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

Instructor: Jaehyuk Choi

Peking University HSBC Business School, Shenzhen, China

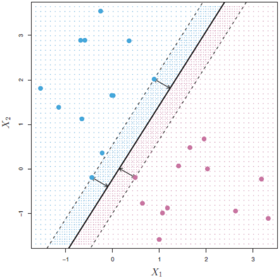
2023-24 Module 3 (Spring 2024)

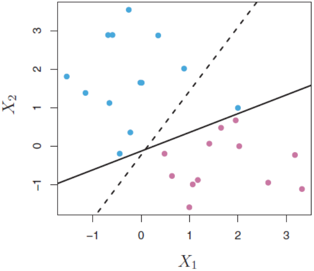
Maximal Margin Classifier

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

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

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





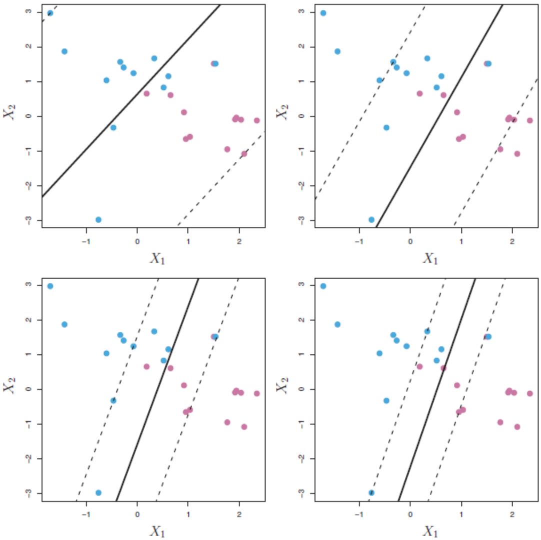
Support Vector Classifier

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

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

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.

Support Vector Classifier: the role of C



The value of C is decreasing from top left to bottom right, being more strict on violation.

Support Vector Classifier (in PML)

- In PML (`sklearn`) implementation, M is absorbed into $\boldsymbol{w}$ ($|\boldsymbol{w}| = 1/M$).
- For maximum margin classifier, we minimize $|\boldsymbol{w}|$ satisfying

$$y_i(w_0 + \boldsymbol{X}_{i*}\boldsymbol{w}) \geq 1 \quad \text{for all} \quad i.$$

- If violation is allowed,

$$y_i(w_0 + \boldsymbol{X}_{i*}\boldsymbol{w}) \geq 1 - \xi_i \quad \text{for all} \quad i \quad (\xi_i \geq 0).$$

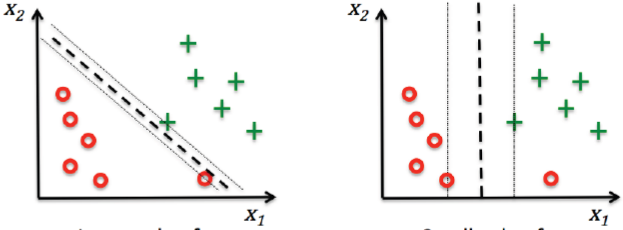
  and $\sum_{i=1}^n \varepsilon_i$ is the sum of violation.
- We minimize:

$$J(w) = \frac{1}{2}|\boldsymbol{w}|^2 + C' \sum_{i=1}^n \xi_i.$$

- When C' is large (small), the violation ξ_i is more (less) important than $|\boldsymbol{w}|$ and the model is more (less) strict in the violation. So the role of C' is opposite to that of C in the original formulation. (The model converges to maximal margin classifier if $C' \to \infty$ or $C \to 0$.)

Support Vector Classifier: the role of C'

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



Large value for parameter *C*

Small value for parameter *C*

Support Vector Machines (SVM)

How can we extend linear classifier to 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:

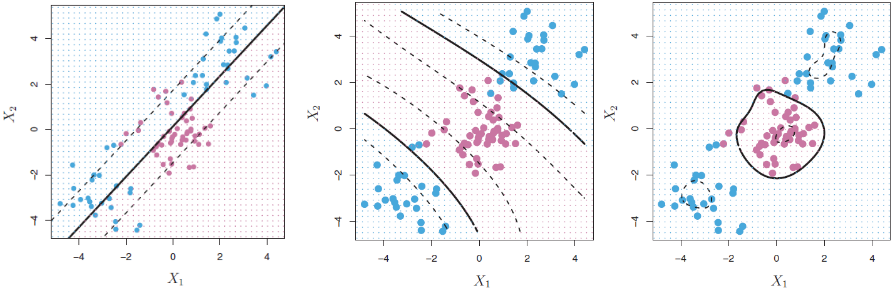
- Linear: $K(\boldsymbol{X}_{i*}, \boldsymbol{X}_{j*}) = \boldsymbol{X}_{i*}\boldsymbol{X}_{j*}^T$
- Polynomial: $K(\boldsymbol{X}_{i*}, \boldsymbol{X}_{j*}) = (1 + \boldsymbol{X}_{i*}\,\boldsymbol{X}_{j*}^T)^d$ (order d)
- Radial basis: $K(\boldsymbol{X}_{i*}, \boldsymbol{X}_{j*}) = \exp(-\gamma|\boldsymbol{X}_{i*}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

Kernel $K(\boldsymbol{X}_{i*}, \boldsymbol{X}_{j*})$ can be understood as a *similarity* (or *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:

$$\hat{y} = f(x) = \frac{1}{K} \sum_{x_i \in N_K(x)} y_i \quad \text{(average of the neighbors)}$$

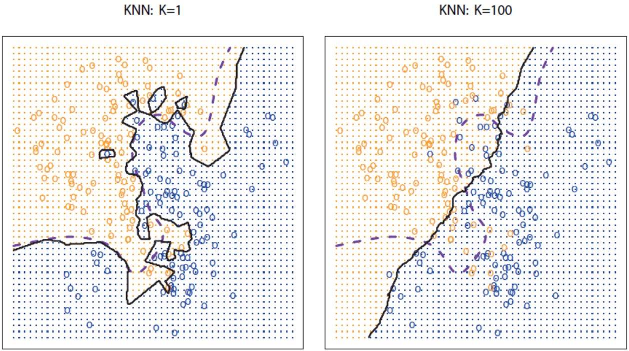
- Classifier:

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

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

- Non-parametric model: there are no parameters (e.g., $\boldsymbol{w}$ or $\boldsymbol{\beta}$) to fit.
- There is no learning step (no $J(\boldsymbol{w})$ to minimize). But KNN is intensive in both computation and storage (*memorize* training data set for prediction)

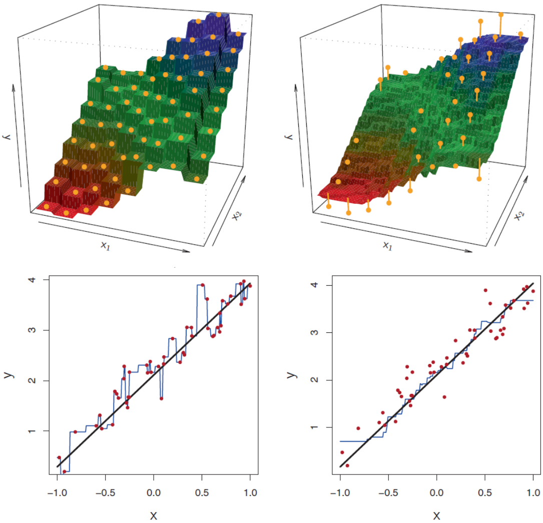
KNN: Classfier example



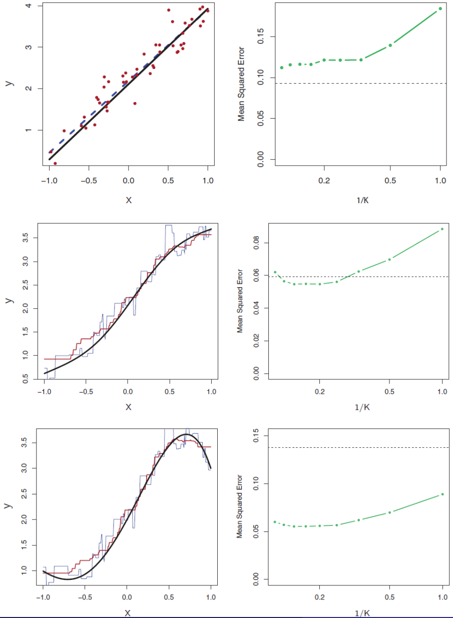
- The number of neighbors, K, is a hyperparameter.
- $K \to 1$: Overfitting. Sensitive to data. Generalization is difficult.
- $K \to \infty$: Allow errors. Insensitive to data.

KNN: Regression example

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



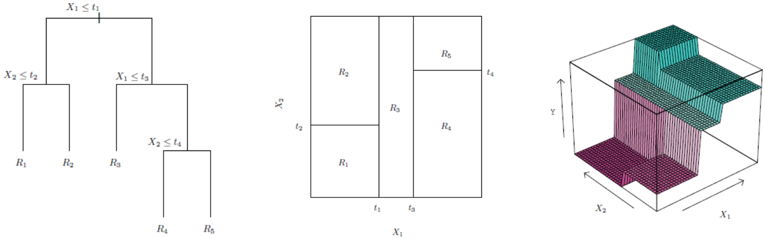
KNN: Parametric vs Non-parametric



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

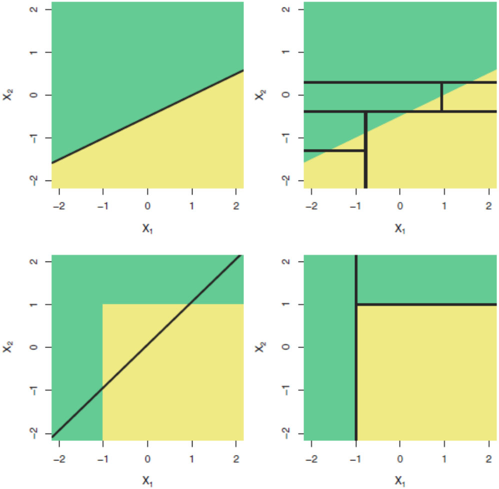
The tree is split (boundary and feature) is determined in such a way that the information gain (or impurity decrease) is maximized:

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

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

Bootstrap

- Given limited sample size, how to measure the model variance? E.g., stdev of regression coefficient?
- Repeat learning on multiple copies of training set which are randomly selected from the original set (with replacement)

Variance minimization between two investments X and Y (ISLR §5.2)

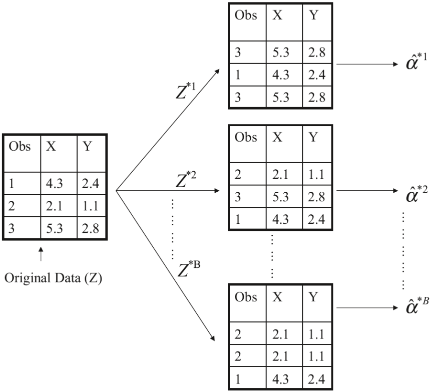
The ratio α minimizes Var$(\alpha X + (1-\alpha)Y)$:

$$\alpha = \frac{\sigma_Y^2 - \rho\sigma_X\sigma_Y}{\sigma_X^2 + \sigma_Y^2 - 2\rho\sigma_X\sigma_Y},$$

where $\sigma_X^2 =$ Var(X), $\sigma_Y^2 =$ Var(Y) and $\rho =$ Corr(X, Y).
What is the stdev of $\hat{\alpha}$ given limited samples of (X, Y)?

$$SE(\hat{\alpha}) = \sqrt{\frac{1}{B-1}\sum_{k=1}^{B-1}\left(\hat{\alpha}_k - E(\hat{\alpha}_r)\right)^2}$$

Bootstrap (illustration)



Original Data (Z)

Obs	X	Y
1	4.3	2.4
2	2.1	1.1
3	5.3	2.8

Z^{*1} → $\hat{\alpha}^{*1}$

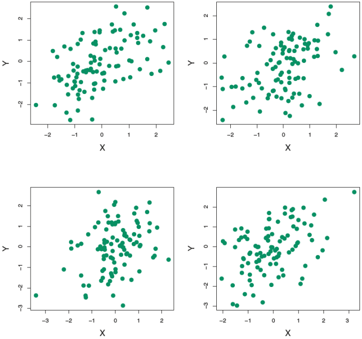
Obs	X	Y
3	5.3	2.8
1	4.3	2.4
3	5.3	2.8

Z^{*2} → $\hat{\alpha}^{*2}$

Obs	X	Y
2	2.1	1.1
3	5.3	2.8
1	4.3	2.4

Z^{*B} → $\hat{\alpha}^{*B}$

Obs	X	Y
2	2.1	1.1
2	2.1	1.1
1	4.3	2.4



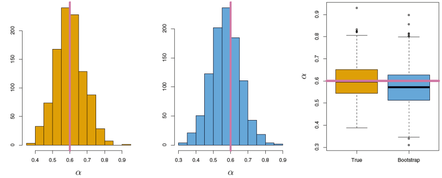


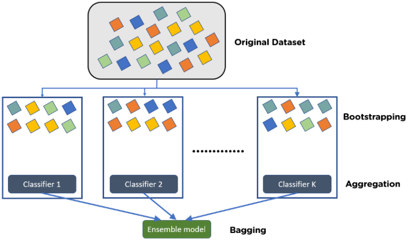
FIGURE 5.10. Left: *A histogram of the estimates of α obtained by generating 1,000 simulated data sets from the true population.* Center: *A histogram of the estimates of α obtained from 1,000 bootstrap samples from a single data set.* Right: *The estimates of α displayed in the left and center panels are shown as boxplots. In each panel, the pink line indicates the true value of α.*

Bagging (Ensemble Learning, Ch 7)

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

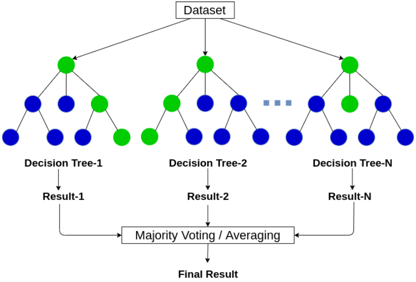
$$\hat{f}_E(X) = \underset{e\in E}{\text{Avg}}\{\hat{f}_e(X)\} \quad \text{or} \quad \hat{f}_E(X) = \underset{e\in E}{\text{Majority}}\{\hat{f}_e(X)\}$$

- The principles of bagging can be applied to any ML methods, but mostly to decision tree.
- Build many trees (models) out of bootstrapped subsets (*bags*) of training set.
- Efficiently reduce model variance. Base model may have overfitting (high variance), e.g., unpruned tree.



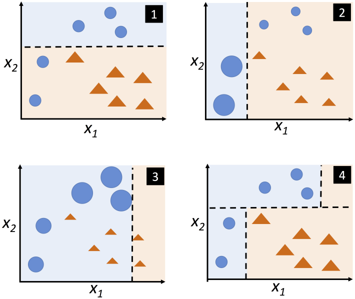
Random Forest (RF)

- In addition to bagging, randomly restrict the features used in each split.
- From the random selection of features, RF decorrelates the base trees.
- Optimal `max_features`?: $m \approx \sqrt{p}$ in classification, $m \approx p/3$ in regression. See `RandomForestClassifier` or `RandomForestRegressor` in `sklearn`
- Used for accessing feature importance for feature selection (**PML** Ch. 4).
- Also used for continuous variable regression. (**PML** Ch. 10).



Adaptive boosting (AdaBoosting)

- Each model is built *sequentially* using information from previously learned models. (vs models are build *independently* in bagging)
- Efficient in leveraging weak learners.
- In each step, increase weight on the samples either incorrectly classified or previous models disagree on. (The weight on the other samples is effectively reduced after normalization.)
- Majority rule is applied on the learners.


1
x_2
x_1
2
x_2
x_1
3
x_2
x_1
4
x_2
x_1