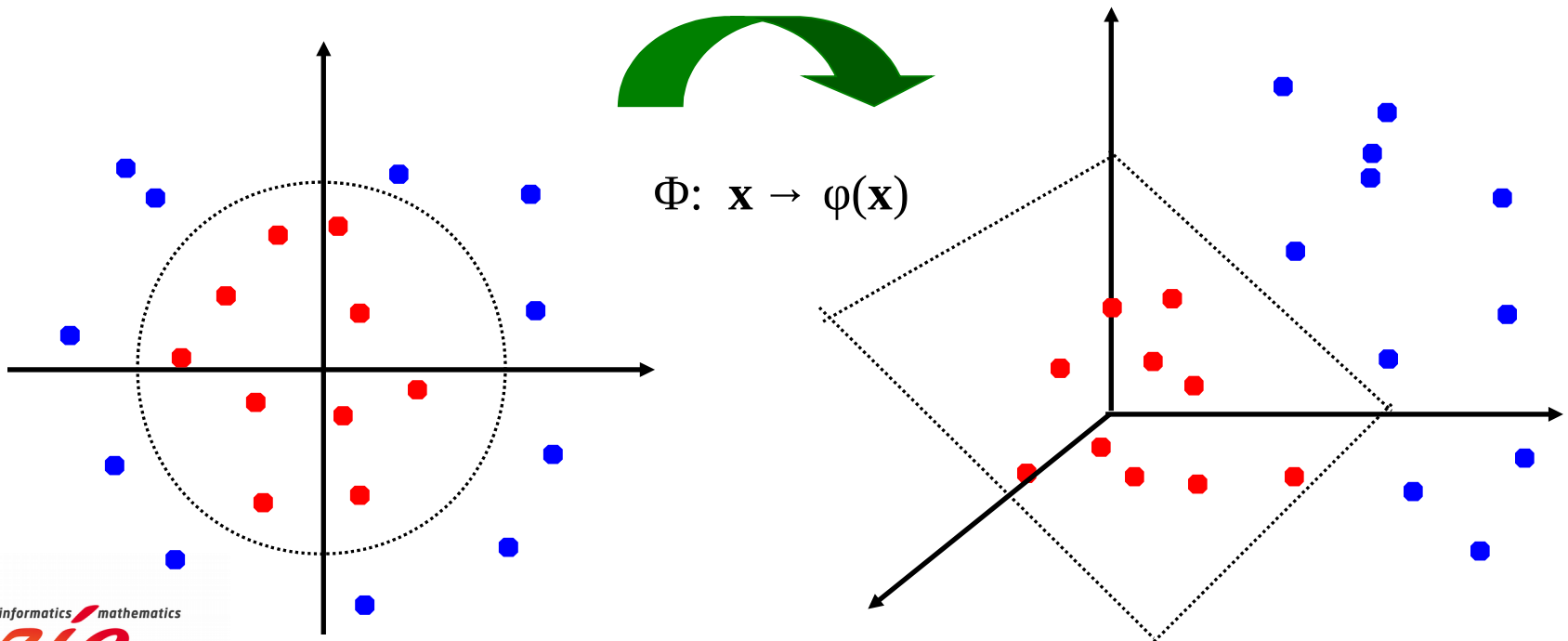
$$\begin{aligned} f(x) &= b + w^T \varphi(x) \\ &= b + \sum_i \alpha_i \varphi(x_i)^T \varphi(x) \\ &= b + \sum_i \alpha_i k(x_i, x) \end{aligned}$$

Kernels for non-linear classification

- What is the kernel function that corresponds to this feature mapping ?

$$\varphi(\mathbf{x}) = \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix}$$

$$\begin{aligned} k(\mathbf{x}, \mathbf{y}) &= \varphi(\mathbf{x})^T \varphi(\mathbf{y}) = ? \\ &= x_1^2 y_1^2 + x_2^2 y_2^2 + 2x_1 x_2 y_1 y_2 \\ &= (x_1 y_1 + x_2 y_2)^2 \\ &= (\mathbf{x}^T \mathbf{y})^2 \end{aligned}$$



Kernels for non-linear classification

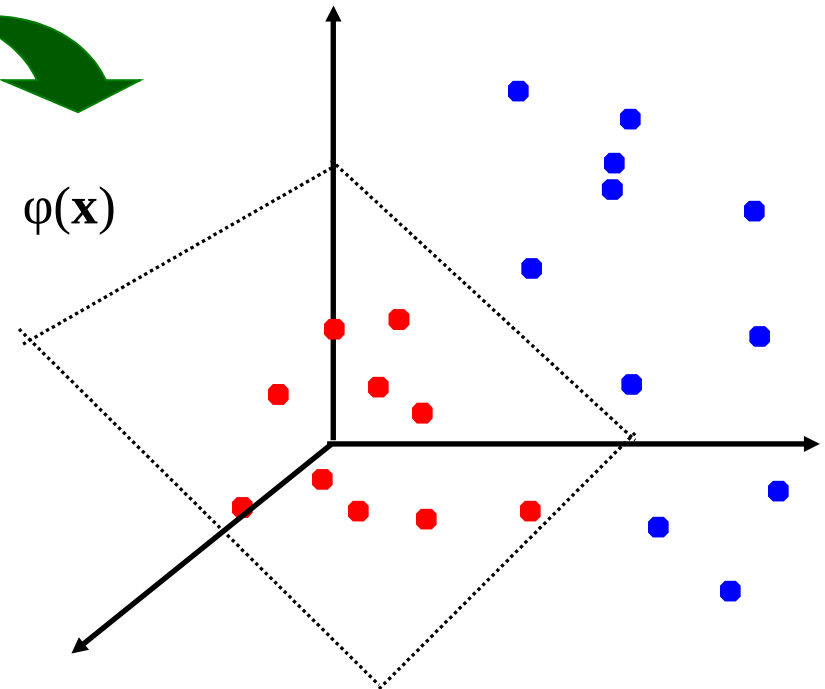
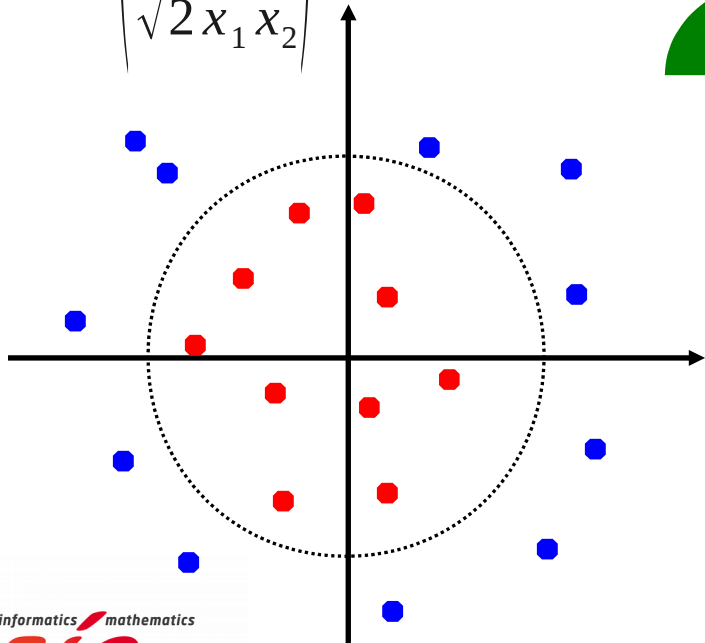
- Suppose we also want to keep the original features to be able to still implement linear functions

$$\begin{aligned}k(x, y) &= \varphi(x)^T \varphi(y) = ? \\ &= 1 + 2x^T y + (x^T y)^2 \\ &= (x^T y + 1)^2\end{aligned}$$

$$\varphi(x) = \begin{pmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix}$$



$\Phi: \mathbf{x} \rightarrow \varphi(\mathbf{x})$



Kernels for non-linear classification

- What happens if we use the same kernel for higher dimensional data
 - ▶ Which feature vector $\varphi(x)$ corresponds to it ?

$$k(x, y) = (x^T y + 1)^2 = 1 + 2x^T y + (x^T y)^2$$

- ▶ First term, encodes an additional 1 in each feature vector
- ▶ Second term, encodes scaling of the original features by sqrt(2)
- ▶ Let's consider the third term $(x^T y)^2 = (x_1 y_1 + \dots + x_D y_D)^2$

$$\begin{aligned} &= \sum_{d=1}^D (x_d y_d)^2 + 2 \sum_{d=1}^D \sum_{i=d+1}^D (x_d y_d)(x_i y_i) \\ &= \sum_{d=1}^D x_d^2 y_d^2 + 2 \sum_{d=1}^D \sum_{i=d+1}^D (x_d x_i)(y_d y_i) \end{aligned}$$

- ▶ In total we have $1 + 2D + D(D-1)/2$ features !
- ▶ But the kernel is computed as efficiently as dot-product in original space

$$\varphi(x) = \left(1, \underbrace{\sqrt{2} x_1, \sqrt{2} x_2, \dots, \sqrt{2} x_D}_{\text{Original features}}, \underbrace{x_1^2, x_2^2, \dots, x_D^2}_{\text{Squares}}, \underbrace{\sqrt{2} x_1 x_2, \dots, \sqrt{2} x_1 x_D, \dots, \sqrt{2} x_{D-1} x_D}_{\text{Products of two distinct elements}} \right)^T$$

Original features

Squares

Products of two distinct elements

Common kernels for bag-of-word histograms

- Hellinger kernel:

$$k(h_1, h_2) = \sum_d \sqrt{h_1(i)} \times \sqrt{h_2(i)}$$

- Histogram intersection kernel:

$$k(h_1, h_2) = \sum_d \min(h_1(d), h_2(d))$$

- ▶ Exercise: find the feature transformation, when $h(d)$ is a bounded integer

- Generalized Gaussian kernel:

$$k(h_1, h_2) = \exp\left(-\frac{1}{A} d(h_1, h_2)\right)$$

- ▶ d can be Euclidean distance, χ^2 distance, Earth Mover's Distance, etc.

See also:

J. Zhang, M. Marszalek, S. Lazebnik, and C. Schmid,
Local features and kernels for classification of texture and object categories: a comprehensive study. Int. Journal of Computer Vision, 2007

Logistic discriminant with kernels

- Let us assume a given kernel, and weight vectors $\mathbf{w}_c = \sum_{i=1}^n \alpha_{ic} \varphi(\mathbf{x}_i)$
 - ▶ Express the score functions using the kernel

$$f_c(\mathbf{x}_j) = b_c + \sum_{i=1}^n \alpha_{ic} \langle \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j) \rangle = b_c + \sum_{i=1}^n \alpha_{ic} k(\mathbf{x}_i, \mathbf{x}_j) = b_c + \boldsymbol{\alpha}_c^T \mathbf{k}_j$$

- Where $\mathbf{k}_j = (k(\mathbf{x}_j, \mathbf{x}_1), \dots, k(\mathbf{x}_j, \mathbf{x}_n))^T$
 $\boldsymbol{\alpha}_c = (\alpha_{1c}, \dots, \alpha_{nc})^T$

- ▶ Express the L2 penalty on the weight vectors using the kernel

$$\langle \mathbf{w}_c, \mathbf{w}_c \rangle = \sum_{i=1}^n \sum_{j=1}^n \alpha_{ic} \alpha_{jc} k(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\alpha}_c^T \mathbf{K} \boldsymbol{\alpha}_c$$

- ▶ Where $[\mathbf{K}]_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$

- MAP estimation of the alpha's and b's amounts to maximize

$$\sum_{i=1}^n \ln p(y_i | \mathbf{x}_i) - \lambda \frac{1}{2} \sum_{c=1}^C \boldsymbol{\alpha}_c^T \mathbf{K} \boldsymbol{\alpha}_c$$

Logistic discriminant with kernels

- Recall that $p(y_i|\mathbf{x}_i) = \frac{\exp(f_{y_i}(\mathbf{x}_i))}{\sum_c \exp f_c(\mathbf{x}_i)}$ and $f_c(\mathbf{x}_i) = b_c + \boldsymbol{\alpha}_c^T \mathbf{k}_i$

- Therefore we want to maximize

$$E(\{\boldsymbol{\alpha}_c\}, \{b_c\}) = \sum_{i=1}^n \left(f_{y_i}(\mathbf{x}_i) - \ln \sum_c \exp f_{y_i}(\mathbf{x}_i) \right) - \lambda \frac{1}{2} \sum_c \boldsymbol{\alpha}_c^T \mathbf{K} \boldsymbol{\alpha}_c$$

- Consider the partial derivative of this function with respect to the b's, and the gradient with respect to the alpha vectors

$$\frac{\partial E}{\partial b_c} = \sum_{i=1}^n ([y_i=c] - p(c|\mathbf{x}_i))$$
$$\nabla_{\boldsymbol{\alpha}_c} E = \sum_{i=1}^n ([y_i=c] - p(c|\mathbf{x}_i)) \mathbf{k}_i - \lambda \mathbf{K} \boldsymbol{\alpha}_c$$

- ▶ Essentially the same gradients as in the linear case, feature vector is replaced with a column of the kernel matrix

Support vector machines with kernels

- Minimize quadratic program

$$\min_{\mathbf{w}, b, \{\xi_i\}} \lambda \frac{1}{2} \mathbf{w}^T \mathbf{w} + \sum_i \xi_i$$

subject to $\forall_i: \xi_i \geq 0$ and $\xi_i \geq 1 - y_i f(\mathbf{x}_i)$

- Let us again define the classification function in terms of kernel evaluations

$$f(\mathbf{x}_i) = b + \boldsymbol{\alpha}^T \mathbf{k}_i$$

- Then we obtain a quadratic program in b, alpha, and the slack variables

$$\min_{\boldsymbol{\alpha}, b, \{\xi_i\}} \lambda \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha} + \sum_i \xi_i$$

subject to $\forall_i: \xi_i \geq 0$ and $\xi_i \geq 1 - y_i (b + \boldsymbol{\alpha}^T \mathbf{k}_i)$

Summary linear classification & kernels

- Linear classifiers learned by minimizing convex cost functions
 - Logistic discriminant: smooth objective, minimized using gradient-based methods
 - Support vector machines: piecewise linear objective, quadratic programming
 - Both require only computing inner product between data points
- Non-linear classification can be done with linear classifiers over new features that are non-linear functions of the original features
 - ▶ Kernel functions efficiently compute inner products in (very) high-dimensional spaces, can even be infinite dimensional in some cases.
- Using kernel functions non-linear classification has drawbacks
 - Requires storing the support vectors, may cost lots of memory in practice
 - Computing kernel between new data point and support vectors may be computationally expensive (at least more expensive than linear classifier)
- The “kernel trick” also applies for other linear data analysis techniques
 - Principle component analysis, k-means clustering, regression, ...

Reading material

- A good book that covers all machine learning aspects of the course is
 - ▶ Pattern recognition & machine learning
Chris Bishop, Springer, 2006
- For clustering with k-means & mixture of Gaussians read
 - ▶ Section 2.3.9
 - ▶ Chapter 9, except 9.3.4
 - ▶ Optionally, Section 1.6 on information theory
- For classification read
 - ▶ Section 2.5, except 2.5.1
 - ▶ Section 4.1.1 & 4.1.2
 - ▶ Section 4.2.1 & 4.2.2
 - ▶ Section 4.3.2 & 4.3.4
 - ▶ Section 6.2
 - ▶ Section 7.1 start + 7.1.1 & 7.1.2