Machine Learning for Finance (FIN 570)
SVM, KNN, Decision Tree

Instructor: Jaehyuk Choi

Peking University HSBC Business School, Shenzhen, China

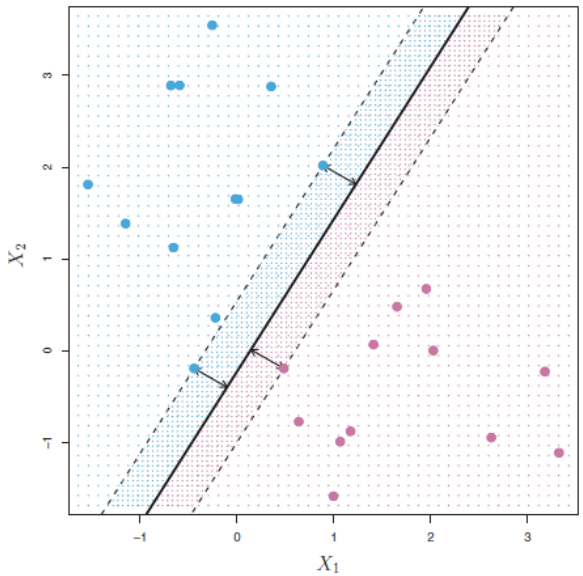
2018-19 Module 1 (Fall 2018)

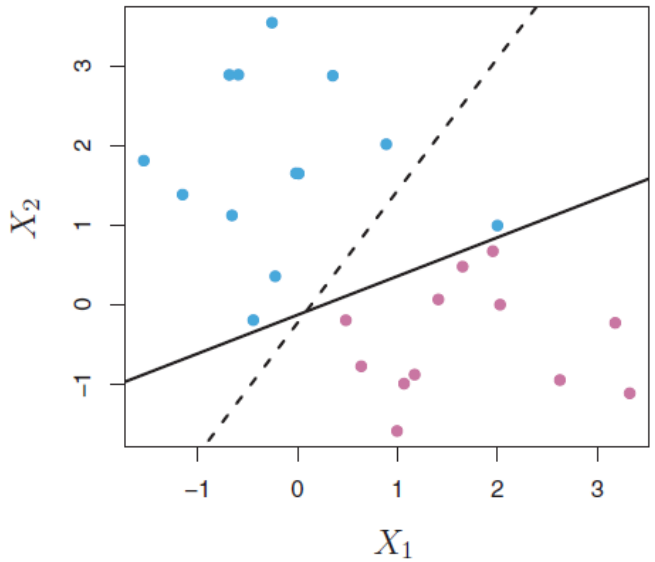
Maximal Margin Classifier

For $y_i \in \{-1, 1\}$, maximize the margin of the separating hyperplane M,

$$y_i(w_0 + \sum_{j=1}^{p} X_{ij} w_j) = y_i(w_0 + \boldsymbol{X}_{i*}\boldsymbol{w}) \geq M > 0 \text{ for all } i, \text{ with } |\boldsymbol{w}| = 1$$

Maximal margin classifier only works for the separable data set and is sensitive to the change in the *support vectors*.





Support Vector Classifier

We make maximal margin classifier flexible: maximize the margin of the separating hyperplane M with $|\boldsymbol{w}| = 1$,

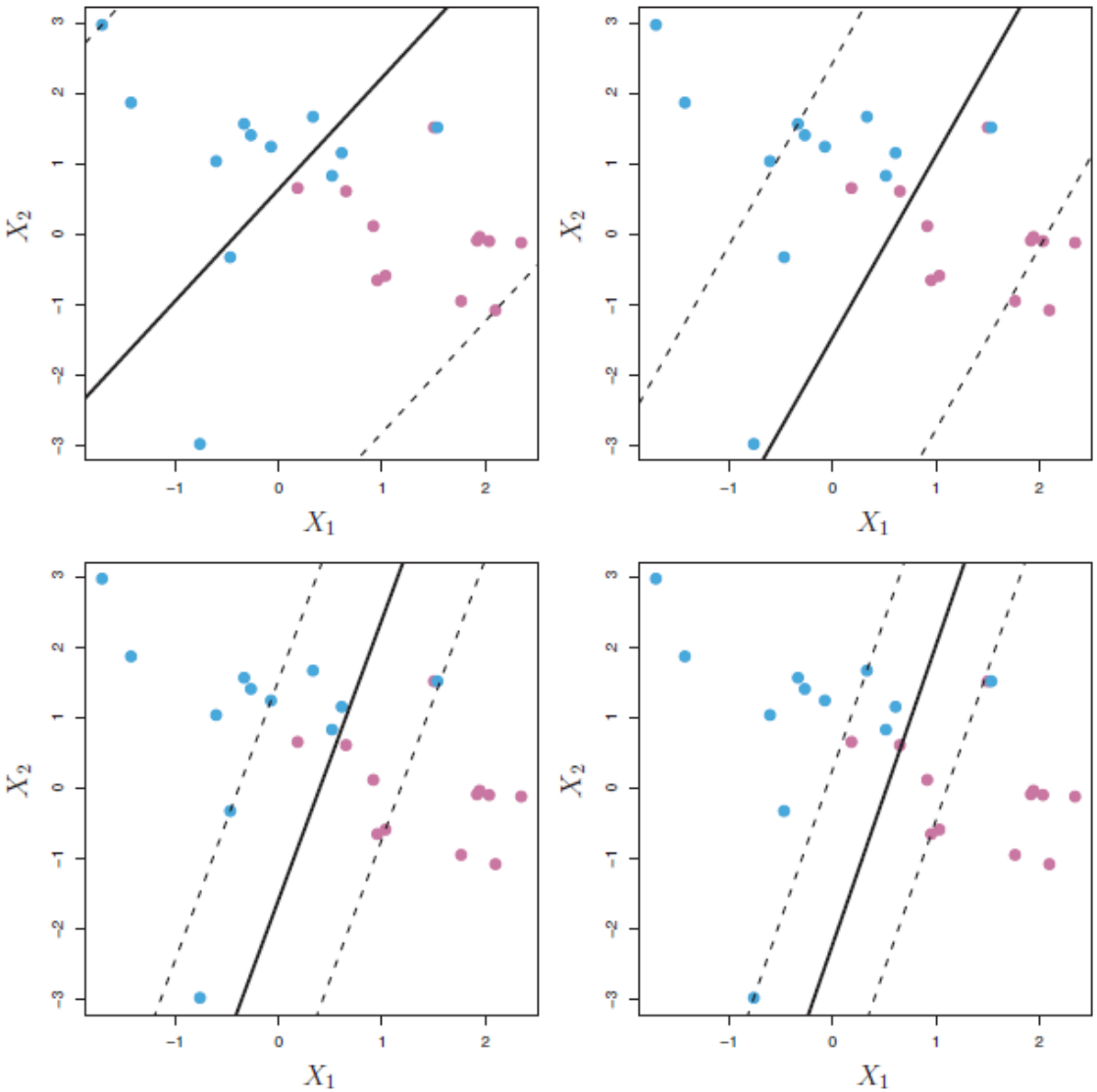
$$y_i(w_0 + \boldsymbol{X}_{i*}\boldsymbol{w}) \geq M(1 - \varepsilon_i) \text{ for all } i, \quad \sum_{i=1}^{n} \varepsilon_i \leq C,$$

where $\varepsilon_i \geq 0$ is *slack variable* indicating the degree of violation ($\varepsilon_i = 0$: no violation, $\varepsilon_i < 1$: margin violation, $\varepsilon_i > 1$: classification violation) and C is a *budget* for the amount of violations by all observations.
Alternatively (in PML), we minimize

$$\frac{1}{2}|\boldsymbol{w}|^2 + C' \sum_{i=1}^{n} \xi_i,$$

where $\{\xi_i \geq 0\}$ satisfies $y_i(w_0 + \boldsymbol{X}_{i*}\boldsymbol{w}) \geq 1 - \xi_i$ for all i. The model converges to maximal margin classifier if C' is very large, so the role of C' is opposite to that of C in the original formulation.

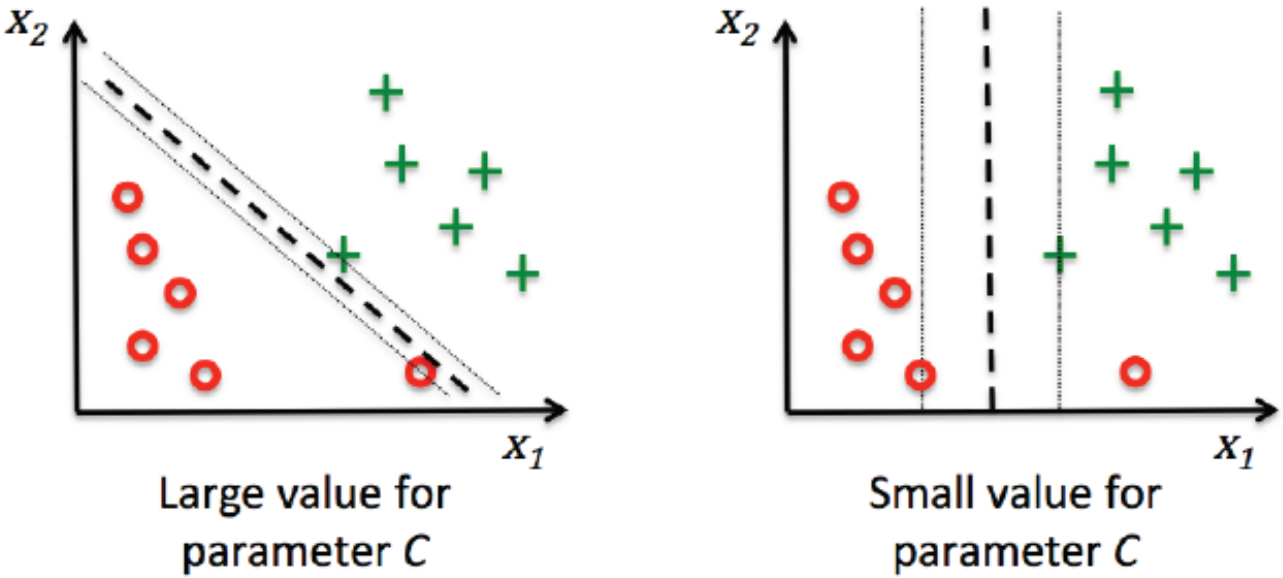
Support Vector Classifier: the role of C



The value of C decreasing from the largest value on top left.

Support Vector Classifier: the role of C'

Large C' (left) vs small C' (right)



Support Vector Machines (SVM)

How can we deal with non-linear decision boundary?

Enlarging feature space

Including high-order terms, $1, X_j, \cdots, X_j^2, \cdots, X_i X_j, \cdots$, can be helpful, but the computation becomes very heavy.

Kernel

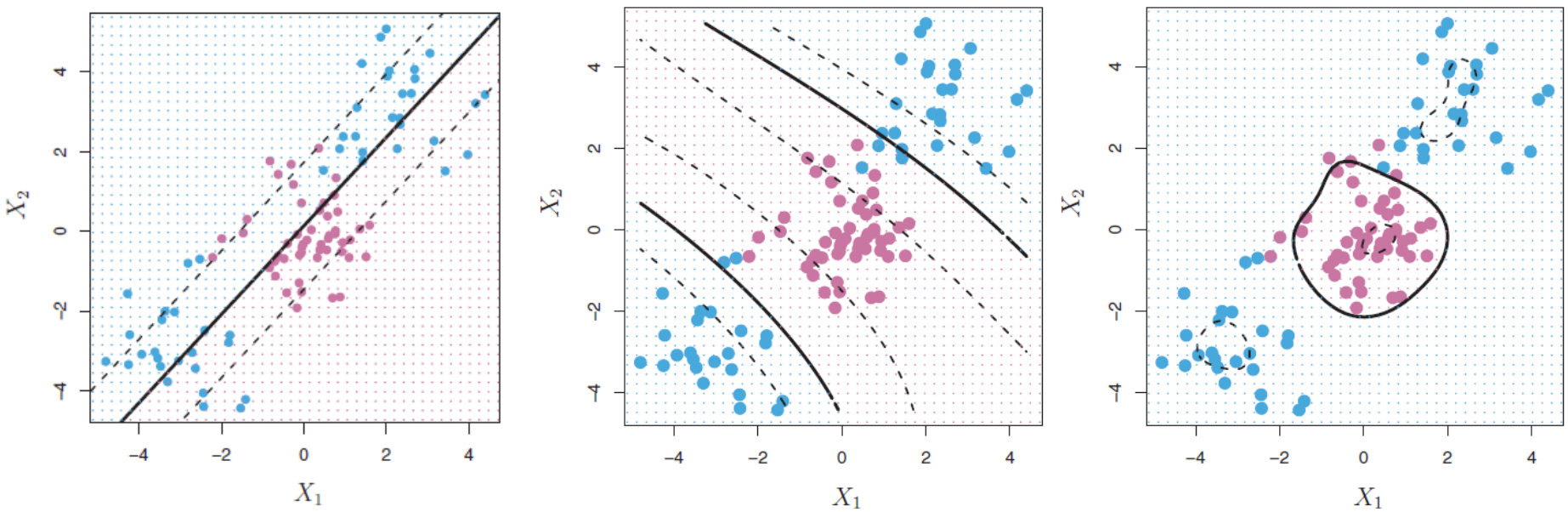
Instead we introduce kernel function, as a generalization of dot product in hyperplane:

- Linear: $K(\boldsymbol{X}_{i*}, \boldsymbol{X}_{j*}) = \boldsymbol{X}_{i*} \boldsymbol{X}_{j*}^T$
- Polynomial: $K(\boldsymbol{X}_{i*}, \boldsymbol{X}_{j*}) = (1 + \boldsymbol{X}_{i*}\, \boldsymbol{X}_{j*}^T)^d$ (order d)
- Radial basis: $K(\boldsymbol{X}_{i*}, \boldsymbol{X}_{j*}) = \exp(-\gamma |\boldsymbol{X}_{i*} - \boldsymbol{X}_{j*}|^2)$ $(\gamma = 1/2\sigma^2)$

Kernel $K(\boldsymbol{X}_{i*}, \boldsymbol{X}_{j*})$ can be understood as a *distance* between two observations: $\boldsymbol{X}_{i*}$ and $\boldsymbol{X}_{j*}$ are similar if the kernel value is high (low) whereas they are different if low (high).

SVM: non-linear decision boundary

SVM classification with linear kernel (left) polynomial kernel of degree 3 (middle) and radial basis kernel (right)



K Nearest Neighbor (KNN)

The method is based on the set of the K nearest neighbors around x, $N_K(x)$:

- Regression:

$$\hat{y} = f(x) = \frac{1}{K} \sum_{x_i \in N_K(x)} y_i$$

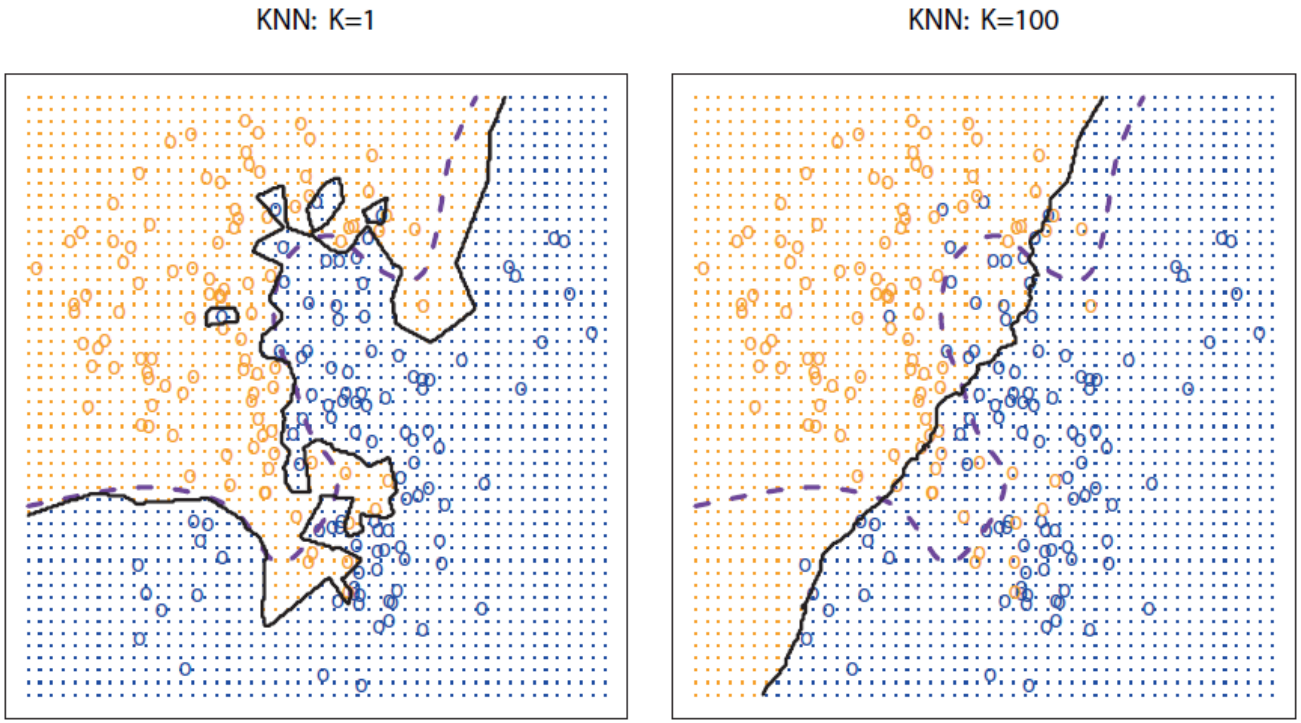
- Classifier:

$$\text{Prob}(y = j|x) = \frac{1}{K} \sum_{x_i \in N_K(x)} I(y_i = j)$$

$$\hat{y} = \text{majority of } \{y_i\} \text{ for } x_i \in N_K(x)$$

- Parametric model (vs non-parametric)
- Learning step is not required, but KNN is intensive in both computation and storage (*memorize* training data set for prediction)

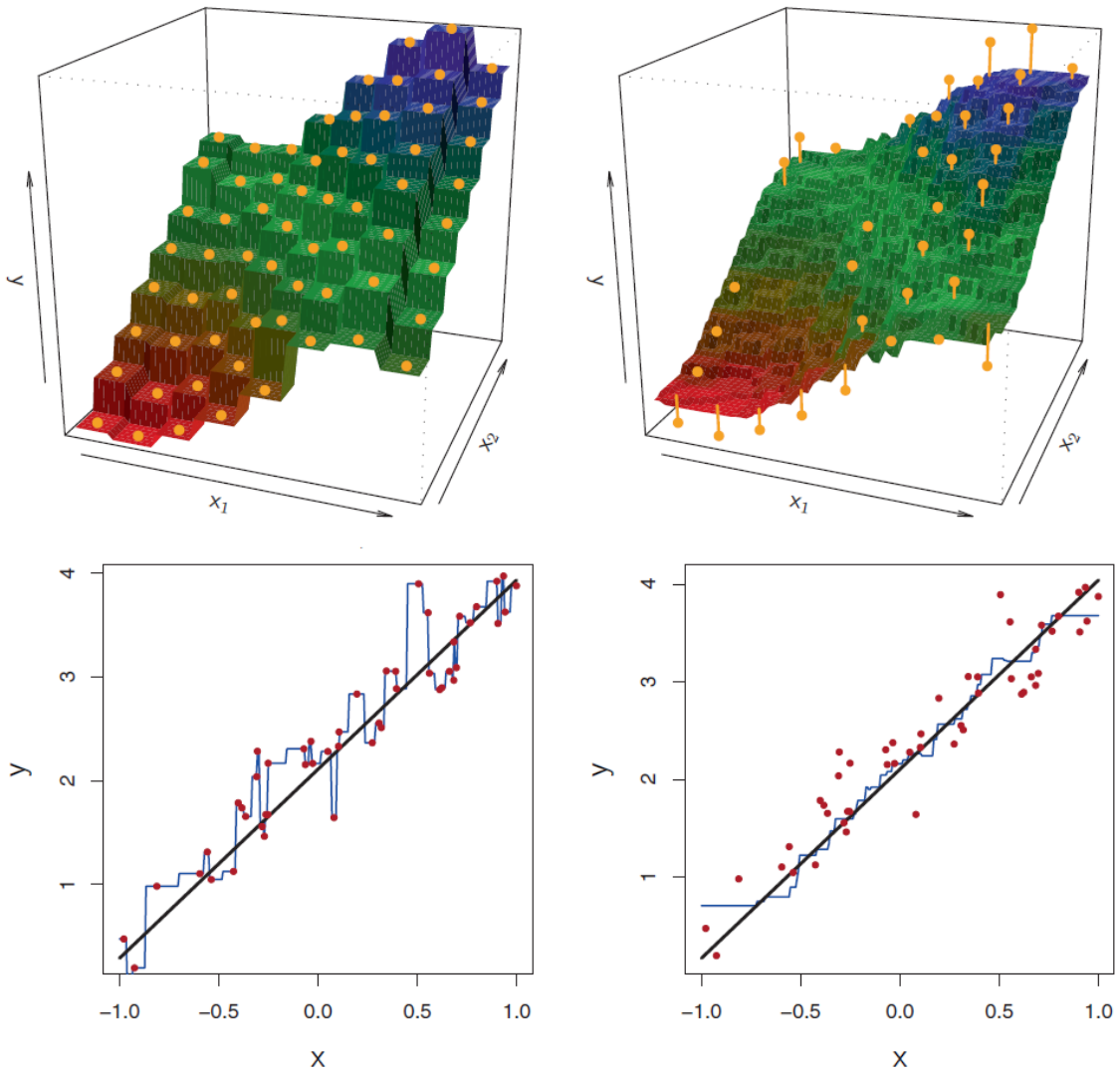
KNN: Classfier example



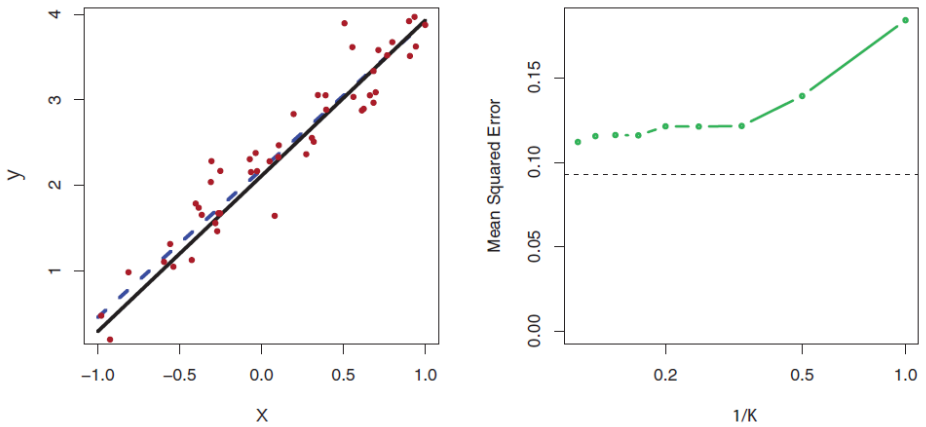
- Overfitting as $K \to 1$
- Generalization (accuracy on test set) is difficult as $K \to 1$

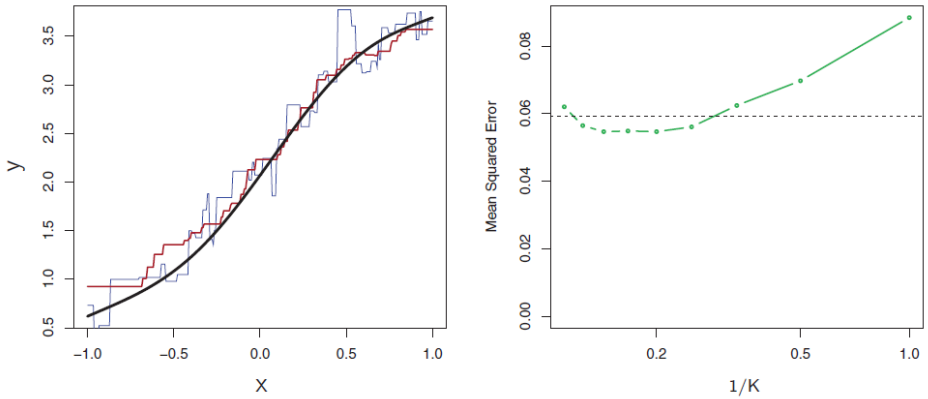
KNN: Regression example

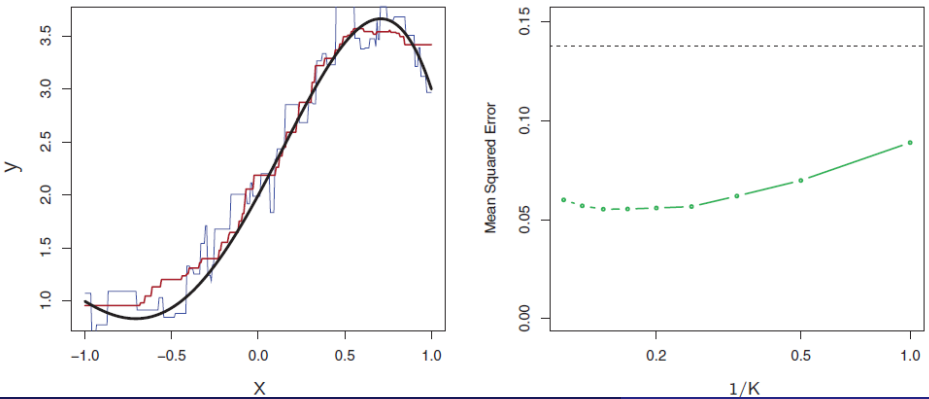
$K = 1$ (left) vs $K = 9$ (right)



KNN: Parametric vs Non-parametric



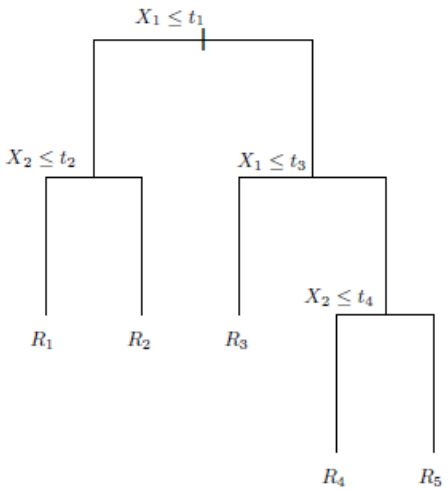


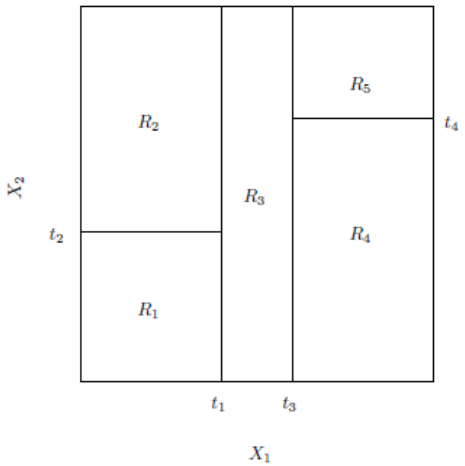


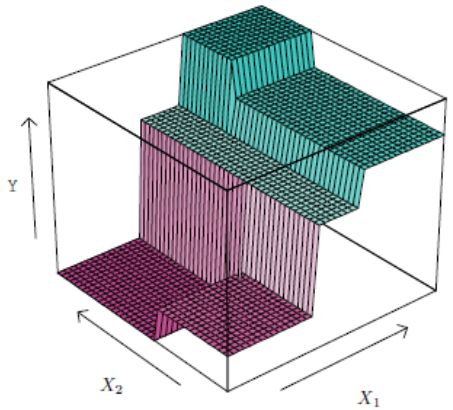
Non-parametric regression works better as the true function deviates from the basis function (linear function in this example). The dashed line is the test set RSS from linear regression.

Decision Tree

- Regression/classification is made on a series of conditions on input variables
- Final conclusion on each *terminal node* or *leaf* of the (upside down) tree.
- The predictor space is broken down to boxes
- Regression: the average value is assigned to each box
- Classification: the majority class is assigned to each box







Growing Tree

We want to minimize the error in each leaf. For a given leaf of tree, t,
Regression Error:

$$RSS(t) = \sum_{i \in t} (y_i - \hat{y}_t)^2$$

Classification Error (measure of impurity):

- Gini index: $I_G(t) = \sum_{k=1}^{K} p(k|t)\big(1 - p(k|t)\big) = 1 - \sum_{k=1}^{K} p^2(k|t)$
- (Cross-)Entropy: $I_H(t) = -\sum_{i=1}^{K} p(k|t) \log_2 p(k|t)$
- Classification Error: $I_E(t) = 1 - \max_{1 \le k \le K} p(k|t)$

  I_E is less sensitive for branching options, so not recommended for growing tree.
- $I_G(t) = I_H(t) = I_E(t) = 0$ if the composition is pure, i.e., $p(k|t) = 1$ for some k. Otherwise $I(t) > 0$.

We can grow tree to the maximum level so that each leaf contains only one sample point. However, it is over-fitting. We need to regularize the number of leaves or the maximum level of branching.

Growing Tree

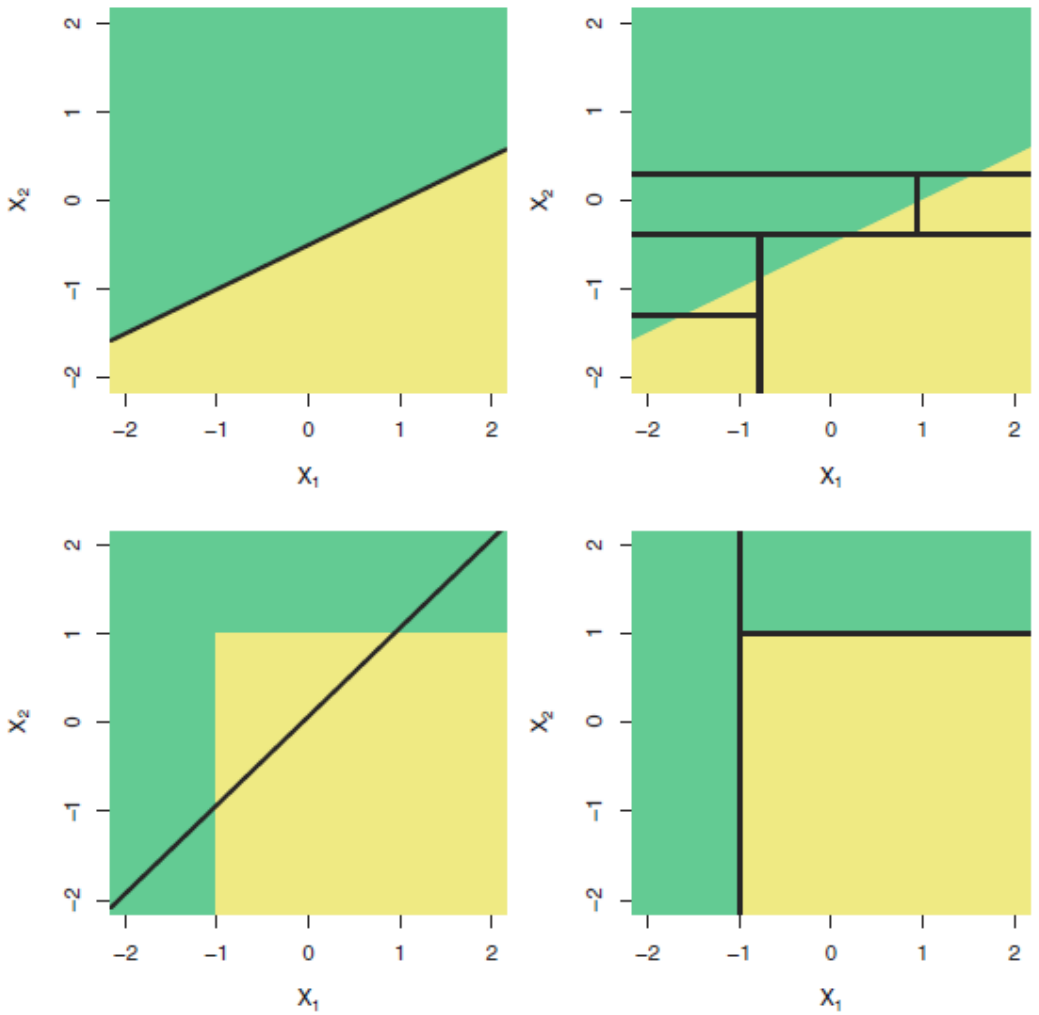
The tree is split (boundary and feature) is determined in such a way that the information gain (or impurity decrease) is maximized:

$$\mathsf{IG}(D_P) = I(D_P) - \frac{N_L}{N_P} I(D_L) - \frac{N_R}{N_P} I(D_R)$$

where D_P, D_L and D_R are the parent, left and right data set and N is the number of the samples in the corresponding sets.

$$I(D) = \sum_{t \in D} I(t) \quad \text{where} \quad I = I_E, I_H \text{ or } I_G$$

Tree vs Linear model



Two classification problems (top vs bottom) approached by linear model (left) and decision tree (right). Linear model outperforms on the problem on the top, whereas decision tree outperforms on the problem on the bottom.

Decision Tree

Pros

- Intuitive and easy to explain. (Even easier than linear regression)
- Closely mirror human decision making
- Can be displayed graphically and easily interpreted by non-experts

Cons

- Prediction is less accurate than other ML methods
- Model variance is high (sensitive to input samples)
  $\longrightarrow$ Bagging, Random Forests, Ada-Boosting

Bagging, Random Forest (Ensemble Learning, Ch 7)

Build an ensemble of different classifiers/regressors: the result is interpreted as majority vote/average.

$$\hat{f}_E(X) = \text{Avg}_{e \in E}\{\hat{f}_e(X)\} \quad \text{or} \quad \hat{f}_E(X) = \text{Majority}_{e \in E}\{\hat{f}_e(X)\}$$

The principles of bagging and RF can be applied to any ML methods.

Bagging

- Build different trees out of subsets (*bags*) of training set (bootstrapping).
- Increase prediction accuracy and reduce model variance

Random Forest (RF)

- Build different trees out of subsets of training set (bootstrapping).
- At each split, randomly select m ($\approx \sqrt{p}$) out of p features.
- By random selection of features, RF reduces the correlation between the trained trees.
- Maximum tree depth is allowed.