Intro to Classification

Evgeny Burnaev

Skoltech, Moscow, Russia

Skoltech

Skolkovo Institute of Science and Technology

Outline

- **Training data**: sample drawn i.i.d. w.r.t. D on $X \subseteq \mathbb{R}^d$

$$S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\} \in \{X \times \{-1, +1\}\}^m$$

- **Problem**: find hypothesis $f : X \to \{-1, +1\}$ in F (classifier) with small generalization error $R(f)$

Example: Churn prediction

- **Object**: customer at some moment of time
- **Label**: $-1/+1$ (not churn/churn)
- **Features**: sex, address, subscription plan, already purchased services, frequency and duration of calls, etc.
- **Challenges**:
  - estimate churn probability
  - huge samples
  - laborious feature engineering

Linear separable case

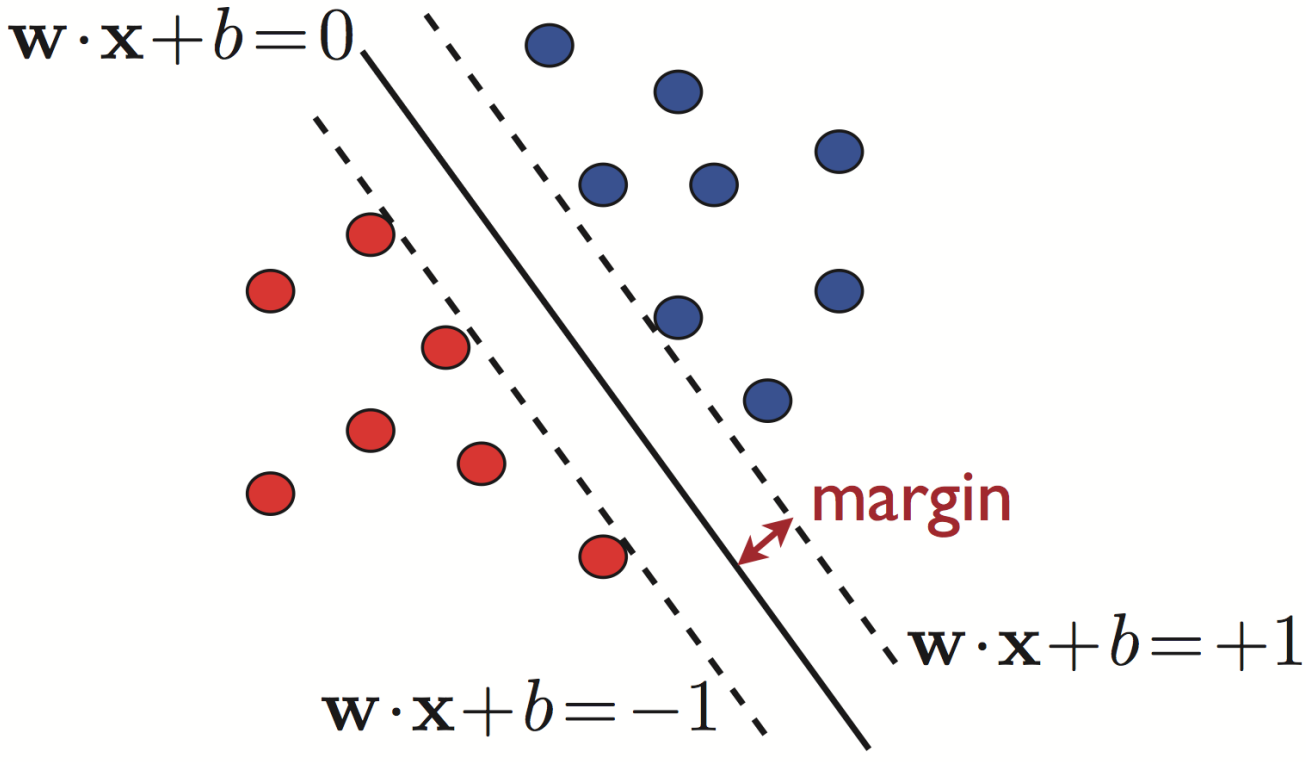


Figure – Linear classifier [Mohri]

- Example of linear classification (hyperplane)
- Set of classifiers: $F = \{\mathbf{x} \to \text{sign}(\mathbf{w}\cdot\mathbf{x}+b), \mathbf{w}\in\mathbb{R}^d, b\in\mathbb{R}^1\}$
- Geometric margin: $\rho = \min_{i\in[1,m]} \frac{|\mathbf{w}\cdot\mathbf{x}_i+b|}{\|\mathbf{w}\|}$

Binary Classification Errors

- Binary classification, $y_i \in \{-1, +1\}$
- Classifier $f(\mathbf{x}_i) \in \{-1, +1\}$

Error type	Classifier	True label
TP, True Positive	$f(\mathbf{x}_i) = +1$	$y_i = +1$
TN, True Negative	$f(\mathbf{x}_i) = -1$	$y_i = -1$
FP, False Positive	$f(\mathbf{x}_i) = +1$	$y_i = -1$
FN, False Negative	$f(\mathbf{x}_i) = -1$	$y_i = +1$

- Proportion of classification errors

$$\text{Accuracy} = \frac{1}{m}\sum_{i=1}^{m} 1_{\{f(\mathbf{x}_i) \neq y_i\}} = \frac{FP + FN}{FP + FN + TP + TN}$$

- **Shortcomings**: Accuracy does not take into account neither disbalance of classes nor different error costs for different classes

Loss function with error penalty

- Binary classification: $y_i \in \{-1, +1\}$
- Classifier: $f(\mathbf{x}; \mathbf{w}, w_0) = \text{sign}(h(\mathbf{x}; \mathbf{w}) - w_0)$
- The bigger w_0 is the bigger the number of $\mathbf{x}_i$ for which $f(\mathbf{x}_i) = -1$
- Let us denote by λ_y a penalty when object from class y is misclassified
- Loss function

$$L(f(\mathbf{x}_i), y_i) = \lambda_{y_i} 1_{\{f(\mathbf{x}_i;\mathbf{w},w_0) \neq y_i\}} = \lambda_{y_i} 1_{\{(h(\mathbf{x}_i;\mathbf{w}) - w_0) y_i < 0\}}$$

- Let us define ROC (receiver operating characteristic) curve
- Each point of the curve corresponds to some

$$f(\mathbf{x}; \mathbf{w}, w_0) = \text{sign}(h(\mathbf{x}; \mathbf{w}) - w_0)$$

- **Abscissa**: we depict values of FPR (false positive rate) as a function of w_0

$$FPR(w_0) = \frac{\sum_{i=1}^{m} 1_{\{y_i=-1\}} 1_{\{f(\mathbf{x}_i;\mathbf{w},w_0)=+1\}}}{\sum_{i=1}^{m} 1_{\{y_i=-1\}}}$$

- **Ordinate**: we depict values of TPR (true positive rate):

$$TPR(w_0) = \frac{\sum_{i=1}^{m} 1_{\{y_i=+1\}} 1_{\{f(\mathbf{x}_i;\mathbf{w},w_0)=+1\}}}{\sum_{i=1}^{m} 1_{\{y_i=+1\}}}$$

- $(1 - FPR)$ is called specificity, TPR is called sensitivity

Example of ROC-curve

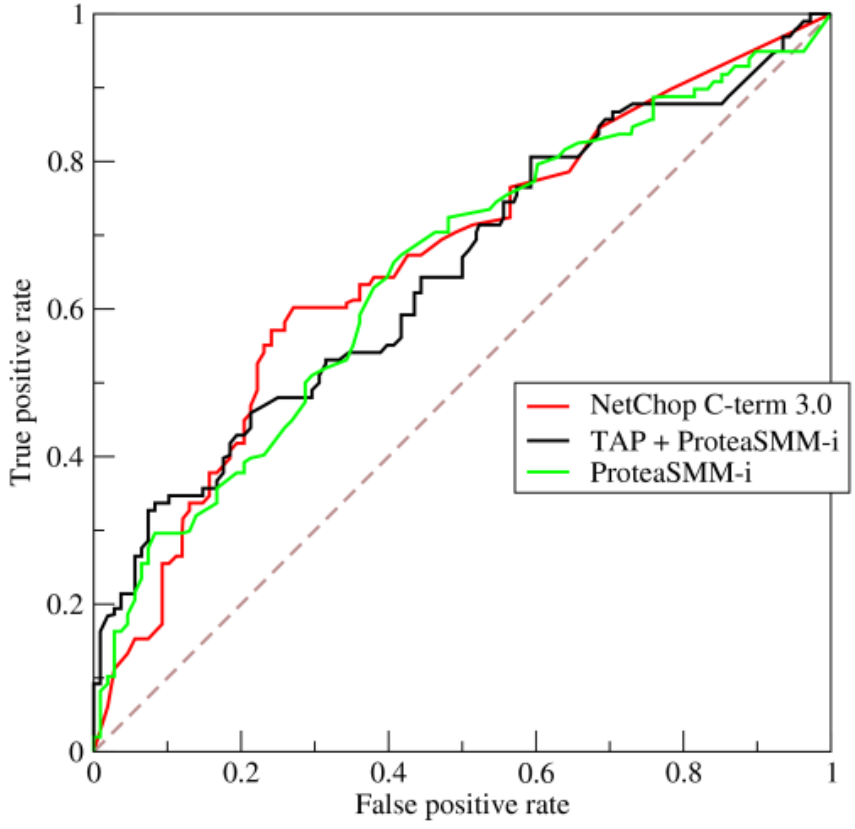


Figure – ROC-curve [Wikipedia]

- AUC is an Area Under ROC-curve, used to characterize classification accuracy (AUC ≥ 0.5)
- Dashed line — the worst accuracy (random guessing)

Precision and Recall in case of Binary Classification

- TP — true positives
- FP — false positives
- FN — false negatives
- Precision and Recall

$$\text{Precision} : P = \frac{TP}{TP + FP},$$

$$\text{Recall} : R = \frac{TP}{TP + FN}$$

In information retrieval

- P is a fraction of relevant objects among retrieved
- R is a fraction of retrieved objects among relevant

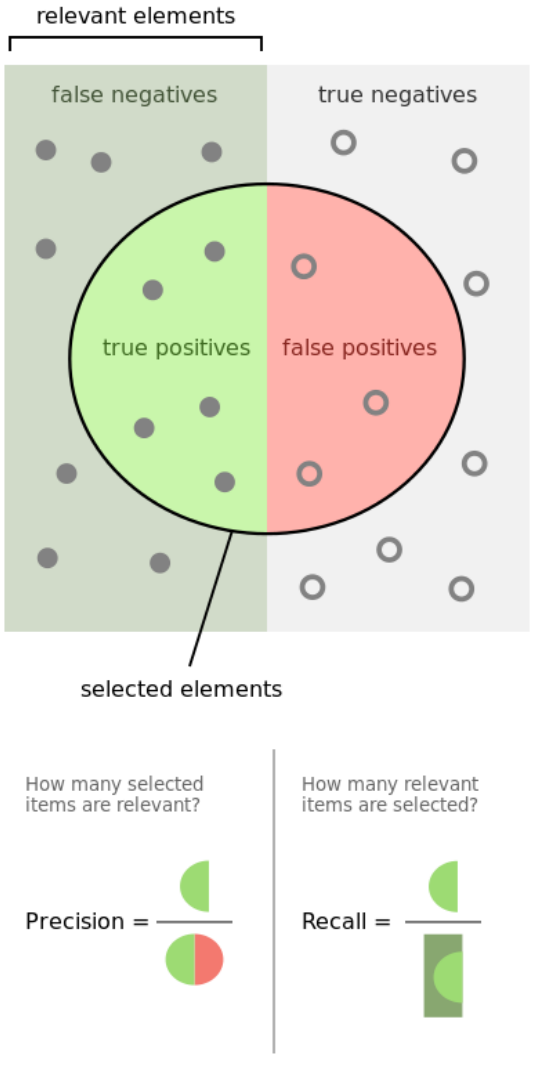


Figure – Graphical illustration [Wikipedia]

Precision and Recall in case of Multi-Class Classification

For each class $y \in Y$

- TP_y — true positives
- FP_y — false positives
- FN_y — false negatives
- Precision and Recall with micro-averaging

$$\text{Precision} : P = \frac{\sum_y TP_y}{\sum_y (TP_y + FP_y)},$$

$$\text{Recall} : R = \frac{\sum_y TP_y}{\sum_y (TP_y + FN_y)}$$

Micro-averaging is not sensitive to errors for rare classes

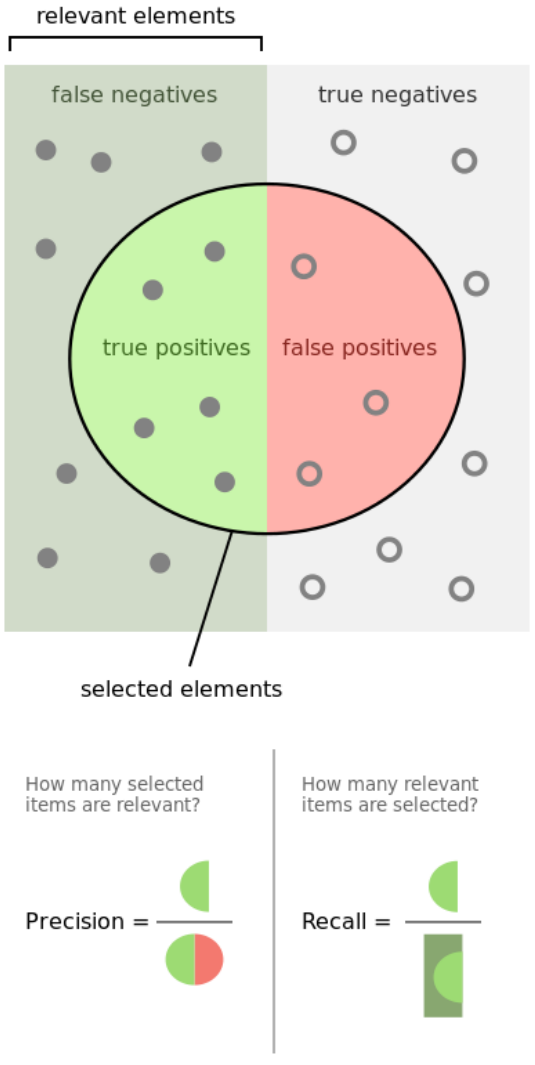


Figure – Graphical illustration [Wikipedia]

Precision and Recall in case of Multi-Class Classification

For each class $y \in Y$

- TP_y — true positives
- FP_y — false positives
- FN_y — false negatives
- Precision and Recall with macro-averaging

$$\text{Precision} : P = \frac{1}{|Y|} \sum_y \frac{TP_y}{TP_y + FP_y},$$

$$\text{Recall} : R = \frac{1}{|Y|} \sum_y \frac{TP_y}{TP_y + FN_y}$$

Macro-averaging is sensitive to errors for rare classes

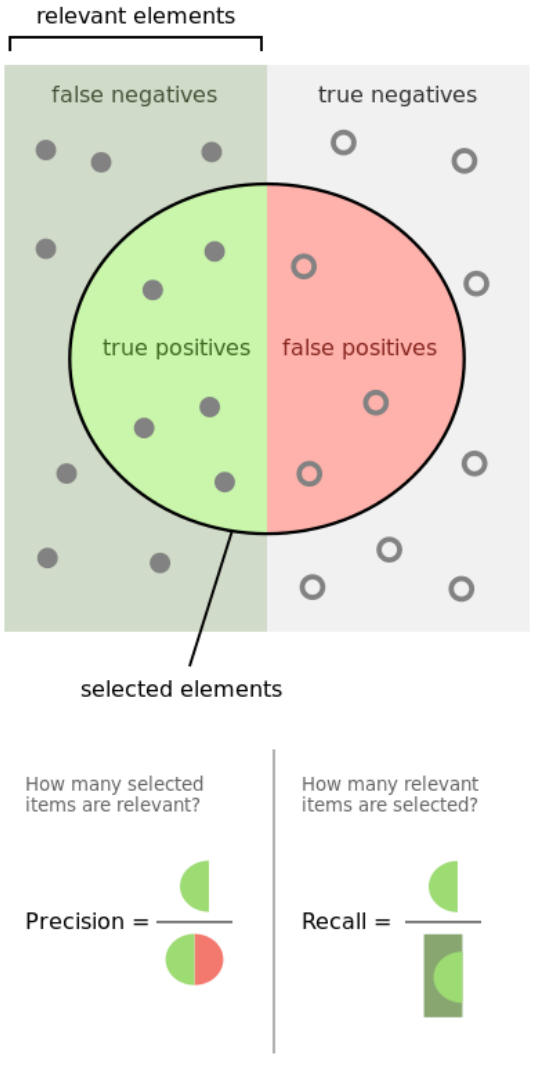


Figure – Graphical illustration [Wikipedia]

Example of Precision-Recall curve

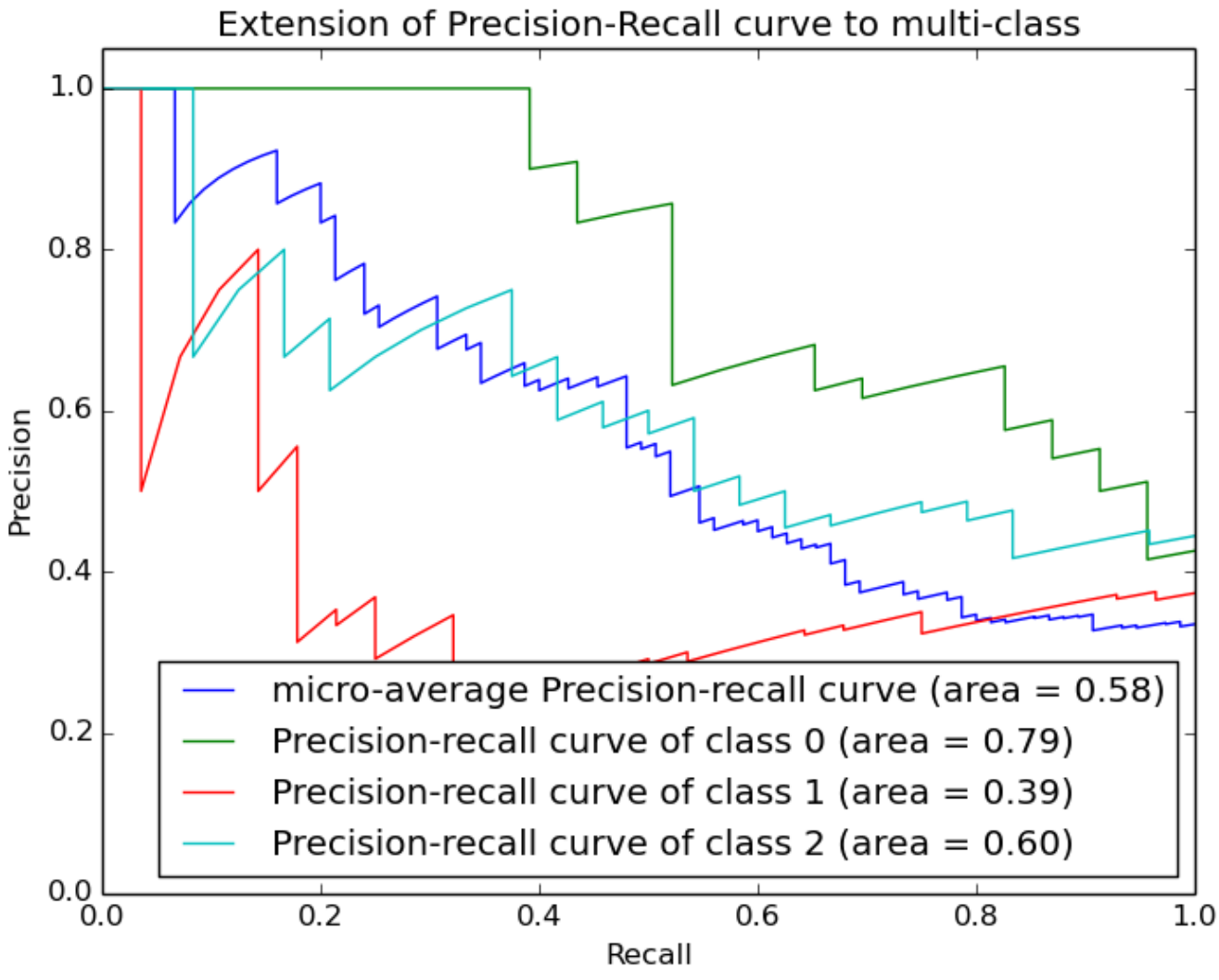


Figure – Precision-Recall curve [scikit-learn manual]

- Sensitivity and specificity are better suited for problems with unbalanced classes
- Precision and Recall are better suited for problems when a class of interest is rare
- Aggregated errors:
  - AUC is better suited when ratio of errors costs for different classes is not fixed
  - AUC-PR — area under precision-recall curve
  - F-measure $F_1 = \frac{2PR}{P+R}$
  - F_β-measure $F_\beta = \frac{(1+\beta^2)PR}{\beta^2 P+R}$ (the bigger β is the more R is important)

Problem setting

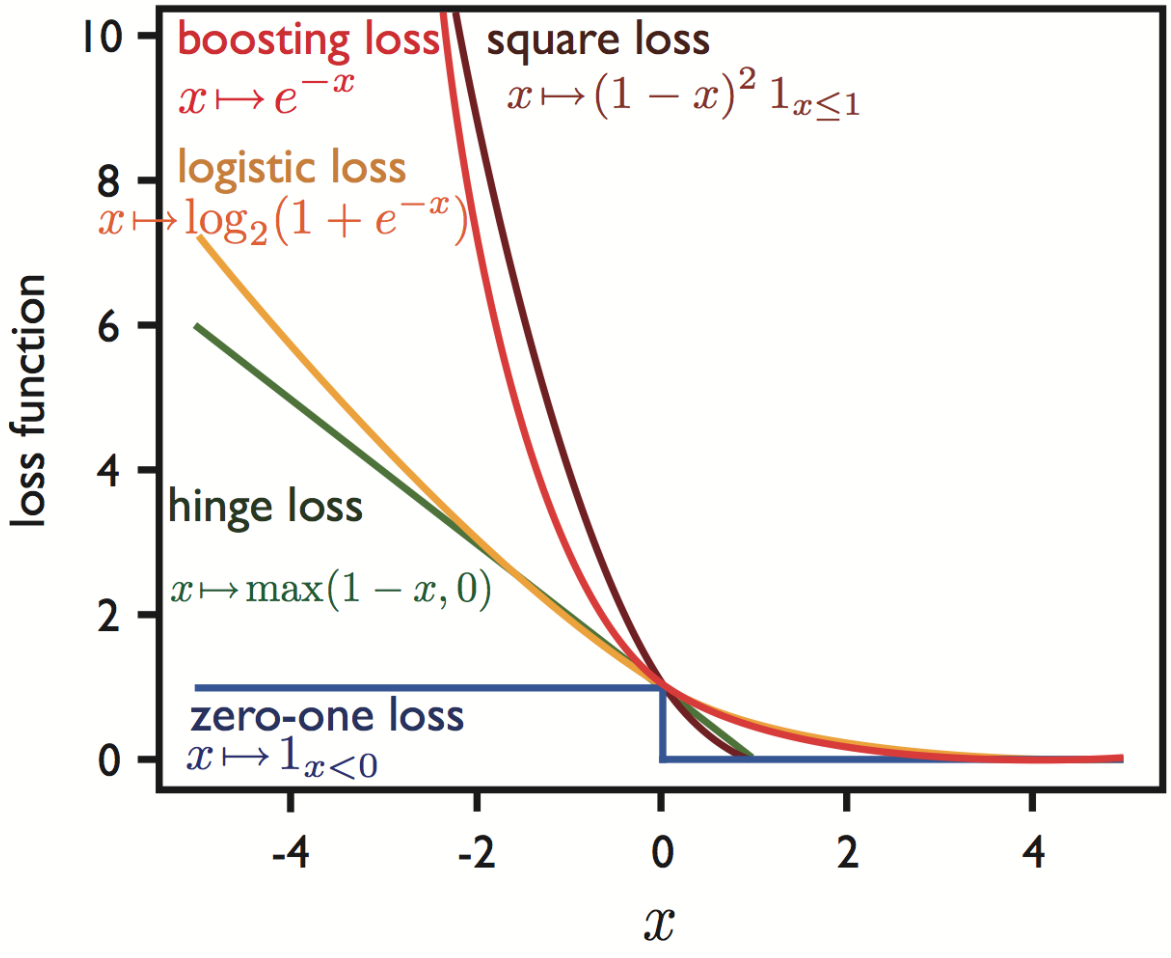
- Learning sample $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, $\mathbf{x}_i \in \mathbb{R}^d$, $y_i \in \{-1, +1\}$
- Classifier
$$f(\mathbf{x}; \mathbf{w}) = \text{sign}(g(\mathbf{x}; \mathbf{w}))$$
- Example of Linear classifier: $g(\mathbf{x}; \mathbf{w}) = \mathbf{w}^\top \mathbf{x}$ (we assume x_1 to account for a constant $-w_0$)
- Loss function
$$\widehat{R}(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^m 1_{\{g(\mathbf{x}_i; \mathbf{w}) y_i < 0\}}$$
can not be efficiently optimized w.r.t. $\mathbf{w}$

Convex upper bound for an indicator function

$$1_{\{x<0\}} \leq L(x),$$

i.e.

$$1_{\{g(\mathbf{x}_i;\mathbf{w})y_i<0\}} \leq L(g(\mathbf{x}_i;\mathbf{w})y_i),$$



Surrogate Loss ERM

- Learning sample $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, $\mathbf{x}_i \in \mathbb{R}^d$, $y_i \in \{-1, +1\}$
- Classifier
$$f(\mathbf{x}; \mathbf{w}) = \text{sign}(g(\mathbf{x}; \mathbf{w}))$$
- Example of Linear classifier: $g(\mathbf{x}; \mathbf{w}) = \mathbf{w}^\top \mathbf{x}$ (we assume x_1 to account for a constant $-w_0$)
- Binary Loss function and its convex upper bound
$$1_{\{g(\mathbf{x}_i;\mathbf{w})y_i<0\}} \leq L(g(\mathbf{x}_i; \mathbf{w})y_i)$$
- Learning $\equiv$ ERM
$$\widehat{R}(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^m 1_{\{g(\mathbf{x}_i;\mathbf{w})y_i<0\}} \leq \frac{1}{m}\sum_{i=1}^m L(g(\mathbf{x}_i; \mathbf{w})y_i) \to \min_{\mathbf{w}}$$
- Estimate a test error using a separate test sample $S'_{m'} = \{(\mathbf{x}'_j, y'_j)\}_{j=1}^{m'}$
$$\frac{1}{m'}\sum_{j=1}^{m'} 1_{\{g(\mathbf{x}'_j;\mathbf{w})y'_j<0\}}$$

Overfitting

- Causes of overfitting
  - too small number of examples
  - too big number of features
  - linear dependence between features (multicollinearity)
- Symptoms of Overfitting
  - too big absolute values of weights $|w_j|$ and different signs of w_j
  - $\widehat{R}(\mathbf{w}^*) \ll \widehat{R}'(\mathbf{w}^*)$ (test error is $\gg$ than train error)
- Regularization is typically used to prevent overfitting

Regularization

- We impose additional penalty for high absolute values of weights

$$\overline{L}(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} L(g(\mathbf{x}_i;\mathbf{w})y_i) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

- In order to tune regularization coefficient λ we can use
  - cross-validation
  - Bayesian inference

Log-loss

- Probabilistic classifier model: a parametric model for

$$\mathbb{P}(y|\mathbf{x}) = p(y|\mathbf{x};\mathbf{w})$$

- MLE for $\mathbf{w}$

$$\prod_{i=1}^{m} p(y_i|\mathbf{x}_i;\mathbf{w}) \to \max_{\mathbf{w}}$$

- Log-likelihood

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{m} \log p(y_i|\mathbf{x}_i;\mathbf{w}) \to \max_{\mathbf{w}}$$

- If we set

$$p(y|\mathbf{x};\mathbf{w}) = e^{-L(g(\mathbf{x};\mathbf{w})y)},$$

then we get the surrogate loss ERM problem

$$\sum_{i=1}^{m} L(g(\mathbf{x}_i;\mathbf{w})y_i) \to \min_{\mathbf{w}},$$

i.e. the surrogate loss function $L(\cdot)$ and $g(\mathbf{x};\mathbf{w})$ define the probabilistic classifier model

Two-Class Logistic Regression

- Linear Classifier in case of $Y = \{-1, +1\}$

$$f(\mathbf{x}; \mathbf{w}) = \text{sign}(g(\mathbf{x}; \mathbf{w})) = \text{sign}(\mathbf{w}^\top \mathbf{x})$$

- Logarithmic loss function

$$L(t) = \log(1 + e^{-t})$$

- A model for conditional probability

$$p(y|\mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}}}$$

- Regularized logistic regression

$$\overline{L}(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^{m} \log(1 + \exp(-(\mathbf{w}^\top \mathbf{x}_i) y_i)) + \frac{\lambda}{2} \|\mathbf{w}\|^2 \to \min_{\mathbf{w}}$$

Multi-class Logistic Regression

- Linear Classifier in a multi-class case, i.e. $|Y| > 1$
- Probability of an object to belong to some class y is equal to

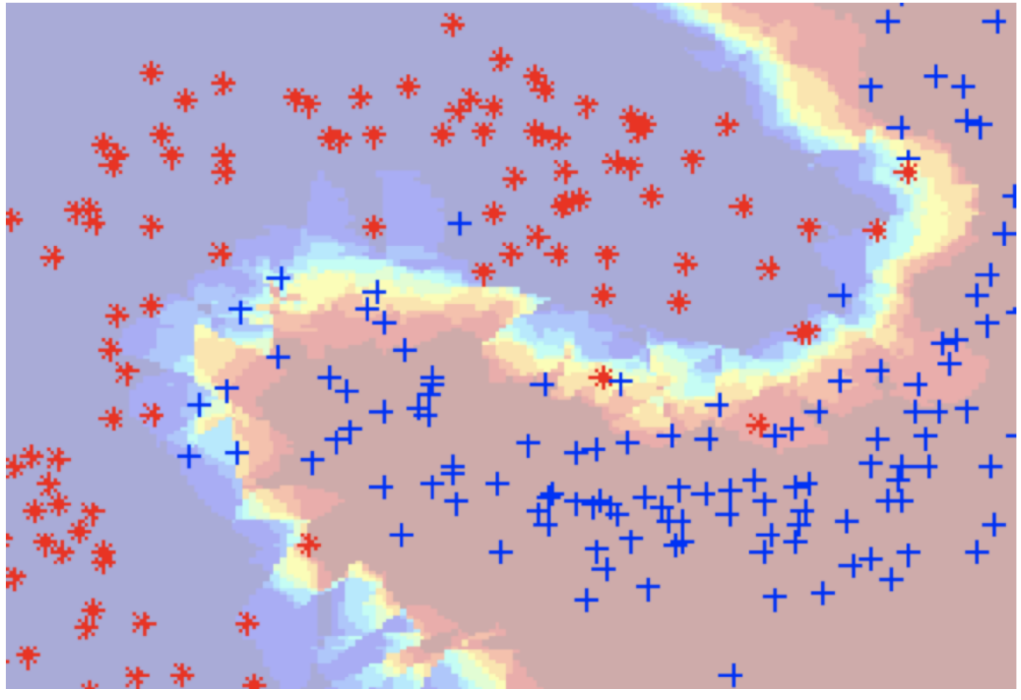
$$p(y|\mathbf{x};\mathbf{w}) = \frac{\exp(\mathbf{w}_y^\top \mathbf{x})}{\sum_{z\in Y}\exp(\mathbf{w}_z^\top \mathbf{x})} = \mathrm{SoftMax}_{y\in Y}(\mathbf{w}_y^\top \mathbf{x})$$

- Regularized logistic regression

$$\overline{L}(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m}\log p(y_i|\mathbf{x}_i,\mathbf{w}) - \frac{\lambda}{2}\sum_{y\in Y}\|\mathbf{w}_y\|^2 \to \min_{\mathbf{w}}$$

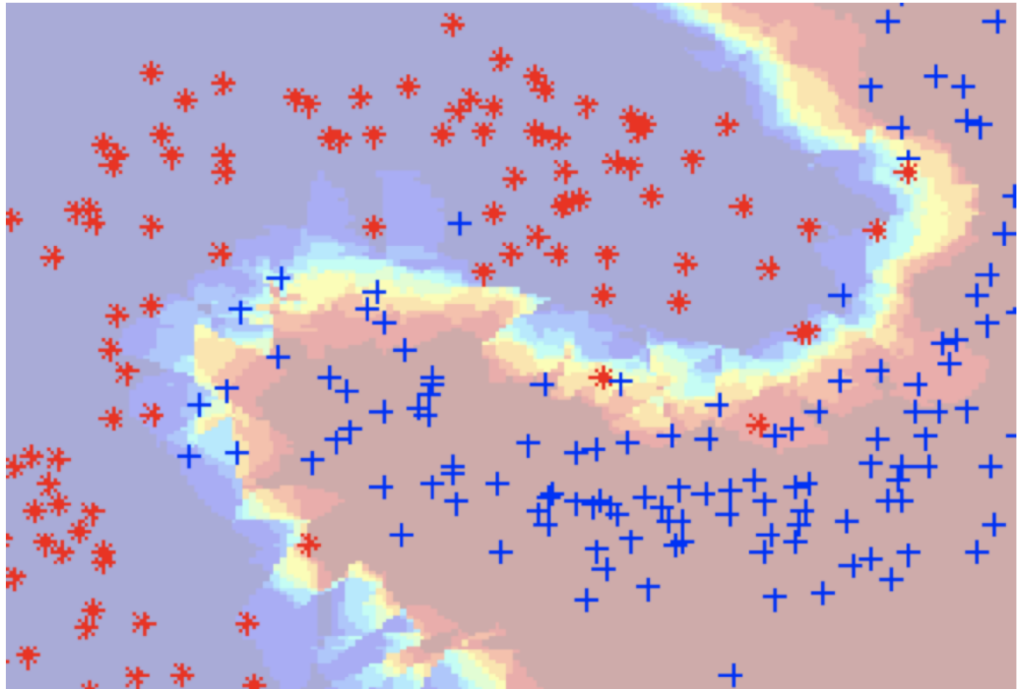
Compactness and continuity hypotheses

- Compactness hypothesis (classification): as a rule similar objects belongs to the same class
- Continuity hypotheses (regression): as a rule similar characteristics correspond to similar objects



Compactness and continuity hypotheses

- Classification: objects as a points in feature space belong to the same class if their feature vectors are close
- Regression: objects as a points in feature space have the same characteristics if their feature vectors are close
- What is “close objects”?



Distance functions

- We define some distance $\rho : X \times X \to [0, \infty)$
- E.g. Euclidian or weighted Euclidian distance:

$$\rho(\mathbf{x}, \mathbf{x}_i) = \left(\sum_{j=1}^{p} |x_j - x_{i,j}|^2 \right)^{1/2}, \; \rho(\mathbf{x}, \mathbf{x}_i) = \left(\sum_{j=1}^{p} \omega_j^2 |x_j - x_{i,j}|^2 \right)^{1/2}$$

- E.g. Manhattan distance

$$\rho(\mathbf{x}, \mathbf{x}') = \sum_{j=1}^{p} |x_j - x'_j|$$

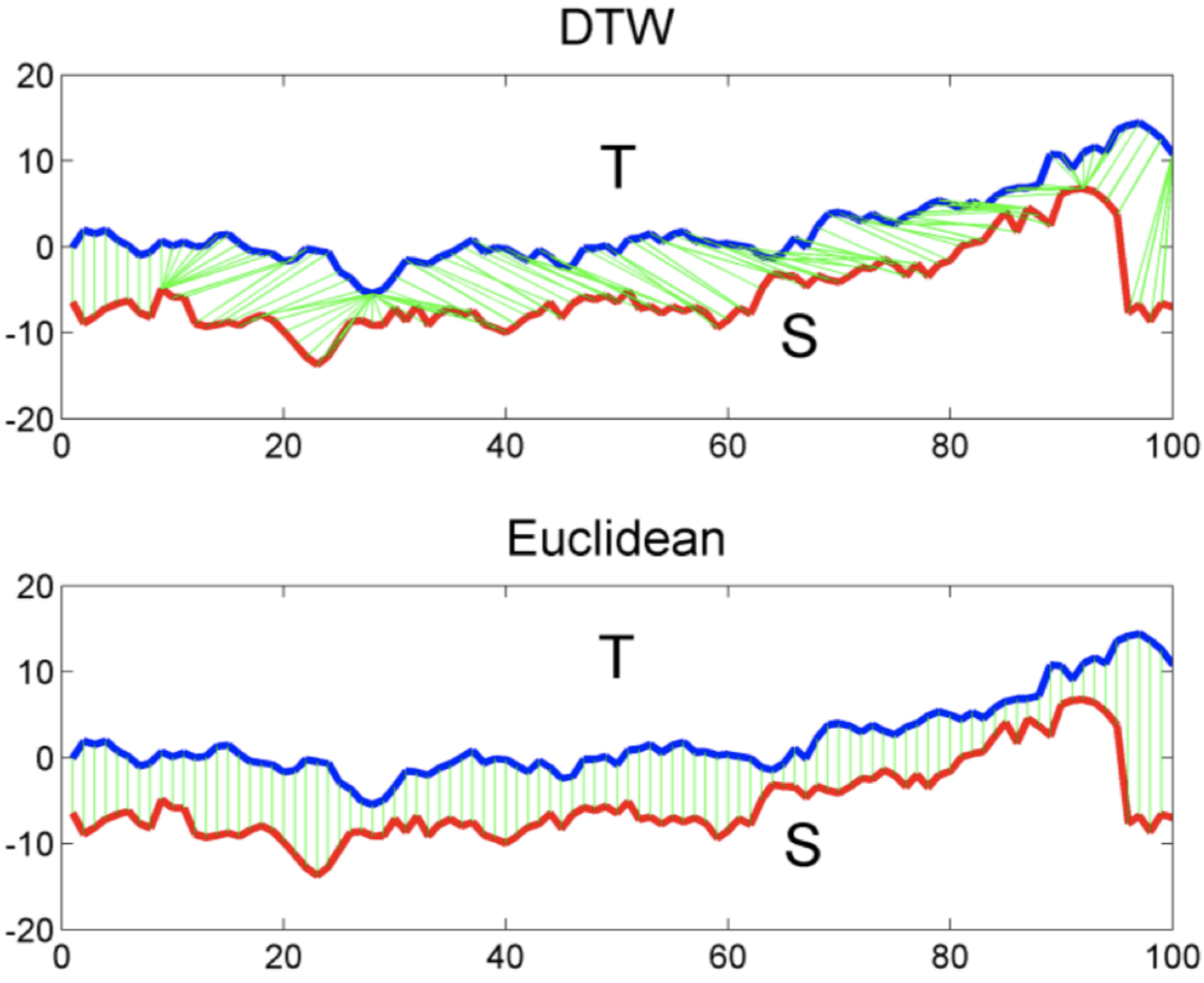
Distance functions

- Distance between strings
- Levenshtein distance is a string metric for measuring the difference between two sequences
- Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other

```
CTGGGCTAAAAGGTCCCTTAGCC..TTTAGAAAAA.GGGCCATTAGGAAATTGC
CTGGGACTAAA....CCTTAGCCTATTTACAAAAATGGGCCATTAGG...TTGC
```

Distance functions

- Distance between time-series
- E.g. total Euclidean distance
- E.g. Dynamic Time Warping and others



k-Nearest Neigbour Classifier

- Let us consider classification problem
- Fix the number k of k-nearest neigbours
- Sample: $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$
- Train: memorize sample S_m

k-Nearest Neigbour Classifier

- For any new $\mathbf{x} \in X$ we rank objects $\mathbf{x}_1, \ldots, \mathbf{x}_m$ such that

$$\rho(\mathbf{x}, \mathbf{x}_{(1)}) \leq \rho(\mathbf{x}, \mathbf{x}_{(2)}) \leq \ldots \leq \rho(\mathbf{x}, \mathbf{x}_{(m)}),$$

  i.e. e.g. $\mathbf{x}_{(1)}$ is the closest object to $\mathbf{x}$.
- So let
  - $\mathbf{x}_{(i)}$ be the ith closest object to $\mathbf{x}$ among $\mathbf{x}_1, \ldots, \mathbf{x}_m$
  - $y_{(i)}$ be the label of $\mathbf{x}_{(i)}$
- We define k-Nearest Neigbour Classifier prediction as the class, which is the most popular among the k nearest neigbours

$$\widehat{f}(\mathbf{x}) = \arg\max_{y \in Y} \#\left\{i,\, i = 1, 2, \ldots, k :\, y_{(i)} = y\right\}$$

- **Pros**: simple realization, interpretable, case-based reasoning
- **Cons**: not-stable enough (for small k) to outliers and noise, not very accurate, the whole sample should be stored, k should be selected

Iris flower data set [Fisher, 1936]

- Quantify the morphologic variation of Iris flowers of three related species (Iris setosa, Iris virginica and Iris versicolor) by four features: the length and the width of the sepals and petals, in centimetres
- $d = 4$ features, $K = 3$ classes (mono-label case), $m = 150$

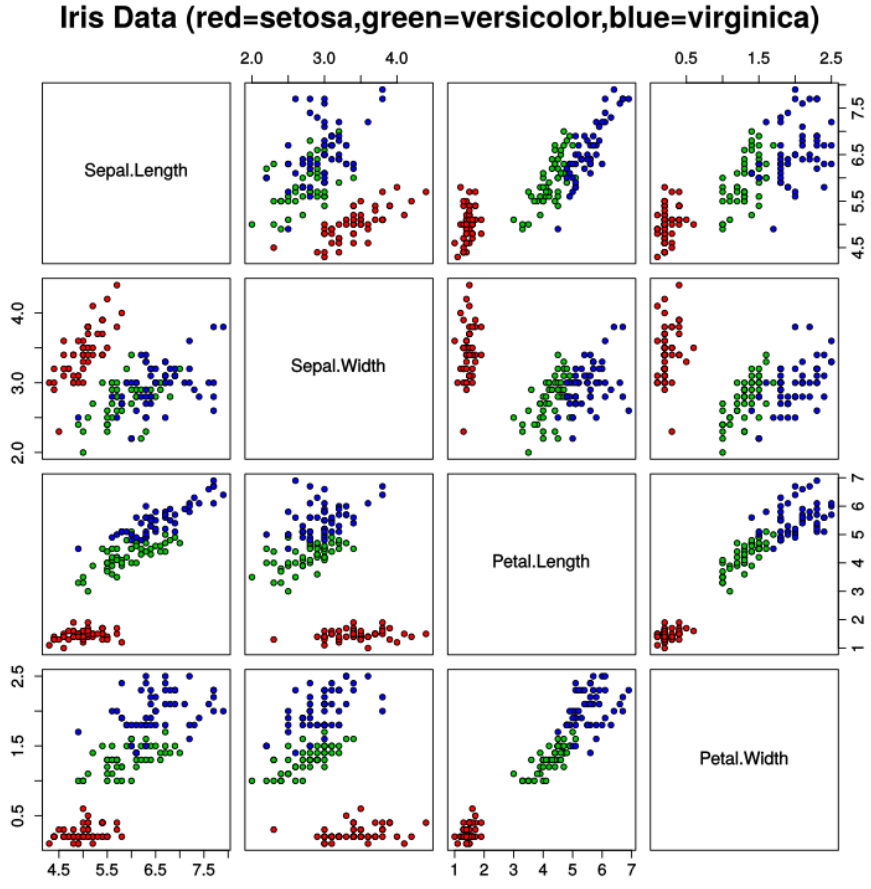
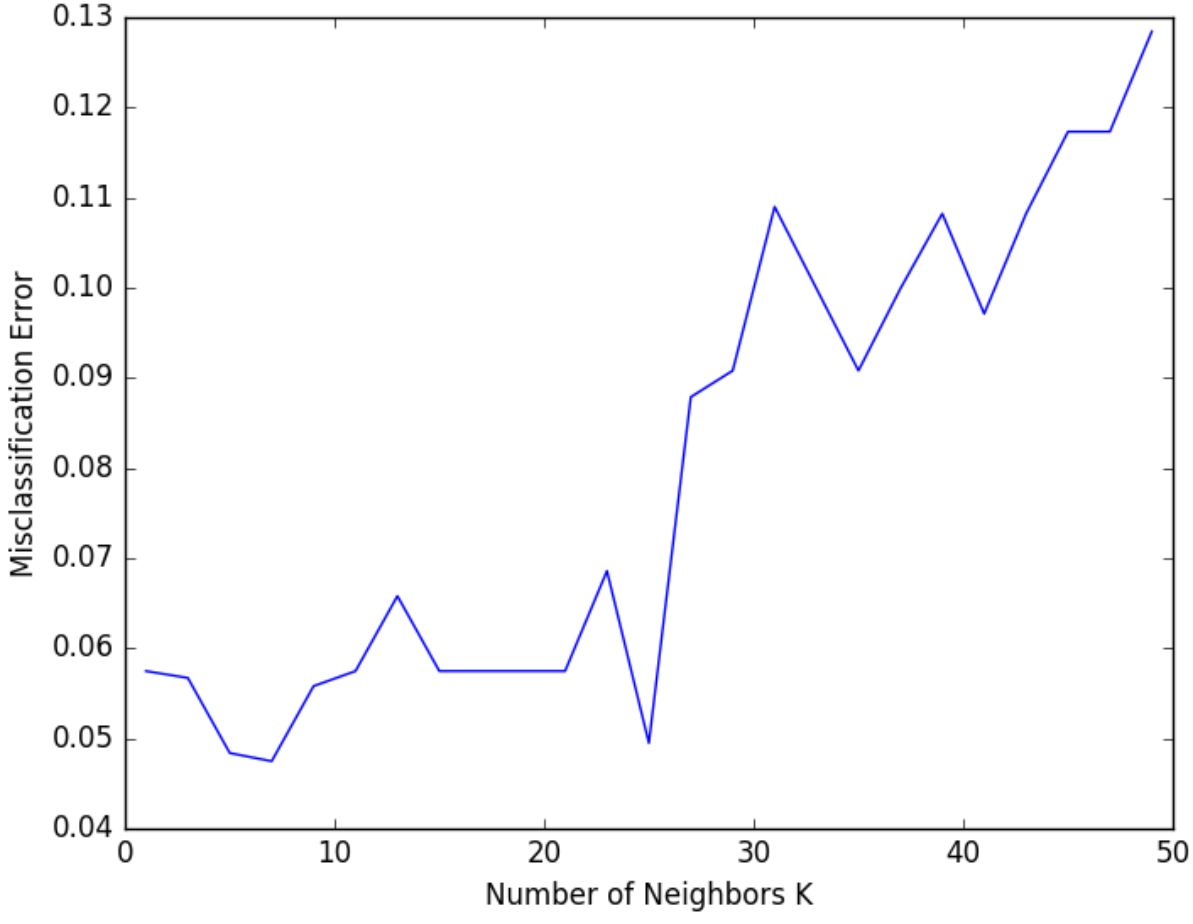


Figure – Scatterplot of the data set [Wikipedia]

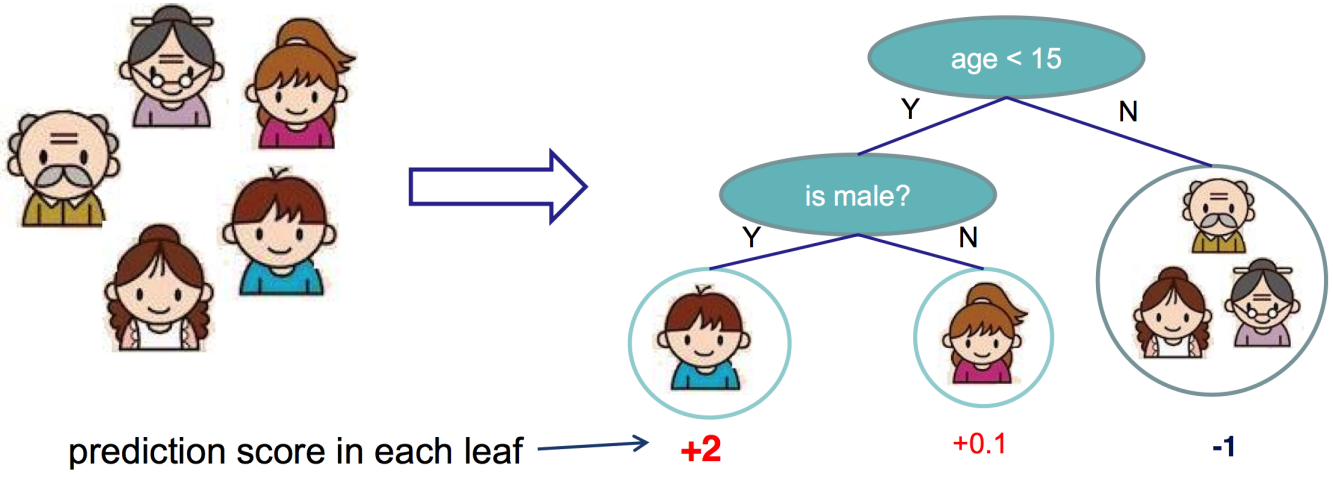
Tuning of k

- Iris Flower Dataset [Fisher]
- 10-fold cross-validation to estimate classification error
- In practice usually $k \in \{1, \ldots, 5\}$

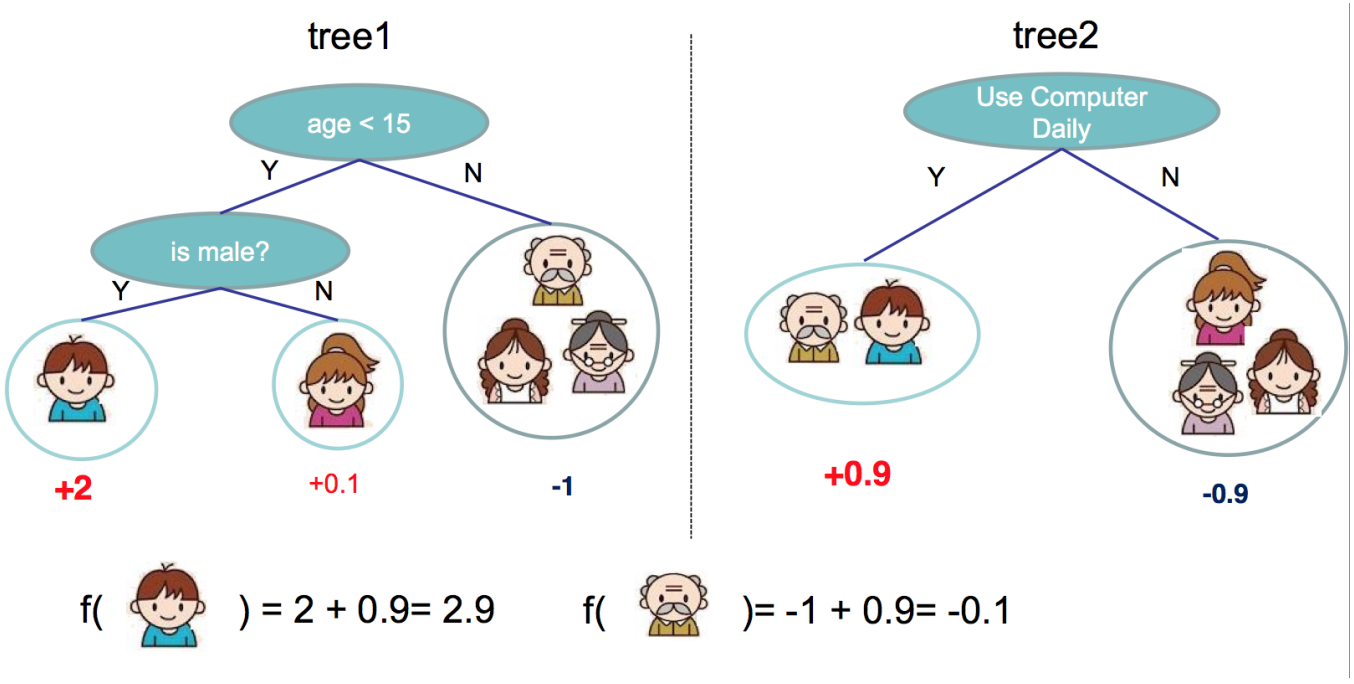


- Classification and Regression Trees:
  - Decision rules
  - Contains one score in each leaf value

Input: age, gender, occupation,... $\Rightarrow$ Does the person like computer games?



Tree Ensembles



Prediction is a sum of scores predicted by each of the tree

Tree Ensembles

- Model: we have T trees

$$\widehat{f}_T(\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} f_t(\mathbf{x}),\ \ f_t(\mathbf{x}) \in F,$$

  where F is a space of functions, containing all regression trees
- Parameters: structure of each tree, and the score in the leaf

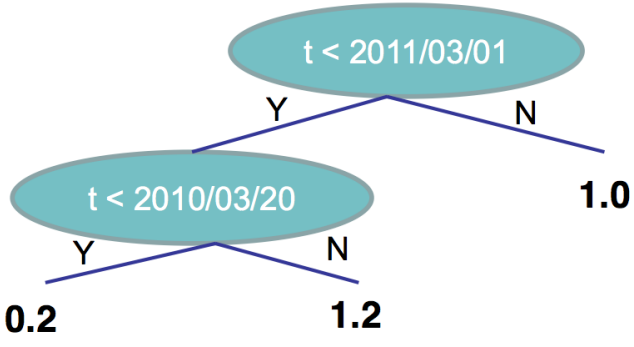
Tree Ensembles

- Very widely used, look for GBM, random forest...
  - Almost half of data mining competitions are won by using some variants of tree ensemble methods
- Invariant to scaling of inputs, so you do not need to do careful features normalization
- Learn higher order interaction between features
- Can be scalable, and are used in Industry

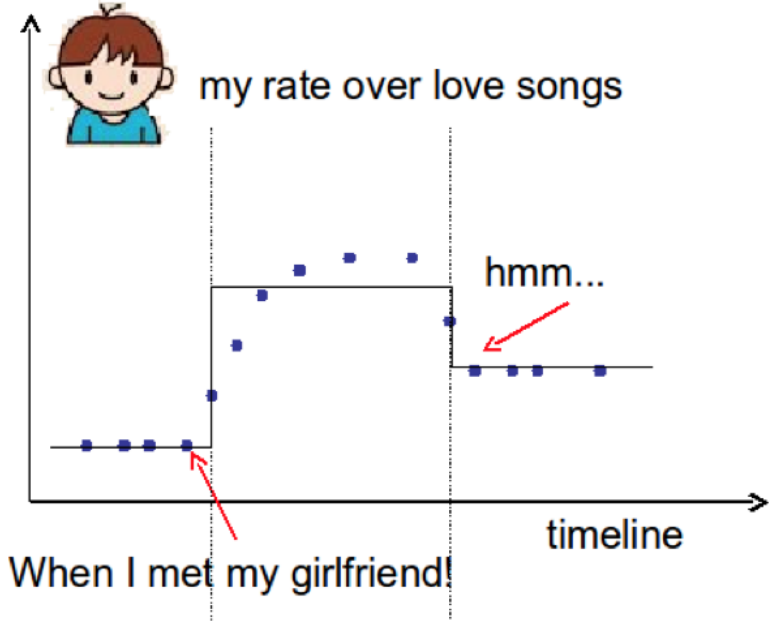
Learning a tree on a single variable

- How can we learn functions?
- Define objective (loss, regularization), and optimize it
- Example:
  - Consider regression tree on single input t (time)
  - We want to predict whether a person like romantic music at time t

The model is a regression tree that splits on time

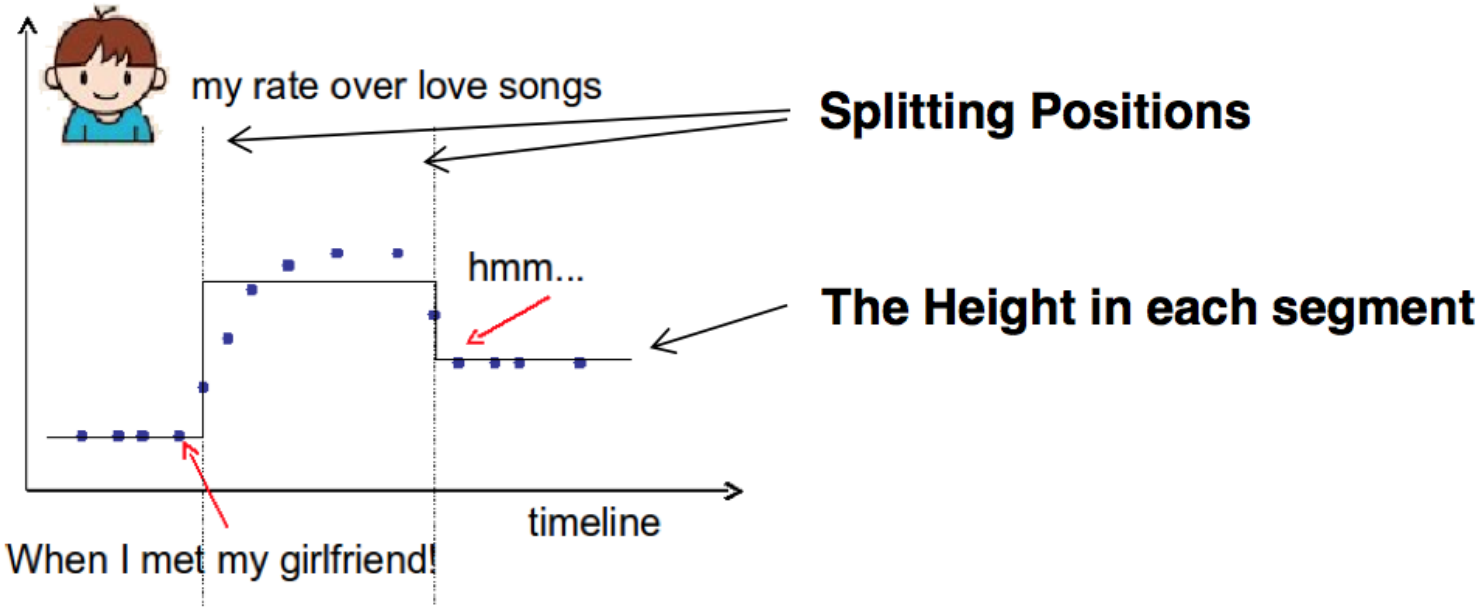


Piecewise step function over time



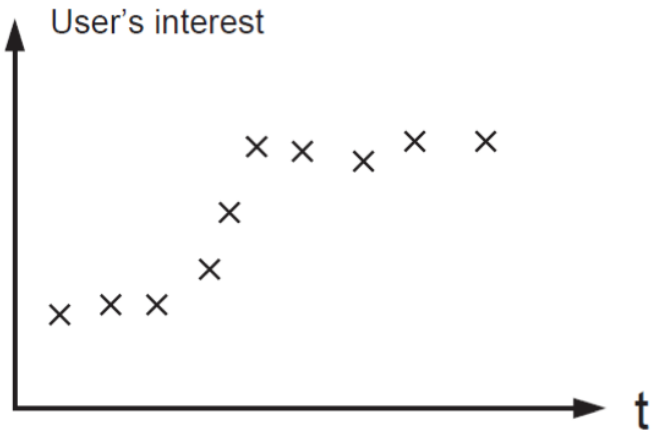
Learning a step function

- Things we need to learn

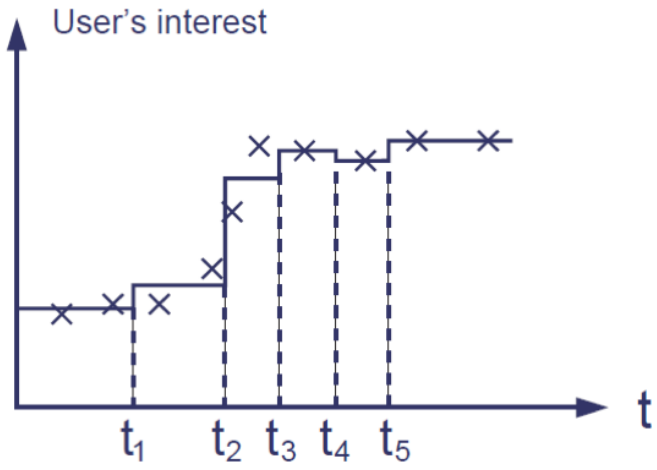


- Objective for single variable regression tree (step functions)
  - Training Loss: How will the function fit on the points?
  - Regularization: How do we define complexity of the function? E.g. number of splitting points, L_2-norm of the height in each segment, ...

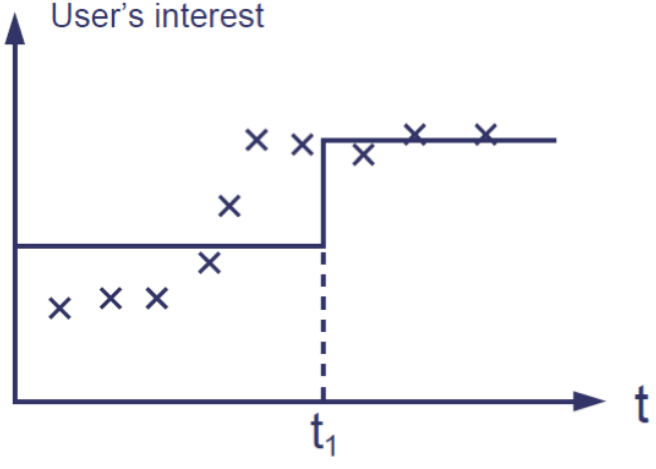
Learning a step function



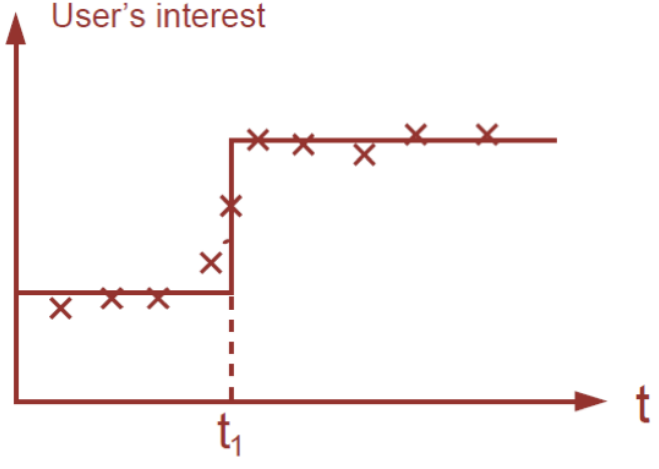
Observed user's interest on topic k against time t



☒ Too many splits, $\Omega(f)$ is high



☒ Wrong split point, $L(f)$ is high



☑ Good balance of $\Omega(f)$ and $L(f)$

Objective for Tree Ensembles

- Assume that we construct T trees

$$\widehat{f}_T(\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} f_t(\mathbf{x}),\ f_t \in F$$

- Objective

$$R(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} l(y_i, \widehat{y}_i) + \sum_{t=1}^{T} \lambda_t \Omega(f_t),$$

  where e.g. $l(y, \widehat{y})$ can be
  - squared loss (regression)
  - log-loss (classification)
- Possible ways to define Ω
  - Number of nodes in the tree, depth
  - L_2-norm of the leaf weight
  - ...

Theory vs. Heuristics

- Decision Trees algorithms contain a lot of heuristics:
  - Split by information gain
  - Prune the tree
  - Maximum depth
  - Smooth leaf values
- Most heuristics map well to objectives
  - Information gain $\rightarrow$ training loss
  - Pruning $\rightarrow$ regularization defined by number of nodes
  - Max depth $\rightarrow$ constraint on the function space
  - Smoothing leaf values $\rightarrow$ L_2-norm regularization on leaf weights
- Decision Trees can be used for Classification, Regression, Ranking, ...
  We just need to define appropriate objective function

Probabilistic model

- Naive Bayes (NB) is a conditional probability model
- Given an object to be classified, represented by features $\mathbf{x} = (x_1, \ldots, x_d)$, Bayes classifier assigns probabilities

$$p(y|x_1, \ldots, x_d)$$

  for each of K possible classes $y \in Y = \{1, \ldots, K\}$
- The conditional probability can be decomposed as

$$p(y|\mathbf{x}) = \frac{p(y)p(\mathbf{x}|y)}{p(\mathbf{x})} \sim p(y)p(\mathbf{x}|y),\ y \in Y$$

- The maximum a posteriori (MAP) decision rule

$$\widehat{y} = \arg \max_{y \in \{1,\ldots,K\}} p(y)p(\mathbf{x}|y)$$

- How to model $p(\mathbf{x}|y)$?

- Let us decompose $p(y, \mathbf{x})$. Using the chain rule we get that

$$
\begin{aligned}
& p(y, x_1, \ldots, x_d) = p(x_1, \ldots, x_d, y) \\
& = p(x_1 | x_2, \ldots, x_d, y) \cdot (x_2, \ldots, x_d, y) \\
& = p(x_1 | x_2, \ldots, x_d, y) \cdot p(x_2 | x_3, \ldots, x_d, y) \cdot p(x_3, \ldots, x_d, y) = \; = \\
& \ldots = \\
& = p(x_1 | x_2, \ldots, x_d, y) \cdot \ldots \cdot p(x_{d-1} | x_d, y) \cdot p(x_d | y) \cdot p(y)
\end{aligned}
$$

- Usually $d \gg 1$ or a feature can take on a large number of values $\Leftrightarrow$ we assume independence of features
- The "naive" conditional independence: assume that each feature x_j is conditionally independent of every other feature x_s given the category y, i.e.

$$p(x_j | x_{j+1}, \ldots, x_d, y) = p(x_j | y)$$

- The joint model

$$p(y|\mathbf{x}) \sim p(y) \prod_{j=1}^{d} p(x_j|y),\ y \in Y$$

- Normalized model

$$p(y|\mathbf{x}) = \frac{1}{Z} p(y) \prod_{j=1}^{d} p(x_j|y),\ y \in Y$$

  where $Z = p(\mathbf{x}) = \sum_{y \in Y} p(y)p(\mathbf{x}|y)$

- The maximum a posteriori (MAP) decision rule

$$\widehat{y} = \arg \max_{y \in \{1,\ldots,K\}} p(y) \prod_{j=1}^{d} p(x_j|y)$$

Event model of the Naive Bayes classifier

- How to define individual models for $p(x_j|y)$?
- We can use any parametric distribution model for $p(x_j|y)$!
- E.g. if x_j is continuous, we can use a Gaussian distribution, i.e.

$$p(x_j = v|y = k) = (2\pi\sigma_{jk}^2)^{-1/2} e^{-(v-\mu_{jk})^2/2\sigma_{jk}^2}$$

- Parameters $(\mu_{jk}, \sigma_{jk}^2)$ are estimated using a subsample S_{jk} of the initial sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$

$$S_{jk} = \{x_{ij} : (\mathbf{x}_i, y_i) \in S,\, y_i = k, i = 1, \ldots, m\}$$

Event model of the Naive Bayes classifier

- We can use various discrete distributions in case some features are categorical
- Multinomial NB: feature x_j counts the number of times event j was observed in a particular instance
- If J is a set of indexes of categorical features, then we can model their joint multimomial distribution as

$$p(x_j, j \in J | y = k) = \frac{(\sum_{j \in J} x_j)!}{\prod_{j \in J} x_j!} \prod_{j \in J} p_{kj}^{x_j}$$

- In log-scale we get a linear classifier

$$\log p(y = k | x_j, j \in J) \sim \log p(y = k) + \sum_{j \in J} x_j \cdot \log p_{kj} = b + \mathbf{w}_k \mathbf{x}_J^\top,$$

with $\mathbf{w}_k = (\log p_{kj}, j \in J)$ and $\mathbf{x}_J = (x_j, j \in J)$

Event model of the Naive Bayes classifier

- Analogously we can define Bernoulli NB: x_j is a boolean expressing the occurrence or absence of the jth term from the vocabulary

$$p(x_j, j \in J | y = k) = \prod_{j \in J} p_{kj}^{x_j} (1 - p_{kj})^{(1 - x_j)}$$

- In order to learn NB:
  - Define models for $p(\mathbf{x}|y)$
  - Estimate parameters of these models using corresponding subsampls, extracted w.r.t. values of y
  - Use MAP to predict $\widehat{y}$

Appendix I: The Newton-Raphson Method for logistic regression

- Surrogate Loss function for a binary logistic regression

$$R(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} L((\mathbf{w}^\top \mathbf{x}_i)y_i)$$

- Steps:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - r_t(R''(\mathbf{w}^t))^{-1}R'(\mathbf{w}^t)$$

- Components of a gradient

$$\frac{\partial R(\mathbf{w})}{\partial w_j} = -\frac{1}{m}\sum_{i=1}^{m}(1-\sigma_i)y_i x_{i,j},\ j = 1,\ldots,d$$

- Hessian

$$\frac{\partial^2 R(\mathbf{w})}{\partial w_j \partial w_k} = \frac{1}{m}\sum_{i=1}^{m}(1-\sigma_i)\sigma_i x_{i,j}x_{i,k},\ j,k = 1,\ldots,d,$$

where $\sigma_i = \sigma(\mathbf{w}^\top \mathbf{x}_i y_i)$, $\sigma(t) = \frac{1}{1+e^{-t}}$

- $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^m \in \mathbb{R}^{m \times d}$ is a matrix of objects features
- $\Gamma = \operatorname{diag}\left(\sqrt{(1-\sigma_i)\sigma_i}\right) \in \mathbb{R}^{m \times m}$ is a diagonal matrix of weights
- $\widetilde{\mathbf{X}} = \Gamma\mathbf{X}$ is a weighted matrix of features
- $\widetilde{y}_i = y_i\sqrt{(1-\sigma_i)/\sigma_i}$, $\widetilde{\mathbf{y}} = \{\widetilde{y}_i\}_{i=1}^m$ is a weighted vector of labels
- Then we get that

$$(R''(\mathbf{w}))^{-1}R'(\mathbf{w}) = -\left(\mathbf{X}^\top\Gamma^2\mathbf{X}\right)^{-1}\mathbf{X}^\top\Gamma\widetilde{\mathbf{y}} = -\left(\widetilde{\mathbf{X}}^\top\widetilde{\mathbf{X}}\right)^{-1}\widetilde{\mathbf{X}}\widetilde{\mathbf{y}}$$

This coincides with a solution of a weighted least-squares problem

$$R(\mathbf{w}) = \left\|\widetilde{\mathbf{X}}\mathbf{w} - \widetilde{\mathbf{y}}\right\|^2 = \frac{1}{m}\sum_{i=1}^m(1-\sigma_i)\sigma_i\left(\mathbf{w}^\top\mathbf{x}_i - \frac{y_i}{\sigma_i}\right)^2 \to \min_{\mathbf{w}}$$

- On each step of the Newton-Raphson method we construct a weighted least-squares regression

$$R(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m}(1-\sigma_i)\sigma_i\left(\mathbf{w}^\top\mathbf{x}_i - \frac{y_i}{\sigma_i}\right)^2 \to \min_{\mathbf{w}}$$

- Here
  - $\sigma_i = p(y_i|\mathbf{x}_i)$ is a probability to correctly classify $\mathbf{x}_i$
  - the close $\mathbf{x}_i$ to the boundary, the bigger the weight $(1-\sigma_i)\sigma_i$
  - the bigger the probability of an error, the bigger the value of y_i/σ_i

  Thus on each iteration we tune $\mathbf{w}$ to perform better on more difficult examples

Iterativey Reweighted Least Squares

- **Input**: $\mathbf{X}$, $\mathbf{y}$, i.e. a matrix and a vector of input features and corresponding labels
- **Output**: estimate of $\mathbf{w}$
- For $t = 1, 2, \ldots$
  - $\sigma_i = \sigma(\mathbf{w}^\top \mathbf{x}_i y_i)$, $i = 1, \ldots, m$
  - $\gamma_i = \sqrt{(1 - \sigma_i)\sigma_i}$, $i = 1, \ldots, m$
  - $\widetilde{\mathbf{X}} = \operatorname{diag}(\gamma_1, \ldots, \gamma_m)\mathbf{X}$
  - $\widetilde{y}_i = y_i \sqrt{(1 - \sigma_i)/\sigma_i}$, $i = 1, \ldots, m$
  - select a gradient step r_t and calculate
    $$\mathbf{w} \leftarrow \mathbf{w} + r_t \left(\widetilde{F}^\top \widetilde{F}\right)^{-1} \widetilde{F}^\top \mathbf{y}$$
  - if σ_i changes not significantly, then stop iterations

Appendix II: Optimal Bayesian Classifier. Problem Statement

- Let
  - X be a feature space
  - Y be a space of labels, e.g. $Y = \{0, 1\}$
  - $p(\mathbf{x}, y)$ be a joint distribution on $X \times Y$
  - $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ be an i.i.d. sample
- We want to construct an optimal classifier $f : X \to Y$

Problem Statement

- We assume that we know joint density

$$p(\mathbf{x}, y) = p(\mathbf{x})p(y|\mathbf{x}) = p(y)p(\mathbf{x}|y)$$

  Here
  - $p(y)$ is a prior distribution on Y
  - $p(\mathbf{x}|y)$ is a likelihood of a class y
  - $p(y|\mathbf{x})$ is a posterior probability of a class y
- Classifier maximizing posterior probability

$$f(\mathbf{x}) = \arg\max_{y \in Y} p(y|\mathbf{x}) = \arg\max_{y \in Y} p(\mathbf{x}|y)p(y)$$

Probability of Error and Risk

- Classifier $f(\mathbf{x})$ divides X into disjoint domains

$$H_y = \{\mathbf{x} \in X | f(\mathbf{x}) = y\},\ y \in Y$$

- We get error for $(\mathbf{x}, y)$ if $\mathbf{x} \in H_z$, $z \neq y$
- Probability of Error: $P(H_z, y) = \int_{H_z} p(\mathbf{x}, y) d\mathbf{x}$
- Losses: λ_{yz} for all $(y, z) \in Y \times Y$
- Average Risk

$$R(f) = \sum_{y \in Y} \sum_{z \in Y} \lambda_{yz} P(H_z, y)$$

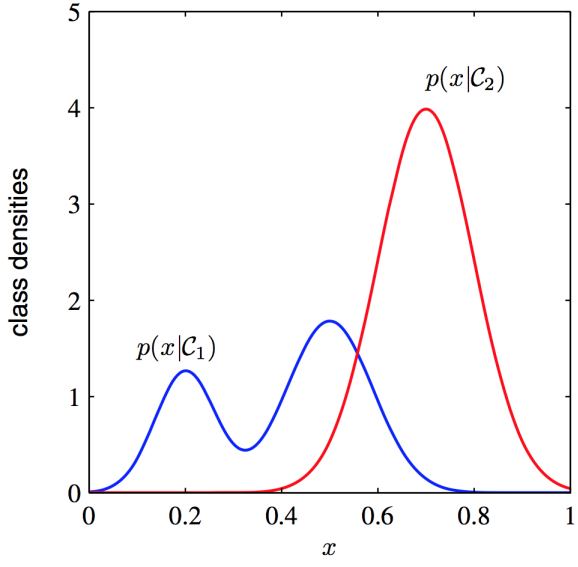
Optimal Bayesian Classifier

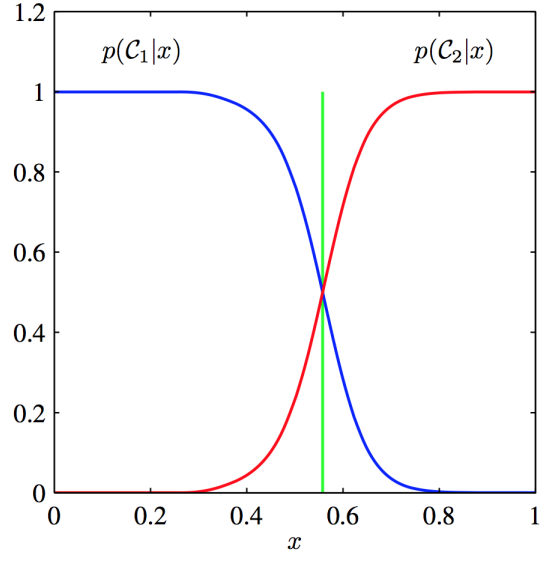
- **Theorem**: Optimal Bayesian Classifier $f(\mathbf{x})$, minimizing average risk $R(f)$, has the form

$$f_{\text{opt}}(\mathbf{x}) = \arg\min_{z \in Y} \sum_{y \in Y} \lambda_{yz} p(y) p(\mathbf{x}|y)$$

- **Corollary**: If $\lambda_{yy} = 0$ and $\lambda_{yz} = \lambda_y$ for all $y, z \in Y$, then

$$f_{\text{opt}}(\mathbf{x}) = \arg\max_{y \in Y} \lambda_y p(\mathbf{x}|y) p(y)$$





- **Theoretical setup**:
  - Assumption: we know probabilities $p(y)$ and $p(\mathbf{x}|y)$, $y \in Y$
  - We now how to construct a classifier $f(\mathbf{x})$, minimizing average risk $R(f)$
- **Applied setup**:
  - Assumption: training set $S_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$
  - Estimate probabilities $\widehat{p}(y)$ and $\widehat{p}(\mathbf{x}|y)$, $y \in Y$ to calculate a Bayesian classifier
- We loose optimality when using empirical probability estimates
- Usually it is more difficult to estimate probability density function then to construct efficient classifier
- Naive Bayesian Classifier:
  - assume that $p(\mathbf{x}|y) = \prod_{j=1}^d p(x_j|y)$, i.e. components of $\mathbf{x}$ are independent
  - simplifying assumption, but it is easier to estimate d one-dimensional distributions $p(x_j|y)$, $j = 1, \ldots, d$
  - although crude approximation, still it sometimes works in practice

Appendix III: Discriminant Analysis

- Let us consider Multidimensional Gaussian Distribution
- We assume that $X = \mathbb{R}^d$ and

$$p(\mathbf{x}|y) = \mathcal{N}(\mathbf{x}; \mu_y, \Sigma_y) = \frac{1}{\sqrt{(2\pi)^N \det \Sigma_y}} e^{-\frac{1}{2}(\mathbf{x}-\mu_y)^\top \Sigma_y^{-1}(\mathbf{x}-\mu_y)},$$

  where $\mu_y \in \mathbb{R}^d$, $\Sigma_y \in \mathbb{R}^{d\times d}$, $y \in Y$
- Optimal Separating Boundary

$$B = \{\mathbf{x} \in X : \lambda_y p(y) p(\mathbf{x}|y) = \lambda_z p(z) p(\mathbf{x}|z)\},$$

  where $y, z \in Y$, $y \neq z$

Linear Discriminant Function

- In case when $\Sigma_y = \Sigma$ for all $y \in Y$ we get a Linear Discriminant function
- E.g. when $Y = \{0, 1\}$ we get the decision boundary

$$B = \Big\{\mathbf{x} \in X : [\Sigma^{-1}(\mu_1 - \mu_0)]^\top \mathbf{x} + \frac{1}{2}\mu_1^\top \Sigma^{-1}\mu_1 - \frac{1}{2}\mu_2^\top \Sigma^{-1}\mu_2 + \log\frac{p(1)}{p(0)} = 0\Big\}$$

- MLE for μ_y and Σ_y are

$$\widehat{\mu}_y = \frac{1}{m_y}\sum_{i:y_i=y} \mathbf{x}_i,\ \widehat{\Sigma}_y = \frac{1}{m}\sum_{y\in Y}\sum_{i:y_i=y}(\mathbf{x}_i - \widehat{\mu}_y)(\mathbf{x}_i - \widehat{\mu}_y)^\top$$

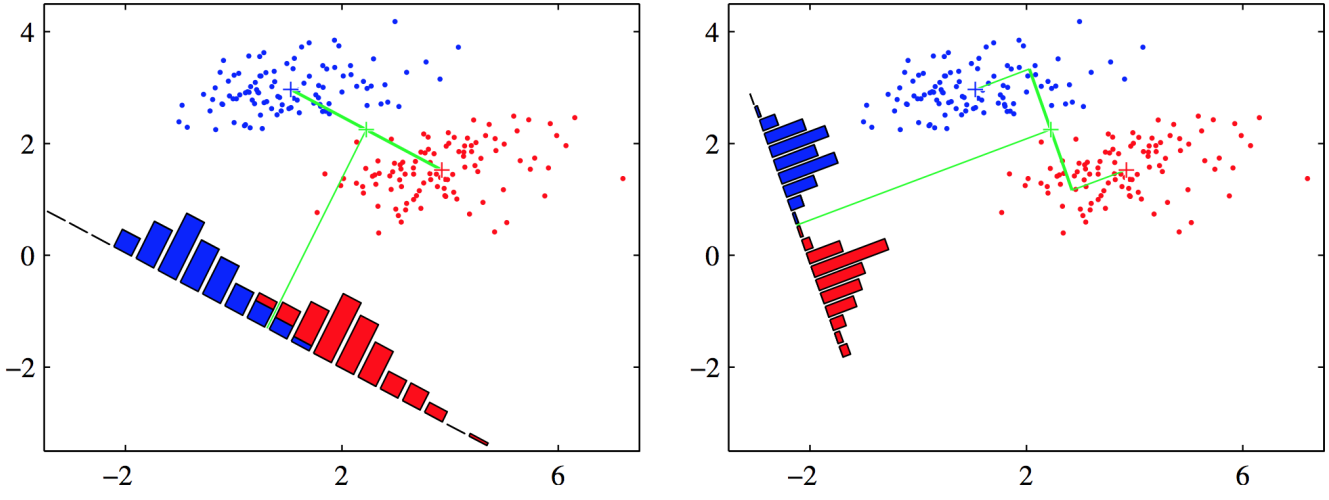
- Regularization of a covariance matrix estimate
  - Use $\widehat{\Sigma} + \tau \mathrm{I}$ instead of $\widehat{\Sigma}$
  - Select τ using cross-validation
- Before constructing a discriminator perform outlier detection and censoring of data

Fisher's Linear Discriminant I

- We consider a two-class classification problem, i.e. $Y = \{0, 1\}$
- We consider a separating hyperplane

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$$

  and we classify $\mathbf{x} : f(\mathbf{x}) \geq -w_0$ for some w_0 as class C_0 and otherwise as class C_1
- We want to select such projection direction that maximizes the class separation



- Let us denote by m_0 and m_1 the number of points, belonging to classes C_1 and C_2 correspondingly
- We define mean vectors of the two classes as

$$\mathbf{m}_0 = \frac{1}{m_0} \sum_{i:y_i=0} \mathbf{x}_i,\ \mathbf{m}_1 = \frac{1}{m_1} \sum_{i:y_i=1} \mathbf{x}_i$$

- The simplest measure of separation $=$ separation of the projected class means, i.e.

$$m_{1,\mathbf{w}} - m_{0,\mathbf{w}} = \mathbf{w}^\top (\mathbf{m}_1 - \mathbf{m}_0)$$

- The within-class variance of the transformed data from class C_k is given by

$$s_k^2 = \sum_{i:y_i=k} (z_i - m_{k,\mathbf{w}})^2,\ k \in \{0, 1\}$$

where $z_i = \mathbf{w}^\top \mathbf{x}_i$

Fisher's Linear Discriminant II

- The Fisher criterion is

$$J(\mathbf{w}) = \frac{(m_{1,\mathbf{w}} - m_{0,\mathbf{w}})^2}{s_1^2 + s_0^2}$$

- In vector form

$$J(\mathbf{w}) = \frac{\mathbf{w}\mathbf{S}_B\mathbf{w}}{\mathbf{w}\mathbf{S}_W\mathbf{w}},$$

  where

$$\mathbf{S}_B = (\mathbf{m}_1 - \mathbf{m}_0)(\mathbf{m}_1 - \mathbf{m}_0)^\top$$

  is the between-class covariance matrix, and

$$\mathbf{S}_W = \sum_{i:y_i=0} (\mathbf{x}_i - \mathbf{m}_0)(\mathbf{x}_i - \mathbf{m}_0)^\top + \sum_{i:y_i=1} (\mathbf{x}_i - \mathbf{m}_1)(\mathbf{x}_i - \mathbf{m}_1)^\top$$

  is the within-class covariance matrix

- $J(\mathbf{w})$ is maximized for

$$\mathbf{w} \sim \mathbf{S}_W^{-1}(\mathbf{m}_1 - \mathbf{m}_0)$$

Fisher's discriminant for multiple classes

- We consider $K > 2$ classes ($d > K$)
- The weight vectors $\{\mathbf{w}_k\}$ can be consider as columns of a matrix $\mathbf{W}$, s.t.

$$\mathbf{y} = \mathbf{W}\mathbf{x},$$

  i.e. $f_k(\mathbf{x}) = \mathbf{w}_k^\top \mathbf{x}$ and we assign a point $\mathbf{x}$ to class C_k if $f_k(\mathbf{x}) > f_j(\mathbf{x})$ for all $j \neq k$
- The generalization of the within-class covariance is $\mathbf{S}_W = \sum_{k=1}^{K} \mathbf{S}_k$, where for $k \in \{0, 1, \ldots, K-1\}$

$$\mathbf{S}_k = \sum_{i:y_i=k} (\mathbf{x}_i - \mathbf{m}_k)(\mathbf{x}_i - \mathbf{m}_k)^\top, \; \mathbf{m}_k = \frac{1}{m_k} \sum_{i:y_i=k} \mathbf{x}_i,$$

Fisher's discriminant for multiple classes II

- The total covariance matrix

$$\mathbf{S}_T = \sum_{i=1}^{m} (\mathbf{x}_i - \mathbf{m})(\mathbf{x}_i - \mathbf{m})^\top,$$

where the mean of the total data set $\mathbf{m} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{x}_i$, and

$$\mathbf{S}_T = \mathbf{S}_W + \mathbf{S}_B,\ \mathbf{S}_B = \sum_{k=0}^{K-1} m_k(\mathbf{m}_k - \mathbf{m})(\mathbf{m}_k - \mathbf{m})^\top$$

- Let us consider *projected* features $\mathbf{z} = \mathbf{W}\mathbf{x}$. In this space we can analogously defined matrices

$$\mathbf{s}_W = \sum_{k=0}^{K-1} \sum_{i:y_i=k} (\mathbf{z}_i - \mu_k)(\mathbf{z}_i - \mu_k)^\top,\ \mathbf{s}_B = \sum_{k=0}^{K-1} m_k(\mu_k - \mu)(\mu_k - \mu)^\top,$$

where $\mu_k = \frac{1}{m_k}\sum_{i:y_i=k} \mathbf{z}_i$, $\mu = \frac{1}{m}\sum_{k=0}^{K-1} m_k \mu_k$

Fisher's discriminant for multiple classes III

- We want to construct a scalar that is large when the between-class covariance is large, and when the within-class covariance is small
- One example is given by

$$J(\mathbf{W}) = \mathrm{Tr}\left\{\mathbf{s}_W^{-1}\mathbf{s}_B\right\} = \mathrm{Tr}\left\{(\mathbf{W}\mathbf{S}_W\mathbf{W}^\top)^{-1}(\mathbf{W}\mathbf{S}_B\mathbf{W})^\top\right\}$$

- The weight vectors are determined by those eigenvectors of $\mathbf{S}_W^{-1}\mathbf{S}_B$ that correspond to the largest eigenvalues