improve model comparison

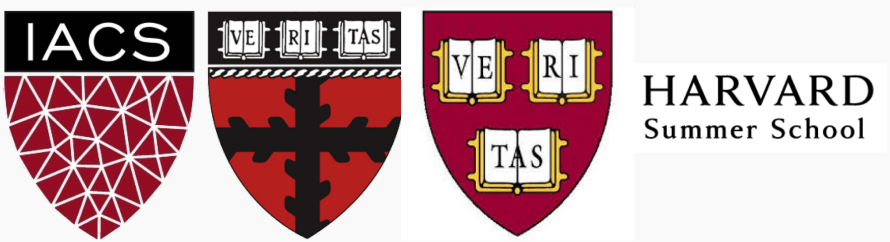
improve prediction

avoid overfitting

Lecture #5: Cross-Validation and Regularization (Ridge and LASSO)

CS-S109A: Introduction to Data Science

Kevin Rader



Lecture Outline

Brief Review

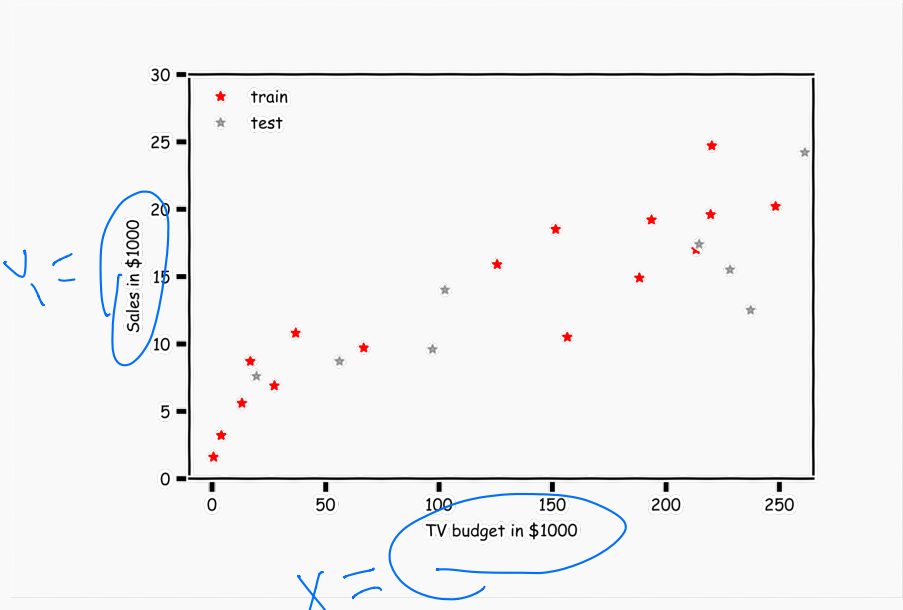
- Overfitting
- Variable Selection

Cross Validation

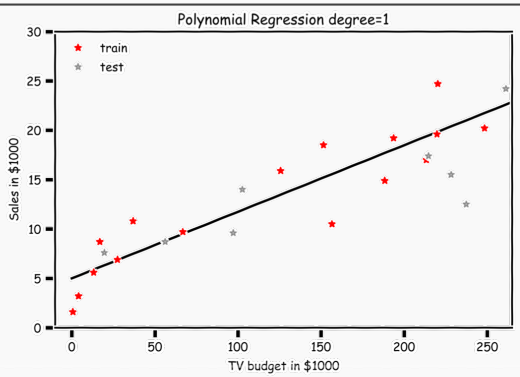
Regularization: LASSO and Ridge

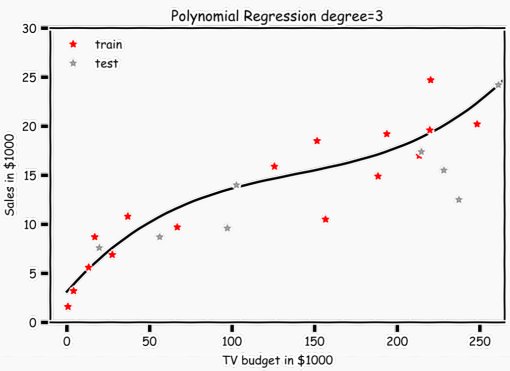
Regularization Methods: A Comparison

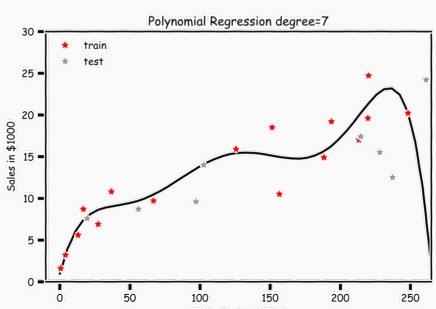
Overfitting

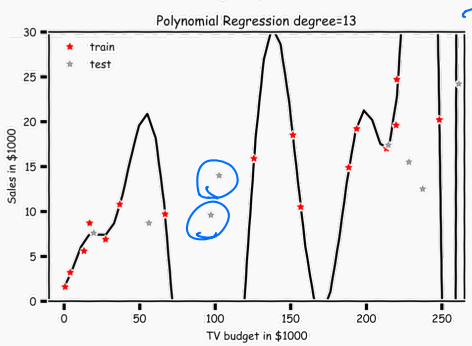


Overfitting with polynomials



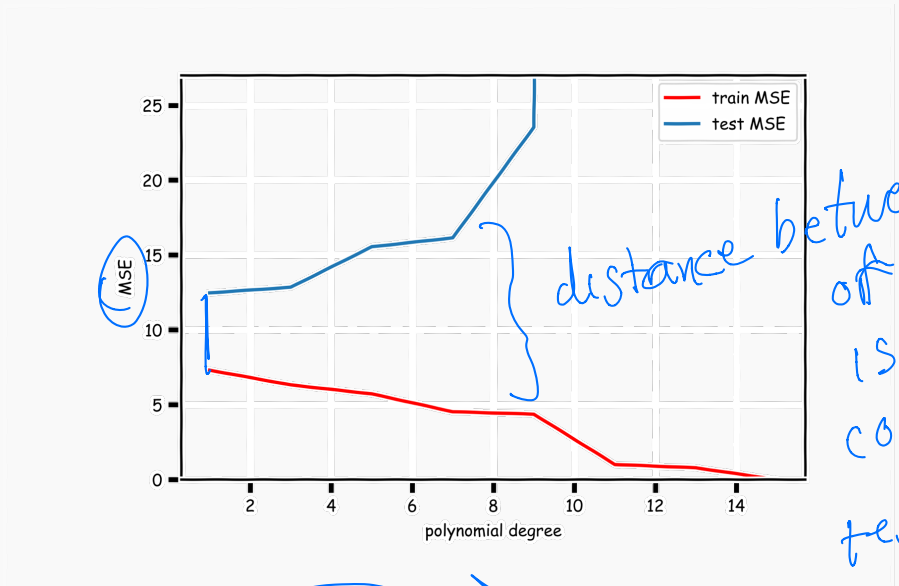






clearly overfit to the train set.
(test set is predicted terribly)

Validation/Test



Overfitting

In statistical modeling, **overfitting** is *the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to properly capture the true relationships or predict future (aka, out-of-sample) observations reliably*.

You can think of overfitting as fitting to the particular intricacies of the residuals in the training set and not just the true signal in the relationship between the response and the predictors:

Think:

$$Y = f(\boldsymbol{X}) + \varepsilon$$

Train-Validation-Test

Question:

What is a more honest representation of the performance of the model?

R2_test(degree=1) = 0.52

R2_train(degree=1) = 0.83

← always report measures of error or accuracy on the out-of-sample (train) set of data.

Variable Selection

Variable selection (also called **model selection**) is the application of a principled method to determine the complexity of the model, e.g. choosing a subset of predictors, choosing the degree of the polynomial model etc.

A strong motivation for performing model selection is to avoid overfitting, which we saw can happen when:

- there are too many predictors: → overfitting
  - the feature space has high dimensionality
  - the polynomial degree is too high
  - too many interaction terms are considered
- the coefficients values are too **extreme** (high multicollinearity)

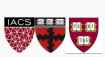
↙ predictors are correlated

Stepwise Variable Selection and Cross Validation

Selecting optimal subsets of predictors (including choosing the degree of polynomial models) through:

- stepwise variable selection - iteratively building an optimal subset of predictors by optimizing a fixed model evaluation metric each time,
- validation - selecting an optimal model by evaluating each model on validation/test set.

We will also address the issue of discouraging extreme values in model parameters later.



Stepwise Variable Selection: Forward method

In **forward selection**, we find an 'optimal' set of predictors by iterative building up our set.

1. Start with the empty set P0, construct the null modelM0.

2. For $k = 1, \ldots, J$:

2.1 Let M_{k-1} be the model constructed from the best set of $k-1$ predictors, P_{k-1}.

2.2 Select the predictor X_{n_k}, not in P_{k-1}, so that the model constructed from $P_k = X_{n_k} \cup P_{k-1}$ optimizes a fixed metric (this can be *p*-value, *F*-stat; validation/test MSE, R^2; or AIC/BIC on training set).

2.3 Let M_k denote the model constructed from the optimal P_k.

3. Select the model M amongst $\{M_0, M_1, \ldots, M_J\}$ that optimizes a fixed metric (this can be validation MSE, R^2; or AIC/BIS on training set)

*Note: this reverse direction (starting with the *full* model) is also possible: *Backwards* Stepwise

Lecture Outline

Brief Review

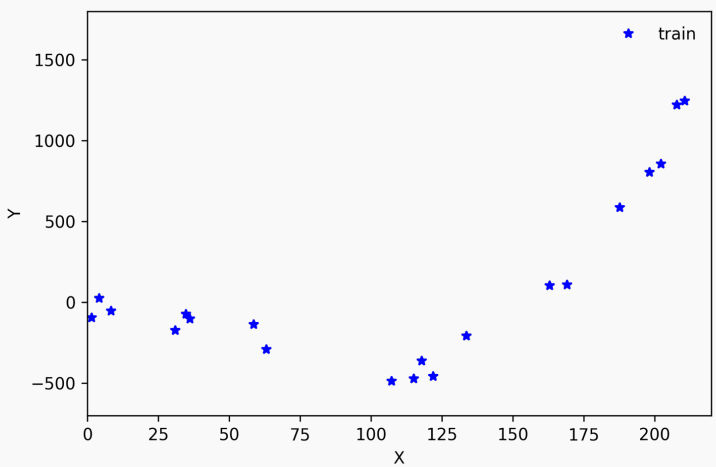
- Overfitting
- Variable Selection

Cross Validation ← to have "multiple test sets" for use to choose between models

Regularization: LASSO and Ridge

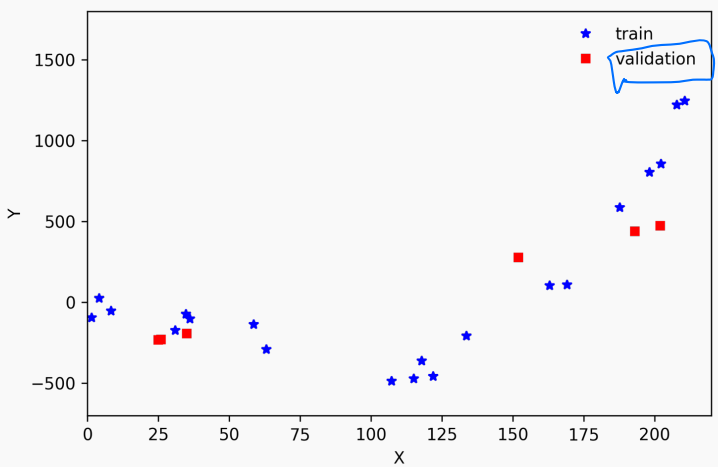
Regularization Methods: A Comparison

Cross Validation



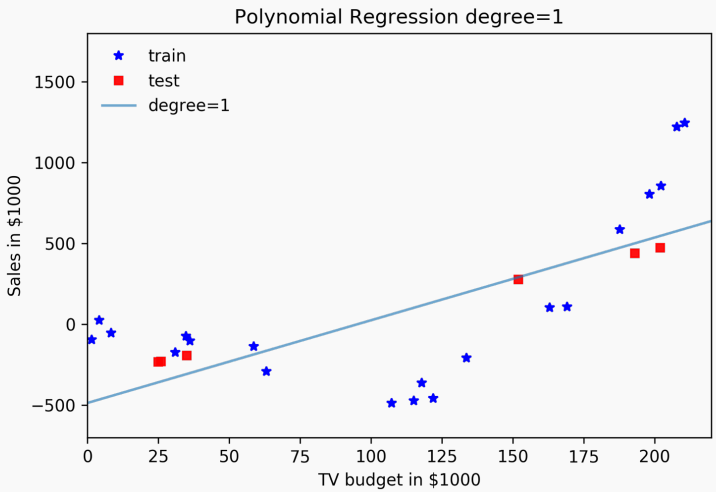
← non-linear relationship

Cross Validation

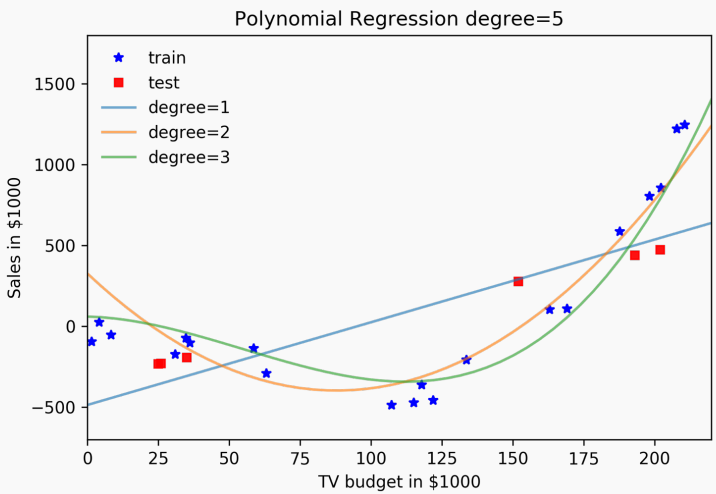


orgind train set
↓
- new train
- new validation

Cross Validation



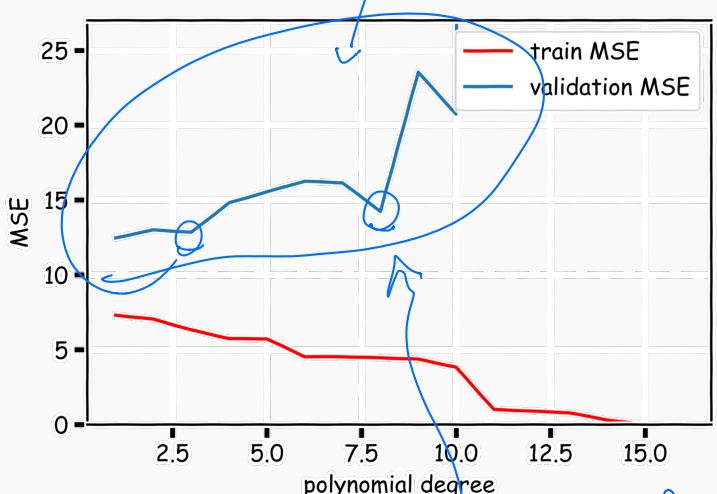
Cross Validation



← n=5 in test

the linear model seems to win out in the test set.

Validation



it's "bouncing" around due to a single, specific set of data

for a single validation/test set has a lot of uncertainty involved

overfit to the randomness in the test set.

Cross Validation: Motivation

thousands of models you are choosing between.

Using a single validation or test set to select amongst multiple models can be problematic - **there is the possibility of overfitting to the test set**.

One solution to the problems raised by using a single validation set is to evaluate each model on **multiple** validation sets and average the validation performance.

One can randomly split the training set into training and validation multiple times **but** randomly creating these sets can create the scenario where important features of the data never appear in our random draws.

cross-validation

Cross Validation (*k*-fold)

10 iterations

based on a single fold

90% will be used to train the model

10% is used to validate (evaluate)

10 folds

($n = 1000$ each

$n_{fold} = 100$ if $k = 10$ folds)

K-Fold Cross Validation

Given a data set $\{X_1, \dots, X_n\}$, where each $\{X_1, \dots, X_n\}$ contains *J* features.

To ensure that every observation in the dataset is included in at least one training set and at least one validation set we use the **K-fold validation**:

- split the data into *K* uniformly sized chunks, $\{C_1, \dots, C_K\}$
- we create *K* number of training/validation splits, using one of the *K* chunks for validation and the rest for training.

We fit the model on each training set, denoted $\hat{f}_{C_{-i}}$, and evaluate it on the corresponding validation set, $\hat{f}_{C_{-i}}(C_i)$. The ***cross validation is the performance*** of the model averaged across all validation sets:

$$CV(\text{Model}) = \frac{1}{K}\sum_{i=1}^{K} L(\hat{f}_{C_{-i}}(C_i))$$

where *L* is a loss function.

Leave-One-Out Cross Validation (n-fold cross-validation)

*Or using the **leave one out*** method:

- validation set: $\{X_i\}$
- training set: $X_{-i} = \{X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n\}$

for $i = 1, \ldots, n$:

We fit the model on each training set, denoted $\hat{f}_{X_{-i}}$, and evaluate it on the corresponding validation set, $\hat{f}_{X_{-i}}(X_i)$.

The ***cross validation score*** is the performance of the model averaged across all validation sets:

$$CV(\text{Model}) = \frac{1}{n}\sum_{i=1}^{n} L(\hat{f}_{X_{-i}}(X_i))$$

where *L* is a loss function.

*Note: for OLS linear regression, there is a closed form solution to this calculation!

Random Subsets (sometimes called bootstrapped) Cross-Validation

- An alternative approach to doing *k*-fold cross-validation is to do many random subsets of random selected observations.
- For example, from the original training set, 80% of the observations may be randomly selected to train the models, and the other 20% are left to validate/evaluate the models to choose between them.
- Then this is done a 2nd time for a new random subset of 80% of the observations. And then a 3rd time, etc.
- What is the advantage to doing CV this way? To doing k-fold?

it is much faster.

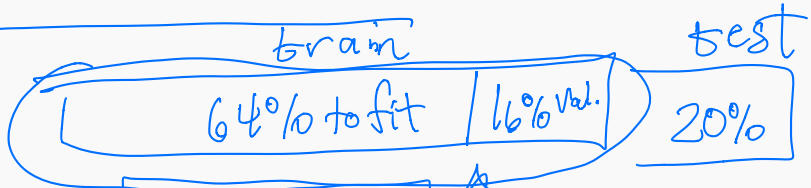
↑ since you are doing many more iterations, you "smooth out" the MSE_{test} curve

Predictor Selection: Cross Validation

80-20 splits: not necessarily the best

Question: What is the right ratio of train/validate/test, how do I choose *k* in *k*-fold?

cross-validation →



Question: What is the difference in multiple predictors and polynomial regression in model selection?

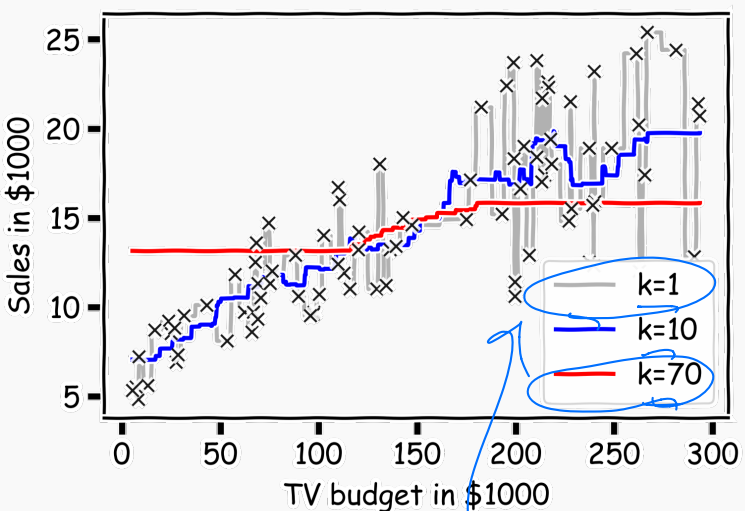
K=5-fold C.V.

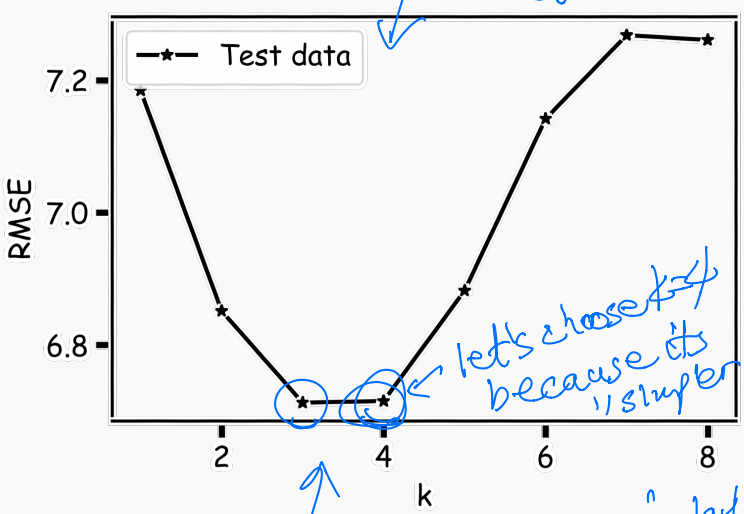
We can frame the problem of degree selection for polynomial models as a predictor selection problem:

which of the predictors $\{x, x^2, \ldots, x^m\}$, should we select for modeling?

← some order or hierarchy in removing or adding variables.

k-NN Revisited





a single test set: MSE (&RMSE)

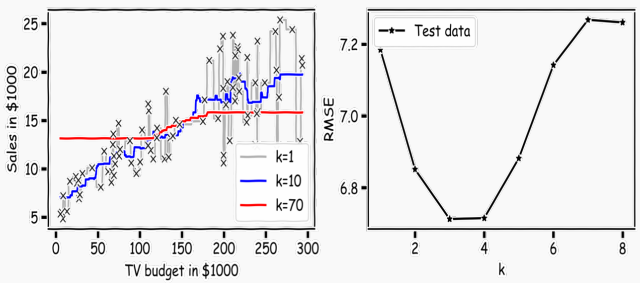
let's choose k=4 because it's "simpler"

perform very similarly

k=1 is overfit

kNN Revisited

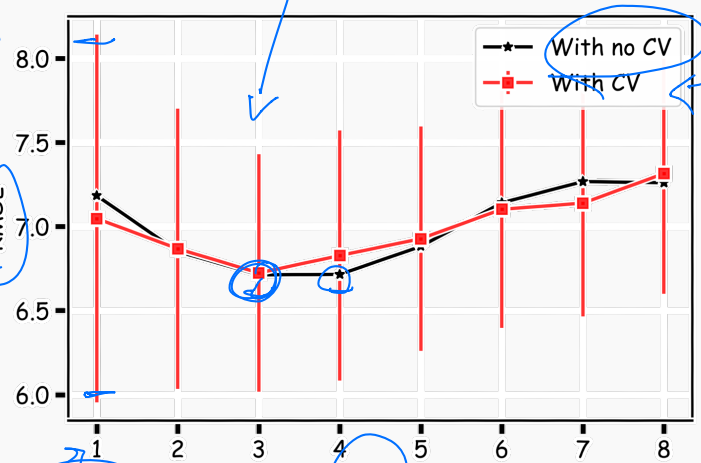
Recall our first simple, intuitive, non-parametric model for regression – the kNN model. We saw that it is vitally important to select an appropriate *k* for the data.



If the *k* is too small then the model is very sensitive to noise (since a new prediction is based on very few observed neighbors), and if the *k* is too large, the model tends towards making constant predictions.

A principled way to choose *k* is **through K-fold cross validation.**

K-fold with k=100 folds



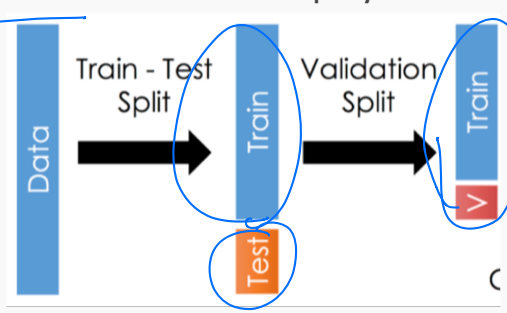
What happens to the test set?

Note: there is still usefulness to a single test set.

Cross-validation is great to choose between many models or tune a hyperparameter (which we will see in a bit). But the error of the model is likely under-evaluated. Why?

↑ overfit to validation sets

This is where the original test set comes in to play: evaluate the model's error.



many cross-validations

leave this alone until the very last step

Lecture Outline

Brief Review

- Overfitting

- Variable Selection

Cross Validation

Regularization: LASSO and Ridge

Regularization Methods: A Comparison

Regularization: An Overview

The idea of regularization revolves around modifying the loss function L; in particular, we add a regularization term that penalizes some specified properties of the model parameters

$$L_{reg}(\beta) = L(\beta) + \lambda R(\beta),$$

where λ is a scalar that gives the weight (or importance) of the regularization term.

Fitting the model using the modified loss function L_{reg} would result in model parameters with desirable properties (specified by R).

LASSO Regression (vs. Ridge Regression)

Since we wish to discourage extreme values in model parameter, we need to choose a regularization term that penalizes parameter magnitudes. For our loss function, we will again use MSE.

Together our regularized loss function is:

$$L_{LASSO}(\beta) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \boldsymbol{\beta}^\top \boldsymbol{x}_i|^2 + \lambda\sum_{j=1}^{J}|\beta_j|.$$

MSE

penalty based on "L_1 norm"

Note that $\sum_{j=1}^{J}|\beta_j|$ is the $\boldsymbol{l_1}$ norm of the vector $\boldsymbol{\beta}$

$$\sum_{j=1}^{J}|\beta_j| = \|\boldsymbol{\beta}\|_1$$

couple specifics:

1. the intercept is not penalized
2. Its helpful (typically) to standardize your x-variables.

Standardize ⇒ mean = 0, sd = 1

LASSO Regression

Hence, we often say that L_{LASSO} is the loss function for l_1 regularization.

Finding the model parameters β_{LASSO} that minimize the l_1 regularized loss function is called ***LASSO regression***.

In [ ]:
```
from sklearn.linear_model import Lasso
```

In [22]:
```
lasso_regression = Lasso(alpha=1.0, fit_intercept=True)
lasso_regression.fit(np.vstack((X_train, X_val)), np.hstack((y_train, y_val)))

print('Lasso regression model:\n {} + {}^T . x'.format(lasso_regression.intercept_, lasso_regression.coe
```

```
Lasso regression model:
 10.424895873901445 + [ 0.24482603  3.48164594  1.84836859 -0.06864603 -0.         -0.
 -0.02249766 -0.          0.          0.          0.          ]^T . x
```

In [23]:
```
print('Train R^2: {}, test R^2: {}'.format(lasso_regression.score(np.vstack((X_train, X_val)),
                                                                  np.hstack((y_train, y_val))),
                                           lasso_regression.score(X_test, y_test)))
```

```
Train R^2: 0.48154992527975765, test R^2: 0.6846451270316087
```

"shrinks" coefficient (β) towards zero (some become exactly zero)

Ridge Regression

Alternatively, we can choose a regularization term that penalizes the squares of the parameter magnitudes. Then, our regularized loss function is:

$$L_{Ridge}(\beta) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \boldsymbol{\beta}^\top \boldsymbol{x}_i|^2 + \lambda \sum_{j=1}^{J}\beta_j^2.$$

Note that $\sum_{j=1}^{J}|\beta_j|^2$ is the $\boldsymbol{l_2}$ norm of the vector $\boldsymbol{\beta}$

$$\sum_{j=1}^{J}\beta_j^2 = \|\boldsymbol{\beta}\|_2^2$$

Ridge Regression

Hence, we often say that L_{ridge} is the loss function for l_2 regularization.

Finding the model parameters β_{ridge} that minimize the l_2 regularized loss function is called ***ridge regression.***

In [ ]:
```
from sklearn.linear_model import Ridge
```

In [20]:
```
X_train = train[all_predictors].values
X_val = validation[all_predictors].values
X_test = test[all_predictors].values

ridge_regression = Ridge(alpha=1.0, fit_intercept=True)
ridge_regression.fit(np.vstack((X_train, X_val)), np.hstack((y_train, y_val)))

print('Ridge regression model:\n {} + {}^T . x'.format(ridge_regression.intercept_, ridge_regression.coe
```

```
Ridge regression model:
 -525.7662550875951 + [ 0.24007312  8.42566029  2.04098593 -0.04449172 -0.01227935  0.41902475
 -0.50397312 -4.47065168  4.99834262  0.          0.          0.29892679]^T . x
```

In [21]:
```
print('Train R^2: {}, test R^2: {}'.format(ridge_regression.score(np.vstack((X_train, X_val)),
                                                                  np.hstack((y_train, y_val))),
                                           ridge_regression.score(X_test, y_test)))
```

```
Train R^2: 0.5319764744847737, test R^2: 0.7881798111697319
```

penalizes / shrinks large values of β

Choosing λ

In both ridge and LASSO regression, we see that the larger our choice of the **regularization parameter** λ, the more heavily we penalize large values in β,

- If λ is close to zero, we recover the MSE, i.e. ridge and LASSO regression is just ordinary regression.
- If λ is sufficiently large, the MSE term in the regularized loss function will be insignificant and the regularization term will force β_{ridge} and β_{LASSO} to be close to zero.

To avoid ad-hoc choices, we should select λ using cross-validation.

Ridge - Computational complexity

Solution to ridge regression:

$$\hat{\beta}_{Ridge} = (X^T X + \lambda I)^{-1} X^T Y$$

$$\hat{\beta}_{OLS} = (X^T X)^{-1} X^T Y$$

The solution of the Ridge/Lasso regression involves three steps

- Select λ
- Find the minimum of the ridge/Lasso regression cost function (using linear algebra) as with the multiple regression and record the R^2 **on the test set**.
- Find the λ that gives the largest R^2

Lecture Outline

Brief Review

- Overfitting

- Variable Selection

Cross Validation

Regularization: LASSO and Ridge

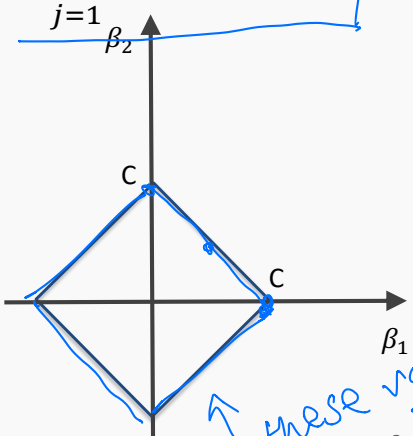
Regularization Methods: A Comparison

The Geometry of Regularization (LASSO)

$$L_{LASSO}(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} |y_i - \boldsymbol{\beta}^T \boldsymbol{x}\Big|^2 + \lambda \sum_{j=1}^{J} |\beta_j|$$

$$\hat{\boldsymbol{\beta}}^{LASSO} = \operatorname{argmin} L_{LASSO}(\boldsymbol{\beta})$$

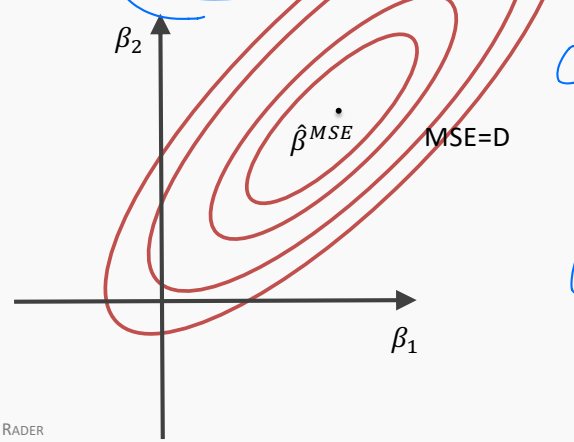
$$\lambda \sum_{j=1}^{J} \left| \hat{\beta}_j^{LASSO} \right| = C$$



these values of (β_1, β_2) have the same penalty factor

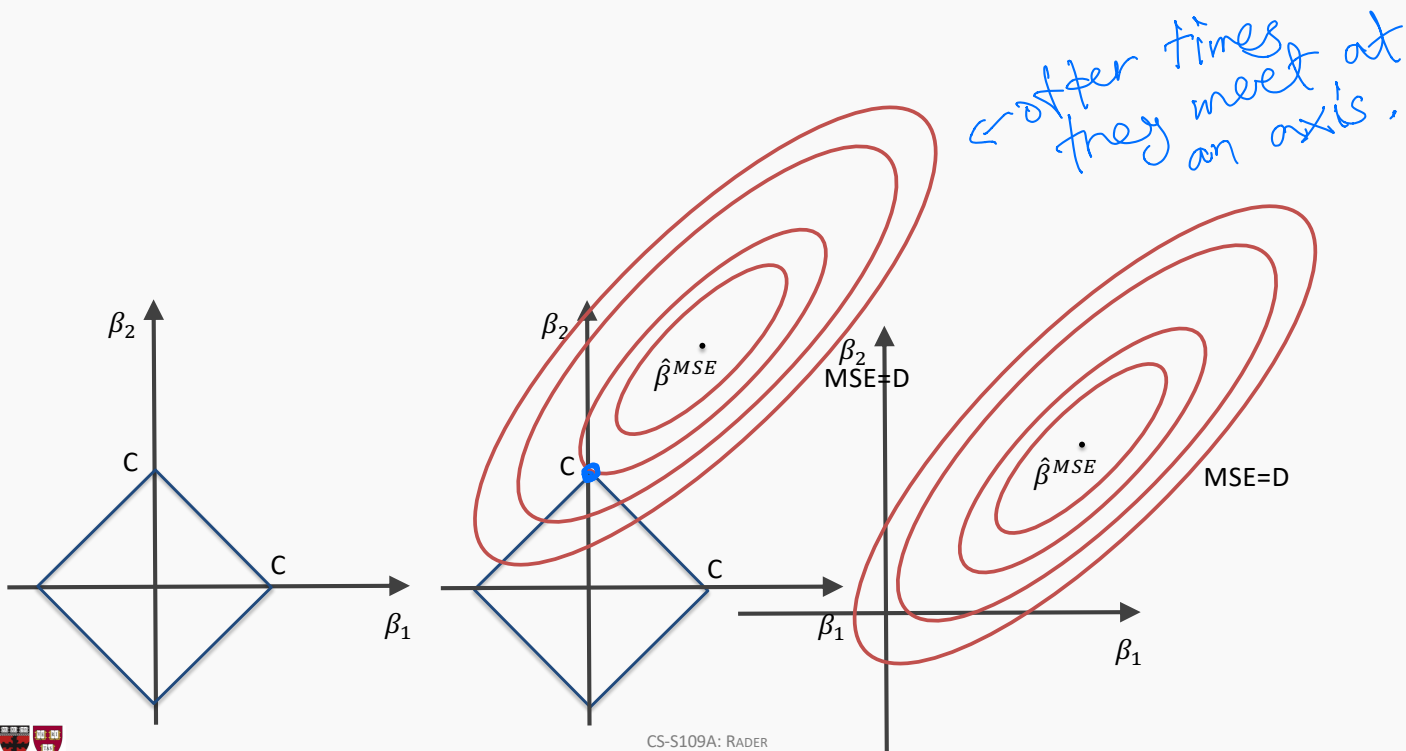
MSE

$$\frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{\boldsymbol{\beta}}^{LASSO^T} \boldsymbol{x}|^2 = D$$



Contour plot of MSE (equal values)

The Geometry of Regularization (LASSO)

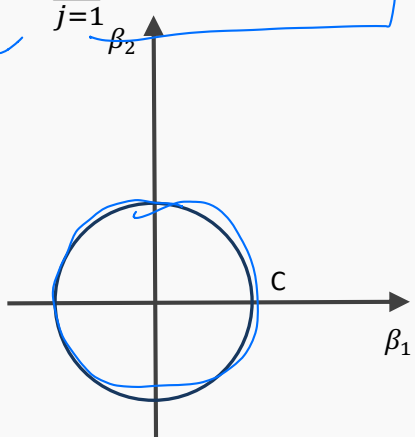


The Geometry of Regularization (Ridge)

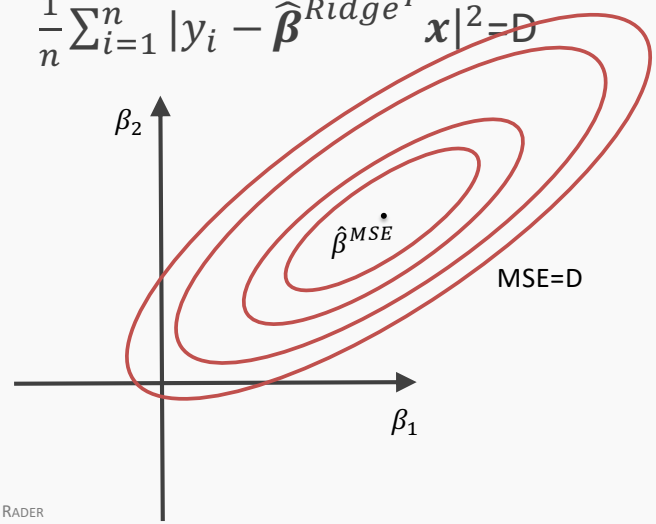
$$L_{Ridge}(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} |y_i - \boldsymbol{\beta}^T \boldsymbol{x}\Big|^2 + \lambda \sum_{j=1}^{J} (\beta_j)^2$$

$$\widehat{\boldsymbol{\beta}}^{Ridge} = \operatorname{argmin} L_{Ridge}(\boldsymbol{\beta})$$

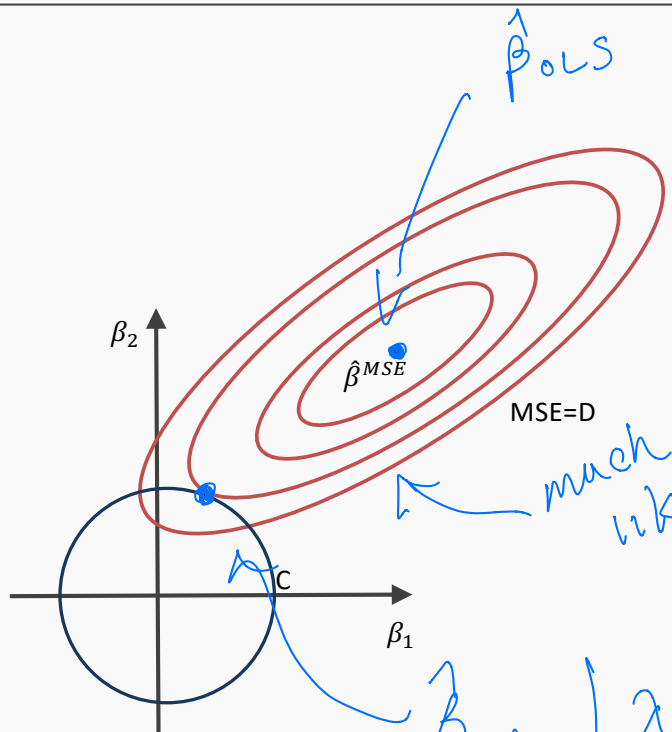
$$\lambda \sum_{j=1}^{J} \left|\hat{\beta}_j^{Ridge}\right|^2 = C$$



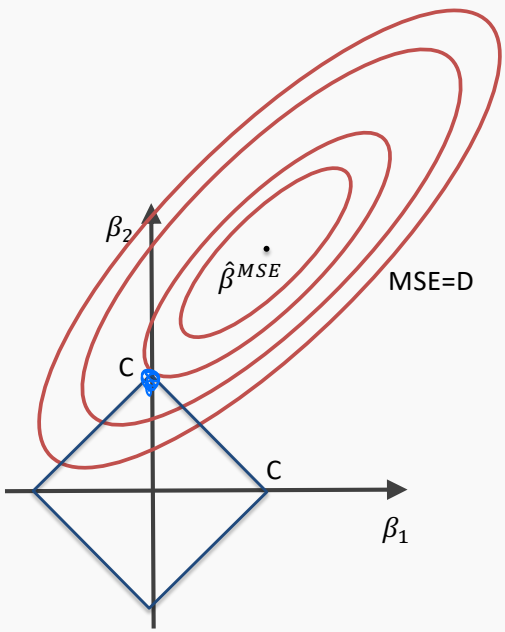
$$\frac{1}{n}\sum_{i=1}^{n} |y_i - \widehat{\boldsymbol{\beta}}^{Ridge^T} \boldsymbol{x}|^2 = D$$

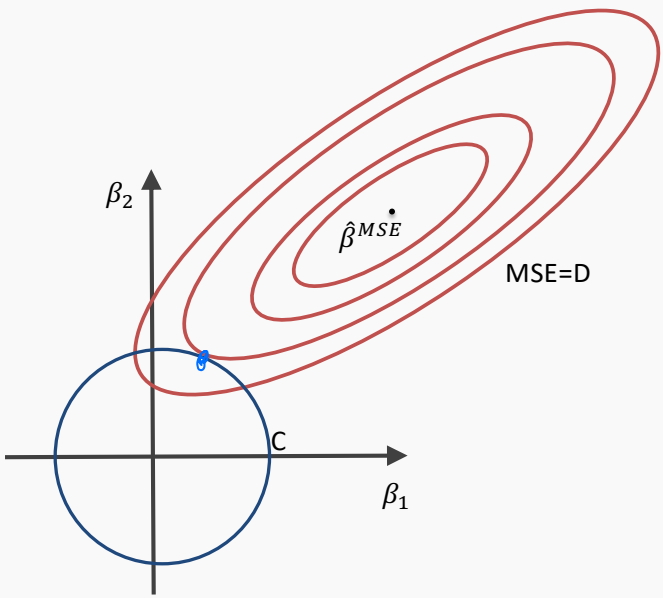


The Geometry of Regularization (Ridge)



The Geometry of Regularization





Examples

In [ ]:
```
from sklearn.linear_model import Lasso
```

In [22]:
```
lasso_regression = Lasso(alpha=1.0, fit_intercept=True)
lasso_regression.fit(np.vstack((X_train, X_val)), np.hstack((y_train, y_val)))

print('Lasso regression model:\n {} + {}^T . x'.format(lasso_regression.intercept_, lasso_regression.coe
```

```
Lasso regression model:
 10.424895873901445 + [ 0.24482603  3.48164594  1.84836859 -0.06864603 -0.         -0.
 -0.02249766 -0.          0.          0.          0.          ]^T . x
```

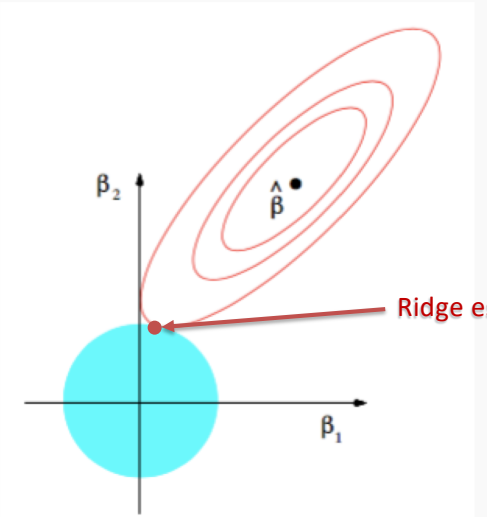
a lot more $\hat{\beta}_j = 0$ in Lasso

In [ ]:
```
from sklearn.linear_model import Ridge
```

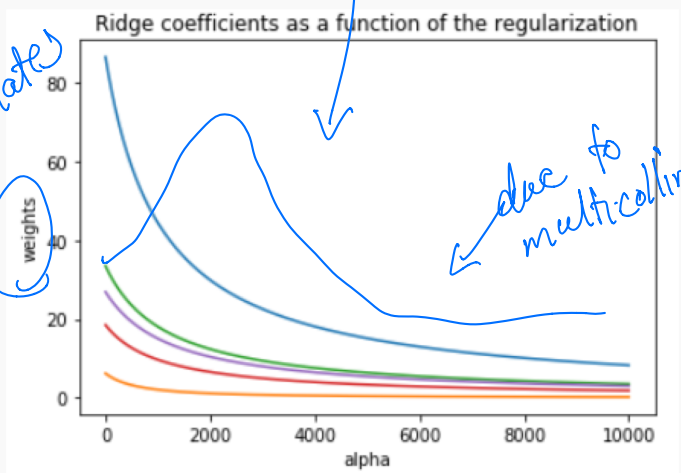
In [20]:
```
X_train = train[all_predictors].values
X_val = validation[all_predictors].values
X_test = test[all_predictors].values

ridge_regression = Ridge(alpha=1.0, fit_intercept=True)
ridge_regression.fit(np.vstack((X_train, X_val)), np.hstack((y_train, y_val)))

print('Ridge regression model:\n {} + {}^T . x'.format(ridge_regression.intercept_, ridge_regression.coe
```

```
Ridge regression model:
 -525.7662550875951 + [ 0.24007312  8.42566029  2.04098593 -0.04449172 -0.01227935  0.41902475
 -0.50397312 -4.47065168  4.99834262  0.          0.          0.29892679]^T . x
```

Ridge visualized

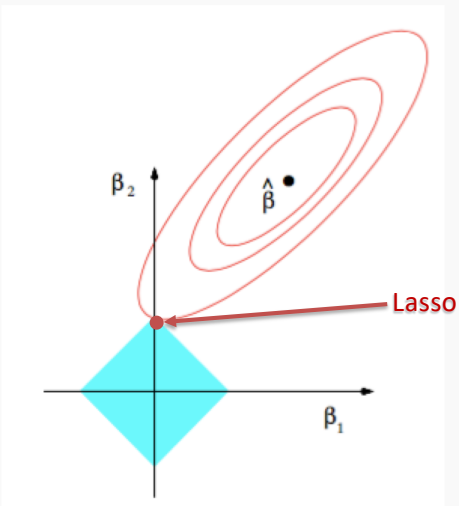


The ridge estimator is where the constraint and the loss intersect.

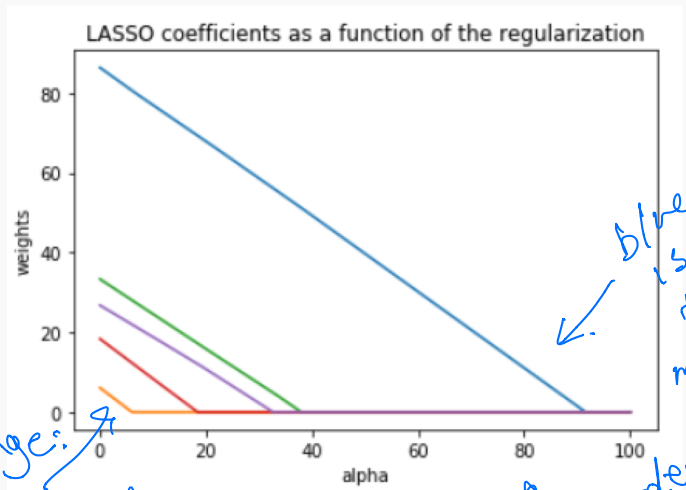


The values of the coefficients decrease as lambda increases, but they are not nullified.

LASSO visualized



The Lasso estimator tends to zero out parameters as the OLS loss can easily intersect with the constraint on one of the axis.

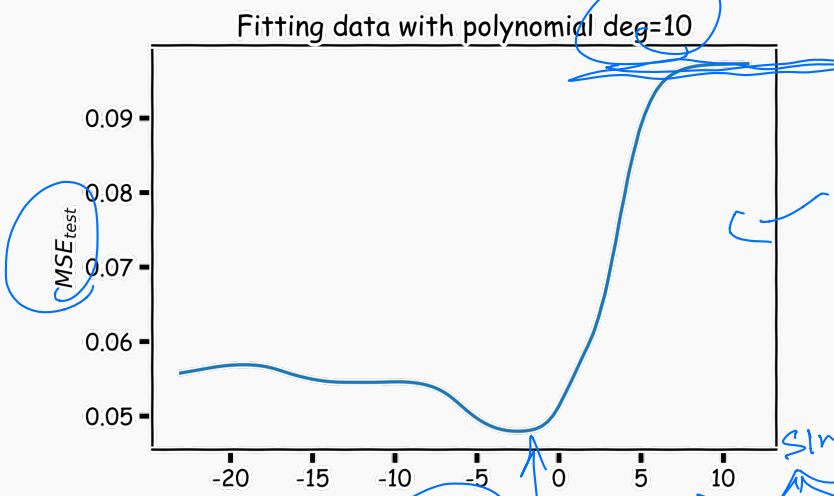


The values of the coefficients decrease as lambda increases, and are nullified fast.

Ridge regularization with **validation** only: step by step

1. split data into $\{\{X,Y\}_{train}, \{X,Y\}_{validation}, \{X,Y\}_{test}\}$
2. for λ in $\{\lambda_{min}, \ldots \lambda_{max}\}$:
   1. determine the β that minimizes the L_{ridge}, $\hat{\beta}_{Ridge}(\lambda) = \left(\mathrm{X}^{\mathrm{T}}\mathrm{X} + \lambda I\right)^{-1} X^T Y$, using the train data.
   2. record $L_{MSE}(\lambda)$ using validation data.
3. select the λ that minimizes the loss on the validation data,
   $$\lambda_{ridge} = \text{argmin}_{\lambda}\, L_{MSE}(\lambda)$$
4. Refit the model using both train and validation data, $\{\{X,Y\}_{train}, \{X,Y\}_{validation}\}$, resulting to $\hat{\beta}_{ridge}(\lambda_{ridge})$
5. report MSE or R^2 on $\{X,Y\}_{test}$ given the $\hat{\beta}_{ridge}(\lambda_{ridge})$

Ridge regularization with **validation** only: step by step



asymptotes at $\frac{\sum (y_i - \bar{y})^2}{n}$

"classic" Ridge-MSE plot

simple

large λ shrinking all β to zero

complex

Goldilocks zone

Lasso regularization with **validation** only: step by step

1. split data into $\{\{X,Y\}_{train}, \{X,Y\}_{validation}, \{X,Y\}_{test}\}$
2. for λ in $\{\lambda_{min}, \ldots \lambda_{max}\}$:
   A. determine the β that minimizes the L_{lasso}, $\hat{\beta}_{lasso}(\lambda)$, using the train data. **This is done using a numerical solver.**
   B. record $L_{MSE}(\lambda)$ using validation data
3. select the λ that minimizes the loss on the validation data,
   $\lambda_{lasso} = \text{argmin}_{\lambda}\, L_{MSE}(\lambda)$
4. Refit the model using both train and validation data, $\{\{X,Y\}_{train}, \{X,Y\}_{validation}\}$, resulting to $\hat{\beta}_{lasso}(\lambda_{lasso})$
5. report MSE or R^2 on $\{X,Y\}_{test}$ given the $\hat{\beta}_{lasso}(\lambda_{lasso})$

- might be slower,
- might not be a unique solution

Ridge regularization with **CV**: step by step

multiple folds

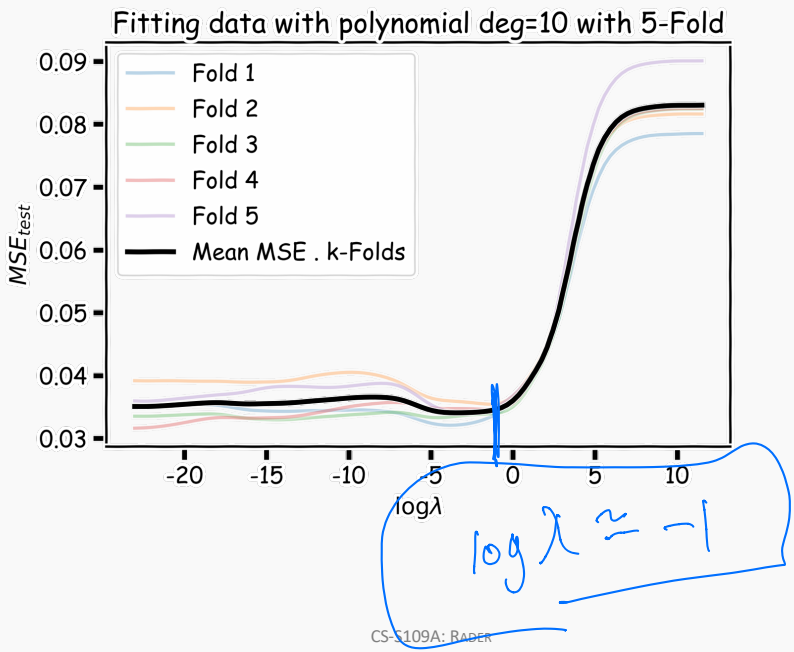
	λ_1	λ_2	...	λ_n
k_1	L_{11}	L_{12}	..	...
k_2	L_{21}	...	..	...
...	..	...	..	...
k_n	...	...	...	...
E[]	$\bar{L}_1$	$\bar{L}_2$	...	$\bar{L}_n$

MSE across all folds

1. remove $\{X,Y\}_{test}$ from data
2. split the rest of data into K folds, $\{\{X,Y\}_{train}^{-k}, \{X,Y\}_{val}^{k}\}$
3. for k in $\{1,\dots,K\}$
    1. for λ in $\{\lambda_0,\dots,\lambda_n\}$:
        A. determine the β that minimizes the L_{ridge}, $\hat{\beta}_{ridge}(\lambda,k) = \left(\mathrm{X}^{\mathrm{T}}\mathrm{X} + \lambda I\right)^{-1} X^T Y$, using the train data of the fold, $\{X,Y\}_{train}^{-k}$.
        B. record $L_{MSE}(\lambda,k)$ using the validation data of the fold $\{X,Y\}_{val}^{k}$

        At this point we have a 2-D matrix, rows are for different k, and columns are for different λ values.
4. Average the $L_{MSE}(\lambda,k)$ for each λ, $\bar{L}_{MSE}(\lambda)$.
5. Find the λ that minimizes the $\bar{L}_{MSE}(\lambda)$, resulting to λ_{ridge}.
6. Refit the model using the full training data, $\{\{X,Y\}_{train}, \{X,Y\}_{val}\}$, resulting to $\hat{\beta}_{ridge}(\lambda_{ridge})$
7. report MSE or R^2 on $\{X,Y\}_{test}$ given the $\hat{\beta}_{ridge}(\lambda_{ridge})$

Ridge regularization with **validation** only: step by step



Variable Selection as Regularization

Since LASSO regression tend to produce zero estimates for a number of model parameters - we say that LASSO solutions are **sparse** - we consider LASSO to be a method for variable selection.

Many prefer using LASSO for variable selection (as well as for suppressing extreme parameter values) rather than stepwise selection, as LASSO avoids the statistic problems that arises in stepwise selection.

Question: What are the pros and cons of the two approaches?

Ridge vs. Lasso vs. Variable Selection: which is better?

So there are 3 different approaches (so far) to dealing with high dimensional data (a large # predictors, *p*). Which choice wins out?

As you might imagine, **it depends** on the setting!

- Sequential Variable Selection: does really well when there are **only a few strong predictors**. But is *slow*.
- Ridge: does really well when there are **a lot of weak to moderately strong predictors**.
- Lasso: does really well when there is **a mix of useless predictors and moderate to strong predictors**.

Elastic Net Regularization: the best of both worlds?

We will not get into the details here, but there is a blended approach of Ridge and LASSO called Elastic Net.

It uses the loss function:

$$\hat{\beta} \equiv \underset{\beta}{\operatorname{argmin}}(\|y - X\beta\|^2 + \lambda_2 \|\beta\|^2 + \lambda_1 \|\beta\|_1)$$

A grid-search cross-validation can then be used to determine the best choices of the 2 penalty terms in this loss function (this idea comes in handy in future models).


Tableau
66.6K views
0:01 / 1:29