Module 5: Linear Model Selection and Regularization

Material primarily drawn from ISLR Chapter 6

Overview

- This module addresses how to identify the appropriate set of predictors to use in a model
- In general, our objectives are:
  - Improve prediction accuracy in situations with large numbers of predictors relative to the number of observations
  - Improve model interpretability by removing irrelevant features
- *Note: these techniques are presented in the context of linear models, but they can be used with a wide range of model types*

Overview

Linear Model Prediction Accuracy

- Provided the true relationship is approximately linear, the least squares estimates will have low bias
  - If $n \gg p$, then least squares estimates tend to have low variance
  - Rule of thumb is you need 10-20 times as many observations as predictors for reasonable performance
- If n is not much more than p, there can be a lot of variance resulting in overfitting and poor predictions
- How about if $p > n$?
  - Then, there is no longer a unique least squares coefficient so the variance is infinite

Overview

Three Approaches

- Subset selection
  - Identifying a subset of the p predictors that we believe are best related to the response
- Shrinkage methods
  - Use all p predictors, but use techniques to shrink the estimated coefficients towards zero (also referred to as regularization)
- Dimension reduction
  - Project the p predictors into an M-dimensional subspace where $M < p$

Subset Selection

Subset Selection Approaches

- Best subset selection
- Stepwise selection
- Choosing the optimal model

Best Subset Selection

- Fit a separate regression model to all possible combinations of the p predictors

Best Subset Selection

Basic Procedure

1. Let $\mathcal{M}_0$ denote the *null model* which contains no predictors
   - Thus, the model simply predicts the sample mean for every observation
2. For $k = 1,2, \ldots, p$:
   - Fit all $\binom{p}{k}$ possible models that contain exactly k predictors
   - Pick the best among these models (smallest training RSS or, equivalently, largest R^2) and call it $\mathcal{M}_k$.
3. Select a single best model from among $\mathcal{M}_0, \mathcal{M}_1, \ldots \mathcal{M}_p$
   - Several candidate techniques for making this selection

Credit Example

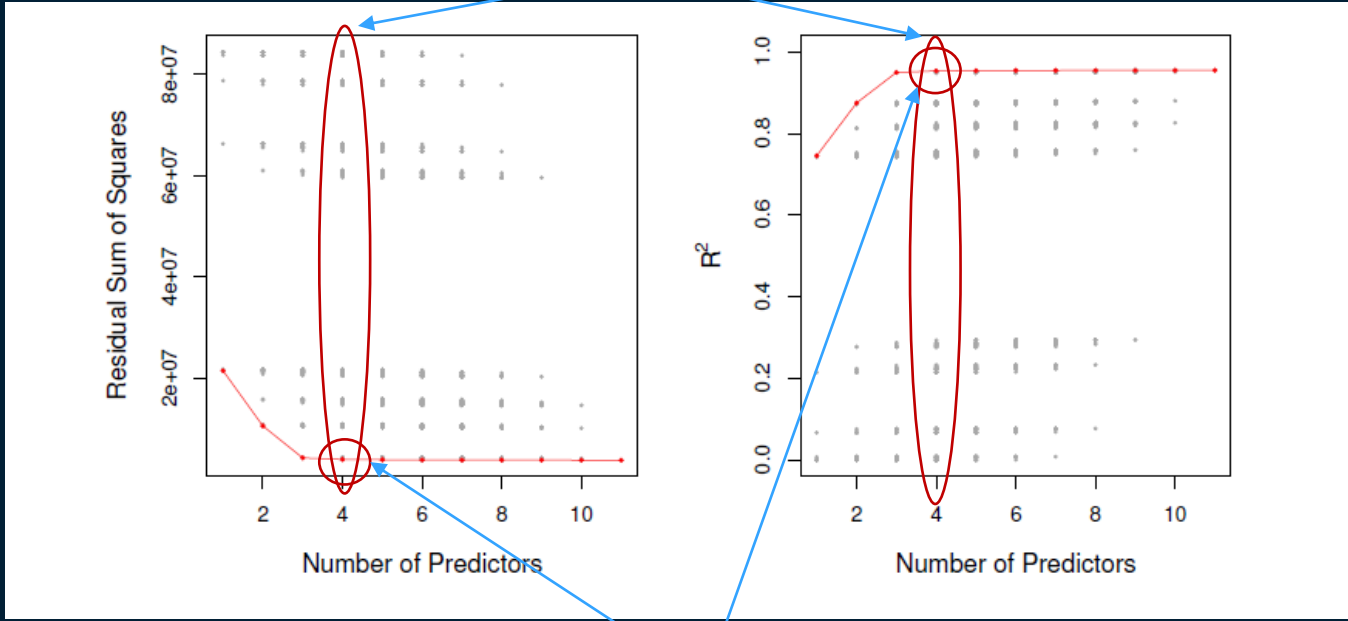
$p = 11$ (Region has 3 levels, so it becomes 2 predictors

$n = 400$

	A	B	C	D	E	F	G	H	I	J	K
1	Income	Limit	Rating	Cards	Age	Education	Own	Student	Married	Region	Balance
2	14.891	3606	283	2	34	11	No	No	Yes	South	333
3	106.025	6645	483	3	82	15	Yes	Yes	Yes	West	903
4	104.593	7075	514	4	71	11	No	No	No	West	580
5	148.924	9504	681	3	36	11	Yes	No	No	West	964
6	55.882	4897	357	2	68	16	No	No	Yes	South	331
7	80.18	8047	569	4	77	10	No	No	No	South	1151
8	20.996	3388	259	2	37	12	Yes	No	No	East	203
9	71.408	7114	512	2	87	9	No	No	No	West	872
10	15.125	3300	266	5	66	13	Yes	No	No	South	279
11	71.061	6819	491	3	41	19	Yes	Yes	Yes	East	1350
12	63.095	8117	589	4	30	14	No	No	Yes	South	1407
13	15.045	1311	138	3	64	16	No	No	No	South	0
14	80.616	5308	394	1	57	7	Yes	No	Yes	West	204
15	43.682	6922	511	1	49	9	No	No	Yes	South	1081
16	19.144	3291	269	2	75	13	Yes	No	No	East	148

Credit Example

All possible p=4 models



$\mathcal{M}_4$

Stepwise Selection

- For computational reasons, best subset selection cannot be applied with very large p. *Why not?*

2^p possible models

- Also, such a large model search space can lead to overfitting and high variance of the coefficient estimates
- For both of these reasons, stepwise methods, which limit the number of models evaluated, are generally employed

Forward Stepwise Selection

- Begins with a model containing no predictors and adds predictors to the model one at a time until all predictors are in the model
  - At each step, the variable that gives the greatest additional improvement to the fit is added

Forward Stepwise Selection

Basic Procedure

1. Let $\mathcal{M}_0$ denote the *null model* which contains no predictors
   - Thus, the model simply predicts the sample mean for every observation
2. For $k = 1,2, \ldots, p$:
   - Consider all $p - k$ models that augment the predictors in $\mathcal{M}_k$ with one additional predictor
   - Pick the best among these models (smallest training RSS or, equivalently, largest R^2) and call it $\mathcal{M}_{k+1}$.
3. Select a single best model from among $\mathcal{M}_0, \mathcal{M}_1, \ldots \mathcal{M}_p$
   - Several candidate techniques for making this selection

Forward Stepwise Selection

Not Guaranteed to Find Best Possible Models

Credit data set example:

# Variables	Best subset	Forward stepwise
One	rating	rating
Two	rating, income	rating, income
Three	rating, income, student	rating, income, student
Four	cards, income student, limit	rating, income, student, limit

Backward Stepwise Selection

- Begins with a model containing all p predictors and removes predictors from the model one at a time until all predictors are in the model
  - At each step, the variable that gives the least additional improvement to the fit is removed

Backward Stepwise Selection

Basic Procedure

1. Let $\mathcal{M}_p$ denote the *full model* with contains all p predictors
2. For $k = p, p-1, \ldots, 1$:
   - Consider all k models that contain all but one of the predictors in $\mathcal{M}_k$, for a total of $k-1$ predictors
   - Pick the best among these models (smallest training RSS or, equivalently, largest R^2)and call it $\mathcal{M}_{k-1}$
3. Select a single best model from among $\mathcal{M}_0, \mathcal{M}_1, \ldots \mathcal{M}_p$
   - Several candidate techniques for making this selection

Choosing the Optimal Model

- Model containing all of the predictors will always have largest training R^2 and smallest RSS
- We wish to choose model with low test error, not model with low training error
- Therefore, RSS and R^2 are not suitable for selecting the best model among a collection of models with different numbers of predictors

Choosing the Optimal Model

Two Approaches

- Indirectly estimate test error by making an adjustment to the training error to account for bias due to overfitting
  - Mallow's C_p
  - AIC/BIC
  - Adjusted R^2

Statistical approach that measures information loss in fitted model. Objective is to minimize.

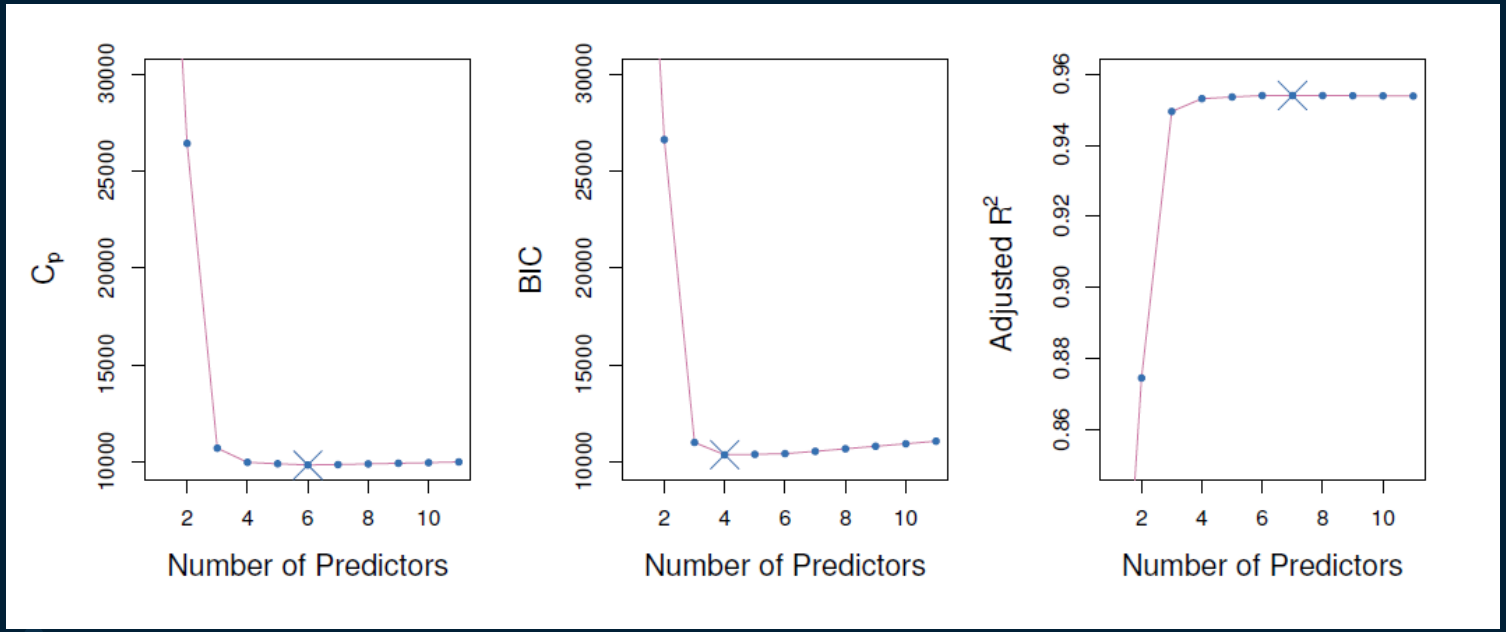
Computational approach

- Directly estimate test error using a validation set approach or cross-validation

Why would we ever not want to directly estimate test error??

Choosing the Optimal Model

Credit Data Example - Indirect Approaches



Indirect Approaches

Mallow's C_p

"Penalty" term for larger numbers of predictors

$$C_p = \frac{1}{n}(RSS + 2d\hat{\sigma}^2)$$

where d is the total number of parameters used and $\hat{\sigma}^2$ is an estimate of the variance of the error ϵ associated with each response measurement (often difficult to estimate)

Indirect Approaches

AIC/BIC

- AIC is defined for the large class of models fit by maximum likelihood:

$$AIC = -2logL + 2d$$

where L is the maximized value of the likelihood function for the estimated model

Note: it can be shown for a linear model that $-2logL = \frac{RSS}{\hat{\sigma}^2}$*, so AIC and* C_p *are equivalent*

Indirect Approaches

AIC/BIC

- Similar to AIC:

$$BIC = \frac{1}{n}(RSS + \log(n)\, d\, \hat{\sigma}^2)$$

Note: BICC replaces the $2d\hat{\sigma}^2$ term in the C_p measure with $\log(n)\, d\, \hat{\sigma}^2$

- Since $\log n > 2$ for any $n > 7$, the BIC statistic places a heavier penalty on models with many variables, generally resulting in smaller models than C_p

Indirect Approaches

Adjusted R^2

Reminder:

R^2 is the proportion of the variation in Y that is explained by variation in $X_1, X_2, \ldots, X_p$

$$R^2 = 1 - \frac{Residual\ Sum\ of\ Squares\ (RSS)}{Total\ Sum\ of\ Squares\ (TSS)} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Adjusted $R^2 = 1 - \frac{RSS/(n-p-1)}{TSS/(n-1)}$

"Penalty" term for larger numbers of predictors (p)

Indirect Approaches

Adjusted R^2

Advantages:

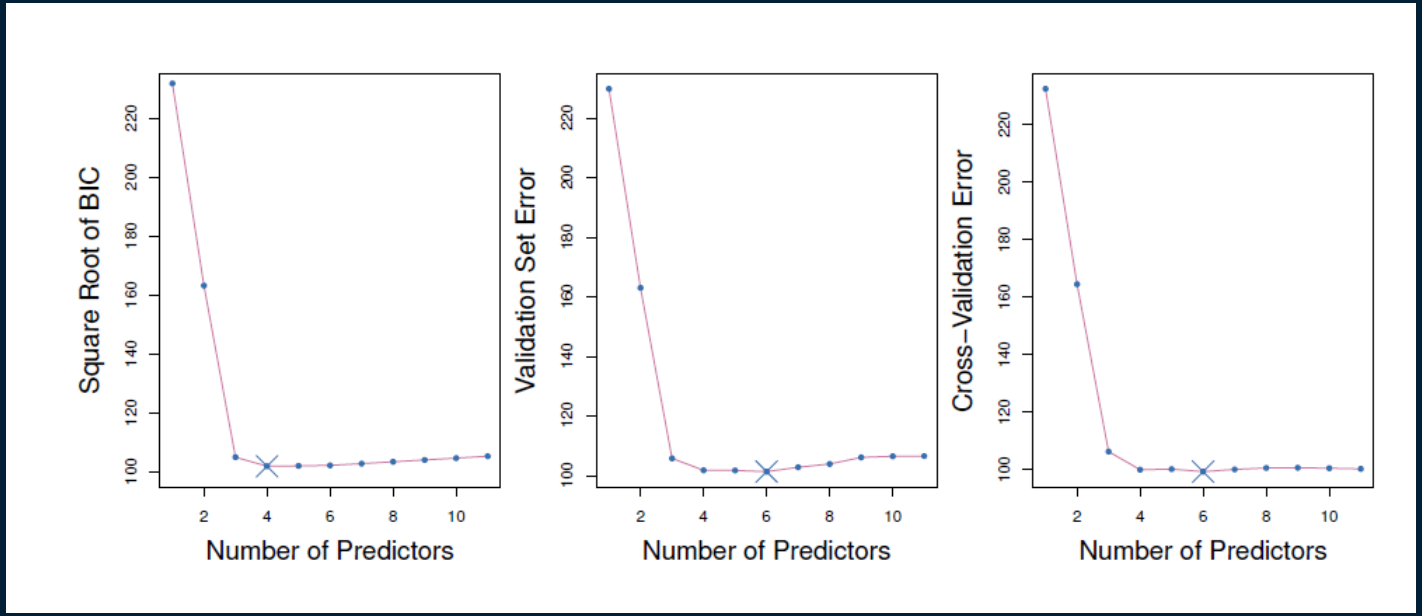
- Don't need to estimate $\hat{\sigma}^2$
- Works with models where $p > n$
- Easier to understand among non-statisticians

Direct Approach

Validation and Cross-Validation

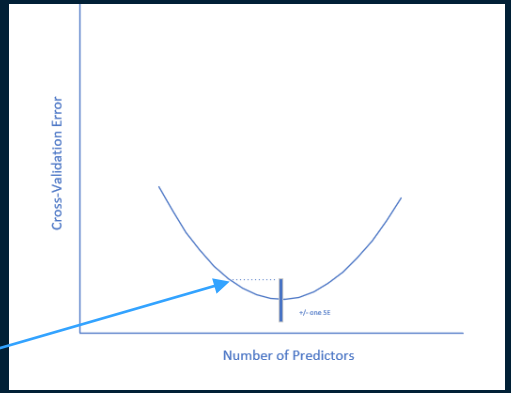
- Generally, the preferred approach if there is enough observations relative to the number of predictors
- Doesn't require an estimate of the error variance $\hat{\sigma}^2$
- Also, doesn't require the number of predictors (p) which is significant for model types where the number of predictors is not obvious
- See Module 4 PPT for details on this approach

Credit Data Example



“One Standard Error Rule”

- Calculate the standard error of the lowest point on the curve
  - Remember, for example that a 10-fold cross validation RSS score is the average of 10 estimates and we can find its standard error
- Select the smallest model for which the estimated test error is within one standard error of the lowest point on the curve:



Select this model

Shrinkage Methods

Shrinkage Methods

Overview

- Subset selection methods involve using least squares to fit a linear model that contains a subset of the predictors
  - That is, they attempt to find models with a subset of the predictors that in some manner improve on the test RSS or, equivalently, MSE
- Shrinkage methods take a different approach
  - Objective is to reduce *model variance*

Shrinkage Methods

Background and Rationale

Two reasons for building regression models:

- Prediction
- Inference (understanding relationship between variables)

Measured by test MSE

Interpreted by assessing the β_j coefficients

Shrinkage Methods

Background and Rationale

- β_j coefficients are themselves random variables, and thus have
  - Expected values
  - Variances (how much do our coefficient estimates change when we take a different set of training data?)
- Overall model accuracy is thus influenced by the variance in the β_j coefficients

Shrinkage Methods

Background and Rationale

Causes of high β_j variance:

- Correlated predictors (multicollinearity)
- Large number of predictors (high dimensional data)

Shrinkage regression models attempt to reduce this variance by reducing the number of predictors

- The two most popular of these techniques are *ridge regression* and the *lasso*

Ridge Regression and the Lasso

Ridge Regression and LASSO

Background

- Modifications to the linear regression “loss function” (function we are trying to minimize when fitting the model) to automate the process of driving regression coefficients of high-VIF factors towards zero

Ridge Regression and LASSO

Loss Functions

Linear Regression

$$Min\ SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Ridge Regression

$$Min\ SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} {\beta_j}^2$$

LASSO Regression

$$Min\ SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

These terms prevent large regression coefficients

Ridge Regression and LASSO

Mathematical Background: Vector Norms

Norm of a vector a measure of the length of a vector

Vector $\underline{b}' = [\, b_1, \ldots, b_m \,]$

ℓ_2 norm $\qquad \|b\|_2 = \sqrt{b_1^2 + \cdots + b_m^2}$

ℓ_1 norm $\qquad \|b\|_1 = |b_1| + \cdots + |b_m|$

Ridge Regression and LASSO

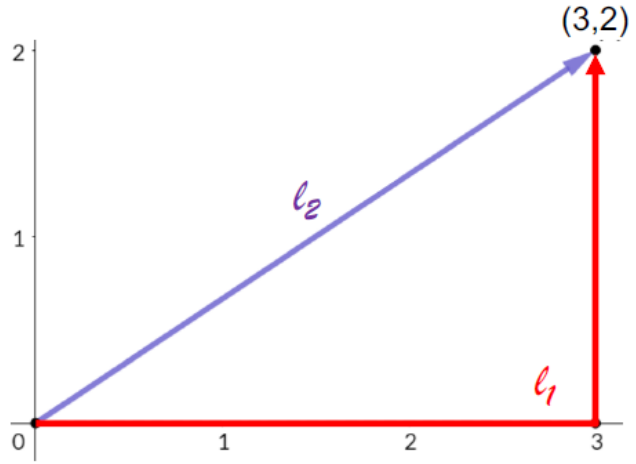
Mathematical Background: Vector Norms

Norm of a vector a measure of the length of a vector

Vector $\underline{b}' = [\, b_1, \ldots, b_m \,]$

ℓ_2 norm $\quad \|b\|_2 = \sqrt{3^2 + 2^2}$

ℓ_1 norm $\quad \|b\|_1 = |3| + |2|$



Ridge Regression and LASSO

Loss Functions

Linear Regression

$$Min\ SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Ridge Regression

$$Min\ SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} {\beta_j}^2$$

"L2 Regularization"

LASSO Regression

$$Min\ SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

"L1 Regularization"

Ridge Regression and LASSO

Loss Functions

- Last term is called *shrinkage penalty*
- λ is the tuning parameter (or *regularization* parameter)
  - If $\lambda = 0$, we get a linear regression model
- The value of λ is selected by cross validation
- Important point: predictors must be standardized before applying Ridge Regression or LASSO
  - Why?

$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2}}$$

Standardized by its standard deviation

Background: Scaling

There are two basic types of scaling:

- Normalization
  - Values shifted and re-scaled so they range between 0 and 1 (“min-max scaling)

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

- Standardization
  - Values are centered around a mean (typically, 0) with a unit standard deviation
  - “Number of standard deviations from the mean:

$$X' = \frac{X - \mu}{\sigma}$$

A Note on Standardization

Background on “Standardizing”

- There are two basic components to standardization:
  - Mean removal (setting the mean = 0)
  - Variance scaling (scaling the variable to represent the number of standard deviations from the man)
- The recently updated Scikit scaling parameter for ridge regression and Lasso has been changed from “normalize” to “StandardScalar” which has options for both of these operations:

sklearn.preprocessing.StandardScaler

class sklearn.preprocessing.StandardScaler(*, *copy=True*, *with_mean=True*, *with_std=True*) ¶ [source]

with_mean : *bool, default=True*

If True, center the data before scaling. This does not work (and will raise an exception) when attempted on sparse matrices, because centering them entails building a dense matrix which in common use cases is likely to be too large to fit in memory.

with_std : *bool, default=True*

If True, scale the data to unit variance (or equivalently, unit standard deviation).

A Note on Standardization

- We note that the formula for standardization is given here (and in our text) as:

$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2}}$$

- This scales the predictor to be the “number of standard deviations from 0”
- A legitimate question is whether it is better to standardize the predictor to be the “number of standard deviations from its mean”:

$$\tilde{x}_{ij} = \frac{x_{ij} - \bar{x}_j}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2}}$$

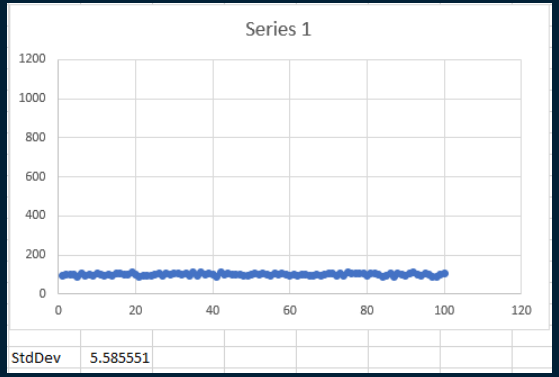
A Note on Standardization

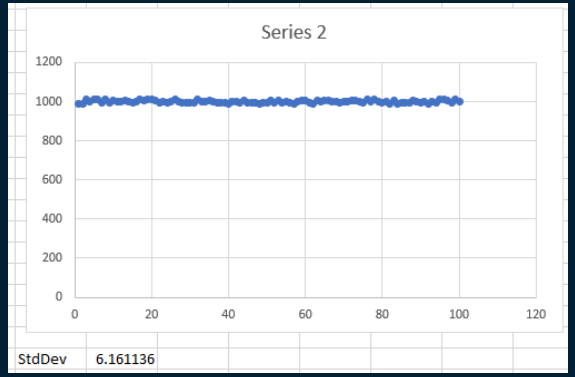
Back to Ridge Regression

- Back to the question … for Ridge Regression, should we scale the variance and subtract the mean or just scale the variance?
- We need to think about what we are trying to try to accomplish:
  - Ensuring that the Ridge Regression loss function is “fair” – the predictors that are “shrunk” to zero or near-zero are not just driven there because of their scale being low (and, thus, their coefficients being larger)
  - Given this, it seems obvious that we want to standardize by the variance AND subtract the mean …

A Note on Standardization

Back to Ridge Regression





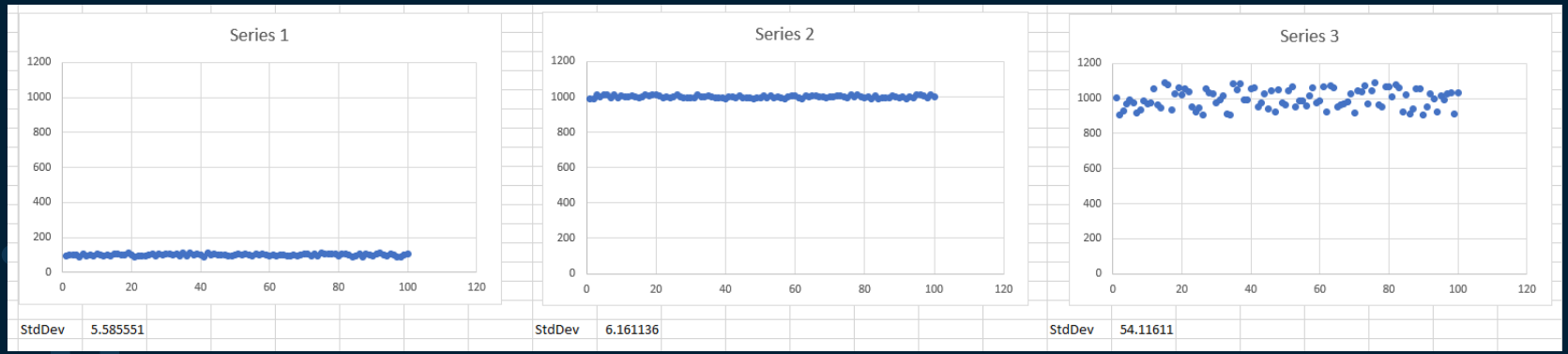
Two series with the same variance standardization, but different means

- If we don't subtract out the means, we will end up with Series 2 predictors having ten times the "weight" in the loss function as Series 1 (and, thus, much more likely to be "shrunk" out of the model

A Note on Standardization

Back to Ridge Regression

- However, … if we standardize all the means to zero, we lose a lot of interpretability of the coefficients
  - What does it mean for a coefficient to be positive or negative??
- Also, back to our example, if one predictor is scaled 10X a second predictor, would its standard deviation not also likely be scaled?



A Note on Standardization

Back to Ridge Regression

- So, what do we do, subtract out the mean or not?

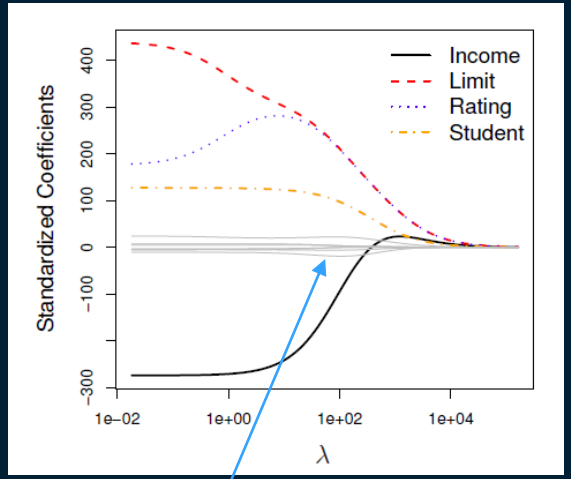
$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2}}$$

$$\tilde{x}_{ij} = \frac{x_{ij} - \bar{x}_j}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2}}$$

Ridge Regression and LASSO

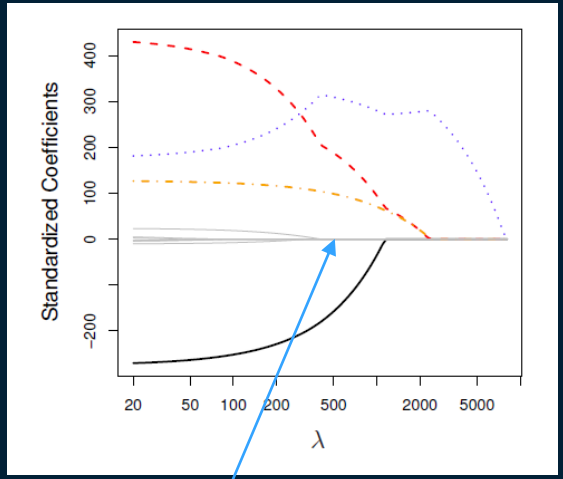
Credit Data Example

Ridge Regression



Higher λ values drive many coefficients "towards 0"

LASSO



Higher λ values drive many coefficients to 0

Ridge Regression and LASSO

Why Does LASSO Drive Coefficients to Zero and Not Ridge Regression?

It can be shown that the ridge regression and lasso solve the problems:

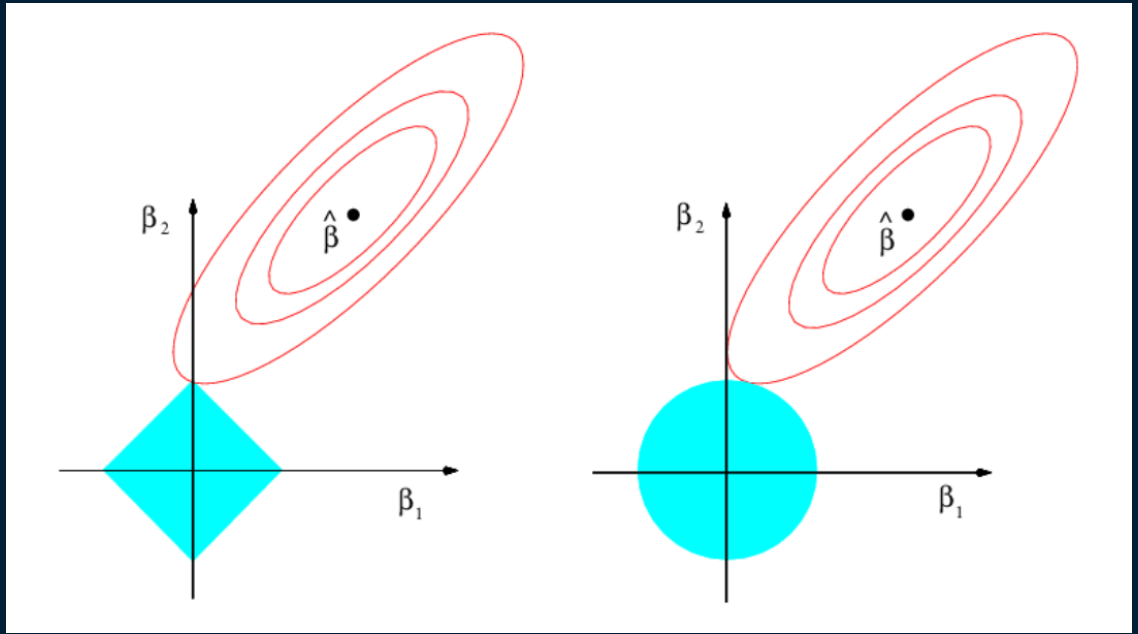
$$minimize\{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\} \text{ subject to } \sum_{j=1}^{p}{\beta_j}^2 < s$$

$$minimize\{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\} \text{ subject to } \sum_{j=1}^{p}|\beta_j| < s$$

Where s is a function of λ

Ridge Regression and LASSO

Why Does LASSO Drive Coefficients to Zero and Not Ridge Regression?



$$\sum_{j=1}^{p} |\beta_j| < s$$

$$\sum_{j=1}^{p} \beta_j{}^2 < s$$

Python Example

Ridge Regression Python Example

Hitters.csv Dataset

19 Predictors

Response Variable

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWalks	League	Division	PutOuts	Assists	Errors	Salary	NewLeague
2	293	66	1	30	29	14	1	293	66	1	30	29	14	A	E	446	33	20	NA	A
3	315	81	7	24	38	39	14	3449	835	69	321	414	375	N	W	632	43	10	475	N
4	479	130	18	66	72	76	3	1624	457	63	224	266	263	A	W	880	82	14	480	A
5	496	141	20	65	78	37	11	5628	1575	225	828	838	354	N	E	200	11	3	500	N
6	321	87	10	39	42	30	2	396	101	12	48	46	33	N	E	805	40	4	91.5	N
7	594	169	4	74	51	35	11	4408	1133	19	501	336	194	A	W	282	421	25	750	A
8	185	37	1	23	8	21	2	214	42	1	30	9	24	N	E	76	127	7	70	A
9	298	73	0	24	24	7	3	509	108	0	41	37	12	A	W	121	283	9	100	A
10	323	81	6	26	32	8	2	341	86	6	32	34	8	N	W	143	290	19	75	N
11	401	92	17	49	66	65	13	5206	1332	253	784	890	866	A	E	0	0	0	1100	A
12	574	159	21	107	75	59	10	4631	1300	90	702	504	488	A	E	238	445	22	517.143	A
13	202	53	4	31	26	27	9	1876	467	15	192	186	161	N	W	304	45	11	512.5	N
14	418	113	13	48	61	47	4	1512	392	41	205	204	203	N	E	211	11	7	550	N
15	239	60	0	30	11	22	6	1941	510	4	309	103	207	A	E	121	151	6	700	A
16	196	43	7	29	27	30	13	3231	825	36	376	290	238	N	E	80	45	8	240	N
17	183	39	3	20	15	11	3	201	42	3	20	16	11	A	W	118	0	0	NA	A
18	568	158	20	89	75	73	15	8068	2273	177	1045	993	732	N	W	105	290	10	775	N
19	190	46	2	24	8	15	5	479	102	5	65	23	39	A	W	102	177	16	175	A
20	407	104	6	57	43	65	12	5233	1478	100	643	658	653	A	W	912	88	9	NA	A
21	127	32	8	16	22	14	8	727	180	24	67	82	56	N	W	202	22	2	135	N
22	413	92	16	72	48	65	1	413	92	16	72	48	65	N	E	280	9	5	100	N
23	426	109	3	55	43	62	1	426	109	3	55	43	62	A	W	361	22	2	115	N
24	22	10	1	4	2	1	6	84	26	2	9	9	3	A	W	812	84	11	NA	A
25	472	116	16	60	62	74	6	1924	489	67	242	251	240	N	W	518	55	3	600	N
26	629	168	18	73	102	40	18	8424	2464	164	1008	1072	402	A	E	1067	157	14	776.667	A
27	587	163	4	92	51	70	6	2695	747	17	442	198	317	A	E	434	9	3	765	A
28	324	73	4	32	18	22	7	1931	491	13	291	108	180	N	E	222	3	3	708.333	N
29	474	129	10	50	56	40	10	2331	604	61	246	327	166	N	W	732	83	13	750	N
30	550	152	6	92	37	81	5	2308	633	32	349	182	308	N	W	262	329	16	625	N
31	513	137	20	90	95	90	14	5201	1382	166	763	734	784	A	W	267	5	3	900	A
32	313	84	9	42	30	39	17	6890	1833	224	1033	864	1087	A	W	127	221	7	NA	A

Ridge Regression Python Example

Approach

- Fit 100 ridge regression models with $10^{-2} < \lambda < 10^{10}$ and plot the coefficients as a function of λ
- Use a traditional validation partition approach to find an optimal value of λ
- Use a cross-validation approach to find an optimal value of λ

Ridge Regression Python Example

Set Up Dataframe for Modeling

Ridge Regression

In [92]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, RidgeCV, Lasso, LassoCV
from sklearn.metrics import mean_squared_error
```

Ridge Regression Python Example

Set Up Dataframe for Modeling

```
In [155]: df = pd.read_csv('Hitters.csv')
          df
```

Out[155]:

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWalks	League	Division	PutOuts	Assists	Errors	Salary
0	293	66	1	30	29	14	1	293	66	1	30	29	14	A	E	446	33	20	NaN
1	315	81	7	24	38	39	14	3449	835	69	321	414	375	N	W	632	43	10	475.00
2	479	130	18	66	72	76	3	1624	457	63	224	266	263	A	W	880	82	14	480.00
3	496	141	20	65	78	37	11	5628	1575	225	828	838	354	N	E	200	11	3	500.00
4	321	87	10	39	42	30	2	396	101	12	48	46	33	N	E	805	40	4	91.50
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
317	497	127	7	65	48	37	5	2703	806	32	379	311	138	N	E	325	9	3	700.00
318	492	136	5	76	50	94	12	5511	1511	39	897	451	875	A	E	313	381	20	875.00
319	475	126	3	61	43	52	6	1700	433	7	217	93	146	A	W	37	113	7	385.00
320	573	144	9	85	60	78	8	3198	857	97	470	420	332	A	E	1314	131	12	960.00
321	631	170	9	77	44	31	11	4908	1457	30	775	357	249	A	W	408	4	3	1000.00

322 rows × 20 columns

Ridge Regression Python Example

Drop Records With NaN for Salary

```
In [4]: d0 = df.dropna()
        d0
```

Out[4]:

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWalks	League	Division	PutOuts	Assists	Errors	Salary
1	315	81	7	24	38	39	14	3449	835	69	321	414	375	N	W	632	43	10	475.0
2	479	130	18	66	72	76	3	1624	457	63	224	266	263	A	W	880	82	14	480.0
3	496	141	20	65	78	37	11	5628	1575	225	828	838	354	N	E	200	11	3	500.0
4	321	87	10	39	42	30	2	396	101	12	48	46	33	N	E	805	40	4	91.5
5	594	169	4	74	51	35	11	4408	1133	19	501	336	194	A	W	282	421	25	750.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
317	497	127	7	65	48	37	5	2703	806	32	379	311	138	N	E	325	9	3	700.0
318	492	136	5	76	50	94	12	5511	1511	39	897	451	875	A	E	313	381	20	875.0
319	475	126	3	61	43	52	6	1700	433	7	217	93	146	A	W	37	113	7	385.0
320	573	144	9	85	60	78	8	3198	857	97	470	420	332	A	E	1314	131	12	960.0
321	631	170	9	77	44	31	11	4908	1457	30	775	357	249	A	W	408	4	3	1000.0

263 rows × 20 columns

Ridge Regression Python Example

Handle Categorical Variables

```
In [156]: y = d0['Salary']
          x0 = d0.drop(['Salary'], axis=1)
          x0.dtypes

Out[156]: AtBat          int64
          Hits           int64
          HmRun          int64
          Runs           int64
          RBI            int64
          Walks          int64
          Years          int64
          CAtBat         int64
          CHits          int64
          CHmRun         int64
          CRuns          int64
          CRBI           int64
          CWalks         int64
          League        object
          Division      object
          PutOuts        int64
          Assists        int64
          Errors         int64
          NewLeague     object
          dtype: object
```

```
In [157]: print(x0['League'].value_counts())
          print(x0['Division'].value_counts())
          print(x0['NewLeague'].value_counts())

          A    139
          N    124
          Name: League, dtype: int64
          W    134
          E    129
          Name: Division, dtype: int64
          A    141
          N    122
          Name: NewLeague, dtype: int64
```

Ridge Regression Python Example

Handle Categorical Variables

```
In [158]: X = pd.get_dummies(x0, columns = ['League', 'Division', 'NewLeague'], drop_first=True)
          X
```

Out[158]:

Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWalks	PutOuts	Assists	Errors	League_N	Division_W	NewLeague_N
81	7	24	38	39	14	3449	835	69	321	414	375	632	43	10	1	1	1
130	18	66	72	76	3	1624	457	63	224	266	263	880	82	14	0	1	0
141	20	65	78	37	11	5628	1575	225	828	838	354	200	11	3	1	0	1
87	10	39	42	30	2	396	101	12	48	46	33	805	40	4	1	0	1
169	4	74	51	35	11	4408	1133	19	501	336	194	282	421	25	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
127	7	65	48	37	5	2703	806	32	379	311	138	325	9	3	1	0	1
136	5	76	50	94	12	5511	1511	39	897	451	875	313	381	20	0	0	0
126	3	61	43	52	6	1700	433	7	217	93	146	37	113	7	0	1	0
144	9	85	60	78	8	3198	857	97	470	420	332	1314	131	12	0	0	0
170	9	77	44	31	11	4908	1457	30	775	357	249	408	4	3	0	1	0

19 columns

Ridge Regression Python Example

Create an Array of 100 Lambda Values from 0.01 to 10^10

```
In [159]: lambdas = 10**np.linspace(10, -2, 100)
          lambdas

Out[159]: array([1.00000000e+10, 7.56463328e+09, 5.72236766e+09, 4.32876128e+09,
                 3.27454916e+09, 2.47707636e+09, 1.87381742e+09, 1.41747416e+09,
                 1.07226722e+09, 8.11130831e+08, 6.13590727e+08, 4.64158883e+08,
                 3.51119173e+08, 2.65608778e+08, 2.00923300e+08, 1.51991108e+08,
                 1.14975700e+08, 8.69749003e+07, 6.57933225e+07, 4.97702356e+07,
                 3.76493581e+07, 2.84803587e+07, 2.15443469e+07, 1.62975083e+07,
                 1.23284674e+07, 9.32603347e+06, 7.05480231e+06, 5.33669923e+06,
                 4.03701726e+06, 3.05385551e+06, 2.31012970e+06, 1.74752840e+06,
                 1.32194115e+06, 1.00000000e+06, 7.56463328e+05, 5.72236766e+05,
                 4.32876128e+05, 3.27454916e+05, 2.47707636e+05, 1.87381742e+05,
                 1.41747416e+05, 1.07226722e+05, 8.11130831e+04, 6.13590727e+04,
                 4.64158883e+04, 3.51119173e+04, 2.65608778e+04, 2.00923300e+04,
                 1.51991108e+04, 1.14975700e+04, 8.69749003e+03, 6.57933225e+03,
                 4.97702356e+03, 3.76493581e+03, 2.84803587e+03, 2.15443469e+03,
                 1.62975083e+03, 1.23284674e+03, 9.32603347e+02, 7.05480231e+02,
                 5.33669923e+02, 4.03701726e+02, 3.05385551e+02, 2.31012970e+02,
                 1.74752840e+02, 1.32194115e+02, 1.00000000e+02, 7.56463328e+01,
                 5.72236766e+01, 4.32876128e+01, 3.27454916e+01, 2.47707636e+01,
                 1.87381742e+01, 1.41747416e+01, 1.07226722e+01, 8.11130831e+00,
                 6.13590727e+00, 4.64158883e+00, 3.51119173e+00, 2.65608778e+00,
                 2.00923300e+00, 1.51991108e+00, 1.14975700e+00, 8.69749003e-01,
                 6.57933225e-01, 4.97702356e-01, 3.76493581e-01, 2.84803587e-01,
                 2.15443469e-01, 1.62975083e-01, 1.23284674e-01, 9.32603347e-02,
                 7.05480231e-02, 5.33669923e-02, 4.03701726e-02, 3.05385551e-02,
                 2.31012970e-02, 1.74752840e-02, 1.32194115e-02, 1.00000000e-02])
```

Ridge Regression Python Example

Use the Array of Lambdas to Fit 100 Models

Fit 100 ridge regression models, one for each alpha (normalizing all columns

In [79]:
```
model = Ridge(normalize=True)
coefs = []
```

In [80]:
```
for l in lambdas:
    model.set_params(alpha = l)  # SKLearn Ridge model uses alpha, not lambda
    model.fit(X,y)
    coefs.append(model.coef_)
```

In [160]:
```
coefs[0] # cofs is a list of 1D arrays (vectors)
```

Out[160]:
```
array([ 1.20896017e-10,  4.38543910e-10,  1.76709456e-09,  7.41610833e-10,
        7.83374132e-10,  9.21983126e-10,  3.77051181e-09,  1.03800944e-11,
        3.82017112e-11,  2.88093612e-10,  7.66411600e-11,  7.90953571e-11,
        8.36822421e-11,  4.84228507e-11,  7.90920206e-12, -3.68777713e-11,
       -1.28820467e-09, -1.73394491e-08, -2.55917587e-10])
```

Ridge Regression Python Example

Create a Dataframe of the Model Coefficients for Each Lambda

Store coefs as the rows in a dataframe model1_coefs

In [164]:
```
model1_coefs = pd.DataFrame(coefs)
model1_coefs.columns = X.columns
model1_coefs.index = lambdas
model1_coefs.index.name = 'lambda'
model1_coefs.round(3)
```

Out[164]:

lambda	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWalks	PutOuts	Assists	Errors	League_N	Div
10000000000.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-0.00	-0.00	
7564633275.55	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-0.00	-0.00	
5722367659.35	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-0.00	-0.00	
4328761281.08	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-0.00	-0.00	
3274549162.88	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-0.00	-0.00	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
0.03	-1.03	3.75	-0.92	0.69	0.54	4.06	-10.58	-0.01	0.16	0.72	0.40	0.30	-0.40	0.27	0.21	-3.84	57.63	
0.02	-1.19	4.22	-0.60	0.49	0.43	4.37	-10.89	-0.02	0.17	0.72	0.46	0.32	-0.46	0.27	0.23	-3.85	58.94	
0.02	-1.33	4.69	-0.22	0.25	0.31	4.66	-10.94	-0.03	0.18	0.72	0.53	0.34	-0.51	0.28	0.25	-3.84	59.91	
0.01	-1.47	5.14	0.20	-0.01	0.19	4.93	-10.75	-0.04	0.19	0.71	0.61	0.36	-0.55	0.28	0.27	-3.81	60.61	
0.01	-1.58	5.55	0.63	-0.28	0.06	5.17	-10.36	-0.05	0.19	0.68	0.69	0.38	-0.59	0.28	0.29	-3.77	61.13	

100 rows × 19 columns

Ridge Regression Python Example

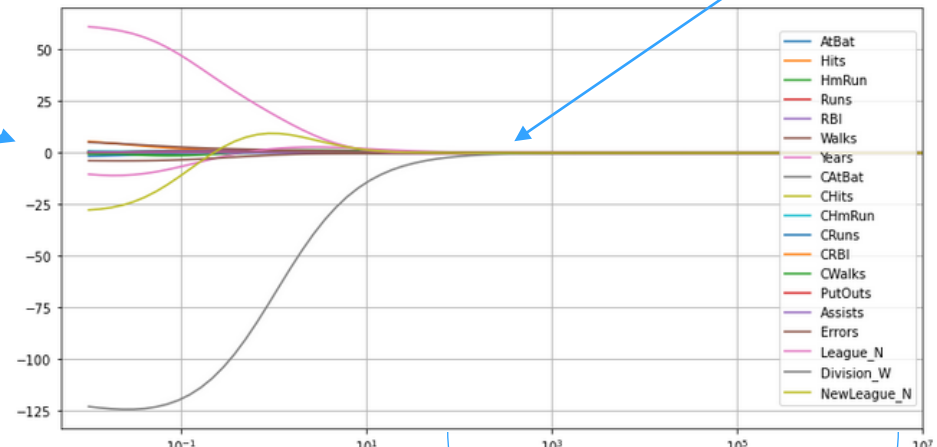
Plot the Model Coefficients as a Function of Lambda

All coefficients shrink to zero as lambda increases. Why??

19 curves – one for each predictor

```
In [165]: model1_coefs.plot(figsize=(12,6), grid=True, logx=True, xlim = (0.01/2, 10**7))
          plt.legend(loc='right')

Out[165]: <matplotlib.legend.Legend at 0x222719c93a0>
```



"Over shrinking"

Ridge Regression Python Example

Evaluate Model With Lambda = 4

Lambda = 4

```
In [184]: X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.5, random_state=1)

In [200]: model2a = Ridge(alpha = 4, normalize = True)
          model2a.fit(X_train, y_train)
          pred2a = model2a.predict(X_test)
          mse2a = mean_squared_error(y_test, pred2a)
          mse2a

Out[200]: 106216.52238005561

In [201]: np.sqrt(mse2a)

Out[201]: 325.90876388961317
```

sklearn uses “alpha” terminology, not lambda

Ridge Regression Python Example

Evaluate Model With Lambda = 1,000,000,000

Lambda = 10^9 (overshrinking case)

```
In [203]: model2b = Ridge(alpha = 10**9, normalize = True)
          model2b.fit(X_train, y_train)
          pred2b = model2b.predict(X_test)
          mse2b = mean_squared_error(y_test, pred2b)
          mse2b

Out[203]: 172862.234750706
```

Ridge Regression Python Example

Evaluate Model With Lambda = 0

Lambda = 0 (linear regression case)

In [204]:
```
model2c = Ridge(alpha = 0, normalize = True)
model2c.fit(X_train, y_train)
pred2c = rr4.predict(X_test)
mse4 = mean_squared_error(y_test, pred4)
mse4
```

Out[204]: 116690.46856660102

Ridge Regression Python Example

MSE as a Function of Lambda

Mean squre prediction error varies with lambda

In [221]:
```
model3 = Ridge(normalize = True)
mses = []
for l in lambdas:
    model3.set_params(alpha = l)
    model3.fit(X_train, y_train)
    mses.append(mean_squared_error(y_test, model3.predict(X_test)))
model3.set_params(alpha = 0)
model3.fit(X_train, y_train)
linear_regression_mse = mean_squared_error(y_test, model3.predict(X_test))
```

In [223]:
```
model3_mpses = pd.DataFrame(mses, columns = ['MPSE'])
model3_mpses.index = lambdas
model3_mpses.index.name = "lambda"
model3_mpses
```

Out[223]:

lambda	MPSE
10000000000.00	172862.24
7564633275.55	172862.24
5722367659.35	172862.24
4328761281.08	172862.24
3274549162.88	172862.24
...	...
0.03	102144.43
0.02	102357.91
0.02	102591.66
0.01	102831.22
0.01	103069.74

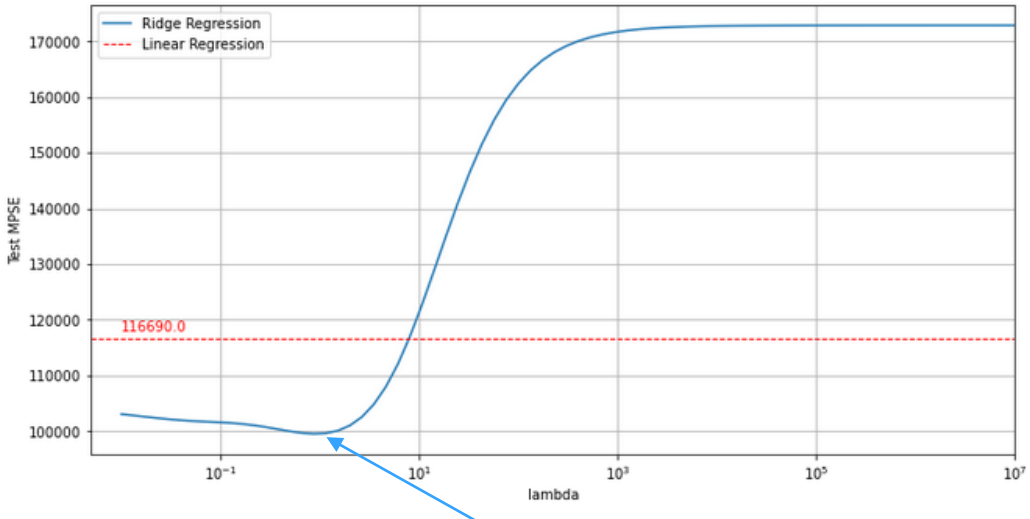
100 rows × 1 columns

Ridge Regression Python Example

Plot MSPE as a Function of Lambda

```
In [224]: model3_mpses.plot(figsize = (12,6), grid=True, logx=True, xlim = (0.01/2, 10**7))  # MPSE as function of lambda
          plt.axhline(y = linear_regression_mse, linestyle = '--', c='r', linewidth = 1)
          plt.annotate(round(mse4,0), xy = (0.01, 1.01*mse4), c = 'r')
          plt.ylabel("Test MPSE")
          plt.legend(("Ridge Regression", "Linear Regression"))

Out[224]: <matplotlib.legend.Legend at 0x222705ce0a0>
```



Best Lambda

Ridge Regression Python Example

Find Best Lambda Minimizing Test MSPE

Find best lambda minimizing test MPSE

```
mses.index(min(mses))
```

```
83
```

```
lambdas[83]
```

```
0.8697490026177834
```

```
model3_mpses.MPSE.iloc[83]
```

```
99541.51776483622
```

```
model3_mpses.iloc[81:86]
```

lambda	MPSE
1.52	100097.70
1.15	99641.81
0.87	99541.52
0.66	99691.76
0.50	99997.35

Ridge Regression Python Example

Use 10-Fold Cross Validation to Find Best Lambda

Ridge regression 10-fold cross validation to find best alpha (minimizing training MSE)

```
]: ridgecv = RidgeCV(alphas = lambdas, cv = 10, normalize = True, scoring = "neg_mean_squared_error")
   ridgecv.fit(X_train, y_train)
   ridgecv.alpha_  # best alpha (minimizing training MSE)

]: 0.6579332246575682
```

Ridge Regression Python Example

Use 10-Fold Cross Validation to Find Best Lambda

Test MSE of best alpha

```
yhatcv = ridgecv.predict(X_test)
best_mspe = mean_squared_error(y_test, yhatcv)
best_mspe
```

```
99691.75835893235
```

```
np.sqrt(best_mpse)
```

```
315.740017037645
```

Ridge Regression Python Example

Use 10-Fold CV With Training Data to Find Best Lambda

Coefficients of best RR model (when fitting training set)

```
model4 = Ridge(alpha = ridgecv.alpha_, normalize = True)
model4.fit(X_train, y_train)
model4_coefs = pd.DataFrame(model4.coef_, index = X.columns, columns = ['Ridge-Coeff'])
model4_coefs
```

	Ridge-Coeff
AtBat	0.01
Hits	0.82
HmRun	-0.04
Runs	0.74
RBI	1.28
Walks	2.01
Years	1.49
CAtBat	0.01
CHits	0.05
CHmRun	0.49
CRuns	0.10
CRBI	0.12
CWalks	0.10
PutOuts	0.27
Assists	-0.02
Errors	-0.22
League_N	17.71
Division_W	-88.02
NewLeague_N	10.26

Ridge Regression Python Example

Use 10-Fold CV With Full Dataset to Find Best Lambda

Coefficients of best RR model (when using full dataset)

```
model4.fit(X, y)
model4_coefs = pd.DataFrame(model4.coef_, index = X.columns, columns = ['Ridge-Coeff'])
model4_coefs
```

	Ridge-Coeff
AtBat	0.07
Hits	0.89
HmRun	0.51
Runs	1.08
RBI	0.88
Walks	1.66
Years	1.15
CAtBat	0.01
CHits	0.06
CHmRun	0.41
CRuns	0.12
CRBI	0.12
CWalks	0.05
PutOuts	0.17
Assists	0.03
Errors	-1.46
League_N	23.19
Division_W	-81.94
NewLeague_N	8.87

A Note on Using Both Test/Training Partitions and CV

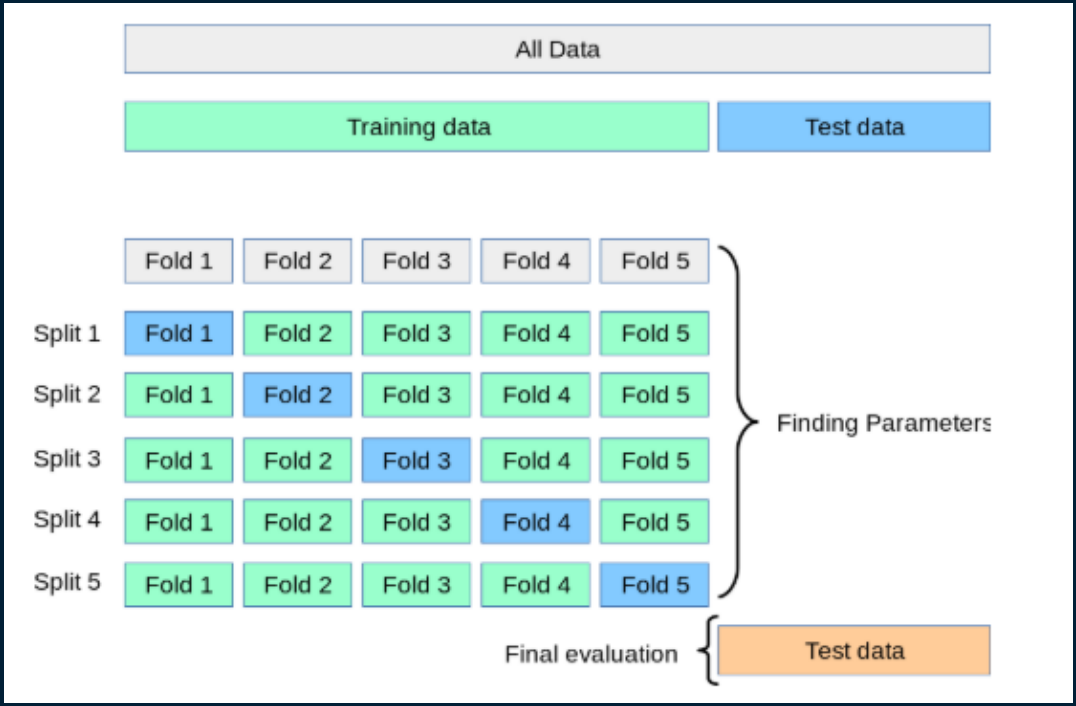
- Although CV techniques can be used as an alternative to a simple test/training partition approach, when used to fit the hyperparameters of a model (in this case, lambda), it is a best practice to use both.
- Recall that it is frequently useful to partition a dataset into three partitions:
  - Training partition – used to fit the model
  - Validation partition – used to select the best model
  - Test partition – used to assess the performance of the final chosen model. USED ONLY ONCE at the end of the process for assessment only

A Note on Using Both Test/Training Partitions and CV

- It is particularly important to have the third test partition when you are using the validation data to make decisions among many different models
- When we are using cross-validation to select a hyperparameter (in this case, lambda), we are doing exactly that – looking at many different models and comparing them
- Thus, it is a best practice to save a final “hold-out” set as depicted in the graphics on the next two pages

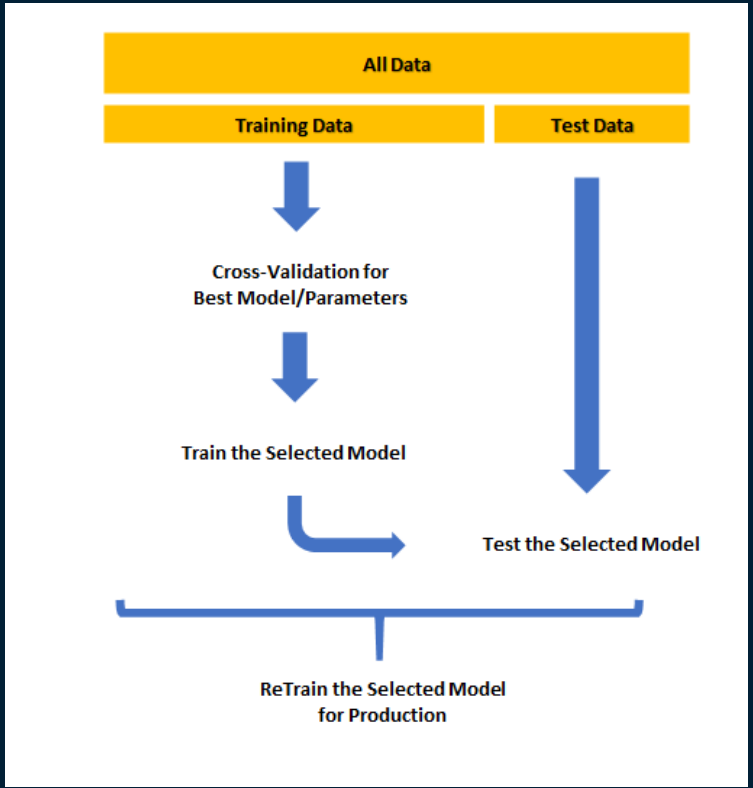
A Note on Using Both Test/Training Partitions and CV

Best Practice When Selecting Hyperparameters (Lambda)



A Note on Using Both Test/Training Partitions and CV

Best Practice When Selecting Hyperparameters (Lambda)



Dimension Reduction

Dimension Reduction

- As previously discussed, high dimensionality (large number of variables) that include redundant (correlated) inputs can degrade your analysis:
  - Destabilizes parameter estimates
  - Increases risk of overfitting
  - Confounds interpretation
  - Increases computation time
  - Increases scoring effort
  - Increases cost of data collection and augmention

Dimension Reduction

Feature Engineering - Objectives

- Simplify models
- Shorter training times
- Improved generalization
- Greater ability to visualize feature space

Reducing Data Complexity

Introductory Example – brand_ratings.csv

- Consumer brand perception survey results on coffee brands
- Scores of 1 to 10 on questions such as “How trendy is brand a”?

	perform	leader	latest	fun	serious	bargain	value	trendy	rebuy	brand
0	2	4	8	8	2	9	7	4	6	a
1	1	1	4	7	1	1	1	2	2	a
2	2	3	5	9	2	9	5	1	6	a
3	1	6	10	8	3	4	5	2	1	a
4	1	1	5	8	1	9	9	1	1	a
...	...	...	...	...	...	...	...	...	...	...
995	2	2	3	6	4	8	5	1	2	j
996	3	2	6	7	1	3	3	2	1	j
997	1	1	10	10	1	6	5	5	2	j
998	1	1	7	5	1	1	2	5	1	j
999	7	4	7	8	4	1	2	5	1	j

1000 rows × 10 columns

Brand Rating Example

Survey Questions

Perceptual Adjective (Column Name	Survey Question
Perform	Brand has strong performance
Leader	Brand is a leader in the field
Latest	Brand is fun
Fun	Brand has the latest products
Serious	Brand is serious
Bargain	Brand products are a bargain
Value	Brand products are a good value
Trendy	Brand is trendy
Rebuy	I would buy from Brand again

Reducing Data Complexity

Brand Rating Example

```
1 brand_ratings.describe()
```

	perform	leader	latest	fun	serious	bargain	value	trendy	rebuy
count	1000.000000	1000.000000	1000.000000	1000.00000	1000.000000	1000.000000	1000.00000	1000.000000	1000.000000
mean	4.488000	4.417000	6.195000	6.06800	4.323000	4.259000	4.33700	5.220000	3.727000
std	3.203454	2.608432	3.078059	2.74425	2.778199	2.667027	2.39858	2.742101	2.544592
min	1.000000	1.000000	1.000000	1.00000	1.000000	1.000000	1.00000	1.000000	1.000000
25%	1.000000	2.000000	4.000000	4.00000	2.000000	2.000000	2.00000	3.000000	1.000000
50%	4.000000	4.000000	7.000000	6.00000	4.000000	4.000000	4.00000	5.000000	3.000000
75%	7.000000	6.000000	9.000000	8.00000	6.000000	6.000000	6.00000	7.000000	5.000000

Brand Rating Example

Rescaling the Data

- Good practice to rescale the raw data by subtracting the mean and dividing by the standard deviation.

```
scaled_brand_ratings = scaler.fit_transform(brand_ratings.drop('brand',1))
scaled_brand_ratings = pd.DataFrame(scaled_brand_ratings, columns = brand_ratings.drop('brand',1).columns)
scaled_brand_ratings['brand'] = brand_ratings['brand']
scaled_brand_ratings
```

	perform	leader	latest	fun	serious	bargain	value	trendy	rebuy	brand
0	-0.777050	-0.159946	0.586702	0.704370	-0.836572	1.778524	1.110796	-0.445137	0.893714	a
1	-1.089370	-1.310638	-0.713468	0.339789	-1.196697	-1.222571	-1.391936	-1.174870	-0.679034	a
2	-0.777050	-0.543510	-0.388426	1.068950	-0.836572	1.778524	0.276552	-1.539736	0.893714	a
3	-1.089370	0.607182	1.236787	0.704370	-0.476446	-0.097160	0.276552	-1.174870	-1.072221	a
4	-1.089370	-1.310638	-0.388426	0.704370	-1.196697	1.778524	1.945040	-1.539736	-1.072221	a
...	...	...	...	...	...	...	...	...	...	...
995	-0.777050	-0.927074	-1.038511	-0.024791	-0.116321	1.403387	0.276552	-1.539736	-0.679034	j
996	-0.464731	-0.927074	-0.063383	0.339789	-1.196697	-0.472297	-0.557692	-1.174870	-1.072221	j
997	-1.089370	-1.310638	1.236787	1.433531	-1.196697	0.653113	0.276552	-0.080271	-0.679034	j
998	-1.089370	-1.310638	0.261659	-0.389372	-1.196697	-1.222571	-0.974814	-0.080271	-1.072221	j
999	0.784546	-0.159946	0.261659	0.704370	-0.116321	-1.222571	-0.974814	-0.080271	-1.072221	j

1000 rows × 10 columns

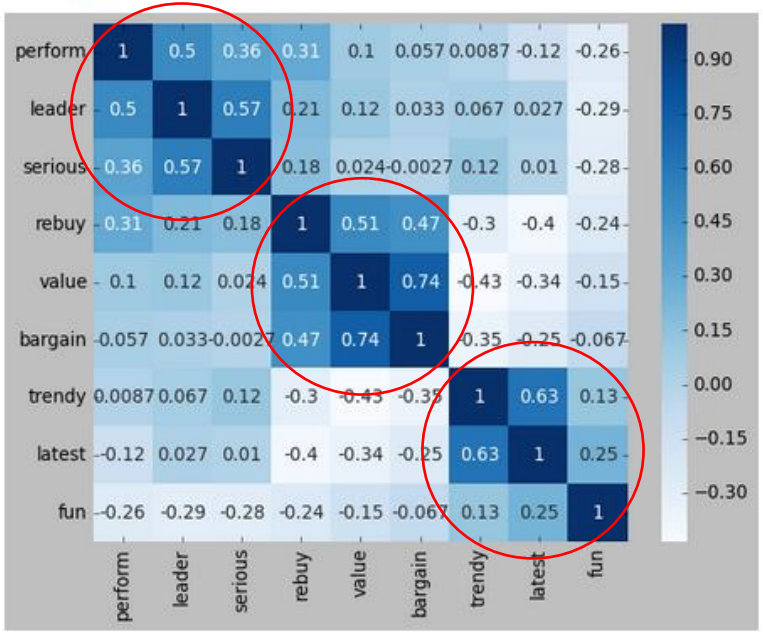
Brand Rating Example

Correlation Matrix

Order the variables by “similarity”

```
corr_matrix = scaled_brand_ratings.corr().sort_values('perform', ascending=False)
sorted_index = corr_matrix.index
sorted_corr_matrix = pd.DataFrame(corr_matrix, columns = sorted_index)
sns.heatmap(sorted_corr_matrix, cmap="Blues", annot=True)
```

<AxesSubplot:>



Brand Rating Example

Aggregate Mean Ratings by Brand

```
1 scaled_brand_ratings.groupby('brand').mean()
```

brand	perform	leader	latest	fun	serious	bargain	value	trendy	rebuy
a	-0.886362	-0.528168	0.411179	0.656974	-0.919400	0.214203	0.184785	-0.525407	-0.596465
b	0.931336	1.071294	0.726470	-0.972701	1.183733	0.041640	0.151415	0.740679	0.237092
c	0.650249	1.163350	-0.102388	-0.845098	1.223346	-0.607347	-0.440898	0.025541	-0.132504
d	-0.680231	-0.593373	0.352671	0.186665	-0.692521	-0.881197	-0.933102	0.737030	-0.494236
e	-0.564673	0.192933	0.456685	0.296039	0.042135	0.551826	0.418373	0.138649	0.036566
f	-0.058716	0.269645	-1.262790	-0.218019	0.589525	0.874444	1.023200	-0.813652	1.357675
g	0.918843	-0.167617	-1.285543	-0.516975	-0.534066	0.896953	1.256789	-1.277032	1.361607
h	-0.014991	-0.298029	0.502191	0.715307	-0.141529	-0.738645	-0.782938	0.864733	-0.604328
i	0.334806	-0.321043	0.355922	0.412705	-0.148732	-0.254718	-0.803794	0.591083	-0.203278
j	-0.630260	-0.788991	-0.154395	0.285102	-0.602490	-0.097160	-0.073831	-0.481624	-0.962129

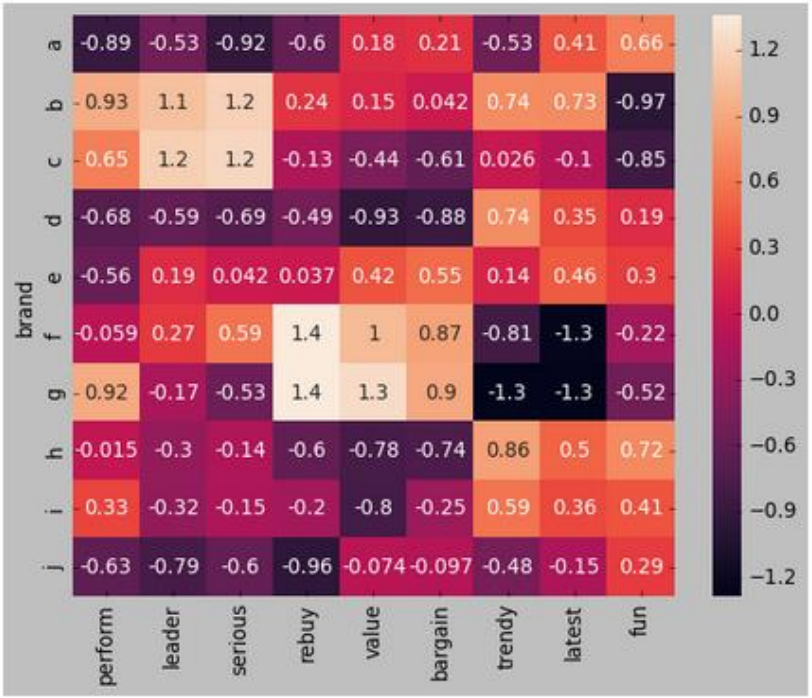
Brand Rating Example

Aggregate Mean Ratings by Brand

Which brand attributes appear to be similar when viewed across brand averages?

```
[ ]: 1 brand_ratings = pd.DataFrame(scaled_brand_ratings.groupby('brand').mean(), columns = sorted_index)
     2 sns.heatmap(brand_ratings, annot=True)

[ ]: <AxesSubplot:ylabel='brand'>
```



Brand Rating Example

Motivating Objective

Given that there appear to be three basic underlying attributes, how can we summarize and quantify these underlying attributes in a way that retains most of the important information but reduces the number of dimensions from nine to three?

Dimension Reduction

Feature Engineering Techniques

Feature engineering (or feature extraction) creates new features (predictors) from the initial set of data. The objective is to encapsulate the central properties of a dataset and represent it in a low-dimensional space

- Principal Components Analysis (PCA)
- Singular Value Decomposition
- Exploratory Factor Analysis (EFA)
- Clustering

Principal Components Analysis

Principal Components Analysis

Introduction

- A classic statistical technique invented in 1901
- In data science, it is used to reduce the dimensionality (number or relevant attributes) of a dataset for visualization and modeling purposes
- Attempts to identify attributes that have significance covariance and use those relationships to reduced the number of dimensions in the data while minimizing the loss of information. For example:
  - In the CARS dataset, there is high covariance between length and weight
  - High covariance is expected between restaurant checks and tips

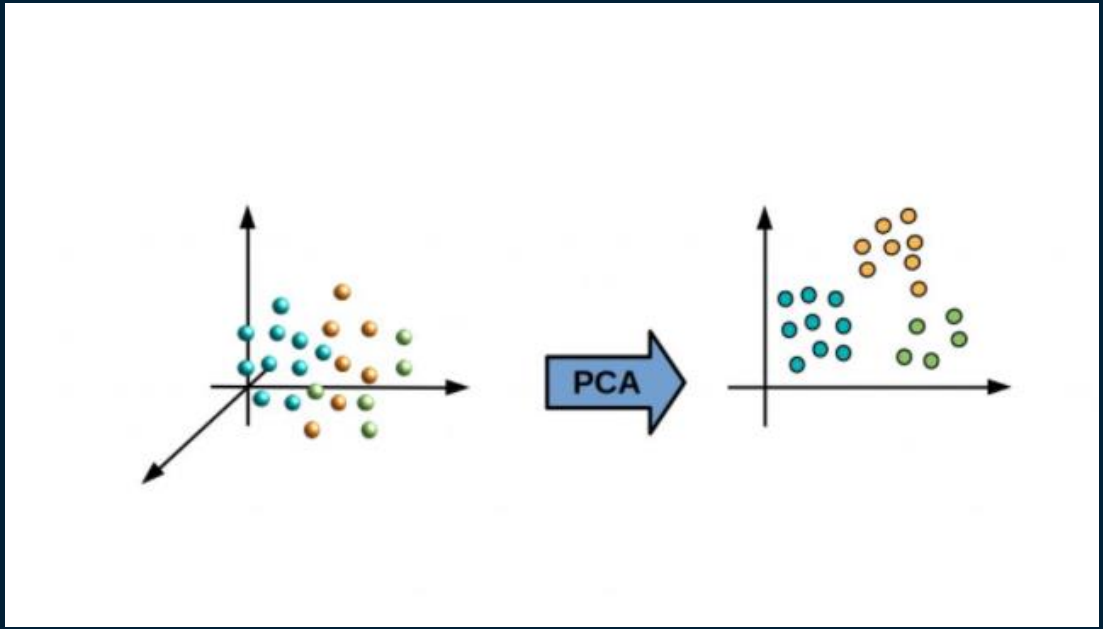
Principal Components Analysis

Objectives

- Combine multiple numeric predictor variables into a smaller set of variables which are a linear combination of or the original sets
  - Smaller set of variables are referred to as “principal components”
  - Objective is to select this smaller set so as to “explain” most of the variability in the full set
- Basic idea is that each of the n observations lies in a p-dimensional space, but not all of these dimensions are equally interesting.
  - PCA attempts to find a lower-dimensional representation of the data that contains the maximum amount of “interesting” information

Principal Components Analysis

Basic Principle



Principal Components Analysis

Approach

The first principal component (Z_1) of a set of attributes $X_1, X_2, \dots, X_p$ is the normalized set of features

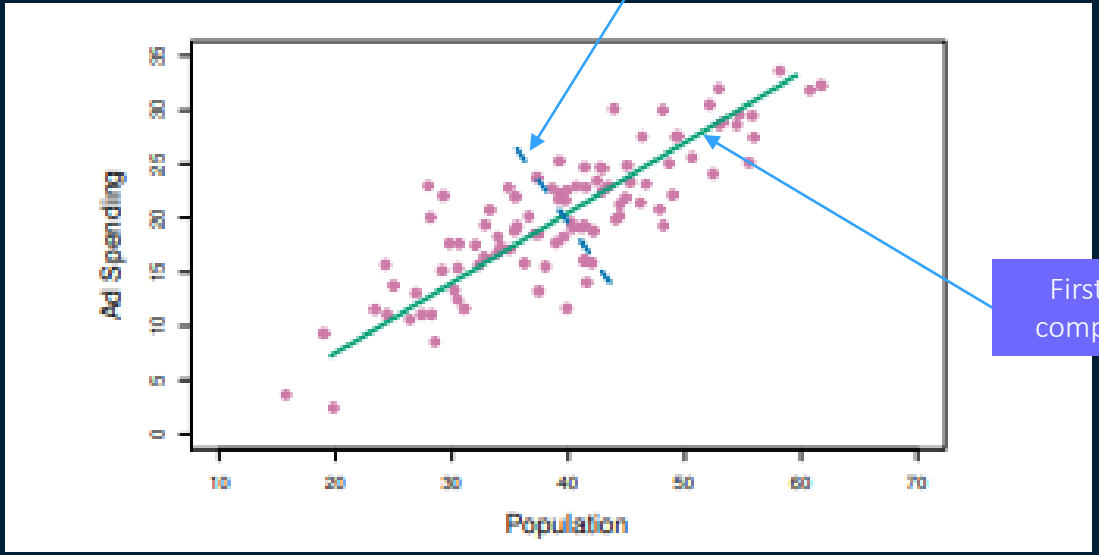
$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \cdots + \phi_{p1}X_p$$

that has the largest variance

by normalized we mean that $\sum_{j=1}^{p} {\phi_{j1}}^2 = 1$

The elements $\phi_{11}, \dots, \phi_{p1}$ are referred to as the *loadings* of the first principal component

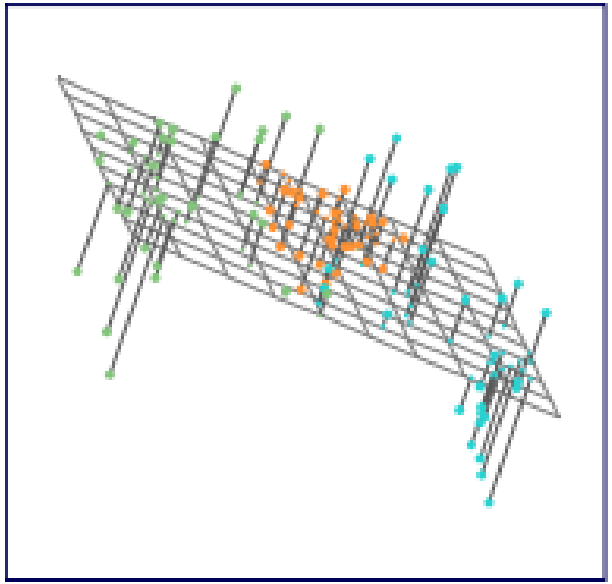
PCA Example

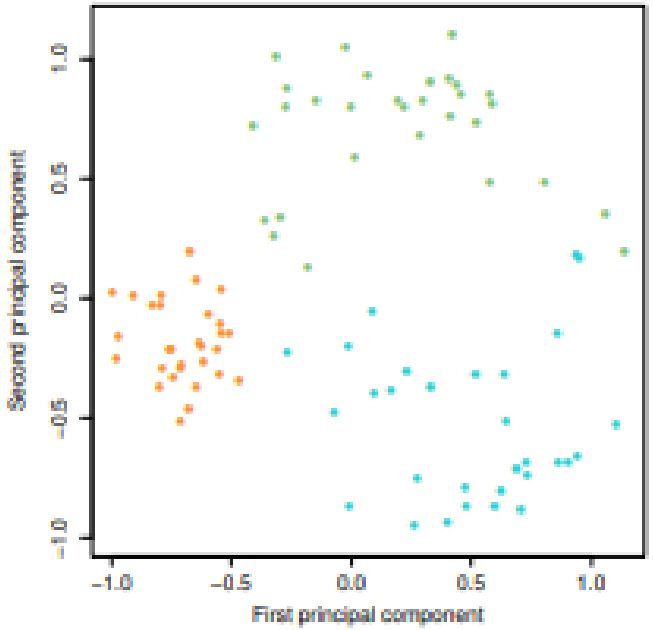


For the first principal component, we are looking for the line that minimizes the sum of the squared distances from the line to each point

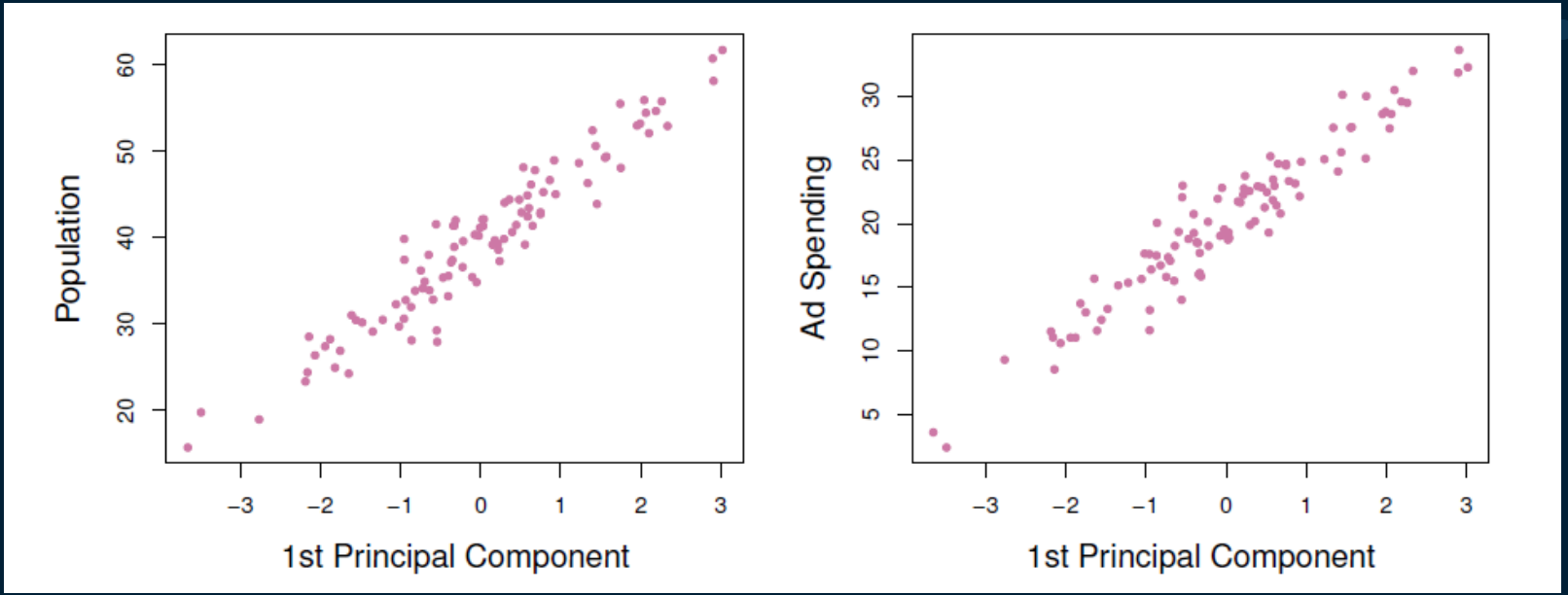
The second and subsequent principal components are similarly defined with the additional constraint that it must be uncorrelated with the other previously defined principal components

PCA Example



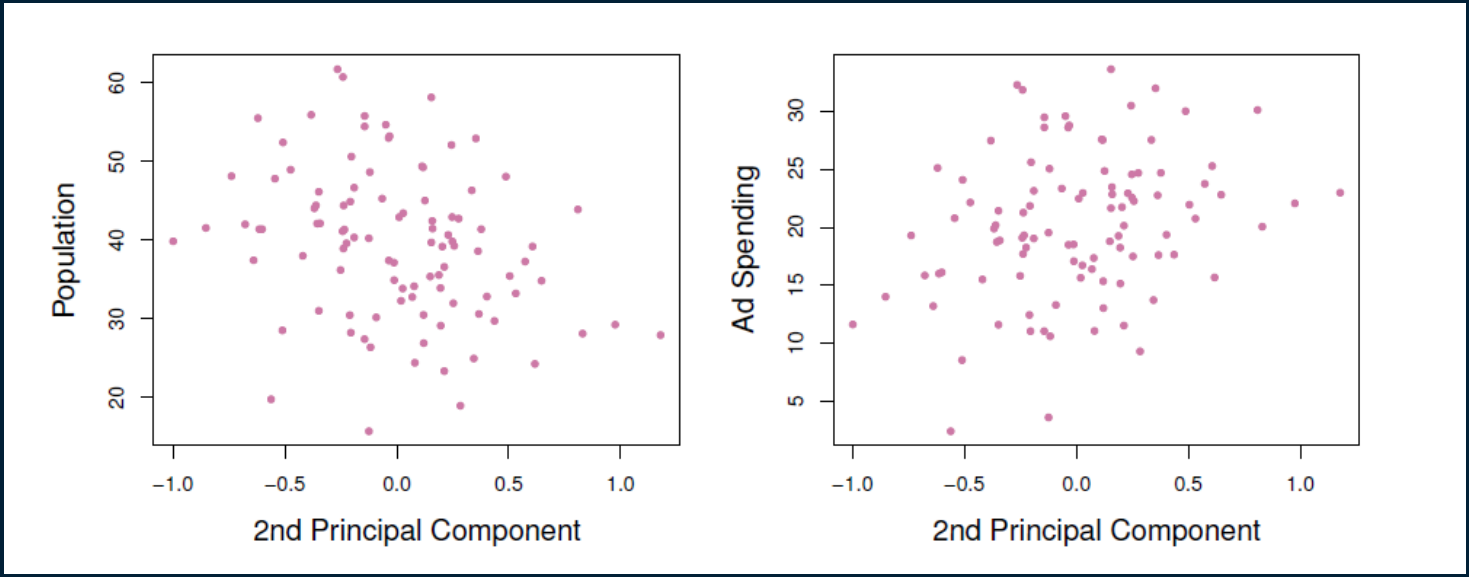

Second principal component
First principal component
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0

First Principal Component



First principal component is highly correlated with both population and ad spending, and thus summarizes these two predictors well

Second Principal Component



Very little relationship with the second principal component

Data Preparation Best Practices

Data Preparation Best Practices

Common Data Problems

Data Problem	Best Practice