STA314 Summer 2023

Week 2: Cross-validation, Model Regularization & Principal Component Analysis

Ziang Zhang

Department of Statistics, University of Toronto

Overview

Moving beyond training and testing..

In last lecture, we said a dataset could be separated into two parts: training and testing.

Training data should only be used to fit the model, and testing data should only be used to assess the model.

But what if we need to do both *simultaneously*?

Example of KNN

Suppose we want to use KNN regression to produce a prediction $\hat{y}$, while knowing its predictive accuracy.

1. Given a choice of $K = k$, we could use the training data to fit a model with prediction $\hat{y}^{(k)}$.
2. There are many choices of K, so we eventually have $\{\hat{y}^{(k)}\}_{k=1}^{n}$, all obtained from the training data.
3. But we could only produce one prediction $\hat{y}$, which one of $\{\hat{y}^{(k)}\}_{k=1}^{n}$ should it be?
4. If we choose the $\{\hat{y}^{(k)}\}_{k=1}^{n}$ that minimizes the training MSE, then it almost always leads to overfitting...
5. If we choose the $\{\hat{y}^{(k)}\}_{k=1}^{n}$ that minimizes the testing MSE, the selected $\hat{y}$ is not independent of the testing data.

Hyperparameter Tuning

The integer K in KNN method is an example of *hyperparameter* or *tuning parameter*.

Conditional on the hyperparameter, the model can be fitted with the training data.

In other words, the hyperparameter controls the structure of the model to be trained.

Unlike other parameters in the model, the hyperparameter is normally not directly estimated using the training data. Instead, we tune it on another dataset, *the validation data*.

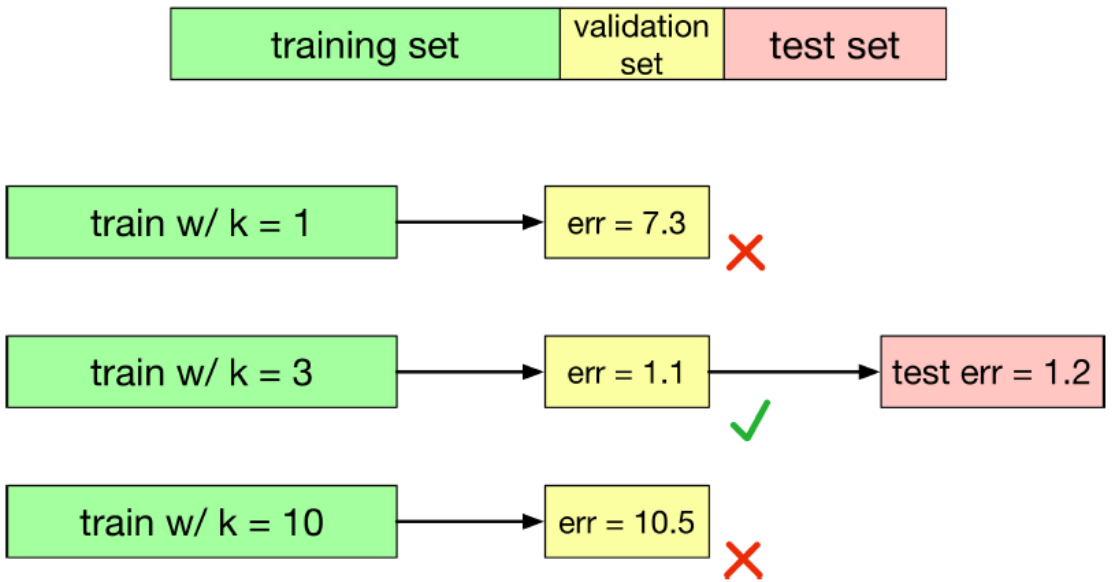


Figure: An example of hyperparameter-tuning with validation set. Picture taken from Prof Maddison's slides.

K-Fold Cross-Validation

However, separating one dataset into three sets is quite wasteful for training purposes...

Particularly for a complex model that has a large number of parameters, the training data needs to be large as well (Example: linear regression with $p \approx n$).

Reducing the size of the validation set seems like a remedy, but makes the model selection unstable...

But we can repeatedly do training/validating! This is called *K-fold Cross-validation*.

K-fold Cross-Validation(CV)

After reserving a set of data as the testing set, the K-fold CV proceeds as the following:

1. Randomly partition the remaining n observations into K *non-overlapping* groups.
2. Reserve the first group as the validation set, and fit the model using the remaining $K-1$ groups.
3. Record the testing performance (i.e. $\text{MSE}^{(1)}$) of the fitted model, using the reserved group.
4. Repeat procedures 2-3, but with the reserved validation set being another group in each iteration.
5. Compute the average test performance (i.e. $\sum_i^K \text{MSE}^{(i)}/K$), and select model based on that.

Question: Why can't we partition the observations into overlapping groups?

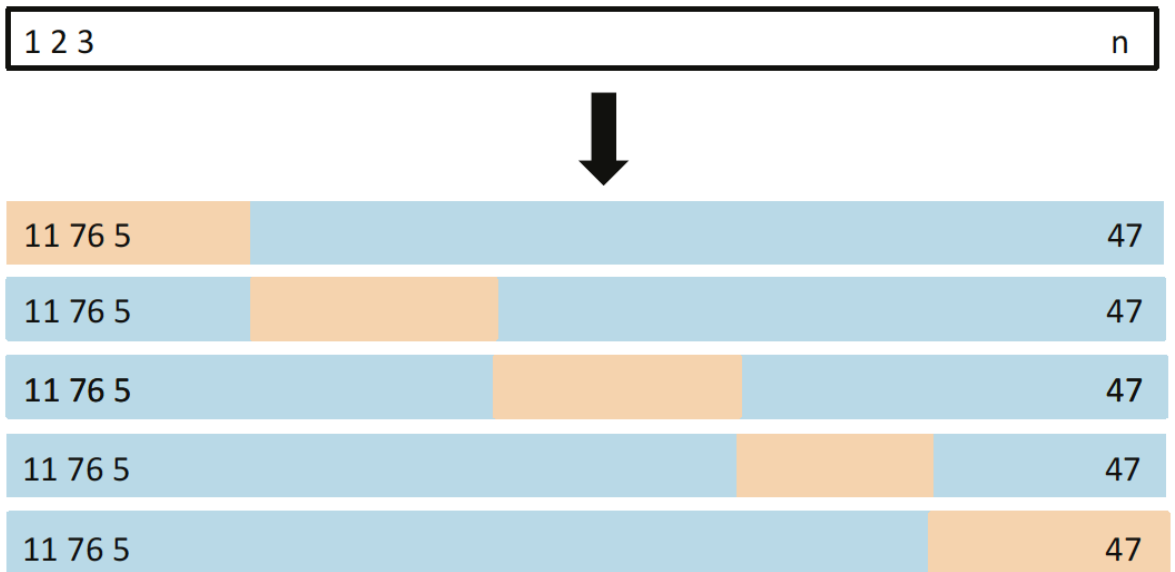


Figure: An example of the 5-fold CV procedure. Source: James et al., 2021.

Special Case: LOOCV

For K-fold CV with $K = n$, it becomes **Leave-one-out cross-validation (LOOCV)**

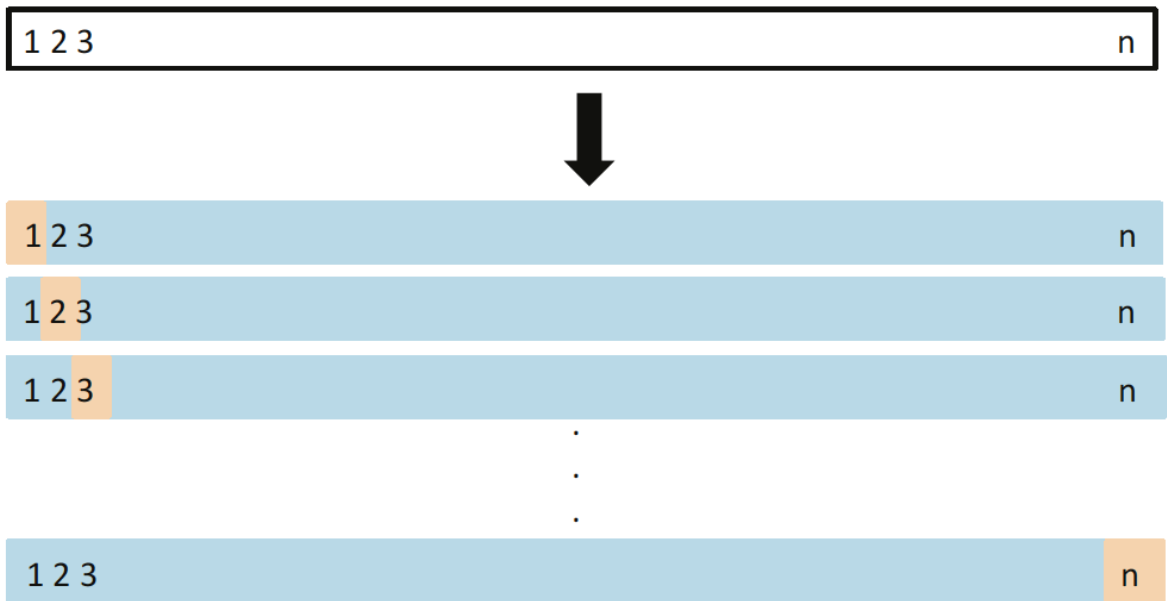


Figure: An example of the LOOCV procedure. Source: James et al., 2021.

LOOCV

- At first, this approach seems computationally intensive...
  - The model needs to be fitted n times.
  - Each model is fitted on a large dataset $(n-1)$.
- However, for certain models, LOOCV has a nice analytical formula for the final average testing MSE.

For linear regression:

$$\mathrm{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{1-h_i}\right)^2$$

- $\hat{y}_i$ is from the model fitted using *the full data*.
- h_i is the leverage of each datapoint.
- In other words, only one model needs to be fitted.

Why Regularization?

Problems with linear regression:

Recall $\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\boldsymbol{y}$ obtained from minimizing

$$\text{RSS} = \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 = (\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta})$$

(or equivalently, negative log-likelihood).

- Sometimes it requires a large number of features p to capture the variation.
- When p is large relative to n, $\hat{\boldsymbol{\beta}}$ is unstable, and likely to overfit the data.
- If $p > n$, $(\mathbf{X}^T\mathbf{X})^{-1}$ is not well-defined...

Model Selection

For linear regression, the most natural approach is:

Algorithm 1 Best Subset Selection

Require: Dataset $\mathbf{X}$ with p predictors, response variable $\mathbf{y}$.

Let $\mathcal{M}_0$ denote then null model with no predictor.

1: **for** $k = 1$ to p **do**
2: Fit all $\binom{p}{k}$ models that contain exactly k predictors.
3: For each model, record its training performance such as the training RSS, log-likelihood or R^2.
4: Pick the best among these $\binom{p}{k}$ models based on training performance, call it $\mathcal{M}_k$.
5: **end for**
6: Select the best model from $\mathcal{M}_0, ..., \mathcal{M}_p$ based on testing performance approximated by the CV approach.

However, for p predictors, the total number of possible models is 2^p.
When $p = 10$, there are 1024 possible models.

- For most problems, best subset selection is computationally infeasible.
- Stepwise selection method (Forward, Backward) will be a faster alternative.

Algorithm 2 Forward Selection

0: Let $\mathcal{M}_0$ denote the null model with no predictor.
1: **for** $k = 0$ to $p - 1$ **do**
2: Augment $\mathcal{M}_k$ with each of the remaining $p - k$ predictors.
3: For each model, record its training performance.
4: Pick the best among these $p - k$ models based on training performance, call it $\mathcal{M}_{k+1}$.
5: **end for**
6: Select the best model from $\mathcal{M}_0, ..., \mathcal{M}_p$ based on testing performance approximated by the CV approach.

Question: What will be the total number of models that we have to fit now?

Stepwise Selection

Algorithm 3 Backward Selection

0: Let $\mathcal{M}_p$ denote then full model with all predictors.
1: **for** $k = p$ to 1 **do**
2: Consider all k models that contain all but one of the predictors in $\mathcal{M}_k$, for a total of $k - 1$ predictors.
3: For each model, record its training performance.
4: Pick the best among these k models based on training performance, call it $\mathcal{M}_{k-1}$.
5: **end for**
6: Select the best model from $\mathcal{M}_0, ..., \mathcal{M}_p$ based on testing performance approximated by the CV approach.

Question: Do these three methods guarantee to select the same model?

Although the stepwise method is computationally faster than the best subset method:

- It does not guarantee finding the best model for each size.
- The method still requires a large number of models to be fitted (how many?), and assessed with CV(how many?)
- When $p \approx n$, the model fitted through OLS/ML becomes unstable or even undefined.

This brings the need for another type of method:

Model Regularization

Ridge Regression

- ML models are fitted by minimizing loss functions.
- The loss function for ordinary linear regression is

$$\mathcal{R} = \text{RSS} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}\right)^2 = (\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta})^T (\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}).$$

- Ridge regression instead minimizes

$$\mathcal{R}_{reg} = \mathcal{R} + \lambda \sum_{j=1}^{p} \beta_j^2.$$

- $\lambda \sum_{j=1}^{p} \beta_j^2$ is called the *penalty term* or *shrinkage term*.
- The hyperparameter (tuning parameter) $\lambda > 0$ controls the size of the penalty.

Ridge Regression Derivation

For simplicity, assume there is no intercept parameter β_0.

- The penalty term can be rewritten as $\lambda \sum_{j=1}^{p} \beta_j^2 = \lambda \boldsymbol{\beta}^T \boldsymbol{\beta}$.
- The ridge regression estimate is obtained as:

$$\hat{\boldsymbol{\beta}}_{reg} = \arg\min_{\boldsymbol{\beta}} \{(\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta})^T (\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda \boldsymbol{\beta}^T \boldsymbol{\beta}\}$$

$\mathcal{R}_{reg}$ is convex (why?), so $\hat{\boldsymbol{\beta}}_{reg}$ is solution to $\nabla_{\boldsymbol{\beta}} \mathcal{R}_{reg} = 0$.

$$\begin{aligned} \nabla_{\boldsymbol{\beta}} \mathcal{R}_{reg} &= \nabla_{\boldsymbol{\beta}} (\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta})^T (\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda \nabla_{\boldsymbol{\beta}} \boldsymbol{\beta}^T \boldsymbol{\beta} \\ &= 2(\mathbf{X}^T \mathbf{X}\boldsymbol{\beta} - \mathbf{X}^T \boldsymbol{y}) + 2\lambda \boldsymbol{\beta} \qquad (1) \\ &= 0. \end{aligned}$$

$$\begin{aligned} \nabla_{\boldsymbol{\beta}} \mathcal{R}_{reg} = 0 &\Rightarrow \mathbf{X}^T \mathbf{X}\boldsymbol{\beta} - \mathbf{X}^T \boldsymbol{y} + \lambda \boldsymbol{\beta} = 0 \\ &\quad (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})\boldsymbol{\beta} = \mathbf{X}^T \boldsymbol{y} \qquad (2) \\ &\Rightarrow \hat{\boldsymbol{\beta}}_{reg} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \boldsymbol{y} \end{aligned}$$

Ridge Regression Properties

For ridge regression: $\mathcal{R}_{reg} = \text{RSS} + \lambda \boldsymbol{\beta}^T \boldsymbol{\beta}$:

- The hyperparameter λ controls the bias-variance tradeoff.
- As $\lambda \to 0$, $\hat{\boldsymbol{\beta}}_{reg} \to \hat{\boldsymbol{\beta}}_{ols}$
- As $\lambda \to \infty$, $\hat{\boldsymbol{\beta}}_{reg} \to \mathbf{0}$
- When the intercept β_0 exists, it is normally unpenalized.
- In practice, λ is either fixed in advance or chosen by CV.

The ridge regression estimate: $\hat{\boldsymbol{\beta}}_{reg} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\boldsymbol{y}$

- Unlike $(\mathbf{X}^T\mathbf{X})$, $(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})$ is always invertible.
- Therefore, ridge regression does not require $p \leq n$.
- The coefficients for trivial features will be shrunk more than for important features.

Ridge Regression: An alternative derivation

Ridge regression, with a tuning parameter λ, minimizes

$$\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}\beta_j^2$$

An alternative way to look at Ridge Regression is to consider it as an unpenalized OLS problem, but with **augmented data**.

- Denote the augmented dataset as $\tilde{\mathbf{X}} = \begin{bmatrix} \mathbf{X} \\ \sqrt{\lambda}\mathbf{I} \end{bmatrix}$ and $\tilde{\mathbf{y}} = \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \end{bmatrix}$.
- In this case, the Ridge Regression estimate is equivalent to the OLS estimate using this augmented data, i.e.,

$$\hat{\boldsymbol{\beta}}_{ridge} = arg \min_{\beta} \|\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\beta\|_2^2$$

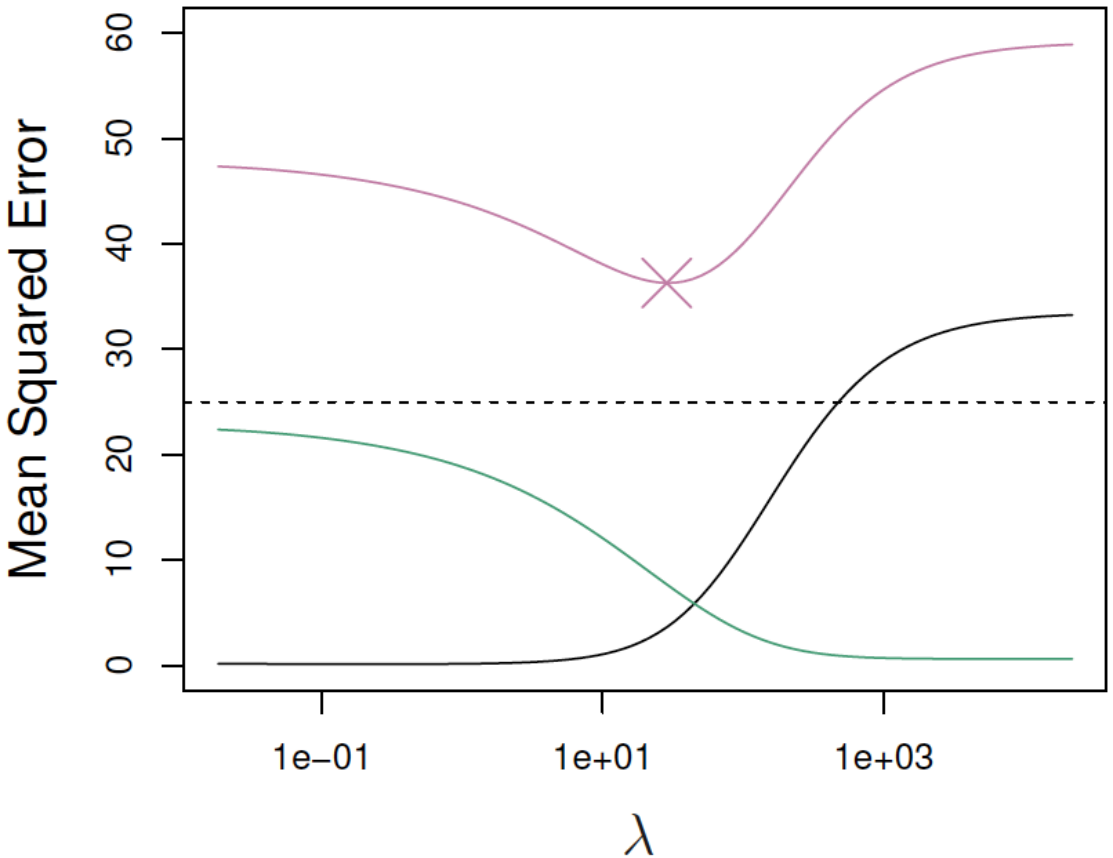


Figure: Bias-variance tradeoff of ridge regression on a simulated dataset. The pink line denotes testing MSE; the black line denotes squared bias; the green line denotes variance. The horizontal line denotes the Bayes error. Source: James et al., 2021.

LASSO Regression

- $\lambda \sum_{j=1}^{p} \beta_j^2$ is not the only penalty that one can apply.
- $\lambda \sum_{j=1}^{p} |\beta_j|$ also penalizes the size of each $\hat{\beta}_j$.
- The LASSO regression considers $\mathcal{R}_{reg} = \mathcal{R} + \lambda \sum_{j=1}^{p} |\beta_j|$.
- LASSO stands for *Least Absolute Shrinkage and Selection Operator*.
- As the name suggested, LASSO does not just shrink the size of $\hat{\beta}_j$, it also selects the important features.
- The $\hat{\beta}_j$ for trivial features will be shrunk to *exactly* zero.
- This improves the *interpretability* of the fitted model.
- Similar to ridge regression, the intercept parameter is normally unpenalized.

Review of the notion of *norm*

Let $\mathbf{x} \in \mathbb{R}^n$ be a vector. The (default) norm of $\mathbf{x}$ is defined as:

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}.$$

In general, a norm can be *any* function that satisfies the following properties:

- $\|\mathbf{x}\| \geq 0$.
- $\|\mathbf{x}\| = 0$ if and only if $\mathbf{x} = \mathbf{0}$.
- $\|\alpha\mathbf{x}\| = |\alpha|\|\mathbf{x}\|$ for any scalar α.
- $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ (triangle inequality).

L_p norm and penalty

- Both the LASSO and the Ridge penalize the size of the *norm* of $\boldsymbol{\beta}$.
- Given $p \in \mathbb{N}$, the L_p norm of $\boldsymbol{x} = (x_1, ..., x_m)^T \in \mathbb{R}^m$ is defined as
$$||\boldsymbol{x}||_p = (|x_1|^p + ... + |x_m|^p)^{1/p}.$$
- LASSO uses the L_1 norm of $\boldsymbol{\beta}$ as the penalty; Ridge uses the squared L_2 norm as the penalty.
- When $p = 0$, the L_0 (pseudo) norm of $\boldsymbol{x}$ is defined as $\sum_{i=1}^{m} I(x_i \neq 0)$.

LASSO, Ridge and the Best Subset regression can be equivalently formulated as the following:

$$\arg\min \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}\right)^2, \text{ subject to } \underbrace{\sum_{i=1}^{p} |\beta_i| \leq s}_{L_1 \ norm},$$

$$\arg\min \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}\right)^2, \text{ subject to } \underbrace{\sum_{i=1}^{p} \beta_i^2 \leq s}_{L_2 \ norm},$$

$$\arg\min \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}\right)^2, \text{ subject to } \underbrace{\sum_{i=1}^{p} I(\beta_i \neq 0) \leq s}_{L_0 \ pseudo \ norm},$$

where s is some function of λ in LASSO and Ridge, and the subset size in best subset regression.

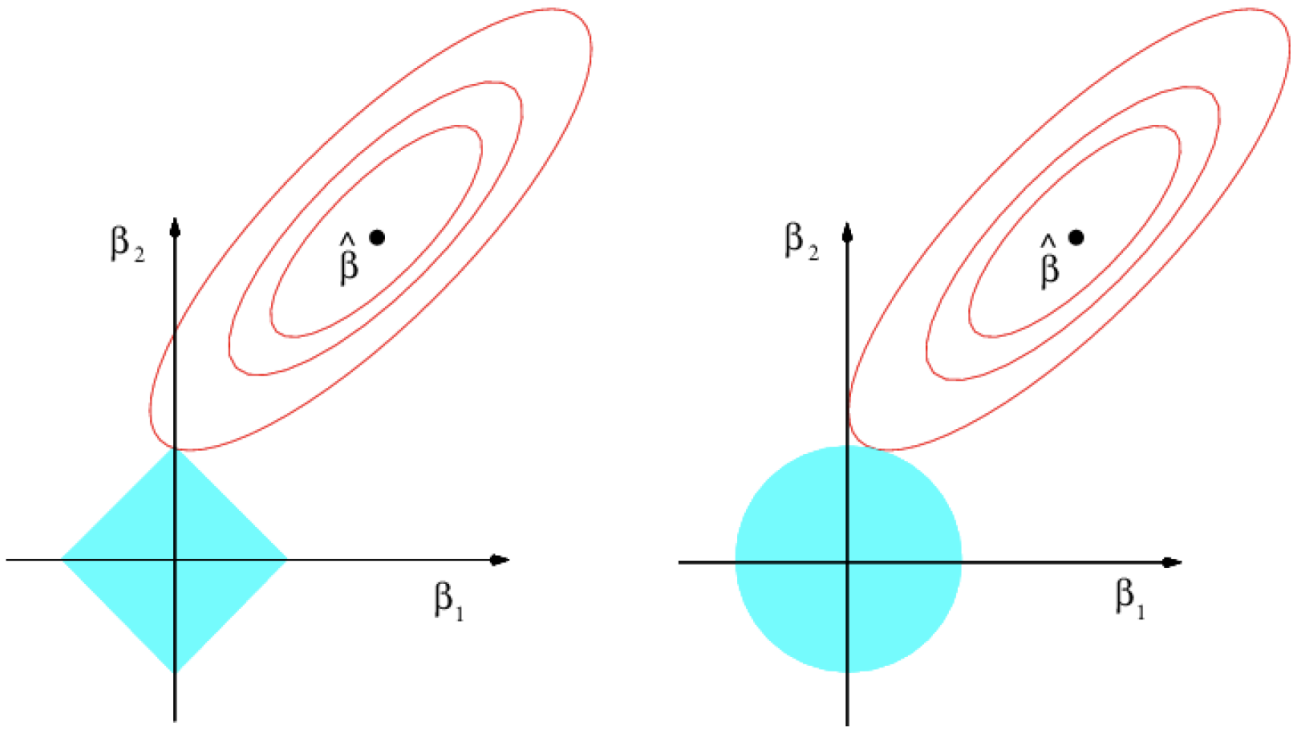


FIGURE 6.7. *Contours of the error and constraint functions for the lasso* (left) *and ridge regression* (right). *The solid blue areas are the constraint regions,* $|\beta_1| + |\beta_2| \leq s$ *and* $\beta_1^2 + \beta_2^2 \leq s$, *while the red ellipses are the contours of the RSS.*

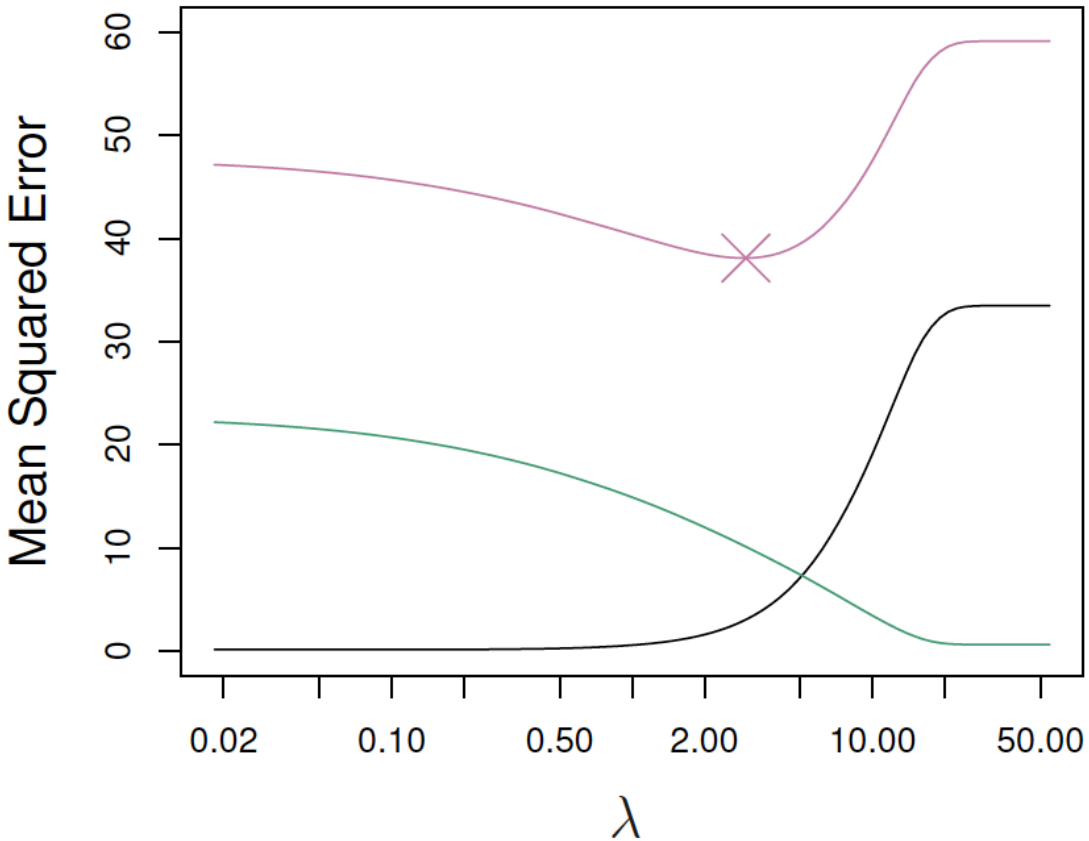


Figure: Bias-variance of LASSO regression on a simulated dataset. The pink line denotes test MSE; the black line denotes squared bias; the green line denotes variance. Source: James et al., 2021.

Computation of LASSO

- The fitted model from OLS and Ridge regression has a closed-form solution.
- However, the solution of LASSO typically *does not have a closed form*.
- The loss function $\mathcal{R}_{reg}$ from LASSO is not *differentiable*.
- Fortunately, $\mathcal{R}_{reg}$ is convex and separable *(the sum of functions each depends only on one variable)*.
- Therefore, the LASSO solution can be obtained numerically through the coordinate descent algorithm *(which we won't cover in this course)*.

Comparing LASSO with Ridge

- Unlike Ridge, LASSO also serves the purpose of feature selection, which improves interpretability (*although the number of selected features will be less than n*).
- For prediction purposes, neither ridge regression nor the lasso will universally dominate the other.
- Lasso performs well when a few predictors have substantial coefficients, while the rest have coefficients close to zero (*sparsity assumption*), unless there are many *highly correlated* features.
- Ridge regression is effective when the response depends on multiple predictors, all with coefficients of similar sizes.
- However, the number of predictors that are related to the response is never known a priori for real data sets.
- In practice, the two approaches should be compared using the CV method.

Combining LASSO with Ridge: Elastic Net

$$\mathcal{R}_{reg} = \mathcal{R} + \underbrace{\lambda_1 ||\boldsymbol{\beta}||_1}_{LASSO} + \underbrace{\lambda_2 ||\boldsymbol{\beta}||_2^2}_{Ridge}.$$

- The computation of Elastic Net is as simple as that of LASSO itself (why?)
- In LASSO, the coefficients of *highly correlated variables* tend to behave arbitrarily. In the elastic net, they tend to be equal.
- Further, the elastic net method can select more than n features when $p \geq n$.
- Some will critique the high bias of the elastic net, due to its double shrinkage. This can be improved by scaling up the Ridge part $\hat{\beta}_j^* = \sqrt{1 + \lambda_2}\hat{\beta}_j$.

fitting models with glmnet

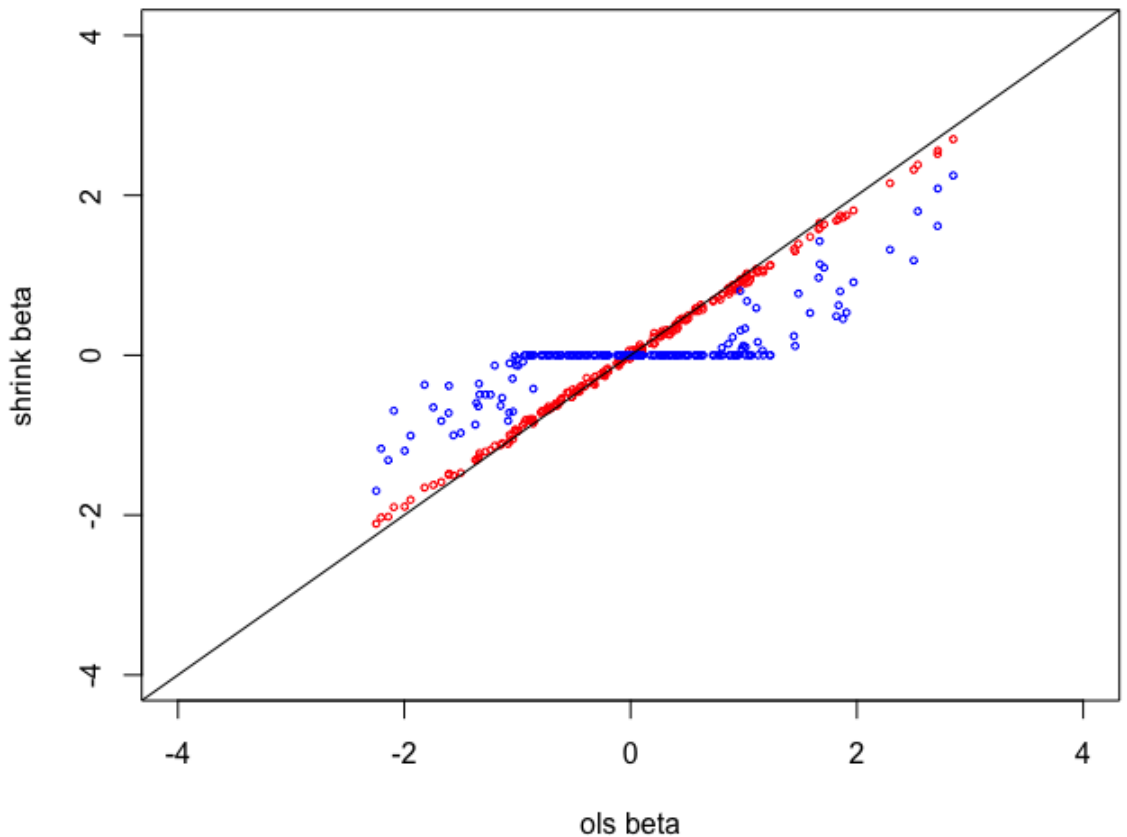
In glmnet, the hyperparameters λ_1 and λ_2 are parametrized as $\lambda\alpha$ and $\lambda(1-\alpha)/2$, where $\lambda \geq 0$ and $\alpha \in (0, 1)$ (When $\alpha = 0$ or 1, the parametrization goes back to original Ridge or LASSO).

$$\mathcal{R}_{reg} = \mathcal{R} + \lambda(\alpha||\boldsymbol{\beta}||_1 + \frac{1-\alpha}{2}||\boldsymbol{\beta}||_2^2).$$

```
> library(glmnet)
>
> ## Simulate 1000 obs, 200 features
> n <- 1000; p <- 200
> ## Assume all features are iid normal
> xall <- rnorm(n = n*p)
> X <- matrix(xall, nrow = n, ncol = p)
> ## Simulate feature coefficients from normal
> beta <- rnorm(n = p, sd = 1)
> y <- X %*% beta + rnorm(n = n)
>
>
> mod_ols <- glmnet(x = X, y = y, lambda = 0, alpha = 0, thresh = 1e-18)
> #mod_ols$beta
> mod_ridge <- glmnet(x = X, y = y, lambda = 1, alpha = 0, thresh = 1e-18)
> #mod_ridge$beta
> mod_lasso <- glmnet(x = X, y = y, lambda = 1, alpha = 1, thresh = 1e-18)
> #mod_lasso$beta
> mod_en <- glmnet(x = X, y = y, lambda = 1, alpha = 0.5, thresh = 1e-18)
> # mod_en$beta
```

fitting models with glmnet

```
> plot(mod_ridge$beta[,1] ~ mod_ols$beta[,1], col = "red", cex = 0.5, ylim = c(-4,4), xlim = c(-4,4), ylab = "shrink beta", xlab = "ols beta")
> points(mod_lasso$beta[,1] ~ mod_ols$beta[,1], col = "blue", cex = 0.5)
> abline(a = 0, b = 1)
```



fitting models with glmnet

```
> ## Simulate 1000 obs, 2000 features
> n <- 1000; p <- 2000
> ## Assume all features are iid normal
> xall <- rnorm(n = n*p)
> X <- matrix(xall, nrow = n, ncol = p)
> ## Simulate feature coefficients from normal
> beta <- rnorm(n = p, sd = 1)
> y <- X %*% beta + rnorm(n = n)
> mod_ridge <- glmnet(x = X, y = y, lambda = 0.05, alpha = 0, thresh = 1e-18)
> mod_lasso <- glmnet(x = X, y = y, lambda = 0.05, alpha = 1, thresh = 1e-18)
> mod_en <- glmnet(x = X, y = y, lambda = 0.05, alpha = 0.5, thresh = 1e-18)
> plot(mod_ridge$beta[,1] ~ beta, col = "red", cex = 0.5, ylim = c(-4,4), xlim = c(-4,4), ylab = "est beta (ridge)", xlab = "true beta")
> plot(mod_lasso$beta[,1] ~ beta, col = "blue", cex = 0.5, ylim = c(-4,4), xlim = c(-4,4), ylab = "est beta (lasso)", xlab = "true beta")
> plot(mod_en$beta[,1] ~ beta, col = "green", cex = 0.5, ylim = c(-4,4), xlim = c(-4,4), ylab = "est beta (en)", xlab = "true beta")
>
> sum(mod_ridge$beta[,1] != 0)
[1] 2000
> sum(mod_lasso$beta[,1] != 0)
[1] 957
> sum(mod_en$beta[,1] != 0)
[1] 1008
```

Regularization & Shrinkage methods are examples of **supervised learning**.

- The penalized regression takes use of both the features $\mathbf{X}$ and the outcome $\mathbf{y}$.
- These methods mostly exploit the information in $p(\mathbf{y}|\mathbf{X})$.

Principal Components Analysis (PCA) is an **unsupervised learning** approach for dimension reduction.

- PCA only looks at the features $\mathbf{X}$.
- The target is to exploit the structure in $p(\mathbf{X})$ alone.
- Apart from reducing the feature dimension in a supervised learning problem, PCA serves as a tool for data visualization, clustering, etc$\cdots$

A motivating example

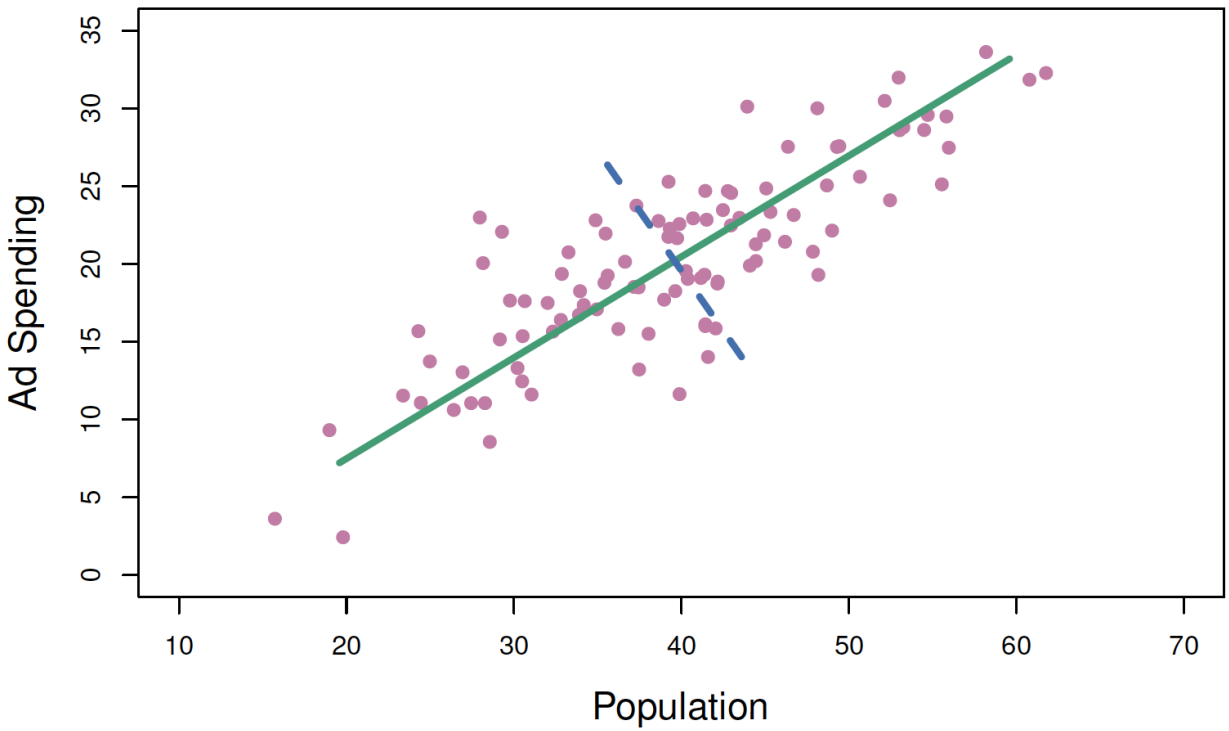


FIGURE 6.14. *The population size (pop) and ad spending (ad) for 100 different cities are shown as purple circles. The green solid line indicates the first principal component, and the blue dashed line indicates the second principal component.*

Motivation of PCA

- The features $\mathbf{X}$ often contain overlapped information.
- Some features tend to behave like linear combinations of other features (i.e. ad $\approx \phi_1 + \phi_2$pop).
- Projecting the *two* features (ad, pop) into that *one* green line will not loose too much of variability in $\mathbf{X}$.
- PCA is eventually about finding the directions to project the original features $\mathbf{X}$, that minimize the lost information (variability).

Review of projection

The projection of a vector $\mathbf{a}$ onto a vector $\mathbf{b}$ is given by:

$$\text{proj}_{\mathbf{b}}\mathbf{a} = \frac{\mathbf{a}^T\mathbf{b}}{\|\mathbf{b}\|_2^2}\mathbf{b}$$

- The projection of $\mathbf{a}$ onto $\mathbf{b}$ results in a vector that lies on the line defined by $\mathbf{b}$ and is closest to $\mathbf{a}$.
- The scalar projection is $\frac{\mathbf{a}^T\mathbf{b}}{\|\mathbf{b}\|_2}$, which is the length of the projection.
- Let θ denotes the angle between the two vectors, then $\cos(\theta) = \frac{\mathbf{a}^T\mathbf{b}}{\|\mathbf{a}\|_2\|\mathbf{b}\|_2}$.

Linear transformation of multivariate normal

1. Assume $\mathbf{X} \sim N(\mathbf{0}, \mathbf{\Sigma})$, then $\mathbf{\Sigma}^{-1/2}\mathbf{X} \sim N(\mathbf{0}, \boldsymbol{I})$.
2. In other words, $\mathbf{X} \overset{d}{=} \mathbf{\Sigma}^{1/2}\mathbf{Z}$, where $\mathbf{Z} \sim N(\mathbf{0}, \boldsymbol{I})$.
3. Write $\mathbf{\Sigma}^{1/2} = \boldsymbol{U}\boldsymbol{D}^{1/2}\boldsymbol{U}^T$, the matrix $\boldsymbol{U}^T$ *projects* $\mathbf{Z}$ onto the eigenvectors. The basis of the system changes from the standard basis to the eigenvectors.
4. Then the matrix $\boldsymbol{D}^{1/2}$ *scales* the projected $\mathbf{Z}$ by the (square root) eigenvalue.
5. The matrix $\boldsymbol{U}$ *projects* the vector back to the original system with the standard basis.

Illustrations

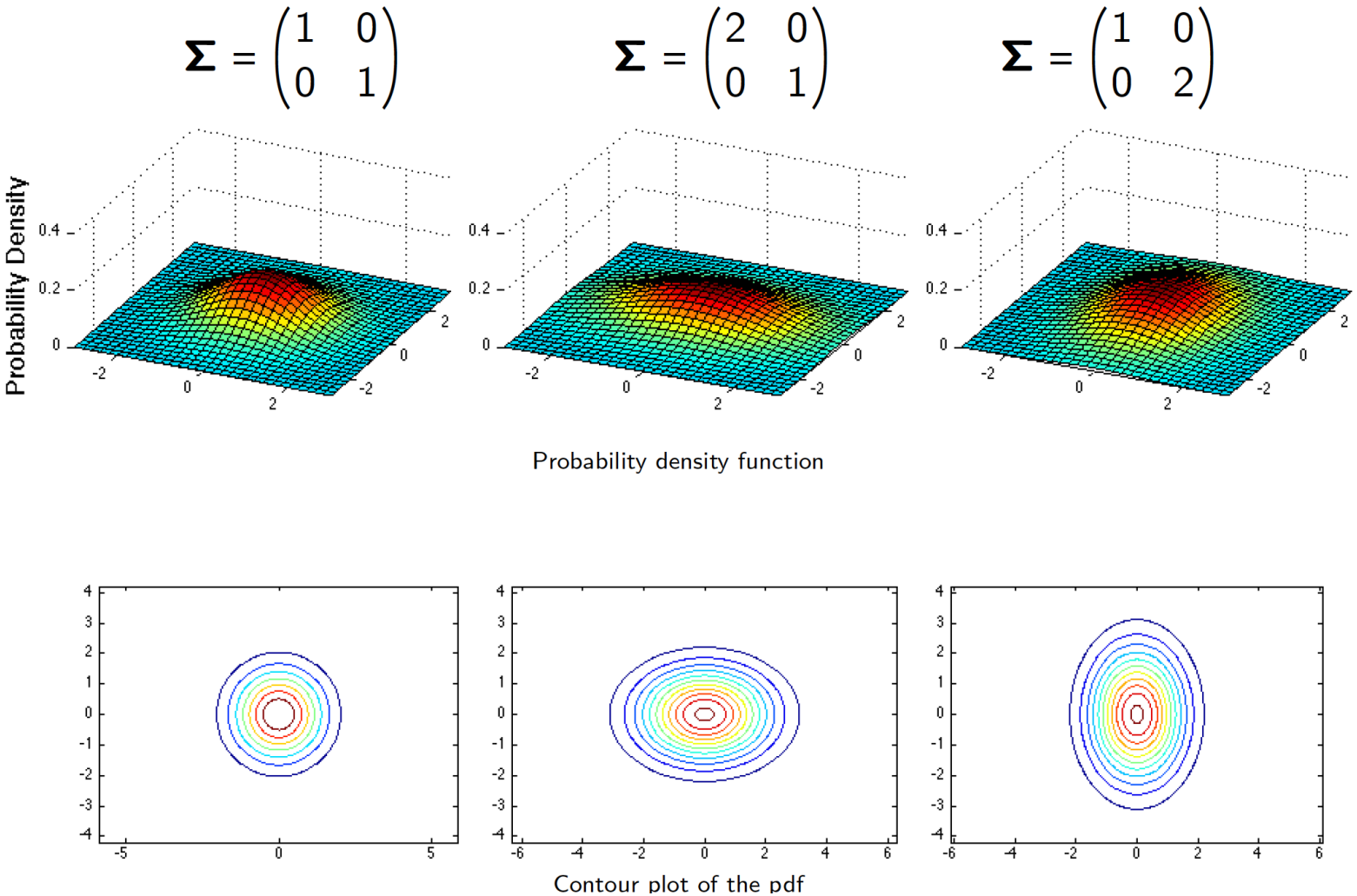


Figure: Example taken from Prof Maddison.

Illustrations

$$\boldsymbol{\Sigma} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \boldsymbol{\Sigma} = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix} \qquad \boldsymbol{\Sigma} = \begin{pmatrix} 1 & 0.8 \\ 0.8 & 1 \end{pmatrix}$$

$$= \mathbf{Q}_1 \begin{pmatrix} 1.5 & 0. \\ 0. & 0.5 \end{pmatrix} \mathbf{Q}_1^\top \qquad = \mathbf{Q}_2 \begin{pmatrix} 1.8 & 0. \\ 0. & 0.2 \end{pmatrix} \mathbf{Q}_2^\top$$

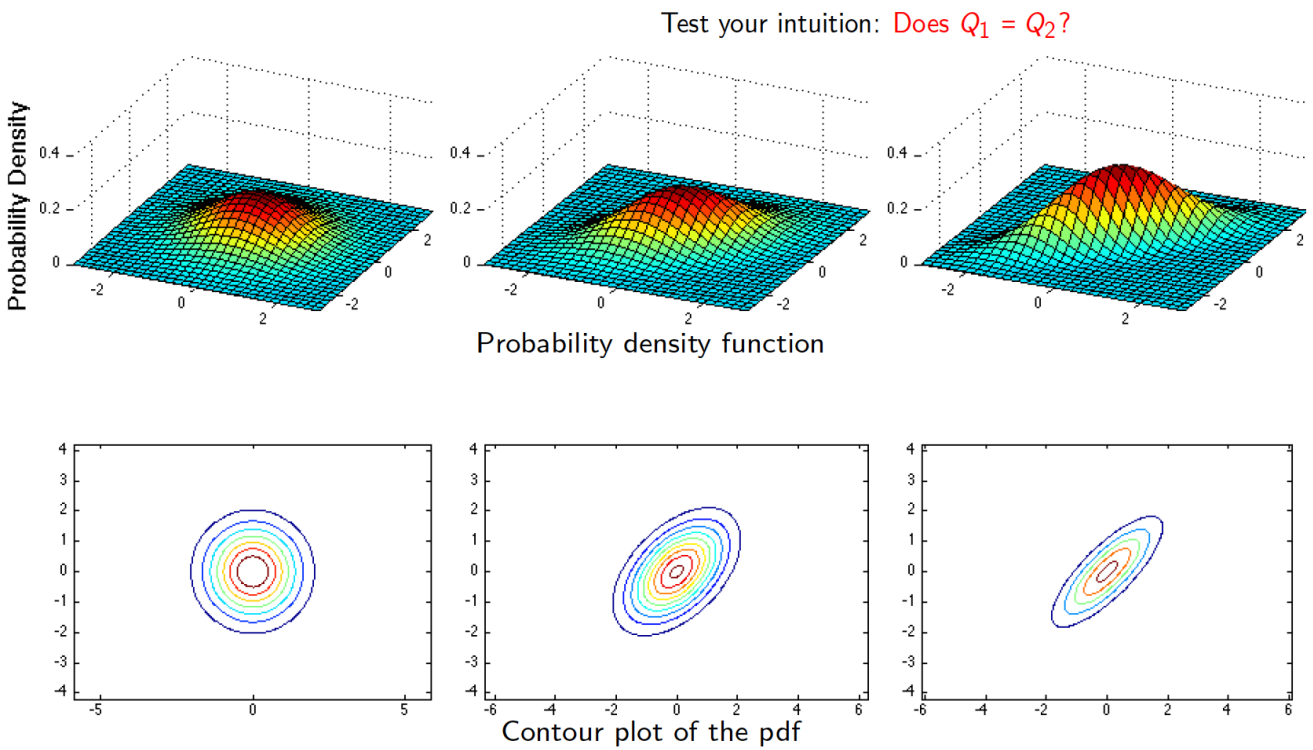


Figure: Example taken from Prof Maddison.

Illustrations

$$\boldsymbol{\Sigma} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \boldsymbol{\Sigma} = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix} \qquad \boldsymbol{\Sigma} = \begin{pmatrix} 1 & -0.5 \\ -0.5 & 1 \end{pmatrix}$$

$$= \mathbf{Q}_1 \begin{pmatrix} 1.5 & 0. \\ 0. & 0.5 \end{pmatrix} \mathbf{Q}_1^\top \qquad = \mathbf{Q}_2 \begin{pmatrix} \lambda_1 & 0. \\ 0. & \lambda_2 \end{pmatrix} \mathbf{Q}_2^\top$$

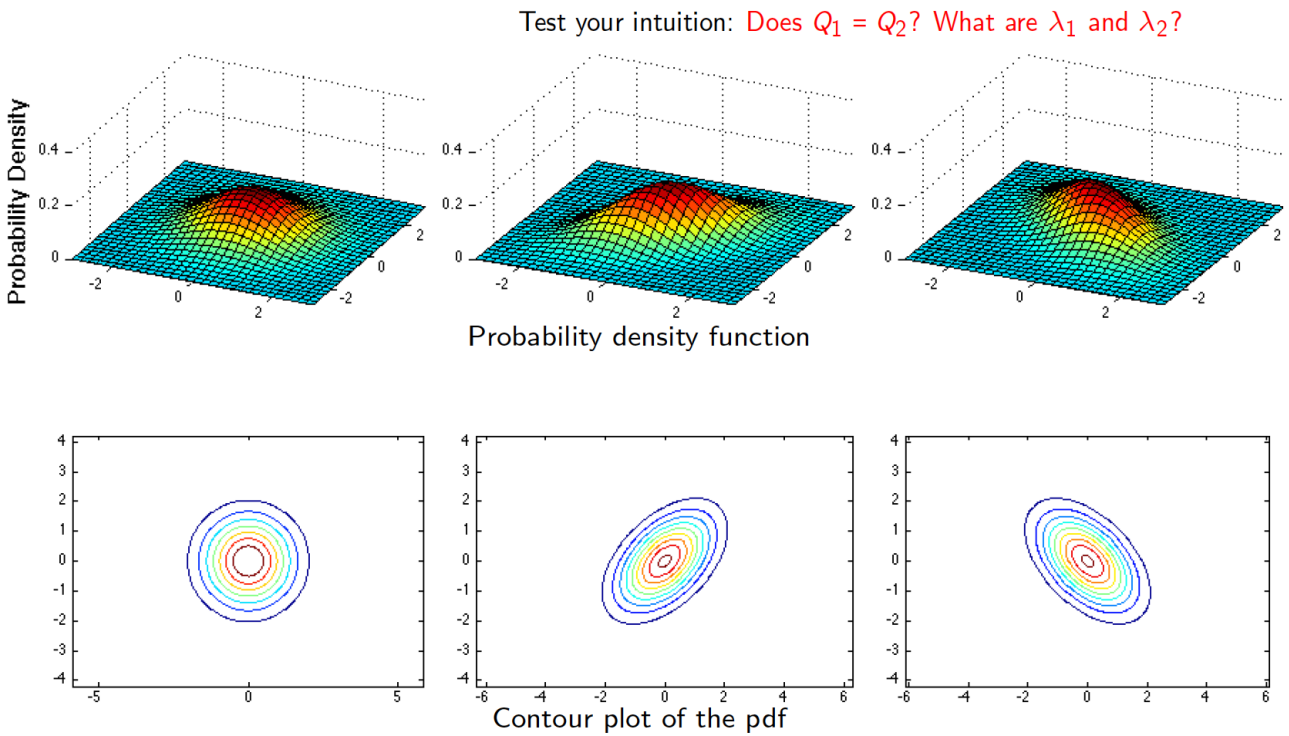


Figure: Example taken from Prof Maddison.

Computing the PC

- Assume the (centered) feature vector $\mathbf{x} \in \mathbb{R}^p$ has covariance $\mathbf{\Sigma}$.
- The first *population* PC direction is defined as

$$v_1 = \underset{||v||=1}{\text{argmax}} \, \text{Var}(v^T \mathbf{x})$$

- The jth PC direction is defined as

$$v_j = \underset{||v||=1; v^T v_i = 0 \ \forall i<j}{\text{argmax}} \, \text{Var}(v^T \mathbf{x})$$

- The jth PC direction is also called the jth PC *loading* vector; $v_j^T x$ is called the jth PC *value or score.*
- In practice, $\mathbf{\Sigma}$ is unknown, hence we estimate it with sample variance $\hat{\mathbf{\Sigma}} = \frac{1}{n}\mathbf{X}^T\mathbf{X}$.
  (*Question: what if the data is uncentered?*)
- The PCs computed from $\hat{\mathbf{\Sigma}}$ are called *sample* PCs.

PC direction vs Eigenvector

Claim: Assume $\text{Var}(\mathbf{x}) = \Sigma = UDU^T$, then the kth population PC direction is the kth eigenvector u_k.

Proof: We will first prove this for the first PC direction. Note that

$$\begin{aligned} \text{Var}(v^T\mathbf{x}) &= v^T\text{Var}(\mathbf{x})v \\ &= v^T\Sigma v \\ &= v^T UDU^T v \\ &= z^T Dz, \end{aligned} \tag{3}$$

where $z = U^T v$. Note that

$$||z||^2 = z^T z = v^T UU^T v = v^T v = ||v||^2 = 1.$$

Since D has decreasing diagonal entries, the unit vector that maximizes $z^T Dz$ is the first standard basis $z = e_1$ (*why?*)
This implies $U^T v = e_1$, hence $v = Ue_1 = u_1$.

Proof continued:

Now to prove the same result holds for other PC directions, assume $v_1 = u_1, ..., v_{k-1} = u_{k-1}$ hold.

To compute the kth PC direction, we require $z_k = U^T v_k$ to maximize $z_k^T D z_k$, while $v_k^T v_i = 0 \; \forall i < k$.

Note that $v_k^T v_i = z_k^T z_i = 0$, and $z_i = e_i$ for $i < k$. It is obvious that the first $k - 1$ entries of z_k must be zero.

Again, since D has decreasing diagonal entries, $z_k = e_k$ will maximize $z_k^T D z_k$. Hence $v_k = u_k$.

The proof concludes by induction.

Properties of PCs

- The similar result holds for sample PCs as well (see exercise).
- The variance of the jth PC value is the jth eigenvalue d_i (see exercise).
- PC values are uncorrelated with each other (see exercise).
- When $\mathbf{x}$ is normally distributed, its PC values are normally distributed as well.

How many PCs should we use?

In practice, we need to choose how many PCs should we keep...

- For visualization purposes, the first two PCs are often used.
- For dimension reduction of feature space, important information may be lost when too few PCs are being used.
- When too many PCs are used, the purpose of dimension reduction is no longer meaningful.
- One way to choose the number of PCs is through the screeplot.
- The screeplot visualizes the *proportion of variance explained* (PVE) by each PC component.
- The total variance of the feature vector is

$$\sum_{i=1}^{p} \text{Var}(x_i) = tr(\Sigma) = \sum_{i=1}^{p} d_i.$$

- The *cumulative variance explained* (CVE) by the first k PCs is $\sum_{i=1}^{k} d_i$, and the jth PVE is $\frac{d_j}{\sum_{i=1}^{p} d_i}$.

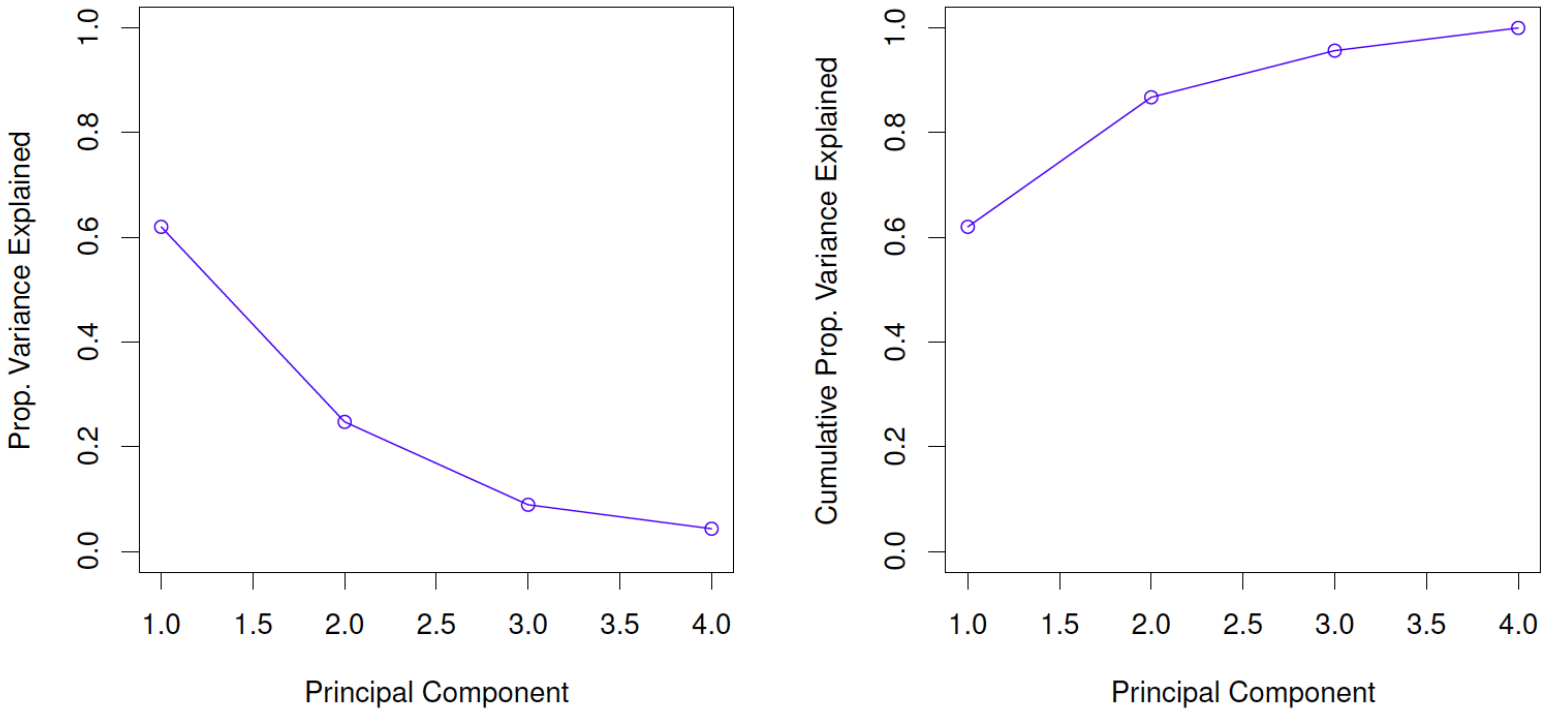


FIGURE 12.3. Left: *a scree plot depicting the proportion of variance explained by each of the four principal components in the* `USArrests` *data.* Right: *the cumulative proportion of variance explained by the four principal components in the* `USArrests` *data.*

Principal Components Regression

Algorithm 4 Principal components regression

0: Let $\mathbf{X} \in \mathbb{R}^{n \times p}$, $\mathbf{y} \in \mathbb{R}^n$. Make sure the features have been centered, such that each column of $\mathbf{X}$ sums to zero.
1: Compute the eigen-decomposition of $\frac{1}{n}\mathbf{X}^T\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{U}^T$.
2: Use the eigenvalues of $\frac{1}{n}\mathbf{X}^T\mathbf{X}$ to decide the number of PCs (k) to used, based on either PVE or CVE.
3: Compute the first k PC values $\mathbf{z}_{ij} = U_j^T\mathbf{x}_i$ using the eigenvectors of $\frac{1}{n}\mathbf{X}^T\mathbf{X}$ ($\mathbf{Z} = \mathbf{X}\mathbf{U}_{[.,1:k]} \in \mathbb{R}^{n \times k}$).
4: Regress the outcome $\mathbf{y}$ on the new design matrix $\mathbf{Z}$.
5: To predict at $\mathbf{x}_{new}$, compute $\mathbf{z}_{new} = \mathbf{x}_{new}\mathbf{U}_{[.,1:k]}$.
6: The prediction is $y_{new} = \mathbf{z}_{new}\hat{\boldsymbol{\beta}}$.

Question: *Why it is not as important to do train/test separation during the PCA* (steps 1- 3)?

For next week:

- The next lecture will be about classification and logistic regression.
- The problem set 2 is posted online. Again, remember to bring your printout of the computation question to the tutorial for the next quiz.
- Reminder, the midterm will be held on July 28th, during the regular lecture time (EX320).
- The midterm is closed-book, and the only allowed item is a non-programmable calculator.
- There will be no quiz during the midterm week. Extra office hours will be announced soon.
- A practice midterm will be posted, which has a representative *structure*.