QTM 347 Machine Learning

Lecture 19: Midterm review

Ruoxuan Xiong

Based on Lectures 1-18

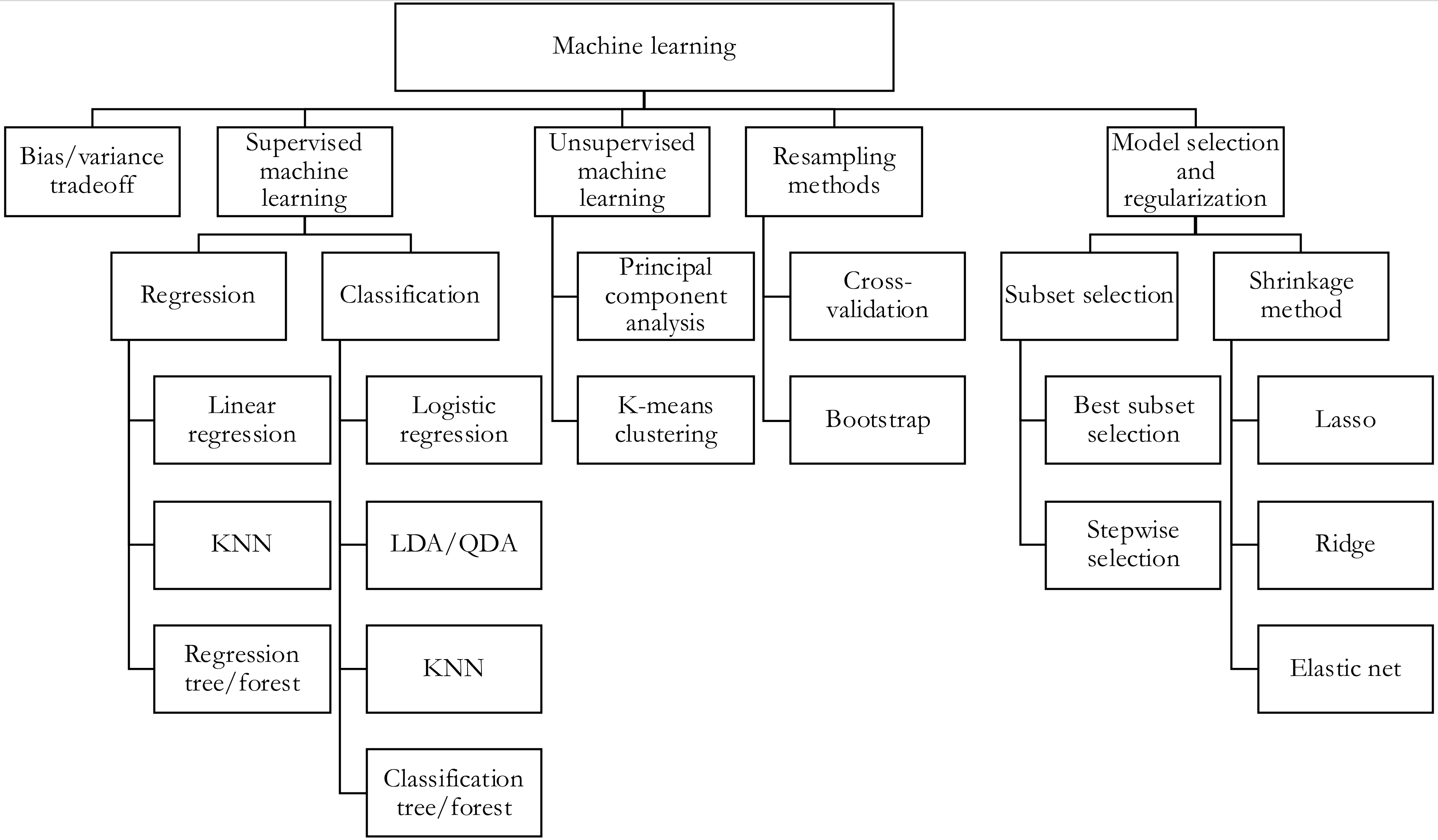
Announcements

- The midterm will be available from 4/9 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 chatGPT
- You cannot talk to anyone about the exam until 4/12 11:59 PM

This course


Machine learning
Bias/variance tradeoff
Supervised machine learning
Unsupervised machine learning
Resampling methods
Model selection and regularization
Regression
Classification
Principal component analysis
K-means clustering
Cross-validation
Bootstrap
Subset selection
Shrinkage method
Linear regression
KNN
Regression tree/forest
Logistic regression
LDA/QDA
KNN
Classification tree/forest
Best subset selection
Stepwise selection
Lasso
Ridge
Elastic net

Supervised vs. unsupervised machine learning

- Supervised machine learning (main focus)
  - Data: $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$
    - 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

- Unsupervised machine learning
  - Data: $x_1, x_2, \cdots, x_n$
  - Task: Understand the relationships between variables/observations
  - E.g., principal component analysis

Supervised machine learning: Regression vs. classification problems

- Suppose we observe n data points: $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$
  - 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, $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$, that we use estimate f (f could be linear, quadratic, etc)

- Training error
  - Regression problem: MSE $= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$
  - Classification problem: classification error $\frac{1}{n}\sum_{i=1}^{n} 1(y_i \neq \hat{y}_i)$

- Test data: the data, $(x_1', y_1'), (x_2', y_2'), \ldots, (x_m', y_m')$, that are previous unseen and not used to fit f

- Test error
  - Regression problem: MSE $= \frac{1}{m}\sum_{i=1}^{m}(y_i' - \hat{f}(x_i'))^2$
  - Classification problem: classification error $\frac{1}{m}\sum_{i=1}^{m} 1(y_i' \neq \hat{y}_i')$

Our goal and challenge in supervised machine learning

- Our goal in supervised learning is to minimize the (test) prediction error

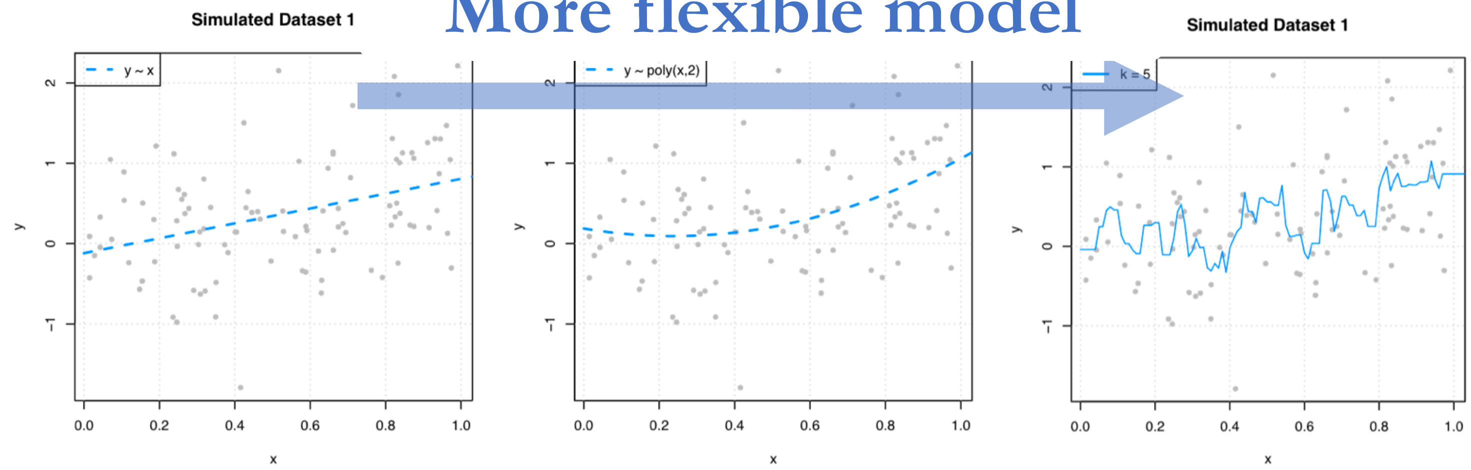
- Regression problem
  - Typically, minimize test Mean Squared Error (MSE)

- Classification problem
  - Typically, minimize test 0-1 loss, Gini index, entropy loss

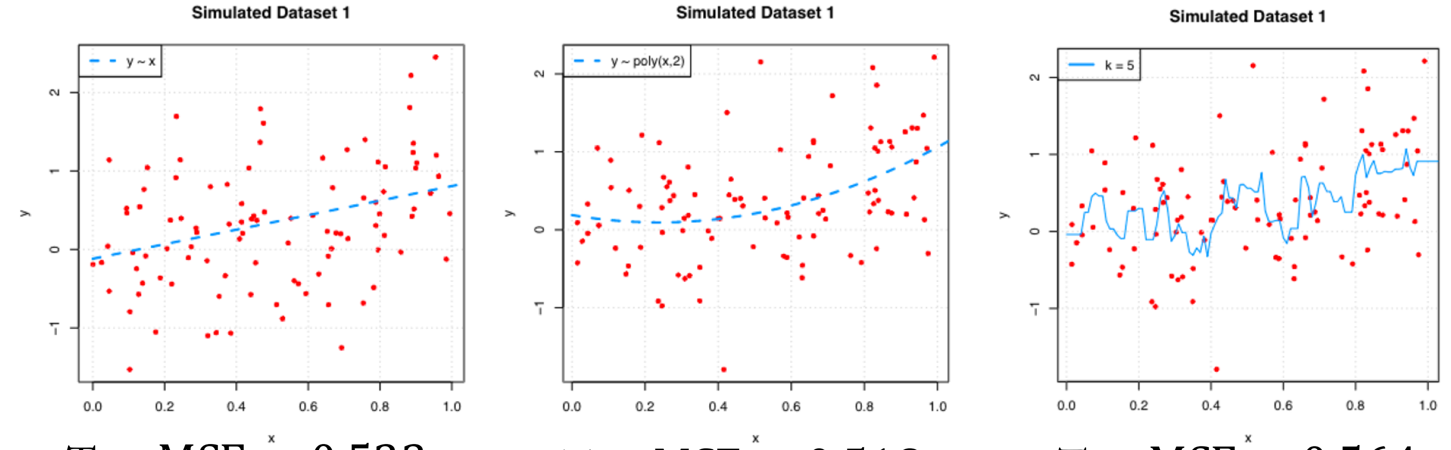
- *A low training MSE/classification error does not imply a low test MSE /classification error …*

MSE varies with model flexibility

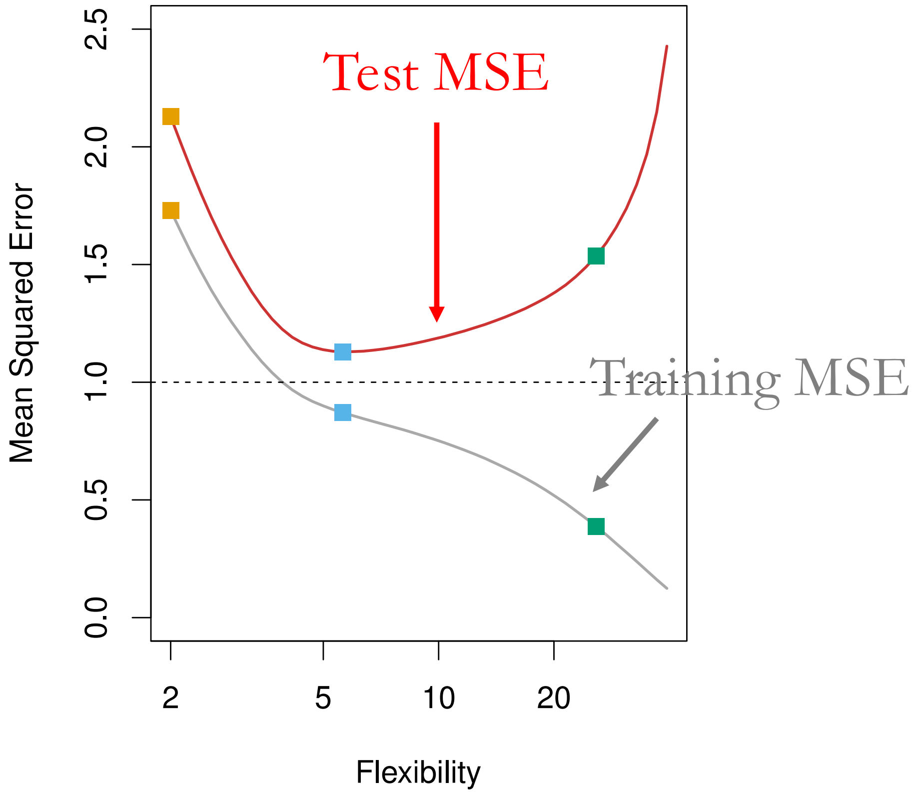
More flexible model



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

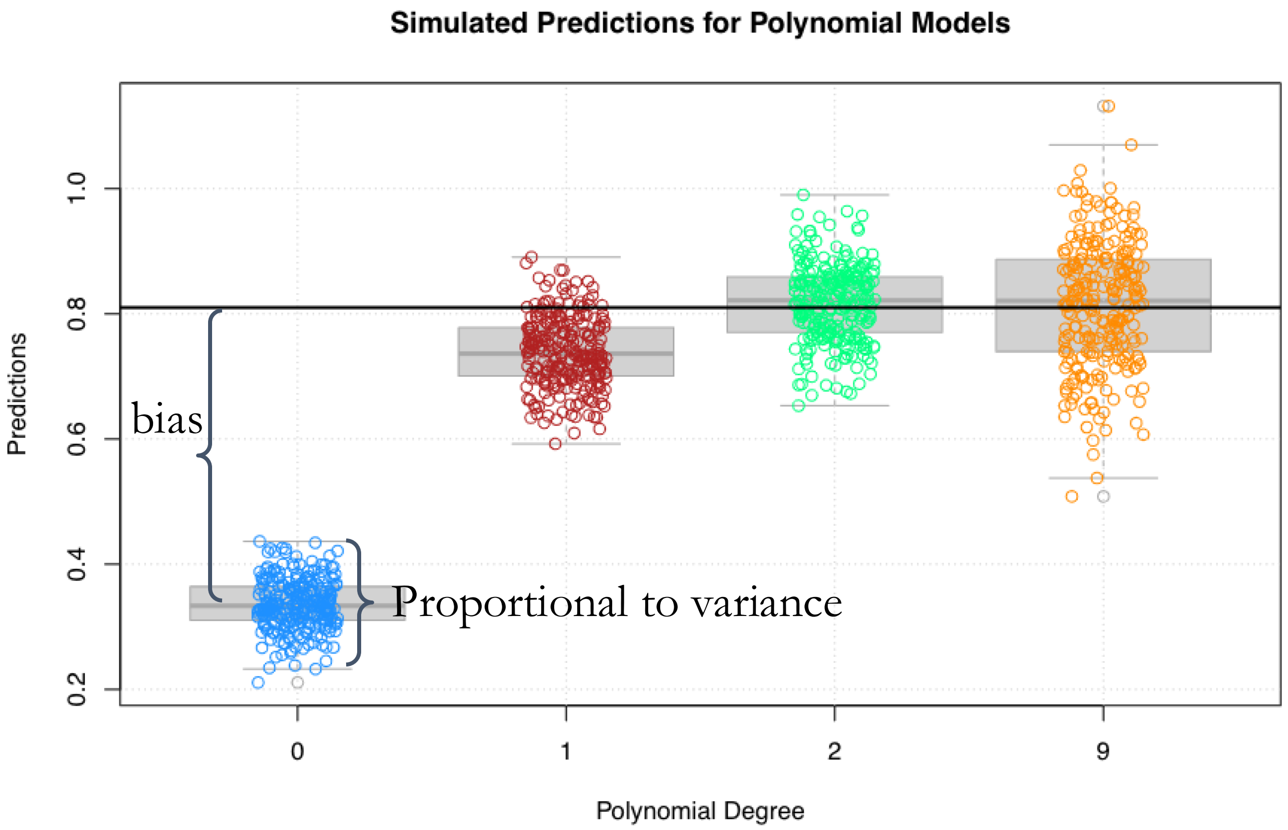
$$\mathrm{MSE}(x_0) = \underbrace{\mathrm{bias}^2\left(\hat{f}(x_0)\right) + \mathrm{var}\left(\hat{f}(x_0)\right)}_{\text{Reducible error}} + \underbrace{V_{Y|X}[Y \mid X = x_0]}_{\text{Irreducible error}}$$

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

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)

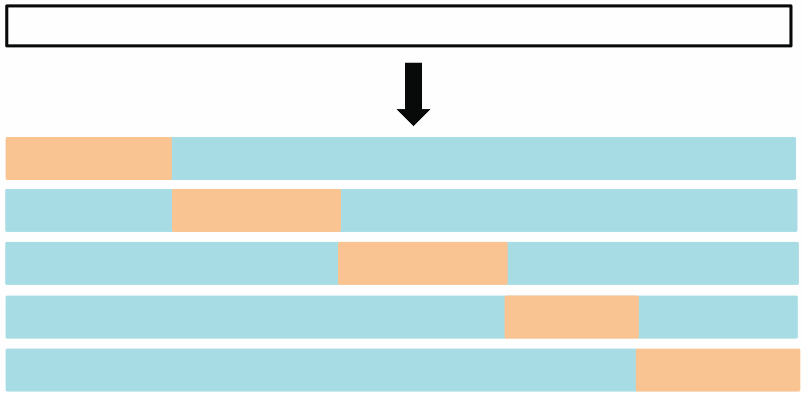
- We use
  - A constant function: $\hat{f}_0(x) = \hat{\beta}_0$
  - A linear function: $\hat{f}_1(x) = \hat{\beta}_0 + x \cdot \hat{\beta}_1$
  - A quadratic function: $\hat{f}_2(x) = \hat{\beta}_0 + x \cdot \hat{\beta}_1 + x^2 \cdot \hat{\beta}_2$
  - A ninth degree polynomial function: $\hat{f}_9(x) = \hat{\beta}_0 + x \cdot \hat{\beta}_1 + \cdots + x^9 \cdot \hat{\beta}_9$



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., λ in lasso/ridge, λ and α 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 $i = 1, \cdots, k$:
       1. train the model on every fold except the ith fold
       2. compute the test error on the ith fold
    3. Average the test errors
  - Leave one out cross-validation (n-fold cross validation)

HW 1 Problem 4

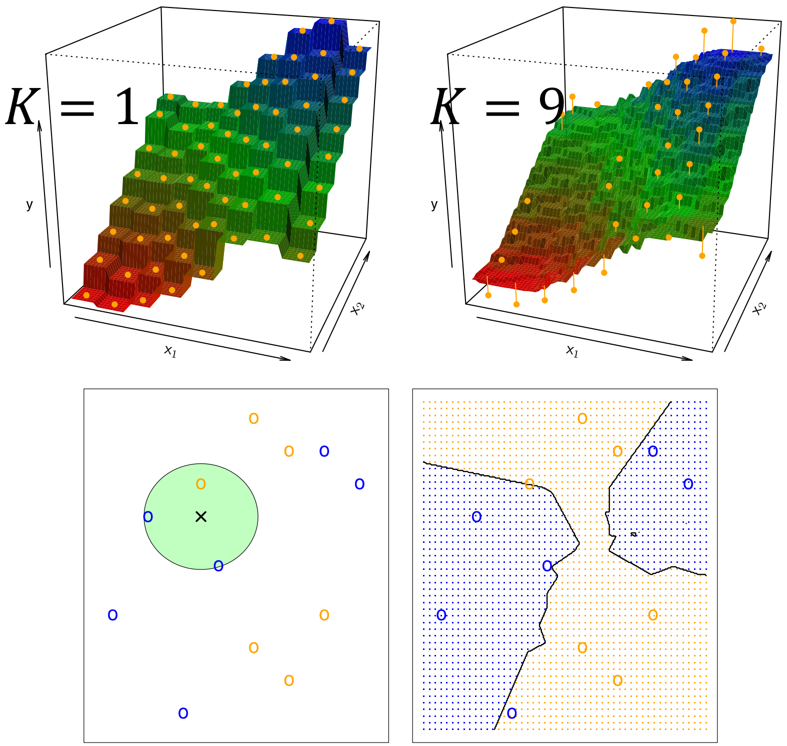
K-nearest neighbors

- 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

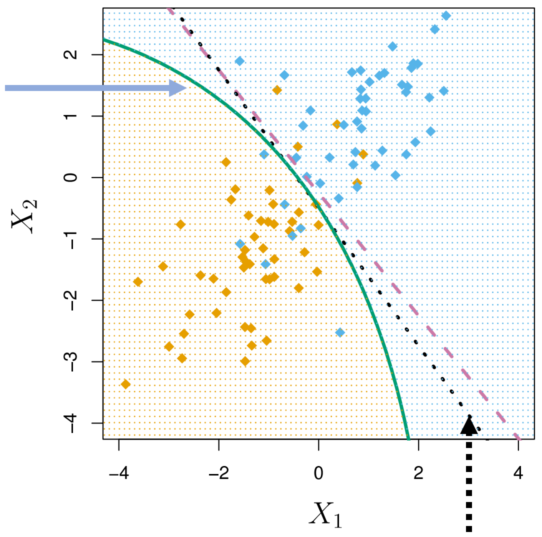
LDA and QDA

- 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 μ_k and Σ (or Σ_k)

     b. $P(Y=k)$
        a. Estimated the fraction of training samples of class k

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

QDA



LDA

HW 1 Problem 4

Classification problem: Bayes classifier

- Bayes classifier (for both discriminative vs. generative methods)
  - $\hat{y}_i = \text{argmax}_j P(Y = j | X = x_i)$
  - 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 $\beta_0, \cdots, \beta_p$ 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 selection or regularization methods

Model selection methods

- Select a small subset of predictors (whether intercept is included)
  - Best subset selection
    - Akaike Information Criterion (AIC or C_p): $C_p = \frac{1}{n}(\text{RSS} + 2k\hat{\sigma}^2)$
      - $\hat{\sigma}^2$ is an estimate of the irreducible error, and k is the number of predictors in the model
    - Bayesian Information Criterion (BIC): $\text{BIC} = \frac{1}{n}(\text{RSS} + \log(n)\, k\hat{\sigma}^2)$
    - Adjusted R^2
  - Stepwise selection
    - Forward selection: Start with a model with no predictors, add predictors to the model one-at-a-time
    - Backward selection: Start with a model with p predictors, remove the least useful predictor one-at-a-time

  - See the notebook (lecture 11 – subset selection.ipynb)

HW 2 Problem 3

Shrinkage methods

- Linear regression minimizes RSS
  - $\text{RSS} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{i1} - \hat{\beta}_2 x_{i2} - \cdots - \hat{\beta}_p x_{ip})^2$

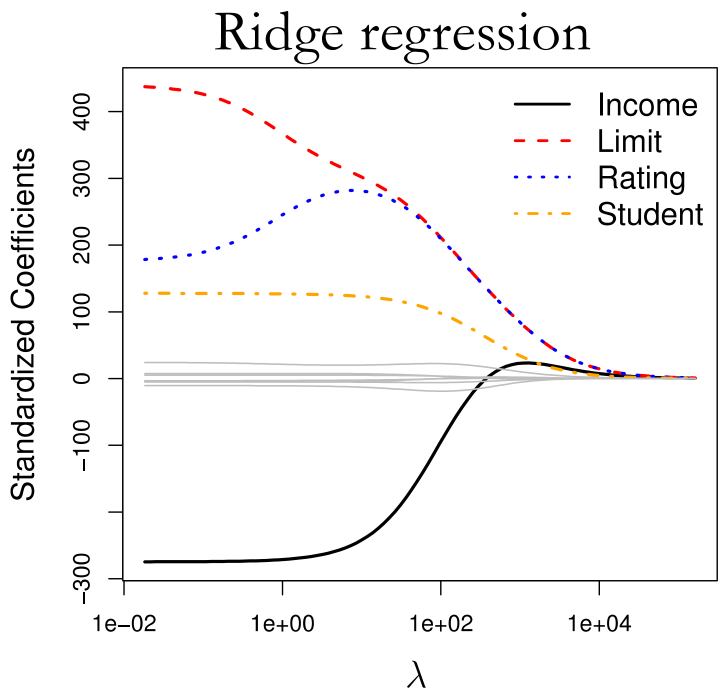
- Ridge regression minimizes RSS+ $\lambda \sum_{j=1}^{p} \beta_j^2$
  - $\lambda \sum_{j=1}^{p} \beta_j^2$: Shrinkage penalty, small if $\beta_1, \cdots, \beta_p$ are close to zero
  - In Python, use ElasticNet() with l1_ratio=0 in sklearn.linear_model

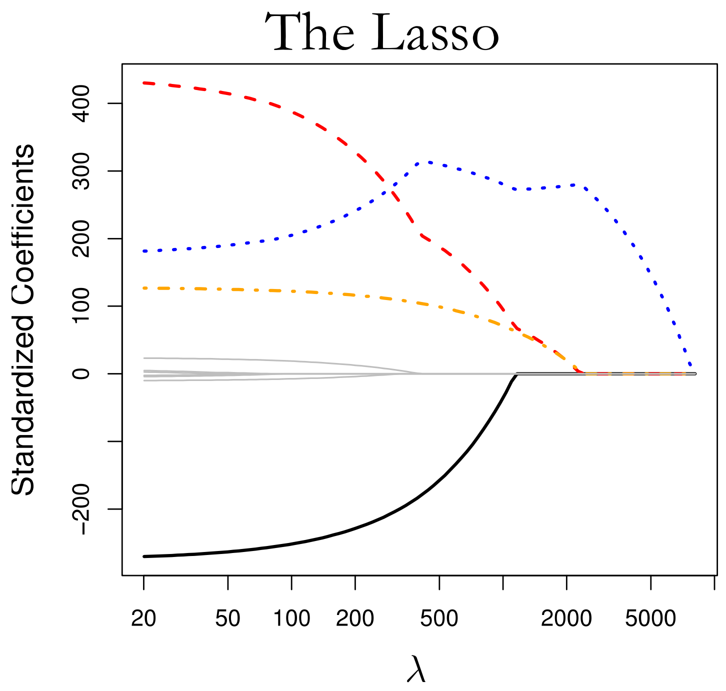
- The lasso minimizes RSS+$\lambda \sum_{j=1}^{p} |\beta_j|$
  - $\lambda \sum_{j=1}^{p} |\beta_j|$: Shrinkage penalty, small if $\beta_1, \cdots, \beta_p$ are close to zero
  - In Python, use ElasticNet() with l1_ratio=1 in sklearn.linear_model

- Elastic net minimizes 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 $\beta_1, \cdots, \beta_p$ 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

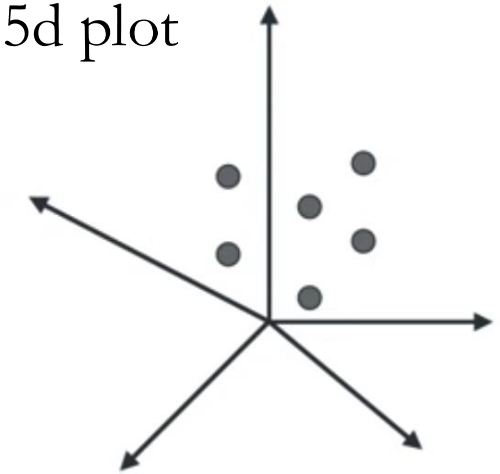
Shrink coefficients to zero, perform variable selection

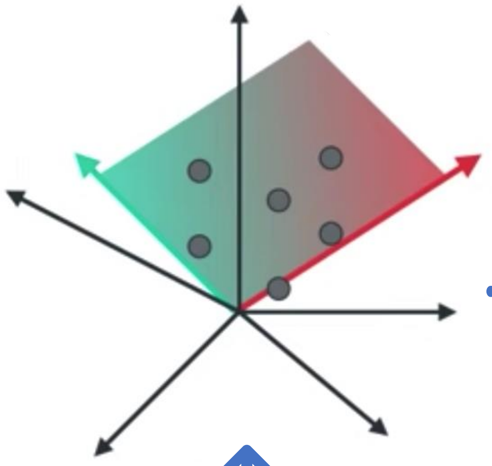
HW 2 Problem 4

Principal component analysis

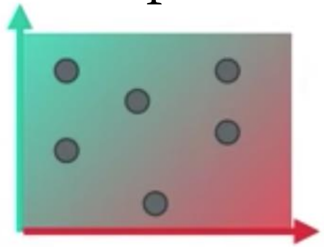
X_1	X_2	X_3	X_4	X_5
*	*	*	*	*
*	*	*	*	*
*	*	*	*	*
*	*	*	*	*
*	*	*	*	*
*	*	*	*	*
*	*	*	*	*
*	*	*	*	*
*	*	*	*	*
*	*	*	*	*
*	*	*	*	*
*	*	*	*	*

5d plot





2d plot



Small table

Z_1	Z_2
*	*
*	*
*	*
*	*
*	*
*	*
*	*
*	*
*	*
*	*
*	*
*	*

Covariance/ Correlation matrix

*****	*****	*****	*****	*****
*	*	*	*	*
*	*	*	*	*
*	*	*	*	*
*	*	*	*	*

Eigenvector	Eigenvalue	
V_1	λ_1	Big
V_2	λ_2	
V_3	λ_3	
V_4	λ_4	
V_5	λ_5	Small

See the notebook (lecture 17 – pca.ipynb)

HW 3 Problem 3

Bootstrap

- Resample the data by drawing n samples ***with replacement*** from the actual observations

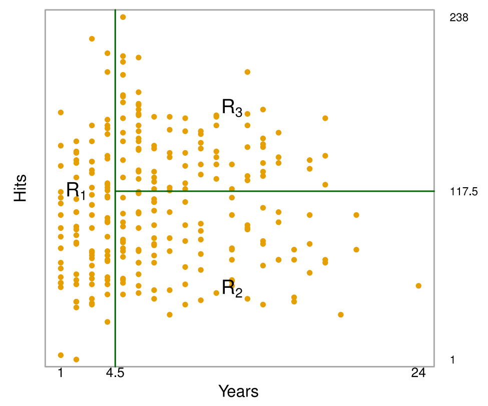
- Can be used to calculate the standard errors of mean, quantile, regression coefficient, prediction at a test point…

- See the notebook (lecture 7 – cross-validation and bootstrap.ipynb)

HW 2 Problems 1 and 2

Decision tree

1. Partition the feature space into J **distinct and non-overlapping** regions, $R_1, R_2, \cdots, R_J$
   - 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 16 - decision tree, random forest and boosting.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

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