

# DATASCI 347 Machine Learning

## Lecture 19: Midterm review

Ruoxuan Xiong

Based on Lectures 1-18

# Announcements

- The midterm will be available from 4/8 12:00 AM until 4/12 11:59 PM at Quizzes on Canvas
- You can choose any 24 hours in between to finish it
- Once you decide to take it, you can open the quiz and the time starts to count
- Once you finish (within 24 hours), upload your solution (two files: one html and one ipynb file) and click submit quiz on Canvas

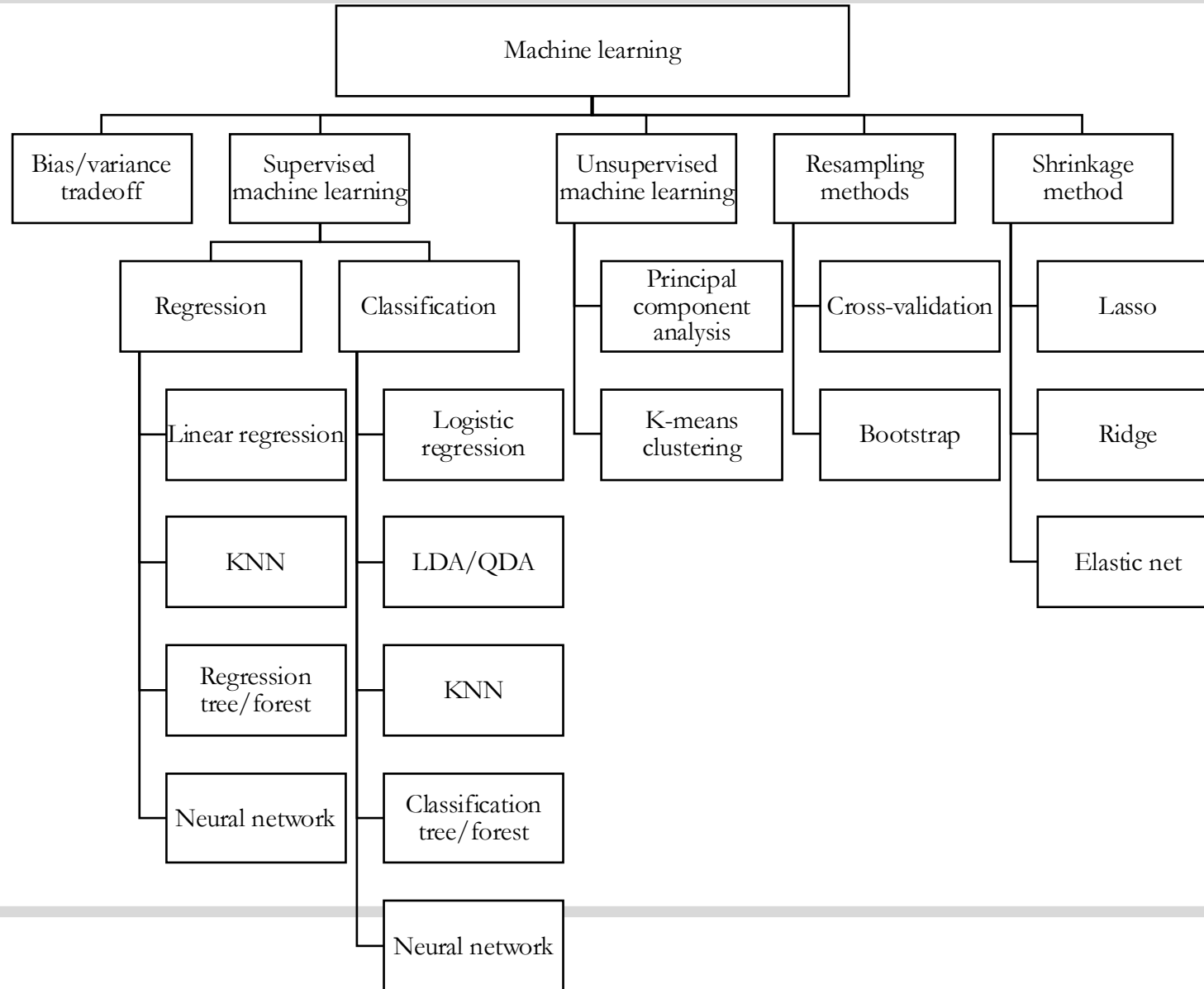


# Midterm

- Cover the material from Lectures 1-18
- Problems are similar to those in homework assignments
- A combination of conceptual and coding questions
- You need to finish it independently
- Open book, open notes. Not allowed to use AI (e.g., ChatGPT, Claude, Copilot, Gemini)
- You cannot talk to anyone about the exam until 4/12 11:59 PM



# This course



# Supervised vs. unsupervised machine learning

- **Supervised machine learning** (main focus)
  - Data:  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ 
    - $x_i$ : predictors
    - $y_i$ : response
  - Task: Fit a model that relates response to predictors
  - E.g., linear regression, logistic regression, KNN, LDA/QDA, tree-based methods, neural networks
  
- **Unsupervised machine learning**
  - Data:  $x_1, x_2, \dots, x_n$
  - Task: Understand the relationships between variables/observations
  - E.g., principal component analysis, K-means clustering



# Supervised machine learning: Regression vs. classification problems

- Suppose we observe  $n$  data points:  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ 
  - $x_i$ : predictors
  - $y_i$ : response
- Supervised machine learning finds a function  $f$  that maps  $X$  to  $Y$
- **Regression problem**
  - Find a function  $f$  that maps  $Y = f(X) + \varepsilon$ , with  $E[\varepsilon] = 0$
  - Example: Predict sales of a product ( $Y$ ) in 200 markets using the expenditure of three media ( $X$ : TV, radio, and newspaper)
- **Classification problem**
  - Estimate  $P(Y|X)$ : conditional distribution of  $Y$  given  $X$
  - Example: Predict whether a customer defaults (binary  $Y$ ) using income, credit card balance, student status, etc.



# Training data, training error, test data, test error

- Training data: the observations,  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ , that we use to estimate  $f$  ( $f$  could be linear, quadratic, etc)
- Training error
  - Regression problem:  $\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}(x_i))^2$
  - Classification problem: classification error  $\frac{1}{n} \sum_{i=1}^n \mathbf{1}(y_i \neq \hat{y}_i)$
- Test data: the data,  $(x'_1, y'_1), (x'_2, y'_2), \dots, (x'_m, y'_m)$ , that are previously unseen and not used to fit  $f$
- Test error
  - Regression problem:  $\text{MSE} = \frac{1}{m} \sum_{i=1}^m (y'_i - \hat{f}(x'_i))^2$
  - Classification problem: classification error  $\frac{1}{m} \sum_{i=1}^m \mathbf{1}(y'_i \neq \hat{y}'_i)$



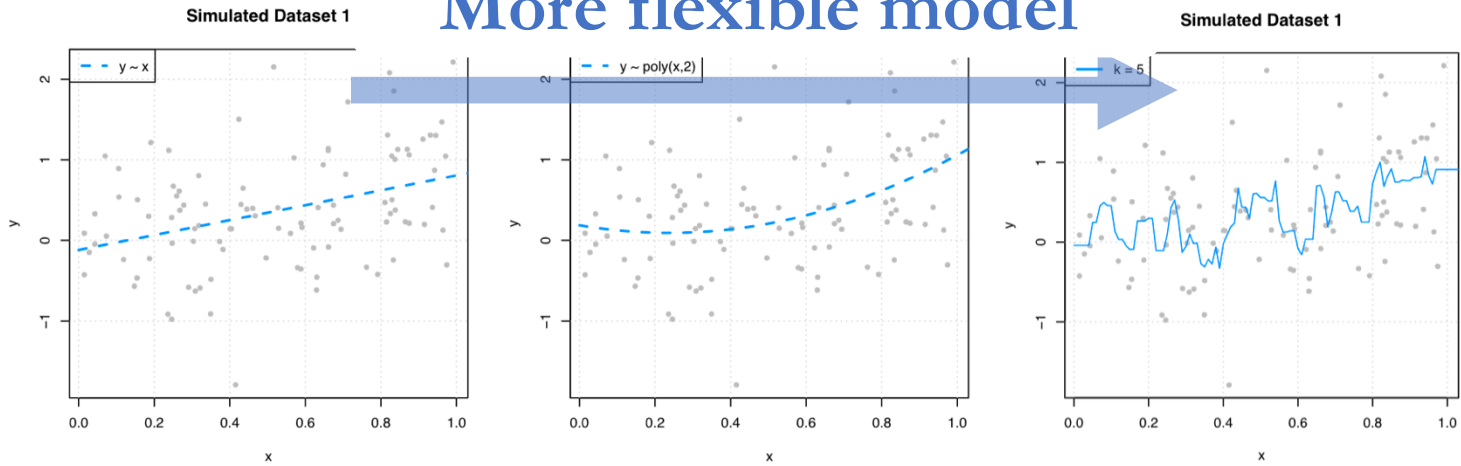
# Our goal and challenge in supervised machine learning

- Our goal in supervised learning is to minimize the (test) **prediction error**
- Regression problem
  - Typically, minimize test Mean Squared Error (MSE)
- Classification problem
  - Typically, minimize test 0-1 loss, Gini index, entropy loss
- *A low training MSE/ classification error does not imply a low test MSE / classification error ...*



# MSE varies with model flexibility

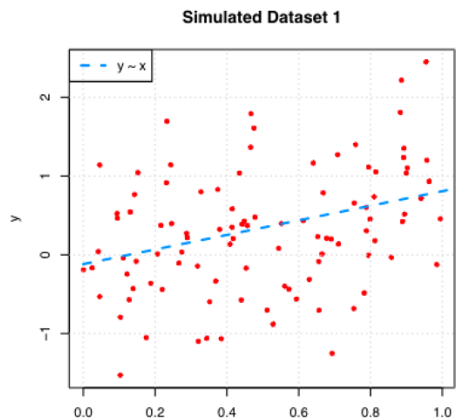
More flexible model



Training MSE = 0.439

Training MSE = 0.425

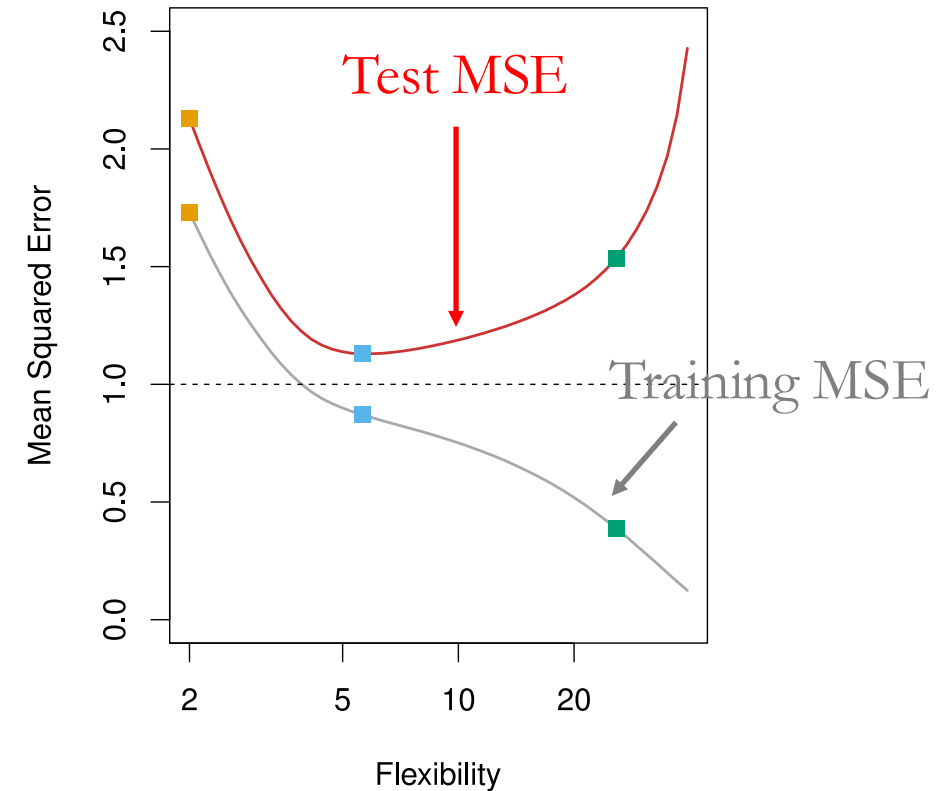
Training MSE = 0.354



Test MSE = 0.533

Test MSE = 0.518

Test MSE = 0.564



# Bias-variance decomposition of MSE

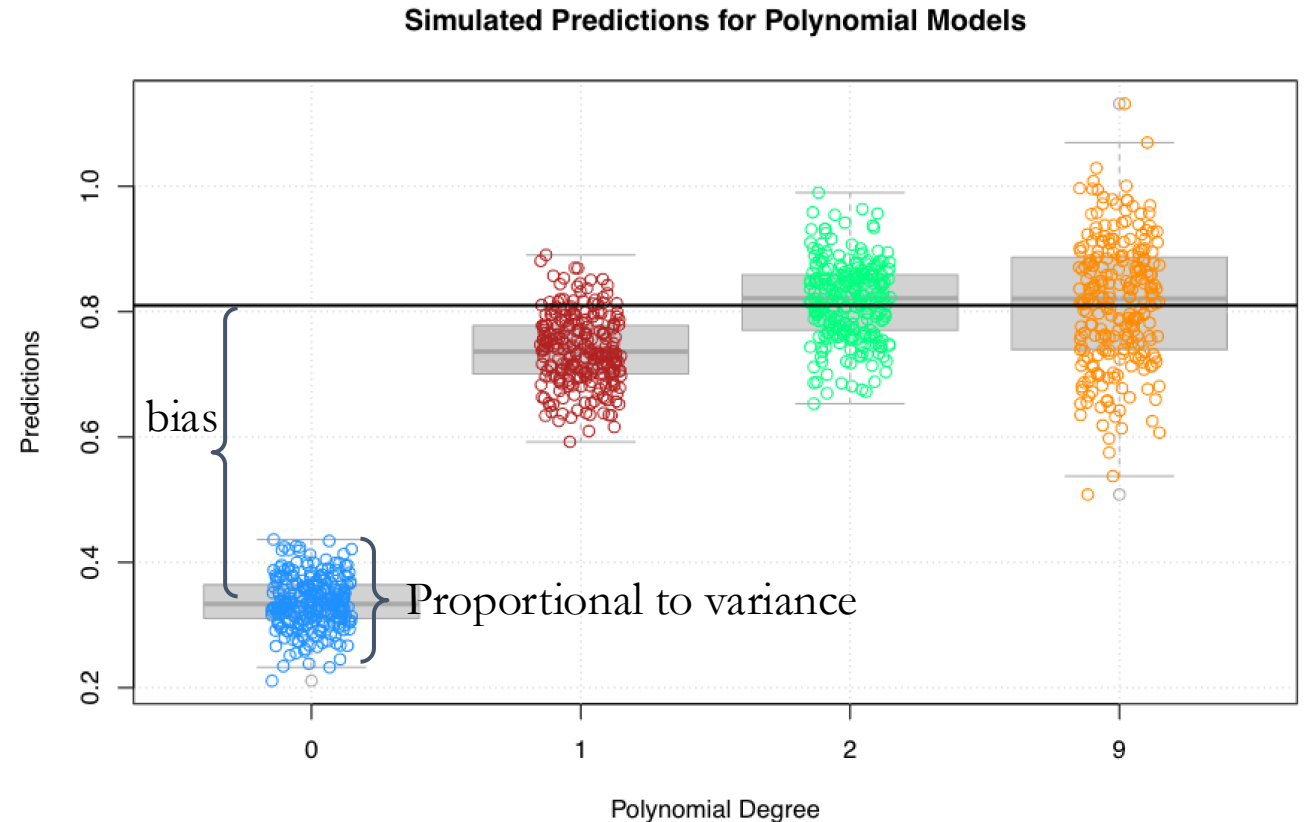
- The MSE at a test point  $x_0$  can be decomposed as

$$\text{MSE}(x_0) = \underbrace{\text{bias}^2(\hat{f}(x_0)) + \text{var}(\hat{f}(x_0))}_{\text{Reducible error}} + \underbrace{V_{Y|X}[Y | X = x_0]}_{\text{Irreducible error}}$$

- $\text{bias}(\hat{f}(x_0)) = f(x_0) - E_{\mathcal{D}}[\hat{f}(x_0)]$ 
  - This measures the deviation of the average prediction  $\hat{f}(x_0)$  from the truth  $f(x_0)$
- $\text{var}(\hat{f}(x_0)) = E_{\mathcal{D}}[(\hat{f}(x_0) - E_{\mathcal{D}}[\hat{f}(x_0)])^2]$ 
  - How much the estimate of  $\hat{f}$  at  $x_0$  changes when we sample new training data
- If  $Y = f(X) + \varepsilon$  with  $E[\varepsilon] = 0$  and  $V[\varepsilon] = \sigma_{\varepsilon}^2$ , then  $V_{Y|X}[Y | X = x_0] = \sigma_{\varepsilon}^2$

# Example of bias-variance decomposition of MSE

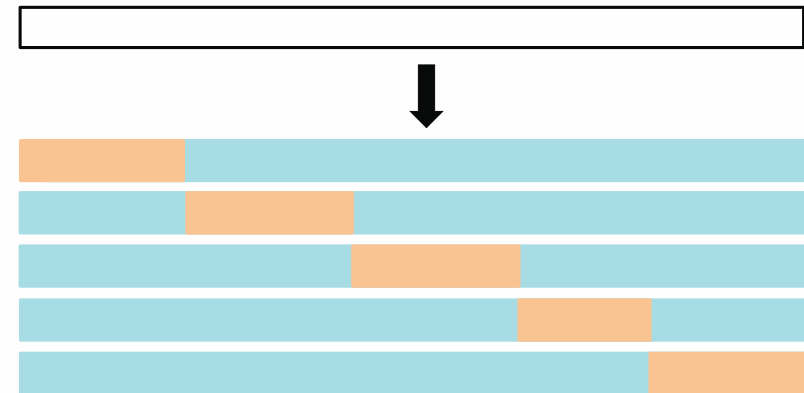
- Suppose we would like to train a model to learn the true regression function  $f(x) = x^2$  ( $x$  is a scalar)
- We use
  - A constant function:  $\hat{f}_0(x) = \hat{\beta}_0$
  - A linear function:  $\hat{f}_1(x) = \hat{\beta}_0 + x \cdot \hat{\beta}_1$
  - A quadratic function:  $\hat{f}_2(x) = \hat{\beta}_0 + x \cdot \hat{\beta}_1 + x^2 \cdot \hat{\beta}_2$
  - A ninth degree polynomial function:  $\hat{f}_9(x) = \hat{\beta}_0 + x \cdot \hat{\beta}_1 + \dots + x^9 \cdot \hat{\beta}_9$



HW 1 Problem 3

# In practice, use data splitting strategy

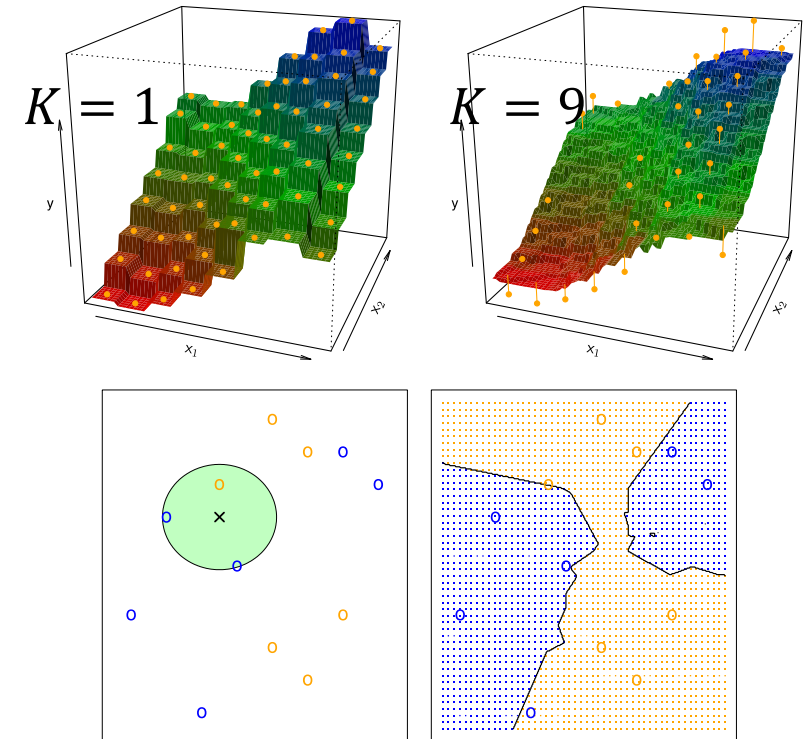
- Split the data into the training and test sets
- Choose parameters by cross-validation on the training data
  - E.g.,  $\lambda$  in lasso/ridge,  $\lambda$  and  $\alpha$  in Elastic net
- Fit various models on the training set using the optimal parameters selected by cross-validation
- Evaluate/select models on the test set
- Cross validation
  - $k$ -fold cross-validation
    1. Split the data into  $k$  subsets or *folds*
    2. For every  $i = 1, \dots, k$ :
      1. train the model on every fold except the  $i$ th fold
      2. compute the test error on the  $i$ th fold
    3. Average the test errors
  - Leave one out cross-validation ( $n$ -fold cross validation)



HW 1 Problem 4

# $K$ -nearest neighbors

- $K$ -nearest neighbors: A simple and well-known nonparametric method
  - Given a value for  $K$  and a prediction point  $x_0$
  - $N_K(x_0)$  represents the set of  $K$  training observations that are closest to  $x_0$
- Regression problem
  - $\hat{f}(x_0) = \frac{1}{K} \sum_{x_i \in N_K(x_0)} y_i$
  - In Python, use `KNeighborsRegressor()` in `sklearn.neighbors`
- Classification problem
  - $\hat{P}(Y = j | X = x_0) = \frac{1}{K} \sum_{x_i \in N_K(x_0)} I(y_i = j)$
  - In Python, use `KNeighborsClassifier()` in `sklearn.neighbors`
- Bias-variance tradeoff for the optimal  $K$ 
  - Large  $K$ , less flexible, large bias, small variance
  - Small  $K$ , more flexible, small bias, large variance



HW 1 Problem 2

# Classification problem: Discriminative vs. generative methods

- Discriminative methods
  - Directly model  $P(Y = k|X = x)$  and classify
  - E.g., logistic regression
  - In Python, use `GLM()` in `statsmodels.api`
- Generative methods
  1. Model the joint probability  $p(x, y)$
  2. Assume some distribution for conditional distribution of  $X$  given  $Y = k$ ,  
 $P(X = x|Y = k)$
  3. Bayes theorem is applied to obtain  $P(Y = k|X = x)$  and classify
    - E.g., linear discriminant analysis (LDA), quadratic discriminant analysis (QDA)
    - In Python, use `LinearDiscriminantAnalysis()` and `QuadraticDiscriminantAnalysis()` in `sklearn.discriminant_analysis`



# LDA and QDA

- To estimate  $P(Y|X)$

1. Estimate  $P(X = x|Y = k)$  and  $P(Y = k)$

- a.  $P(X = x|Y = k)$

- I. LDA: Assume  $P(X = x|Y = k) = N(\mu_k, \Sigma)$

- II. QDA: Assume  $P(X = x|Y = k) = N(\mu_k, \Sigma_k)$

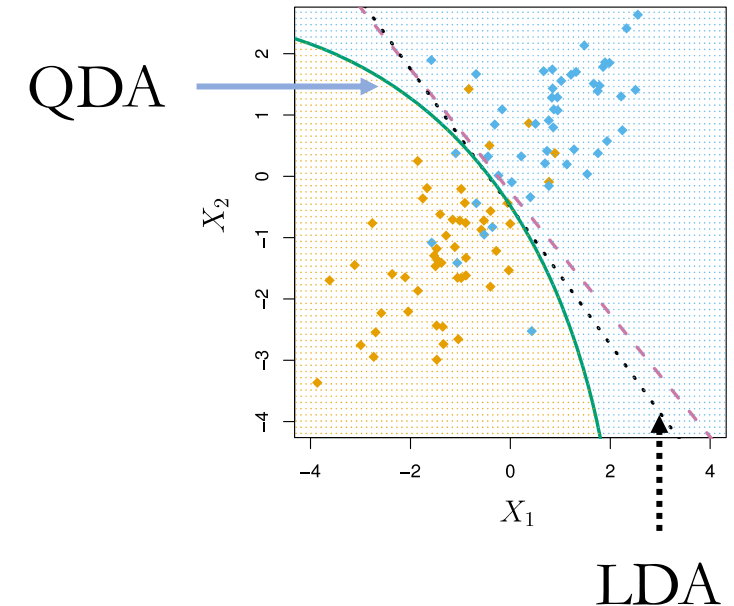
- III. Estimate  $\mu_k$  and  $\Sigma$  (or  $\Sigma_k$ )

- b.  $P(Y = k)$

- a. Estimated the fraction of training samples of class  $k$

2. Apply Bayesian rule  $P(Y = k|X = x) = \frac{P(X=x|Y=k)P(Y=k)}{\sum_j P(X=x|Y=j)P(Y=j)}$

See the notebook (lecture 6 - classification methods.ipynb)



HW 1 Problem 4

# Classification problem: Bayes classifier

- Bayes classifier (for both discriminative vs. generative methods)
  - $\hat{y}_i = \operatorname{argmax}_j P(Y = j | X = x_i)$
  - Assign unit  $i$  the class with largest probability



# Regression problem

- Suppose a linear model  $y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \varepsilon_i$
- If  $p$  is small compared to  $n$ ,
  - We can estimate  $\beta_0, \dots, \beta_p$  by linear regression, that minimizes the RSS
    - $\text{RSS} = \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \sum_{i=1}^n (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{i1} - \hat{\beta}_2 x_{i2} - \cdots - \hat{\beta}_p x_{ip})^2$
- If  $p$  is large compared to  $n$ , use model regularization methods

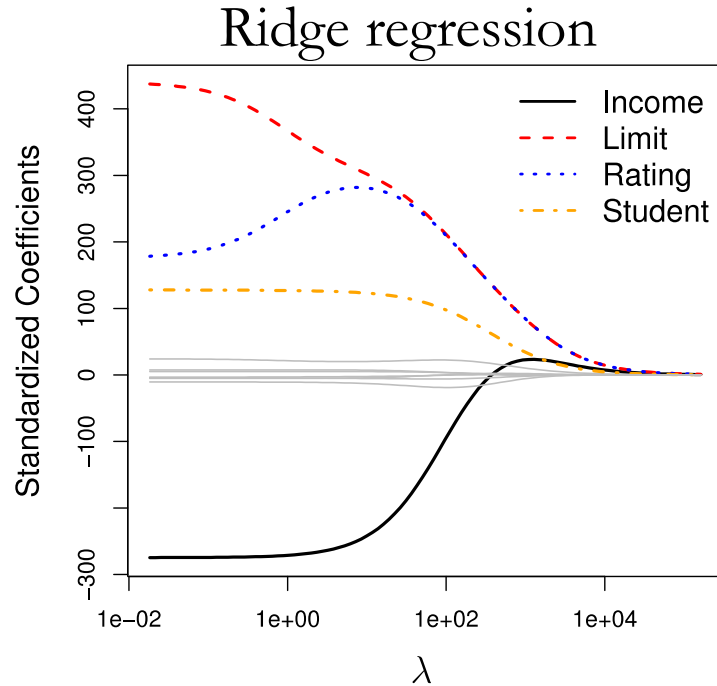
# Shrinkage methods

- Linear regression minimizes RSS
  - $\text{RSS} = \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \sum_{i=1}^n (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{i1} - \hat{\beta}_2 x_{i2} - \dots - \hat{\beta}_p x_{ip})^2$
- Ridge regression minimizes  $\text{RSS} + \lambda \sum_{j=1}^p \beta_j^2$ 
  - $\lambda \sum_{j=1}^p \beta_j^2$ : Shrinkage penalty, small if  $\beta_1, \dots, \beta_p$  are close to zero
  - In Python, use `ElasticNet()` with `l1_ratio=0` in `sklearn.linear_model`
- The lasso minimizes  $\text{RSS} + \lambda \sum_{j=1}^p |\beta_j|$ 
  - $\lambda \sum_{j=1}^p |\beta_j|$ : Shrinkage penalty, small if  $\beta_1, \dots, \beta_p$  are close to zero
  - In Python, use `ElasticNet()` with `l1_ratio=1` in `sklearn.linear_model`
- Elastic net minimizes  $\text{RSS} + \lambda \left( (1 - \alpha) \cdot \sum_{j=1}^p \beta_j^2 / 2 + \alpha \cdot \sum_{j=1}^p |\beta_j| \right)$ 
  - $\lambda \left( (1 - \alpha) \cdot \sum_{j=1}^p \beta_j^2 / 2 + \alpha \cdot \sum_{j=1}^p |\beta_j| \right)$ : Shrinkage penalty, small if  $\beta_1, \dots, \beta_p$  are close to zero
  - In Python, use `ElasticNet()` in `sklearn.linear_model`

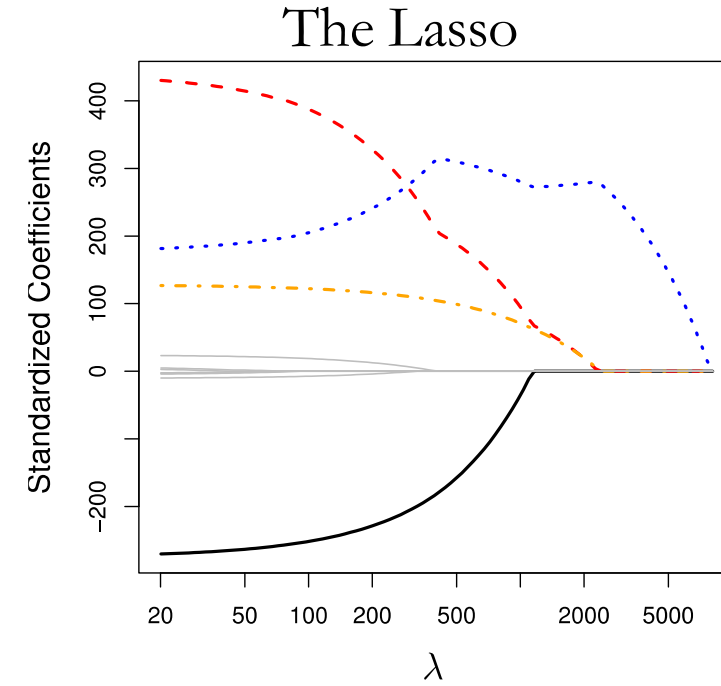


# Coefficients of Ridge and the Lasso

- Predict default in the Credit dataset



A lot of small coefficients throughout the regularization path



Shrink coefficients to zero, perform variable selection

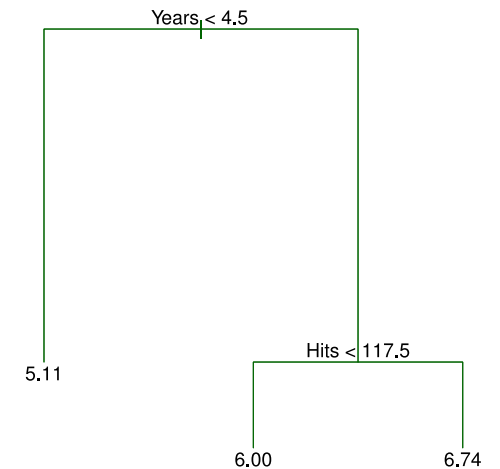
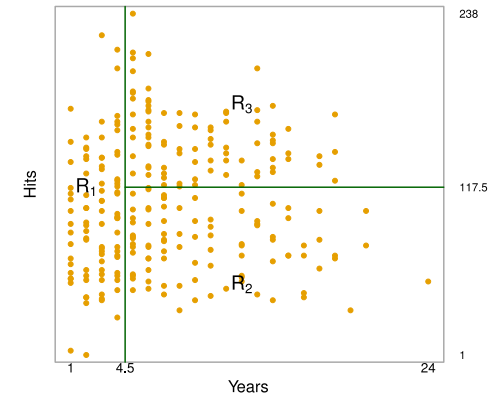
HW 2 Problem 3

# Bootstrap

- Resample the data by drawing  $n$  samples *with replacement* from the actual observations
- Can be used to calculate the standard errors of mean, quantile, regression coefficient, prediction at a test point...
- See the notebook (lecture 8 – bootstrap.ipynb)

# Decision tree

1. Partition the feature space into  $J$  **distinct and non-overlapping** regions,  $R_1, R_2, \dots, R_J$ 
  - Regression tree: Based on MSE
  - Classification tree: Based on Gini index or entropy
2. Make the **same** prediction for every observation in region  $R_j$ 
  - Regression tree: **Mean** of the training observations in  $R_j$
  - Classification tree: **Mode** of the training observations in  $R_j$
  - In Python, use `DecisionTreeClassifier()` in `sklearn.tree` for classification tree; use `DecisionTreeRegressor()` in `sklearn.tree` for regression tree
3. Prune a large tree from leaves to the root to control overfitting
  - In Python, use `cost_complexity_pruning_path()`



See the notebook (lecture 12 - decision tree.ipynb)

# Bagging and random forest

- We fit a decision tree to different Bootstrap samples
- When growing the tree
  - Bagging: Use all predictors
  - Random forest: use  $m < p$  predictors
    - Lead to very different (or “uncorrelated”) trees from each sample
- Finally, average the prediction of each tree
- Pro: reduce variance of decision trees
- Generalization of KNN
- In Python, use `RandomForestRegressor()` in `sklearn.ensemble`

See the notebook (lecture 13 - bagging.ipynb, lecture 13 - bagging classification.ipynb, lecture 14 - random forest and boosting.ipynb)

# Gradient boosting

- **Boosted trees**

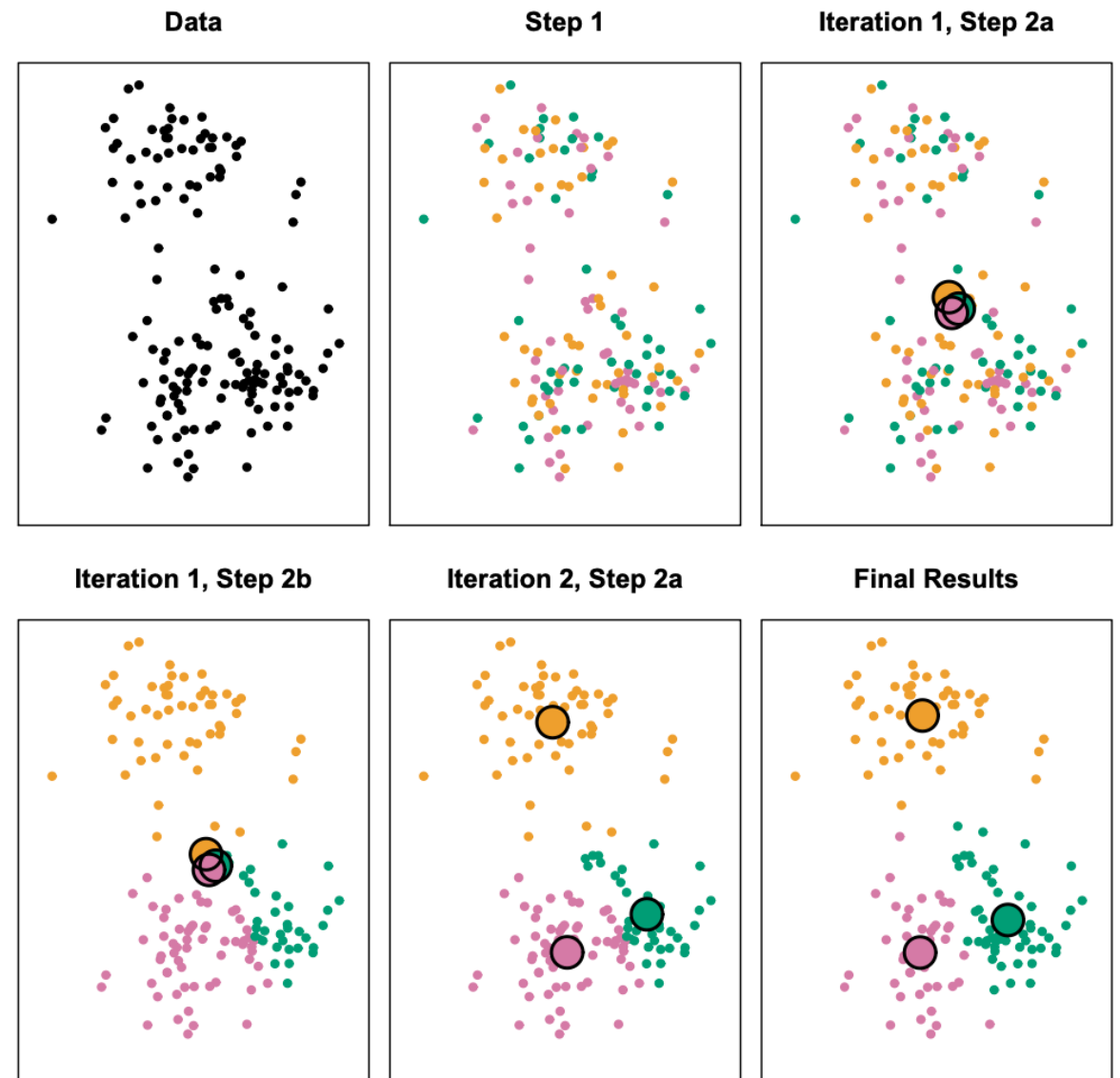
- Trees are grown sequentially using the information left from previously grown trees
- Each tree is fit on a **modified version** of the original data
- Idea is similar to **partial least squares**
  
- In Python, use `GradientBoostingRegressor()` for regression problems and `GradientBoostingClassifier()` for classification problems in `sklearn.ensemble`

See the notebook (lecture 14 - random forest and boosting.ipynb)



# K-means clustering

- *Randomly assign* a number, from 1 to  $K$ , to each observation, i.e., *initial cluster assignment*
- *Iterate* until the cluster assignments stop changing
  - For each cluster, *compute the cluster centroid*, i.e., vector of the  $p$  feature means for observations in the  $k$ th cluster
  - *Assign each observation* to the *cluster* whose *centroid is closest*



# Neural network

- Neuron / hidden unit
  - $A_k = h_k(X) = g\left(w_{k0} + \sum_{j=1}^p w_{kj}X_j\right)$
  - $g$  is a **nonlinear activation function** (e.g., ReLU, sigmoid)
- Outcome prediction
  - $f(X) = \beta_0 + \sum_{k=1}^K \beta_k h_k(X)$
- Loss function to fit a neural network
  - Regression: MSE; Classification: Cross-entropy
- (Stochastic) gradient descent to update parameters  $w$  and  $\beta$

