LECTURE 5: LASSO REGULARIZATION AND FEATURE SELECTION

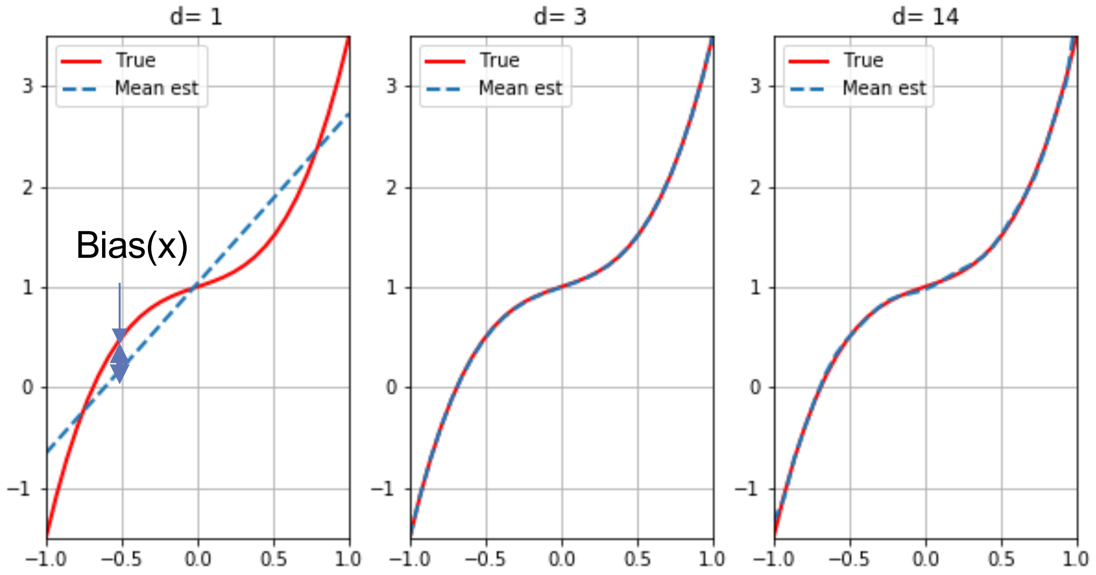
Ehsan Aryafar
earyafar@pdx.edu
http://web.cecs.pdx.edu/~aryafare/ML.html

Recall: Bias and Variance

- To understand potential problem of using a large model class introduce two key quantities:
- Bias: $Bias(\boldsymbol{x}_{test}) := f_0(\boldsymbol{x}_{test}) - E\left[f\left(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}\right)\right]$
  - How much the average value of the estimate differs from the true function
- Variance: $Var(\boldsymbol{x}_{test}) \coloneqq E\left[f\left(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}\right) - E\left[f\left(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}\right)\right]\right]^2$
  - How much the estimate varies around its average
- Bias and variance are (conceptually) measured as follows:
  - Get many independent training data sets, each with same size N and input values x_i
  - Each dataset has different output values y_i because of independent noise in the training data
  - Obtain $\widehat{\boldsymbol{\beta}}$ for each training data set
  - Bias and variances are computed over the different sets
- Of course, in reality, we have only one training dataset
  - Used to study theoretical averages over different experiments

Recall: Bias Illustrated

- Red: True function
- Repeat 100 trials
  - Each trial has independent data
  - Obtain estimate for each trial
- Dashed line:
  Mean estimate among all trials
- Bias=True – Mean estimate
- Conclusions:
  - Low model orders ⇒ bias high
  - High model orders ⇒ bias low



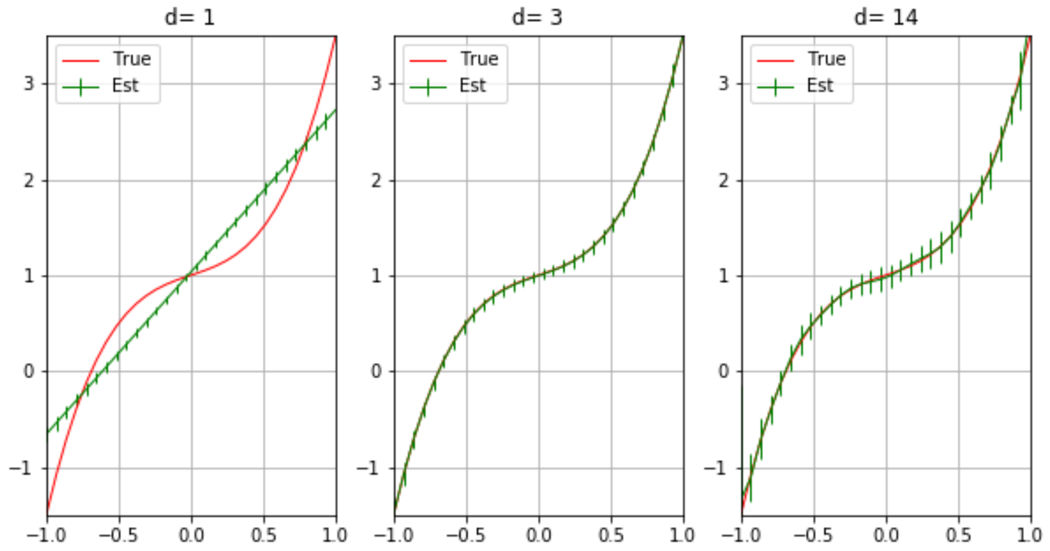
High bias $d < d_{true}$

No bias $d = d_{true}$

No bias $d > d_{true}$

Recall: Variance Illustrated

- Red: True function
- Repeat 100 trials
  - Each trial has independent data
  - Obtain estimate for each trial
- Variance=STD around mean
- Conclusions:
  - Low model orders ⇒ low variance
  - High model orders ⇒ high variance



High bias
Low variance

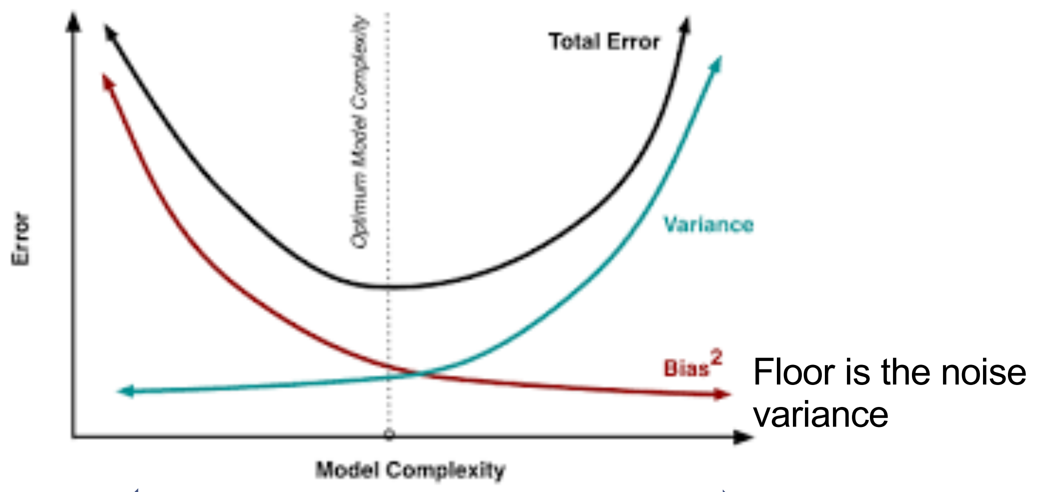
No bias
Med variance

No bias
High variance

Recall: Bias-Variance Formula

- Recall definitions:
  - Function MSE: $\mathrm{MSE}_f(\boldsymbol{x}_{test}) \coloneqq E\left[f_0(\boldsymbol{x}_{test}) - f\left(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}\right)\right]^2$:
  - Bias: $Bias(\boldsymbol{x}_{test}) := f_0(\boldsymbol{x}_{test}) - E\left[f\left(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}\right)\right]$
  - Variance: $Var(\boldsymbol{x}_{test}) \coloneqq E\left[f\left(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}\right) - E\left[f\left(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}\right)\right]\right]^2$
- Bias-Variance formula : $\boxed{MSE_f(\boldsymbol{x}_{test}) = Bias(\boldsymbol{x}_{test})^2 + Var(\boldsymbol{x}_{test})}$
  - Will be proved soon
- Bias-Variance tradeoff
- Bias due to under-modeling
  - Reduced with high model order
- Variance is due to noise in training data and number of parameters to estimate
  - Increases with higher model order

Recall: Bias-Variance Tradeoff



Simpler models
Less parameters
Under-fitting

Richer models
More parameters
Over-fitting

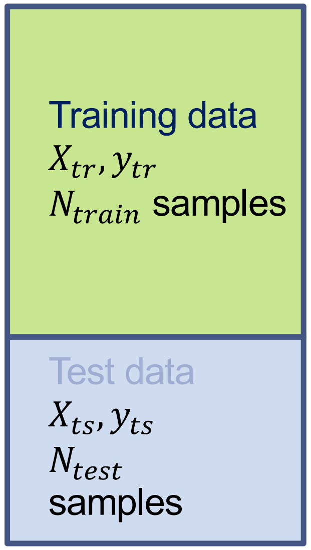
- Bias:
  - Due to under-modeling
  - Reduced with high model order
- Variance:
  - Increases with noise in training data
  - Increase with high model order
- Optimal model order depends on:
  - Amount of samples available
  - Underlying complexity of the relation

Recall: Cross Validation

- Key idea: Evaluate on samples different from training
- Get data X, y
- Split into training X_{tr}, y_{tr} and test X_{ts}, y_{ts}
- For $p = 1$ to p_{max} // Loop over model order
  - Fit on training data with model order p
  - Predict values on test data
  - Score fit on test data (e.g. measure RSS)
- Select model order with smallest score:

$$\hat{p} = \arg\min_{p} S[p]$$

  - Maximize if higher score is better



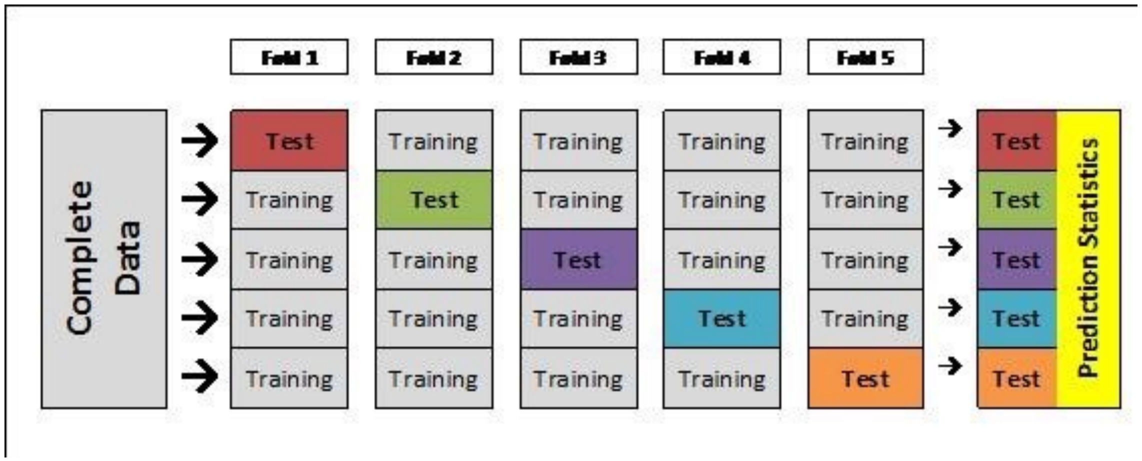
$\hat{\beta} = \mathrm{fit}(X_{tr}, y_{tr}, p)$

$\hat{y}_{ts} = \mathrm{predict}(X_{ts}, \hat{\beta})$

$S[p] = \mathrm{score}(y_{ts}, \hat{y}_{ts})$

Recall: K-Fold Cross Validation

- K-fold cross validation
  - Divide data into K parts
  - Use $K-1$ parts for training. Use remaining for test.
  - Average over the K test choices
  - More accurate, but requires K fits of parameters
  - Typical choice: K=5 or 10
  - Average MSE over K folds estimates the total MSE
  - (=Bias^2+Variance+irreducible error)
- Leave one out cross validation (LOOCV)
  - Take $K = N$ so one sample is left out.
  - Most accurate, but requires N model fittings
  - Necessary when N is small



From http://blog.goldenhelix.com/goldenadmin/cross-validation-for-genomic-prediction-in-svs/

Learning Objectives

- Describe model selection and identify when it may be needed
- Mathematically describe linear regression with regularization
- Select regularizers to impose constraints such as sparsity
- Compute an L1-regularized estimate (LASSO) using sklearn
- Compute the optimal regularization level using cross validation
- Interpret results from a LASSO path
- Set regularizer based on a probabilistic prior

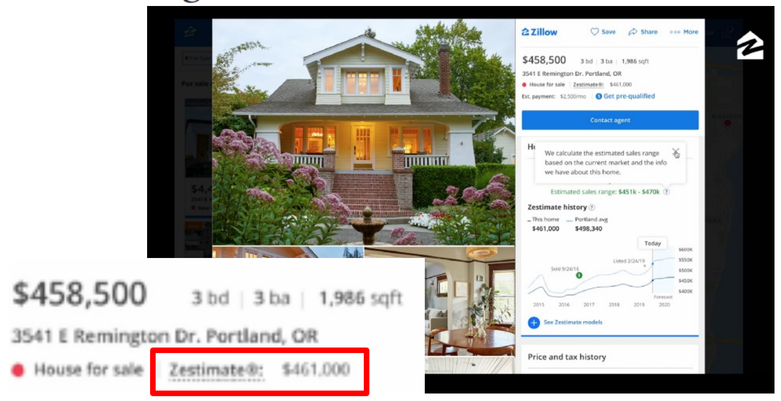
Outline

- Motivating Example: Feature selection in predicting house prices
- Model selection and regularization
- Housing price prediction with LASSO

Predicting Housing Prices

AI, MACHINE LEARNING & RESEARCH

Introducing a new and improved Zestimate algorithm



$458,500 3 bd | 3 ba | 1,986 sqft

3541 E Remington Dr. Portland, OR

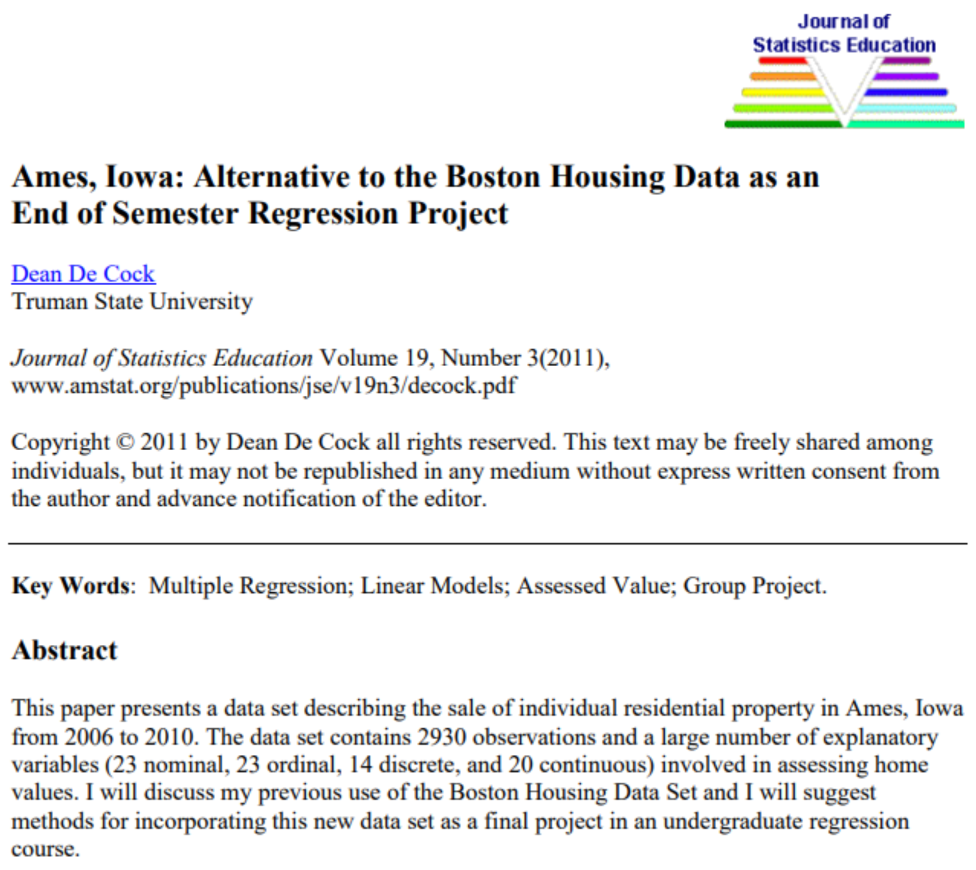
● House for sale Zestimate®: $461,000

https://www.zillow.com/tech/introducing-a-new-and-improved-zestimate-algorithm/

- Many services now predict house prices
- Data science enters real estate!
- Many possible variables:
  - Square meters
  - Condition
  - Zip code
  - Education quality
  - …
- What variables *really* determine the price?

Zillow StreetEasy

Ames, Iowa Dataset



Journal of Statistics Education

Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project

Dean De Cock
Truman State University

Journal of Statistics Education Volume 19, Number 3(2011),
www.amstat.org/publications/jse/v19n3/decock.pdf

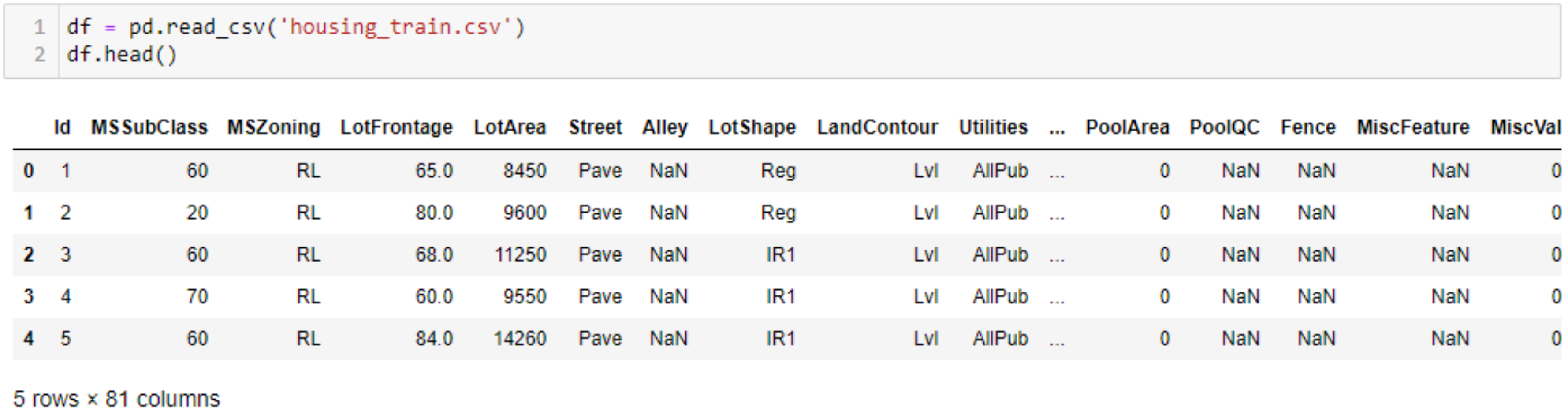
Key Words: Multiple Regression; Linear Models; Assessed Value; Group Project.

Abstract

This paper presents a data set describing the sale of individual residential property in Ames, Iowa from 2006 to 2010. The data set contains 2930 observations and a large number of explanatory variables (23 nominal, 23 ordinal, 14 discrete, and 20 continuous) involved in assessing home values. I will discuss my previous use of the Boston Housing Data Set and I will suggest methods for incorporating this new data set as a final project in an undergraduate regression course.

- Ames, Iowa Dataset
  - Sales from 2006 to 2010
  - From Dean De Cock
  - Undergraduate student
- Alternative to Boston Housing dataset
- Many more variables to explore
  - Approximately 81 variables
  - 2930 samples

Loading the Dataset



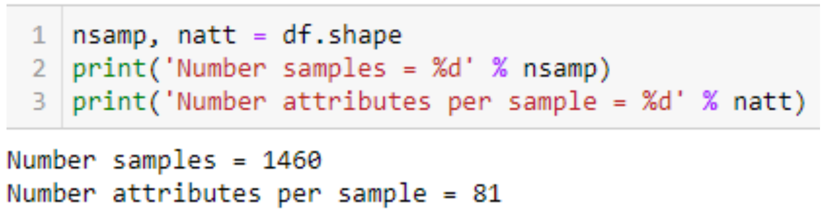
```python
df = pd.read_csv('housing_train.csv')
df.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0

5 rows × 81 columns

- Issues:
  - Many different types of data: Discrete and continuous
  - Missing values (NaN)

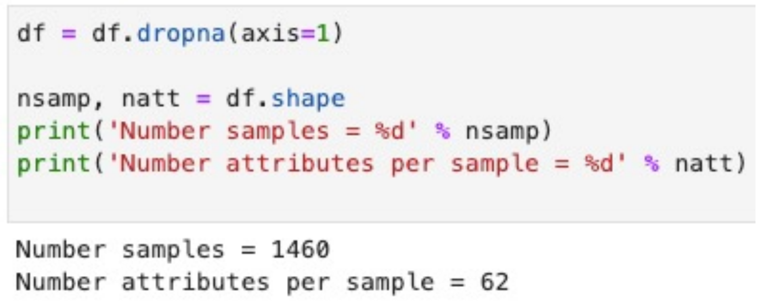
Data Cleaning



```python
nsamp, natt = df.shape
print('Number samples = %d' % nsamp)
print('Number attributes per sample = %d' % natt)
```

```
Number samples = 1460
Number attributes per sample = 81
```

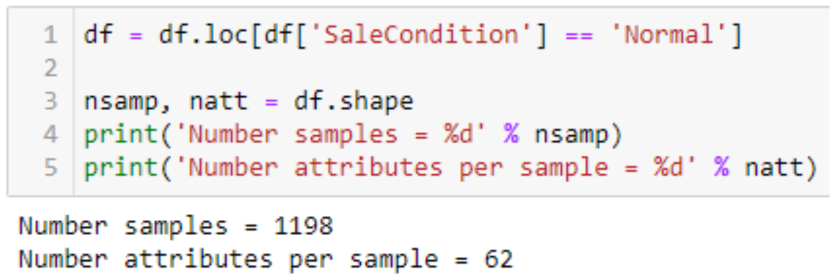
- Original data



```python
df = df.dropna(axis=1)

nsamp, natt = df.shape
print('Number samples = %d' % nsamp)
print('Number attributes per sample = %d' % natt)
```

```
Number samples = 1460
Number attributes per sample = 62
```

- Remove columns with NaN values
  - Could use more sophisticated methods



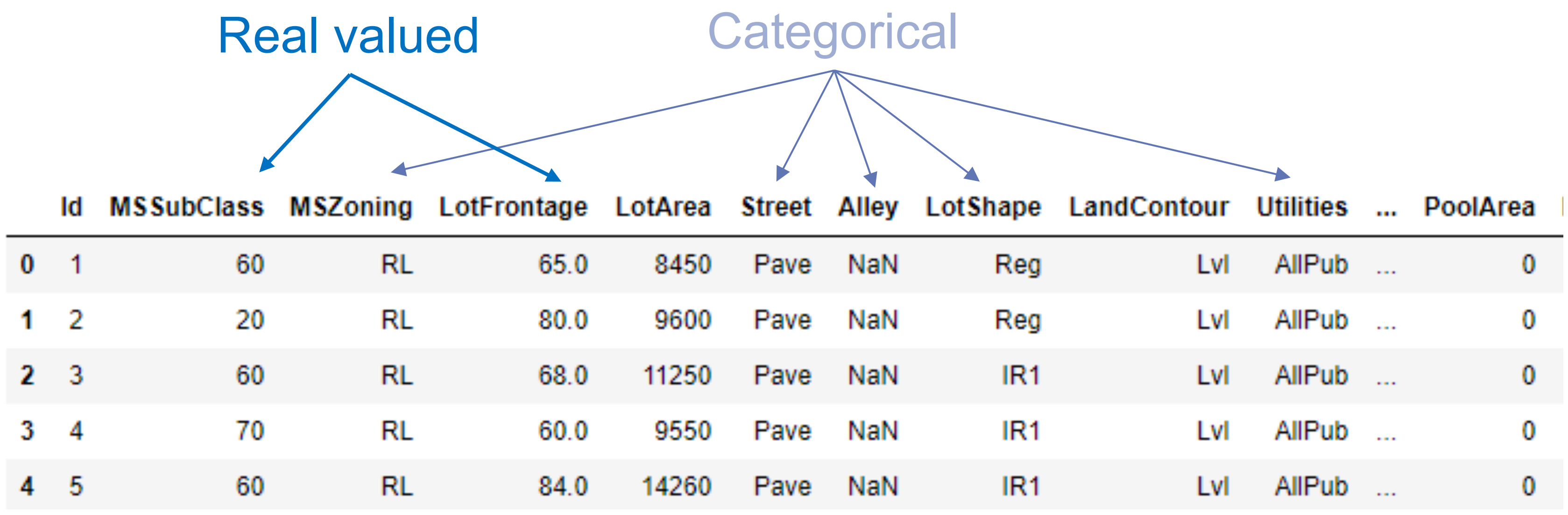
```python
df = df.loc[df['SaleCondition'] == 'Normal']

nsamp, natt = df.shape
print('Number samples = %d' % nsamp)
print('Number attributes per sample = %d' % natt)
```

```
Number samples = 1198
Number attributes per sample = 62
```

- Keep only normal sales
  - Recommended in De Cock paper
  - Makes fitting much easier

Categorical Variables

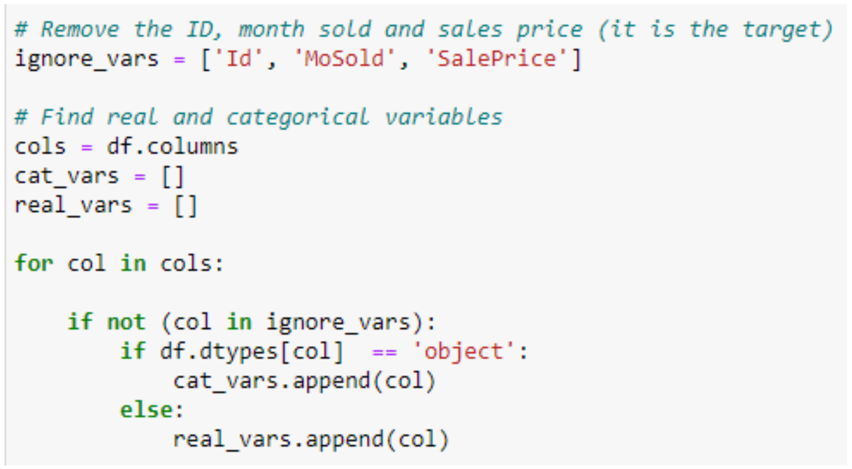
- Data has many categorical variables
- Need to code the categorical variables to numerical values


Real valued
Categorical

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0

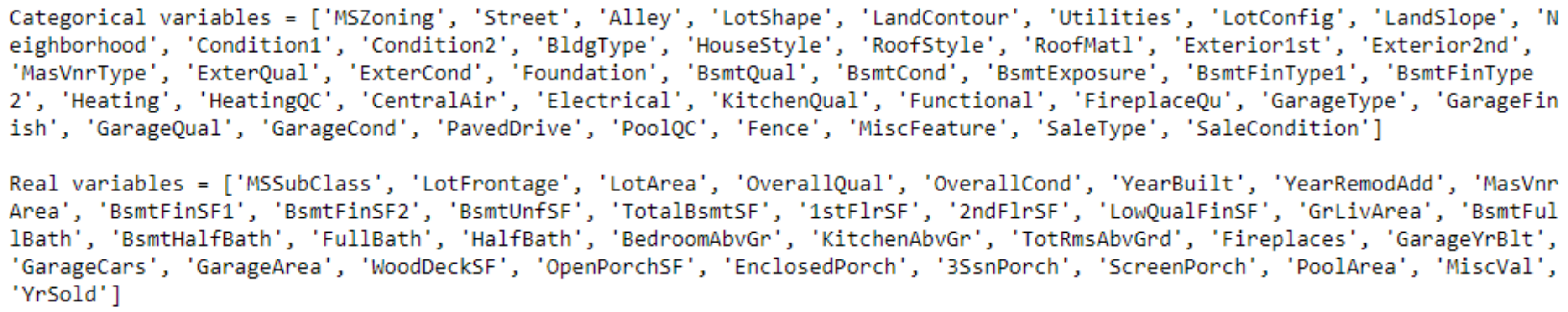
Splitting the Variables

- First, split the variables into categorical and real
- Attributes are of different types (int64, object) and you can check their types



```python
# Remove the ID, month sold and sales price (it is the target)
ignore_vars = ['Id', 'MoSold', 'SalePrice']

# Find real and categorical variables
cols = df.columns
cat_vars = []
real_vars = []

for col in cols:

    if not (col in ignore_vars):
        if df.dtypes[col]  == 'object':
            cat_vars.append(col)
        else:
            real_vars.append(col)
```

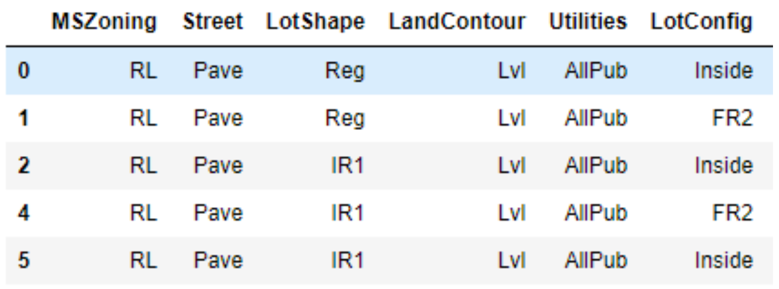
Categorical variables have type *object*



```
Categorical variables = ['MSZoning', 'Street', 'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig', 'LandSlope', 'N
eighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd',
'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType
2', 'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual', 'Functional', 'FireplaceQu', 'GarageType', 'GarageFin
ish', 'GarageQual', 'GarageCond', 'PavedDrive', 'PoolQC', 'Fence', 'MiscFeature', 'SaleType', 'SaleCondition']

Real variables = ['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnr
Area', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFul
lBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageYrBlt',
'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal',
'YrSold']
```

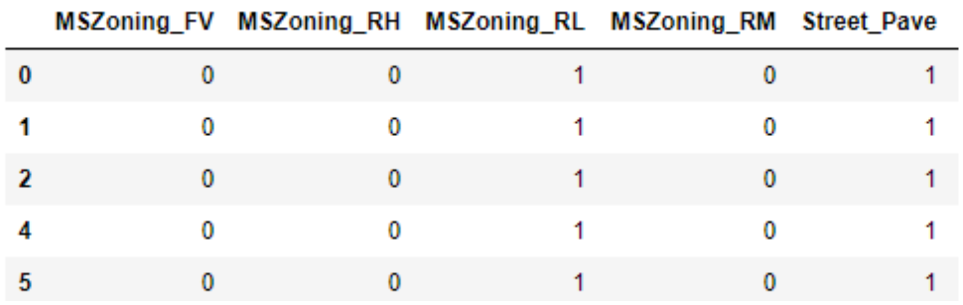
One Hot Coding

Original



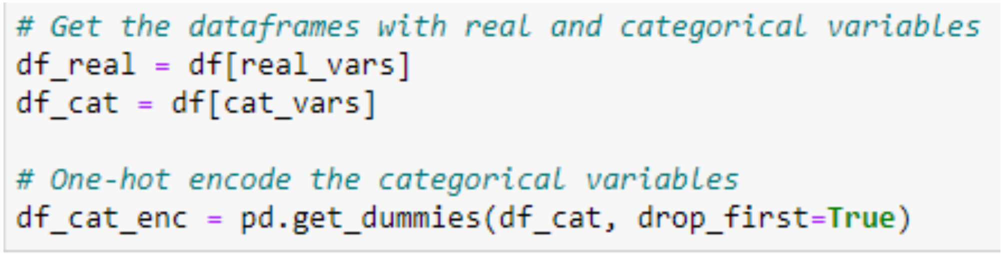
	MSZoning	Street	LotShape	LandContour	Utilities	LotConfig
0	RL	Pave	Reg	Lvl	AllPub	Inside
1	RL	Pave	Reg	Lvl	AllPub	FR2
2	RL	Pave	IR1	Lvl	AllPub	Inside
4	RL	Pave	IR1	Lvl	AllPub	FR2
5	RL	Pave	IR1	Lvl	AllPub	Inside

One Hot coded



	MSZoning_FV	MSZoning_RH	MSZoning_RL	MSZoning_RM	Street_Pave
0	0	0	1	0	1
1	0	0	1	0	1
2	0	0	1	0	1
4	0	0	1	0	1
5	0	0	1	0	1

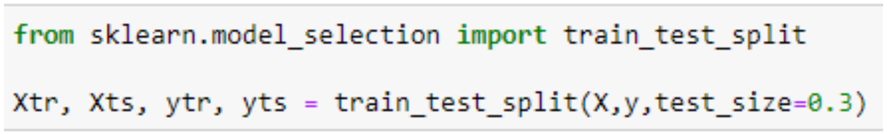
- Use pandas get_dummies
- Replaces categorical variables with one-hot coded values
  - Ex: MSZoning
  - Becomes MSZoning_FV, MSZoning_RH, …



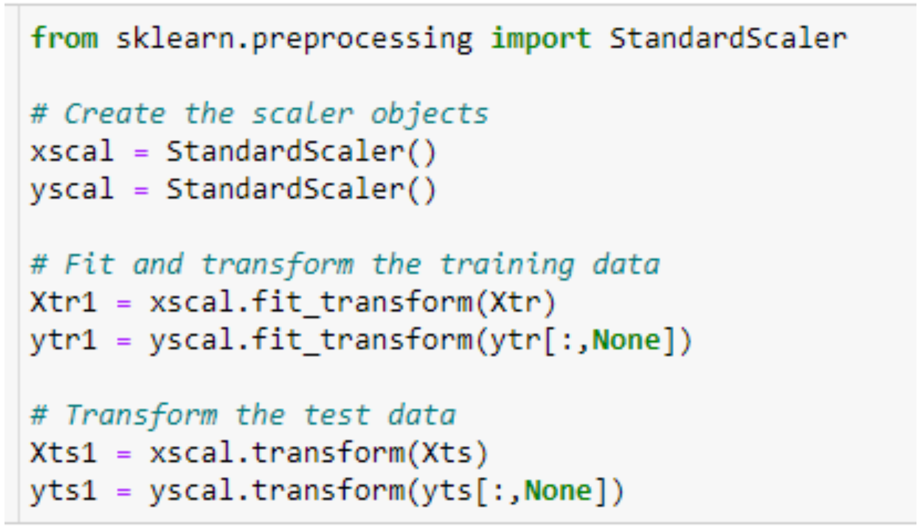
```
# Get the dataframes with real and categorical variables
df_real = df[real_vars]
df_cat = df[cat_vars]

# One-hot encode the categorical variables
df_cat_enc = pd.get_dummies(df_cat, drop_first=True)
```

Scaling Data

- Split data into training and test
- Scale data
  - Remove mean for each column and divide by standard deviation
- Needed to compare coefficients
  - Ensures that all variables have same range
  - Methods need same range for all features
- Note: The scaling transform is
  - Fit on the training data
  - Transform on test data
  - Performed on training and test



```
from sklearn.model_selection import train_test_split

Xtr, Xts, ytr, yts = train_test_split(X,y,test_size=0.3)
```



```
from sklearn.preprocessing import StandardScaler

# Create the scaler objects
xscal = StandardScaler()
yscal = StandardScaler()

# Fit and transform the training data
Xtr1 = xscal.fit_transform(Xtr)
ytr1 = yscal.fit_transform(ytr[:,None])

# Transform the test data
Xts1 = xscal.transform(Xts)
yts1 = yscal.transform(yts[:,None])
```

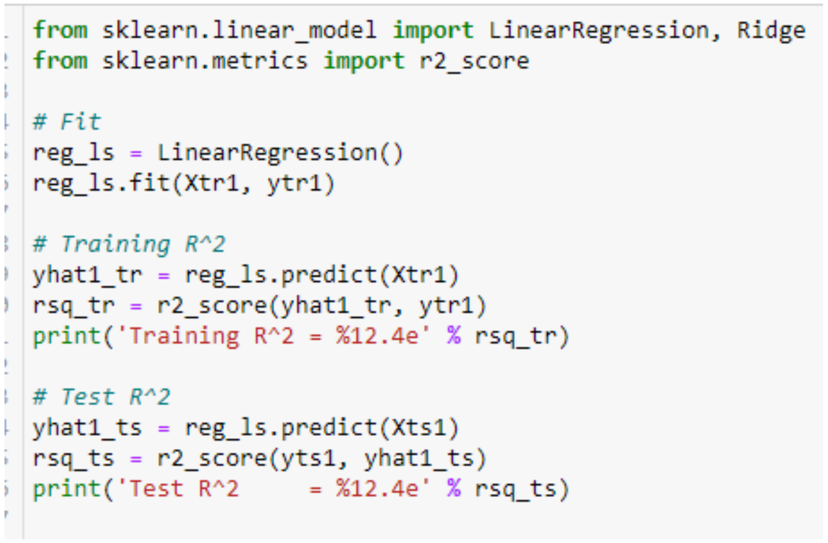
Why Fit_Transform() on Training and Transform() on Test Data

- Both are methods of class sklearn.preprocessing.StandardScaler()

- We want scaling be applied to our test data set too but we do not want to be biased with our model

- Using the transform method we can use the same mean and variance as it is calculated from our training data to transform our test data

First Try: Linear Regression



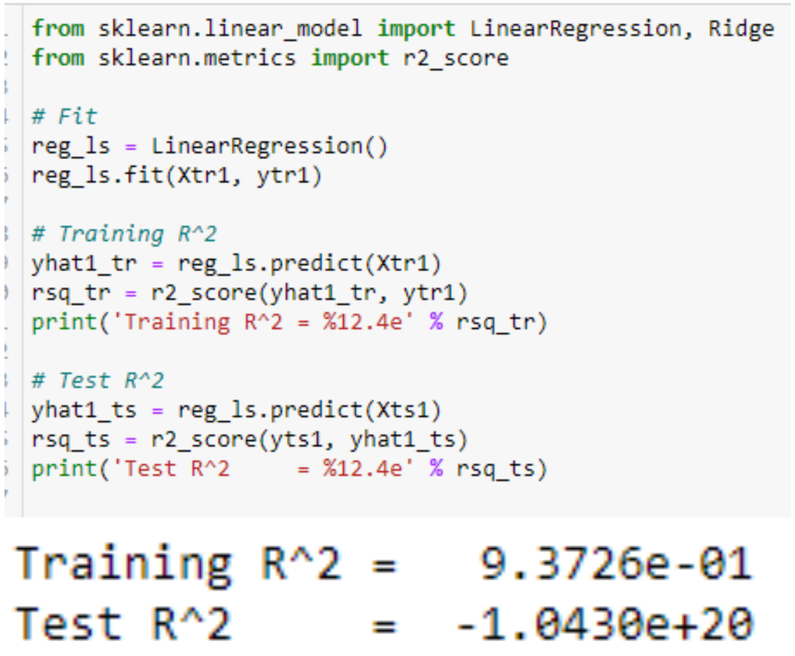
```
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.metrics import r2_score

# Fit
reg_ls = LinearRegression()
reg_ls.fit(Xtr1, ytr1)

# Training R^2
yhat1_tr = reg_ls.predict(Xtr1)
rsq_tr = r2_score(yhat1_tr, ytr1)
print('Training R^2 = %12.4e' % rsq_tr)

# Test R^2
yhat1_ts = reg_ls.predict(Xts1)
rsq_ts = r2_score(yts1, yhat1_ts)
print('Test R^2     = %12.4e' % rsq_ts)
```

```
Training R^2 =   9.3726e-01
Test R^2     =  -1.0430e+20
```

- Simple idea:
  - Use linear regression over features
- Fits the training data very well!
  - R^2 ≈ 0.937
- But, completely fails on the test data
  - $R^2 > 10^{20}$

Conditioning

- What went wrong?
- Recall LS solution is: $\hat{\beta} = (A^T A)^{-1} A^T y$
- Matrix $A^T A$ may be ill-conditioned
  - Eigenvalues close to zero
  - Inverse blows up
- With ill-conditioned data:
  - Training error is fine
  - But the test error blows up
- Overfits data



```
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.metrics import r2_score

# Fit
reg_ls = LinearRegression()
reg_ls.fit(Xtr1, ytr1)

# Training R^2
yhat1_tr = reg_ls.predict(Xtr1)
rsq_tr = r2_score(yhat1_tr, ytr1)
print('Training R^2 = %12.4e' % rsq_tr)

# Test R^2
yhat1_ts = reg_ls.predict(Xts1)
rsq_ts = r2_score(yts1, yhat1_ts)
print('Test R^2     = %12.4e' % rsq_ts)
```

```
Training R^2 =   9.3726e-01
Test R^2     =  -1.0430e+20
```

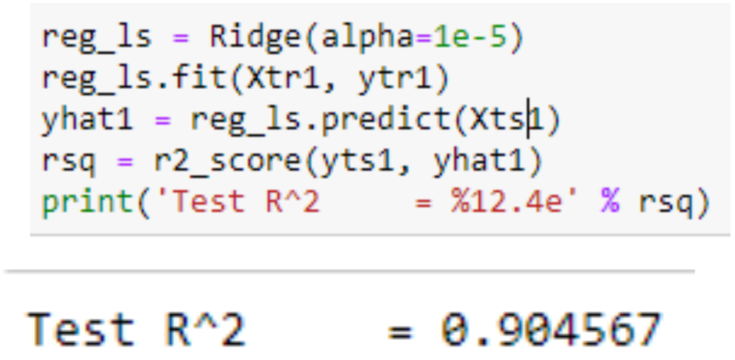
Improving Conditioning via Ridge Regression

- Standard LS solution: $\hat{\beta} = (A^T A)^{-1} A^T y$
- Ridge Regression: Add a conditioning term:

$$\hat{\beta} = (A^T A + cI)^{-1} A^T y$$

  - c is a small positive value.
  - Makes inverse well-behaved
  - We will see this technique more later

- Get good test R^2

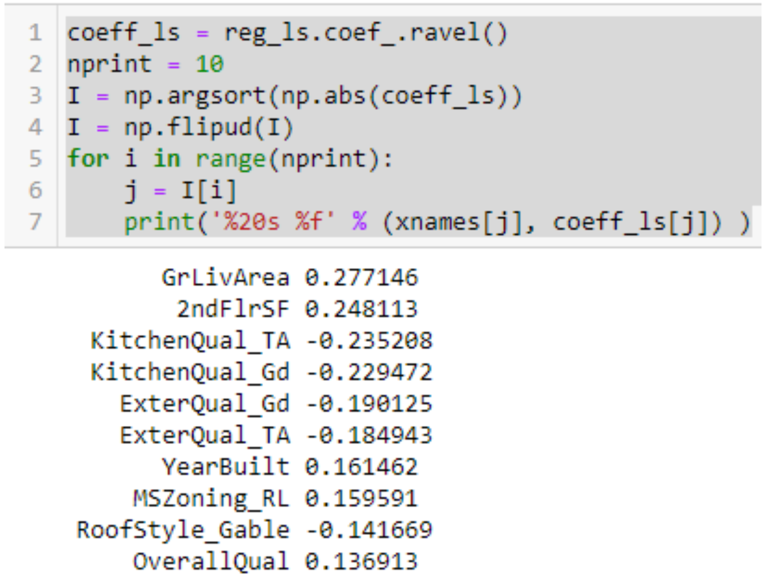


```
reg_ls = Ridge(alpha=1e-5)
reg_ls.fit(Xtr1, ytr1)
yhat1 = reg_ls.predict(Xts1)
rsq = r2_score(yts1, yhat1)
print('Test R^2       = %12.4e' % rsq)
```

```
Test R^2       = 0.904567
```

What Components Matter?

- Simple idea: Look at large coefficients
- We see variables that we may expect:
  - Square footage
  - Quality
  - Zoning
- But there are some issues
  - Some variables seem highly correlated
  - Ex: GrLivArea and 2ndFlrSF
  - KitchenQual and OverallQual
  - Do we need both?

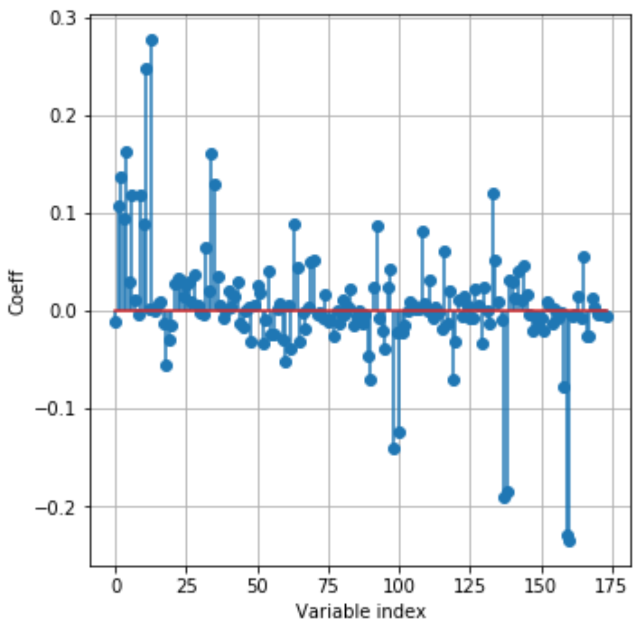


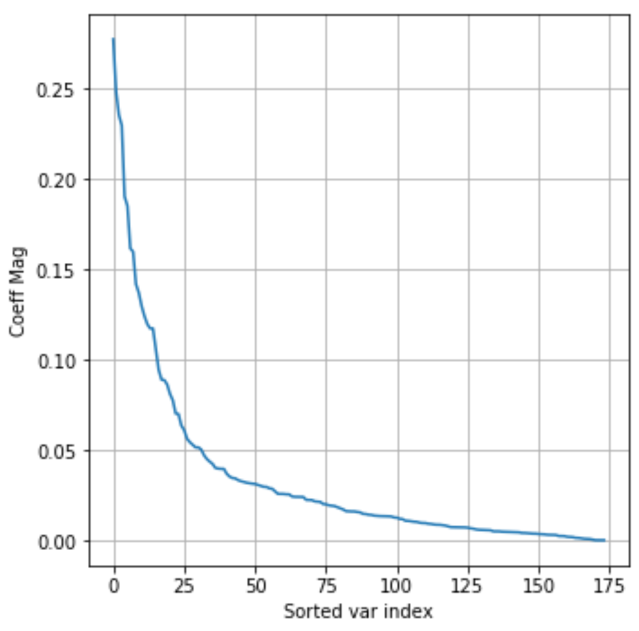
```
coeff_ls = reg_ls.coef_.ravel()
nprint = 10
I = np.argsort(np.abs(coeff_ls))
I = np.flipud(I)
for i in range(nprint):
    j = I[i]
    print('%20s %f' % (xnames[j], coeff_ls[j]) )
```

```
           GrLivArea 0.277146
            2ndFlrSF 0.248113
       KitchenQual_TA -0.235208
       KitchenQual_Gd -0.229472
         ExterQual_Gd -0.190125
         ExterQual_TA -0.184943
           YearBuilt 0.161462
         MSZoning_RL 0.159591
      RoofStyle_Gable -0.141669
         OverallQual 0.136913
```

What Components Do *Not* Matter?

- All coefficients are far from zero
  - Very few coefficients that can be removed
- Does this mean all variables matter?
- Model or feature selection problem:
  - *How do we find the variables that matter?*

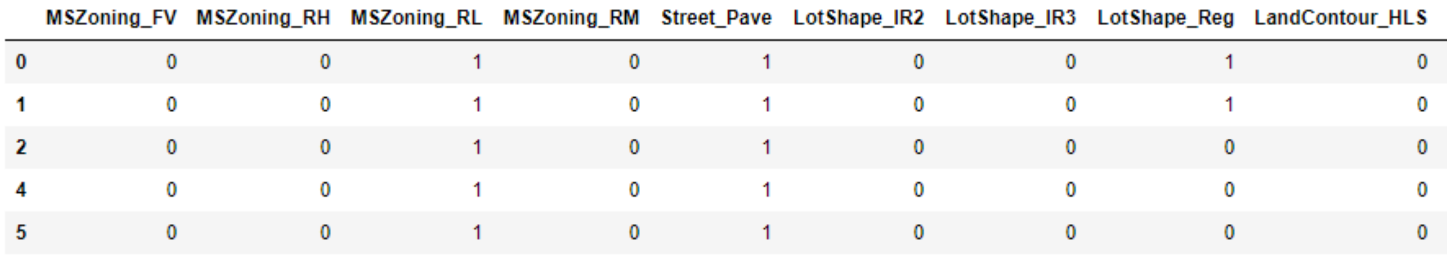




Outline

- Motivating Example: Feature selection in predicting house prices
- Model selection and regularization
- Housing price prediction with LASSO

Model Selection via Sparsity



	MSZoning_FV	MSZoning_RH	MSZoning_RL	MSZoning_RM	Street_Pave	LotShape_IR2	LotShape_IR3	LotShape_Reg	LandContour_HLS
0	0	0	1	0	1	0	0	1	0
1	0	0	1	0	1	0	0	1	0
2	0	0	1	0	1	0	0	0	0
4	0	0	1	0	1	0	0	0	0
5	0	0	1	0	1	0	0	0	0

174 variables after one-hot coding

- Model selection problem: Need to identify the parameters that *really* matter. Two reasons to do that:
  - Help interpret results
  - Improves generalization error (less parameters, close to model order)
- Idea: Fit model under sparsity constraint:
  - Linear model: $\hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$
  - Feature x_j is ignored if $\beta_j = 0$
  - Try to force most $\beta_j = 0 \Rightarrow$ Model only uses a few of the variables
    - Sparse coefficient constraint

Regularized LS Estimation

- Regularization: General method for finding constrained solutions
    - E.g. solutions that are sparse
- Standard least squares estimation (from Lecture 3, unregularized form):

$$\hat{\beta} = \arg\min_{\beta} MSE(\beta), \qquad MSE(\beta) = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- Regularized estimator:

$$\hat{\beta} = \arg\min_{\beta} J(\beta), \qquad J(\beta) = MSE(\beta) + \phi(\beta)$$

- $MSE(\beta) = \text{mean-squared prediction error from before}$
- $\phi(\beta)$ = regularizing function
- Concept: Regularizer penalizes β that are "unlikely"
    - Constrains estimate to smaller set of parameters

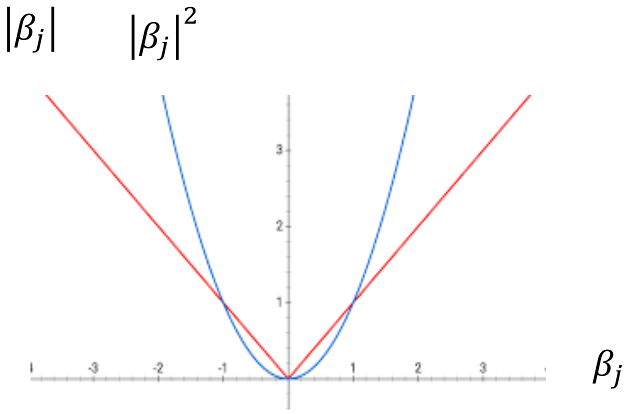
Two Common Regularizers

- Ridge regression (called L2)

$$\phi(\beta) = \frac{\alpha}{n}\sum_{j=1}^{d}|\beta_j|^2$$

- LASSO regression (called L1)

$$\phi(\beta) = 2\alpha\sum_{j=1}^{d}|\beta_j|$$



- Coefficient $\alpha > 0$ determines regularization level
  - Higher $\alpha \Rightarrow$ Higher level of regularization, more constrained
  - Will show how to select α later via cross-validation
  - Scaling factors adjusted to match sklearn convention
- Both penalize large β_j: Tries to make β_j small
  - Will see that L1 also promotes **sparsity**
- Convention: Do not include intercept term β_0
  - In general, no reason to make this term small

L1 and L2 Norm

- Ridge and LASSO Regularization can be written with norms

- Ridge cost function:

$$J(\boldsymbol{\beta}) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^{d} \left|\beta_j\right|^2 = ||\boldsymbol{y} - \boldsymbol{A\beta}||^2 + \alpha||\boldsymbol{\beta}||_2^2$$

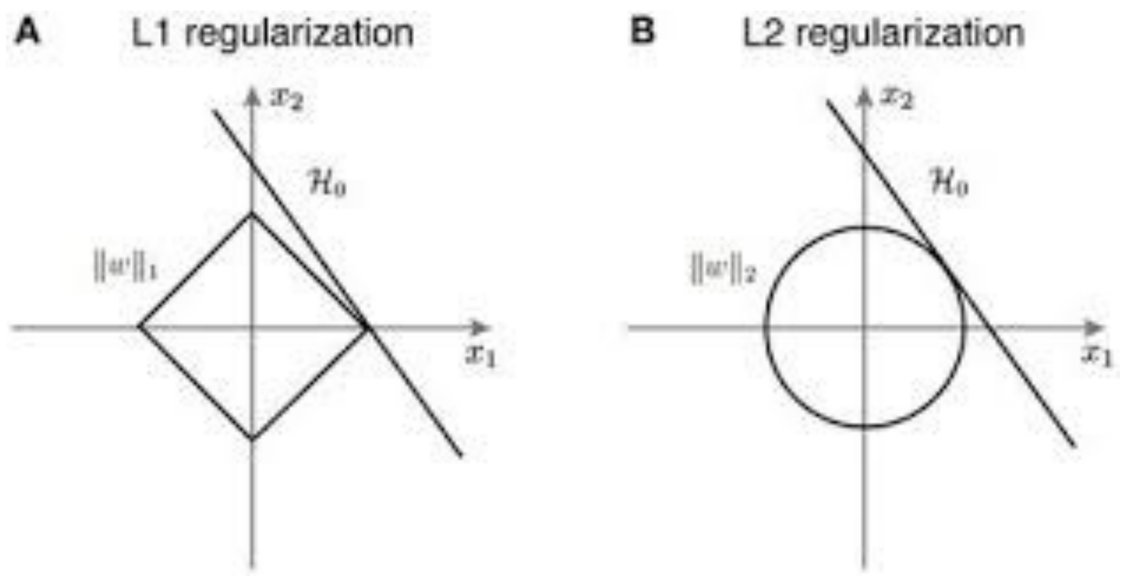
- LASSO cost function:

$$J(\boldsymbol{\beta}) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^{d} \left|\beta_j\right|$$
$$= \frac{1}{2n} ||\boldsymbol{y} - \boldsymbol{A\beta}||^2 + \alpha||\boldsymbol{\beta}||_1$$

  - $||\boldsymbol{\beta}||_1$ = L1 norm (pronounced ell-1)

Ridge vs LASSO

- L2 tends to lead to have many “small” coefficients
  - But solutions are not exactly zero
  - Not ideal for feature selection
- L1 tends to lead to more sparse solutions
  - Several coefficients are zero



Solving Ridge Regression

- Ridge regression problem: Find β to minimize

$$J(\boldsymbol{\beta}) = ||\boldsymbol{y} - A\boldsymbol{\beta}\ ||^2 + \alpha||\boldsymbol{\beta}\ ||^2$$

- Solution for given regularization level

$$\boldsymbol{\beta}_{ridge} = (\boldsymbol{A}^T\boldsymbol{A} + \alpha\boldsymbol{I})^{-1}\boldsymbol{A}^T\boldsymbol{y}$$

  - Set gradient = 0

- Sklearn function for ridge regression:
  - http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

Solving LASSO Regression

- LASSO cost function:

$$J(\boldsymbol{\beta}) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha\sum_{j=1}^{d}\left|\beta_j\right| = \frac{1}{2n}\|\boldsymbol{y} - \boldsymbol{A\beta}\|^2 + \alpha\|\boldsymbol{\beta}\|_1$$

- Because derivative of $\left|\beta_j\right|$ is not continuous, there is no closed-form solution.
- Many methods to solve iteratively
  - Least angle regression (LAR), coordinate descent, ADMM
  - However, the cost function is convex $\Rightarrow$ no local minima
  - Beyond the scope of this class
  - See textbook [Hastie2008] for LAR method
  - We will talk about a method later.

Data Scaling

- Scaling: Whenever using regularization:
  - Scale each feature and the target to have zero mean and unit variance (or STD)
  - $x_{ij} \rightarrow (x_{i,j} - \bar{x}_j)/\mathrm{STD}(x_{ij})$
  - $y_i \rightarrow (y_i - \bar{y})/\mathrm{STD}(y_i)$
- After predictor for the scaled data are determined:
  - Derive the equivalent predictor on the original data
- Motivation:
  - Without scaling, the regularization level depends on the data range
  - With mean removal, we do not need the intercept term β_0
  - So that the regularization term is simply a L2 or L1 norm of coefficient vector

Selecting the Regularization Level

- How do we select regularization level α?
  - Higher $\alpha \Rightarrow$ More constrained / simpler model
  - Lower $\alpha \Rightarrow$ More complex model
- Similar to inverse of model order
- Find α via cross-validation

Pseudo-code

```
Split in training $X_{tr}, y_{tr}$ and test $X_{ts}, y_{ts}$.

For $\alpha$ in $\alpha_{test}$:
    • $\hat{\beta} = fit(X_{tr}, y_{tr}, \alpha)$   // Fit on training data

    • $\hat{y}_{ts} = predict(X_{ts})$   // Predict on test data
    • $S[\alpha] = score(y_{ts}, \hat{y}_{ts})$   // Score on test data

$\hat{\alpha} = argmax\ S[\alpha]$ // Select $\alpha$ with highest test score
```

Summary

Method	Regularizer	Effect on parameters	Solution for Fitting
None	$\phi(\boldsymbol{\beta}) = 0$	Leaves parameters unconstrained	$\widehat{\boldsymbol{\beta}} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\,\boldsymbol{A}^T\boldsymbol{y}$
Ridge	$\phi(\boldsymbol{\beta}) = \frac{\alpha}{n}\|\boldsymbol{\beta}\|_2^2$	Makes parameters small Close to zero	$\widehat{\boldsymbol{\beta}} = (\boldsymbol{A}^T\boldsymbol{A} + \alpha\,\boldsymbol{I})^{-1}\,\boldsymbol{A}^T\boldsymbol{y}$
LASSO	$\phi(\boldsymbol{\beta}) = 2\alpha\|\boldsymbol{\beta}\|_1$	Makes parameters sparse. Many coefficients exactly zero	No analytic solution. Need to run an optimizer

- Regularized least squares

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\beta} J(\boldsymbol{\beta}), \qquad J(\boldsymbol{\beta}) = \frac{1}{n}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\beta}\|^2 + \phi(\boldsymbol{\beta})$$

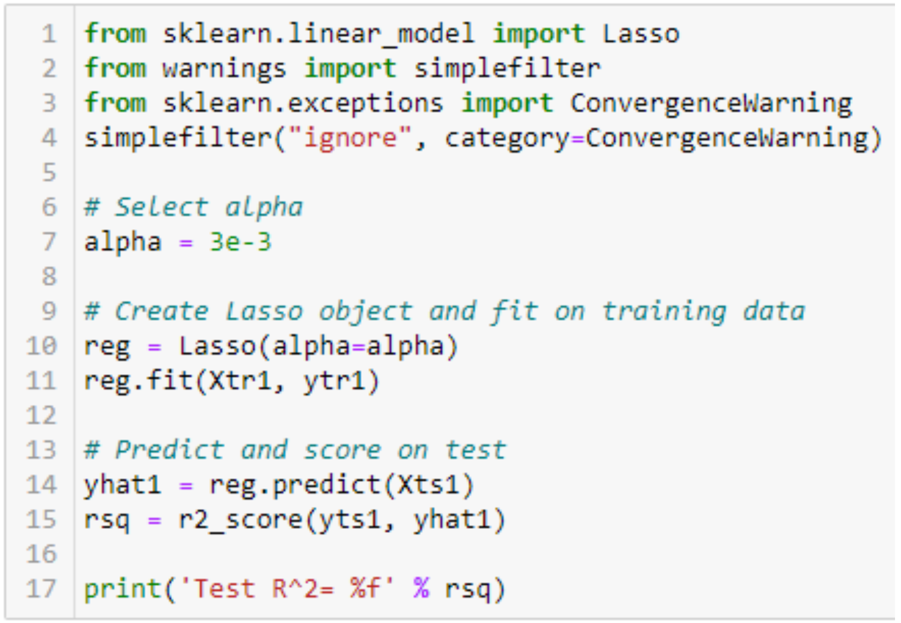
- Whatever you choose for the regularizer:
  - Scale data before training
  - Select regularization level with cross-validation

Outline

- Motivating Example: Feature selection in predicting house prices
- Model selection and regularization
- Housing price prediction with LASSO

LASSO Regression in Python

- Sklearn built in Lasso class
- Easy to use
  - Set alpha
  - Fit on training data
  - Predict and score on test

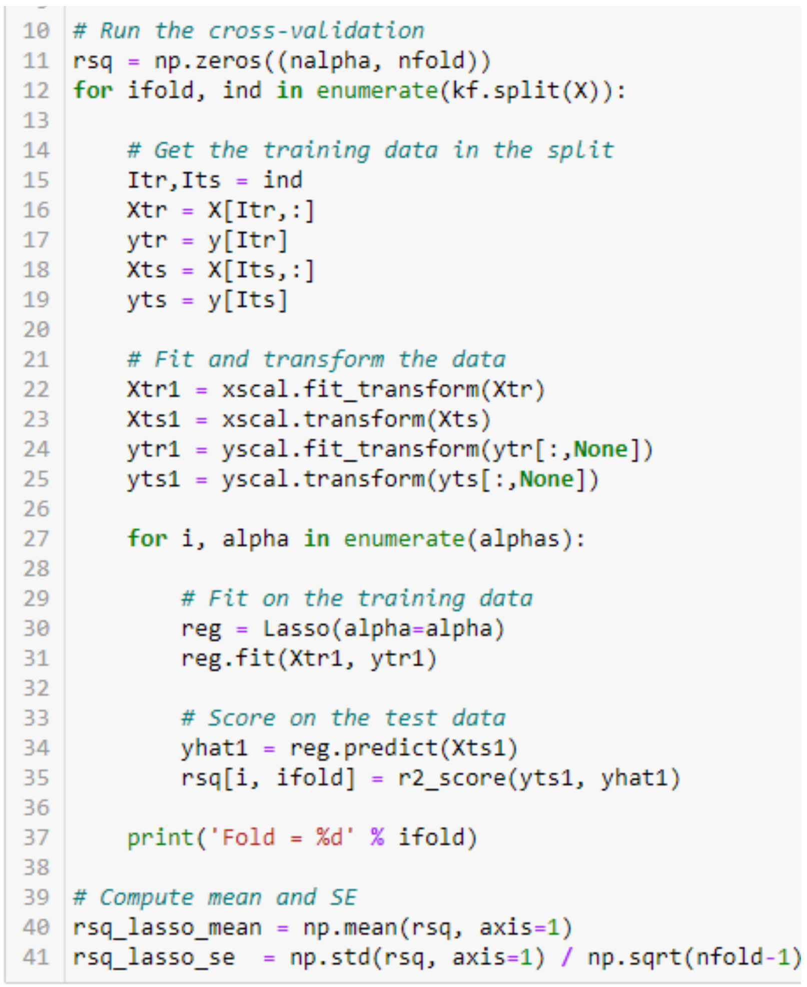


```python
from sklearn.linear_model import Lasso
from warnings import simplefilter
from sklearn.exceptions import ConvergenceWarning
simplefilter("ignore", category=ConvergenceWarning)

# Select alpha
alpha = 3e-3

# Create Lasso object and fit on training data
reg = Lasso(alpha=alpha)
reg.fit(Xtr1, ytr1)

# Predict and score on test
yhat1 = reg.predict(Xts1)
rsq = r2_score(yts1, yhat1)

print('Test R^2= %f' % rsq)
```

```
Test R^2= 0.899122
```

Optimizing Alpha via Cross Validation

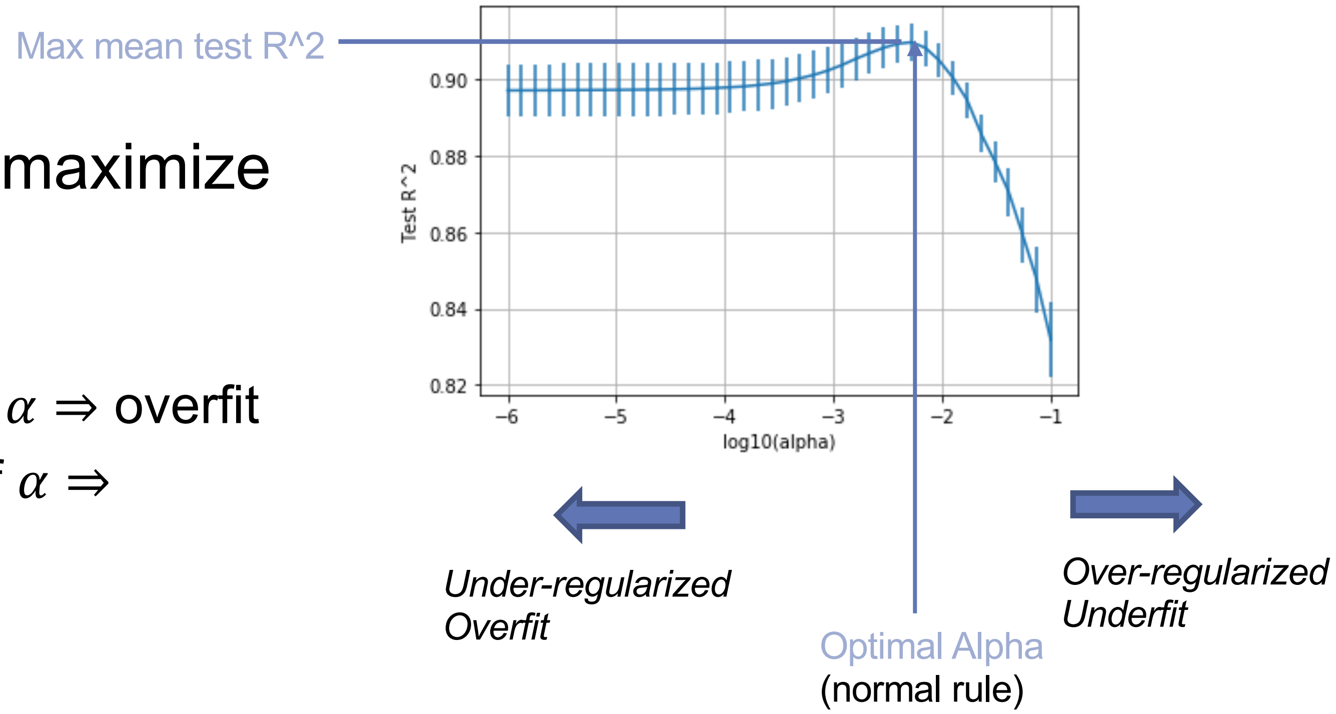
- In each fold we:
  - Split data into training and test
  - Fit the scale on the training
  - Transform training and test
  - For each alpha:
  - Fit training and score on test
- Note: Scaling is redone on each fold
  - Ensures scaling is part of the training



```python
# Run the cross-validation
rsq = np.zeros((nalpha, nfold))
for ifold, ind in enumerate(kf.split(X)):

    # Get the training data in the split
    Itr,Its = ind
    Xtr = X[Itr,:]
    ytr = y[Itr]
    Xts = X[Its,:]
    yts = y[Its]

    # Fit and transform the data
    Xtr1 = xscal.fit_transform(Xtr)
    Xts1 = xscal.transform(Xts)
    ytr1 = yscal.fit_transform(ytr[:,None])
    yts1 = yscal.transform(yts[:,None])

    for i, alpha in enumerate(alphas):

        # Fit on the training data
        reg = Lasso(alpha=alpha)
        reg.fit(Xtr1, ytr1)

        # Score on the test data
        yhat1 = reg.predict(Xts1)
        rsq[i, ifold] = r2_score(yts1, yhat1)

    print('Fold = %d' % ifold)

# Compute mean and SE
rsq_lasso_mean = np.mean(rsq, axis=1)
rsq_lasso_se   = np.std(rsq, axis=1) / np.sqrt(nfold-1)
```

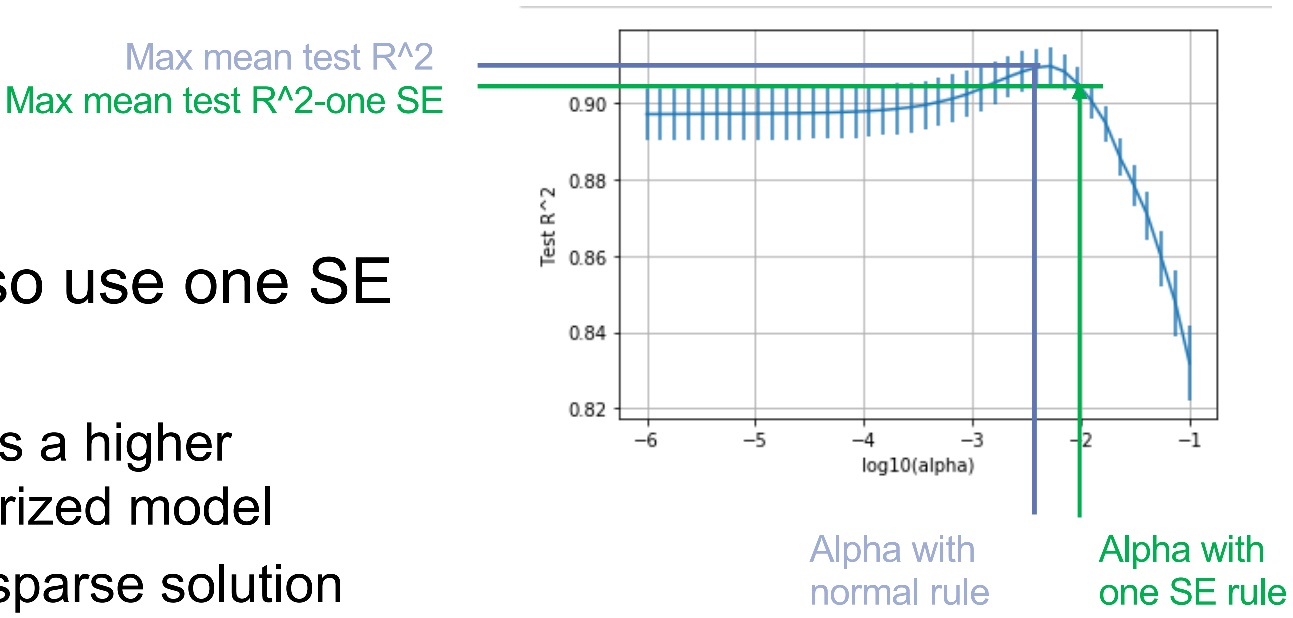
Cross Validation: Normal Rule

- Select alpha to maximize mean test R^2
  - Normal rule
  - Lower values of $\alpha \Rightarrow$ overfit
  - Higher values of $\alpha \Rightarrow$ underfit



```
Alpha optimal (normal rule) =   5.2233e-03
Mean test R^2 (normal rule) =   0.910
```

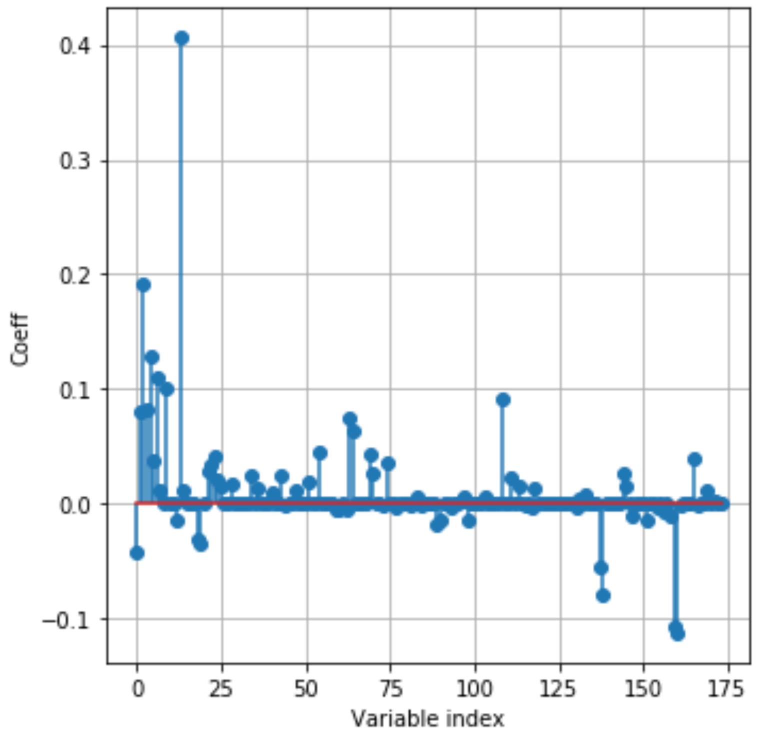
Cross Validation: One SE Rule

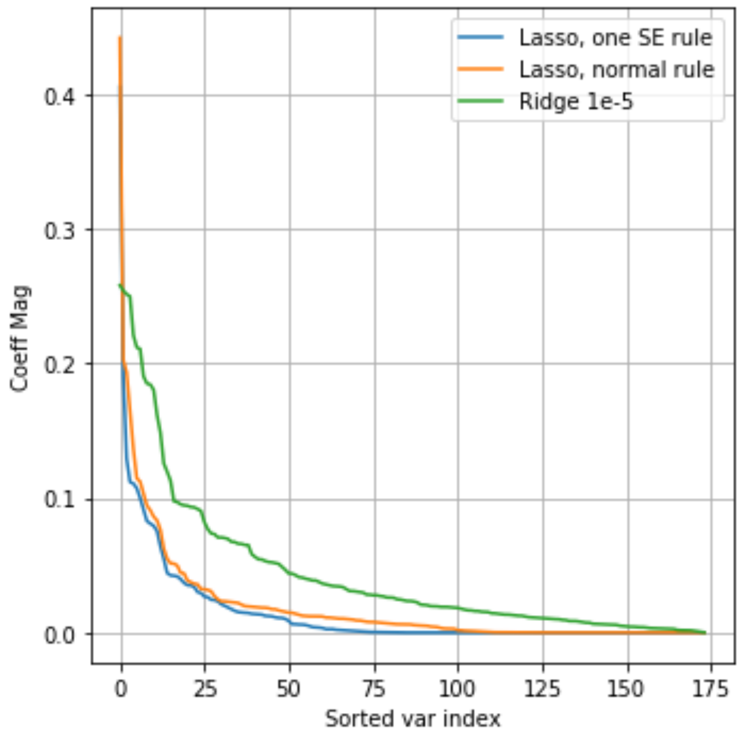


- Can also use one SE rule:
  - Selects a higher regularized model
  - More sparse solution

Sparsity in the Coefficients

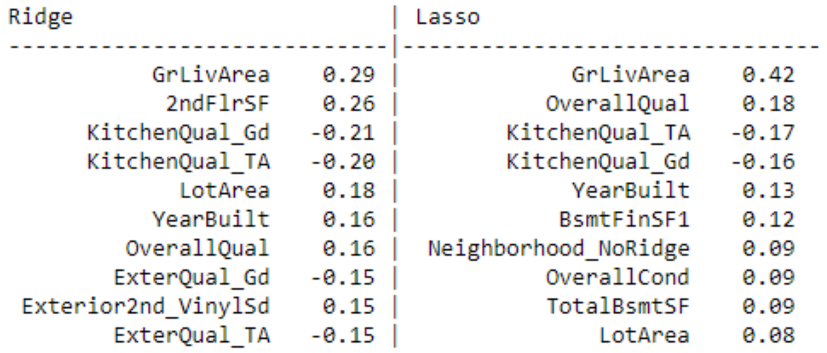
- Adding L1 regularization:
  - Makes coefficient smaller
  - Many coefficients approx. 0





Most Important Variables

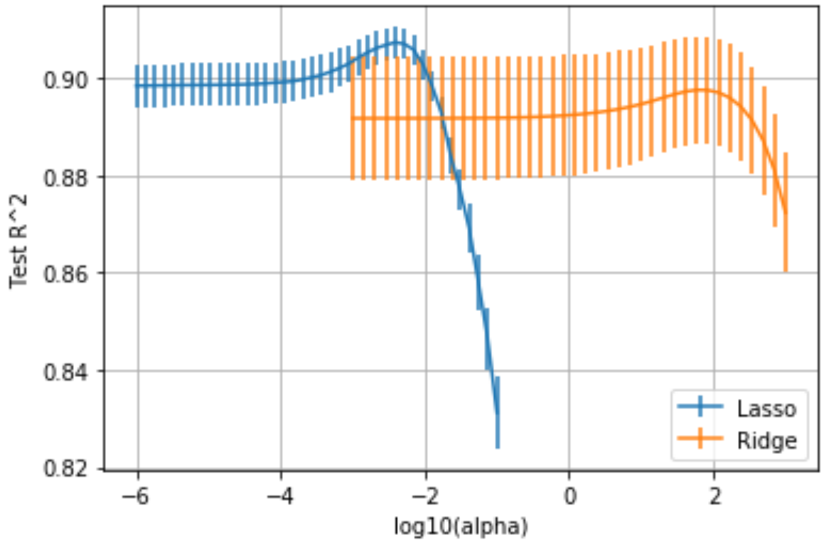
- Right table: Variables with 10 large coefficient magnitudes
- Minimally regularized (Ridge) has:
  - Variables that are highly correlated
  - Ex: GrLivArea and 2ndFlrSF
  - Several large variables
- Lasso:
  - Reduces correlated variables
  - Selects GrLivArea alone
  - Gives the variables more importance
    - GrLivArea coefficient is much higher



```
Ridge                                 | Lasso
--------------------------------------|-----------------------------------
           GrLivArea       0.29 |            GrLivArea      0.42
            2ndFlrSF       0.26 |          OverallQual      0.18
      KitchenQual_Gd      -0.21 |       KitchenQual_TA     -0.17
      KitchenQual_TA      -0.20 |       KitchenQual_Gd     -0.16
             LotArea       0.18 |            YearBuilt      0.13
           YearBuilt       0.16 |           BsmtFinSF1      0.12
         OverallQual       0.16 | Neighborhood_NoRidge      0.09
        ExterQual_Gd      -0.15 |          OverallCond      0.09
 Exterior2nd_VinylSd       0.15 |          TotalBsmtSF      0.09
        ExterQual_TA      -0.15 |              LotArea      0.08
```

Ridge Vs. Lasso

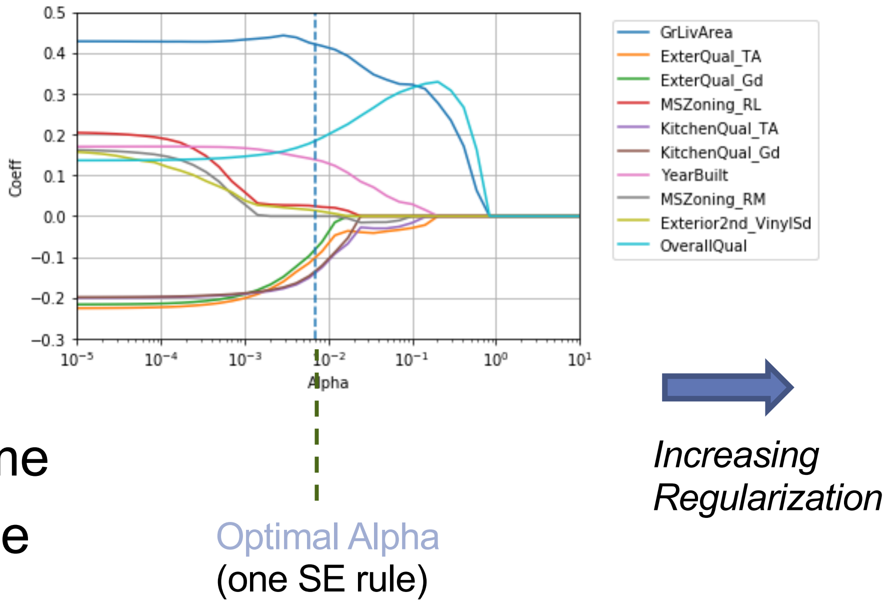
- Can optimize alpha for both regularizers
- Optimal mean test R^2 is better for LASSO
- Offers better feature selection



```
Optimal R^2 Lasso:  0.907283
Optimal R^2 Ridge:  0.897498
```

Lasso Path

- Plot of coefficients vs. alpha
- For large alpha:
  - All coefficients are zero
- As alpha is decreased:
  - One coefficient is activated at a time
  - Indicates an ordering of importance


Coeff
Alpha
GrLivArea
ExterQual_TA
ExterQual_Gd
MSZoning_RL
KitchenQual_TA
KitchenQual_Gd
YearBuilt
MSZoning_RM
Exterior2nd_VinylSd
OverallQual
Increasing Regularization
Optimal Alpha
(one SE rule)