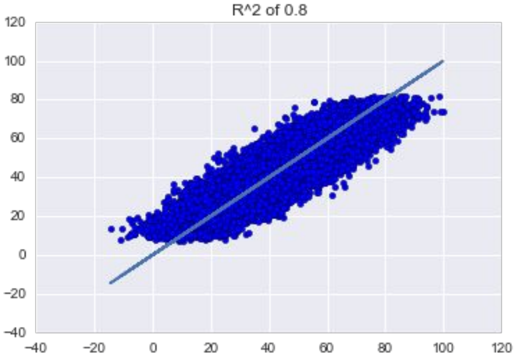
Model Evaluation, Complexity and Regularization

Lecture 3

How to train and evaluate a model?

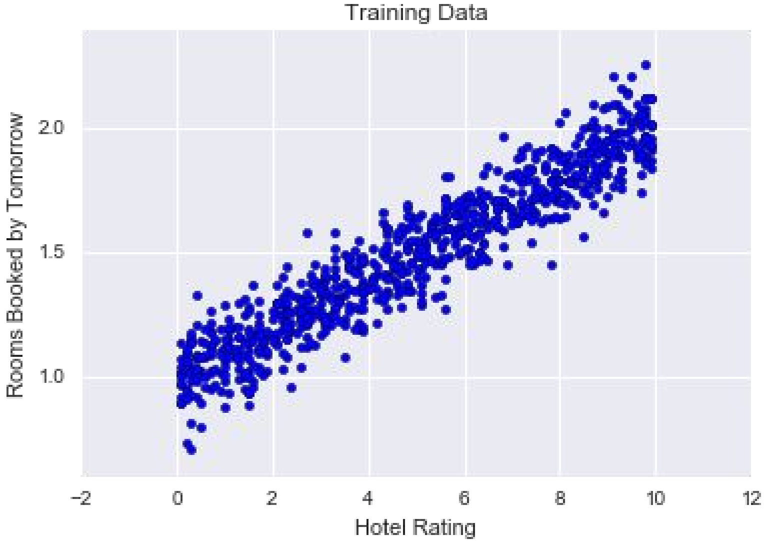
- **Training set** - the data your model ‘learns’ from.
- **Testing set** - the data your model uses to test what it’s learned.
- This is true of all unsupervised machine learning models.

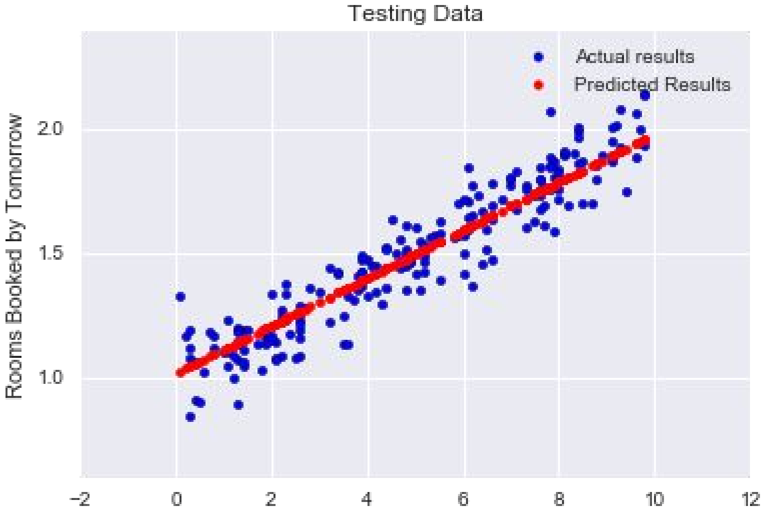


This leaves 4 questions:

1. What **methods** do we use to **train the model** ? (we have focused on linear regression)
2. What **metric** do we use to **evaluate performance**?
3. How do we **choose** what will be **training** and what will be **testing** data?
4. How do we **select features** optimally to maximize performance on testing set.

How to train and evaluate a model?

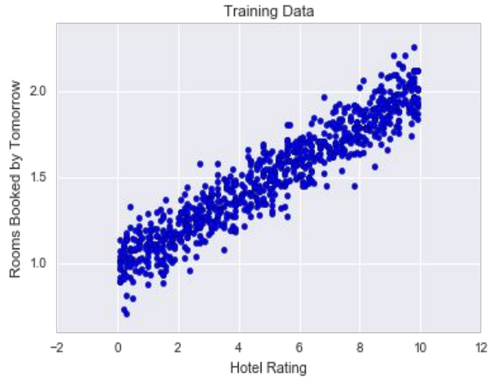


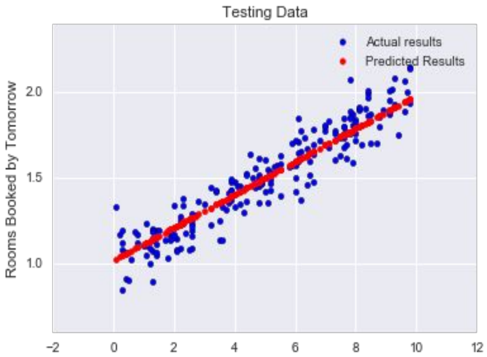


$$\mathcal{L}(y, f(x)) := \frac{1}{N}\sum_{i=1}^{N} |y_i - f(x_i)|^p$$

Minimized using **gradient descent** on training set (left picture)

How to train and evaluate a model?





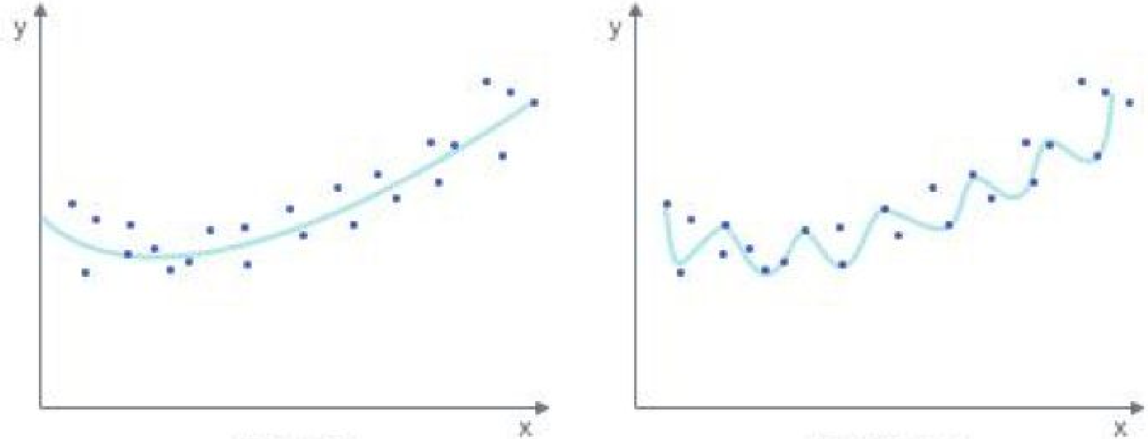
Training: $$\mathcal{L}(y, f(x)) := \frac{1}{N}\sum_{i=1}^{N} |y_i - f(x_i)|^p$$

- It's extremely important that you test on a different subset of data from what you trained on (why?)
- In general you want these selections to be as random as possible to minimize potential bias.

Complexity and Regularization

How less is more sometimes - the problem of overfitting

Model Complexity and Regularization



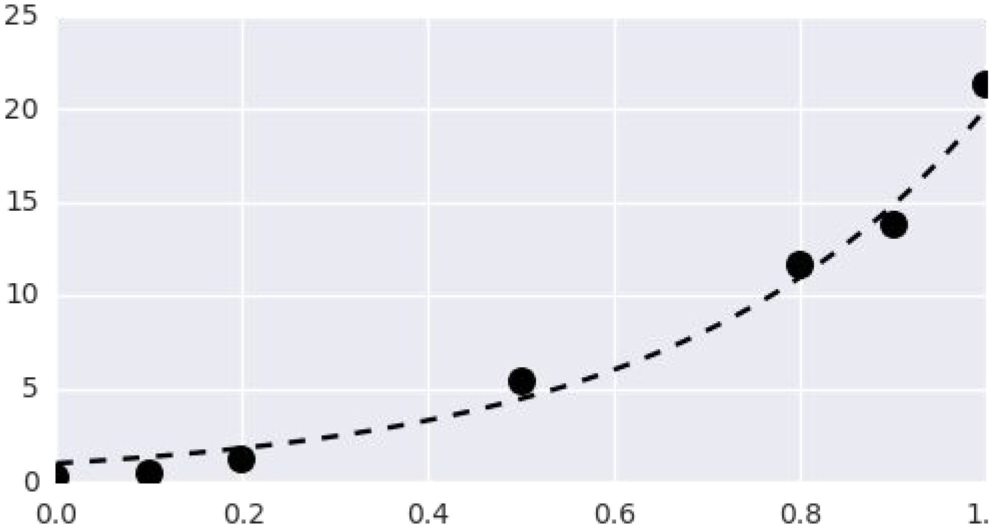
Good fit

Overfitting

Over fitting occurs when the trained model picks up too much ‘variance’ , which isn’t reflective of its true behavior, and therefore will perform badly on new data.

Example - Polynomial Overfitting

$$y = \exp(3x) + \text{noise}$$



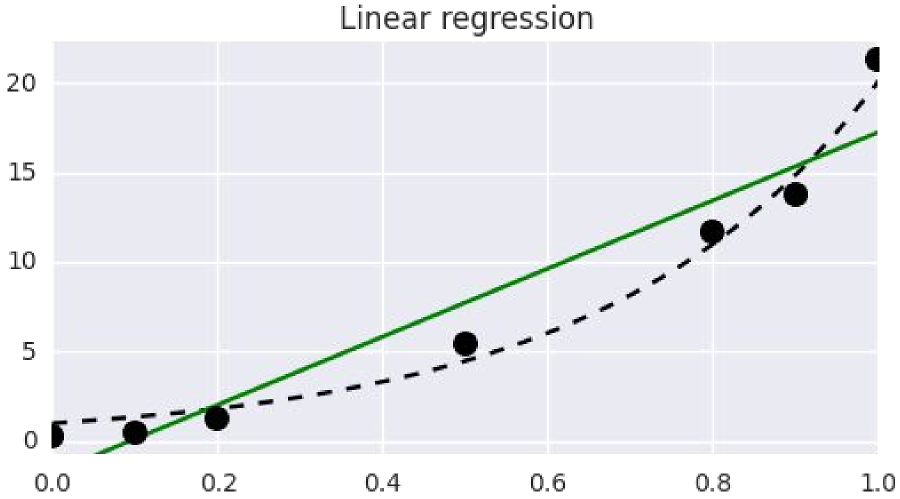
- What is the right degree of polynomial to fit this curve?

- We will see that despite the exponential technically having an infinite degree, we still can risk picking up too much variance if we don’t limit the order.

Example - Polynomial Overfitting

$$y = \exp(3x) + \text{noise}$$

- A straight line doesn’t seem complex enough to model this curve as the figure shows.

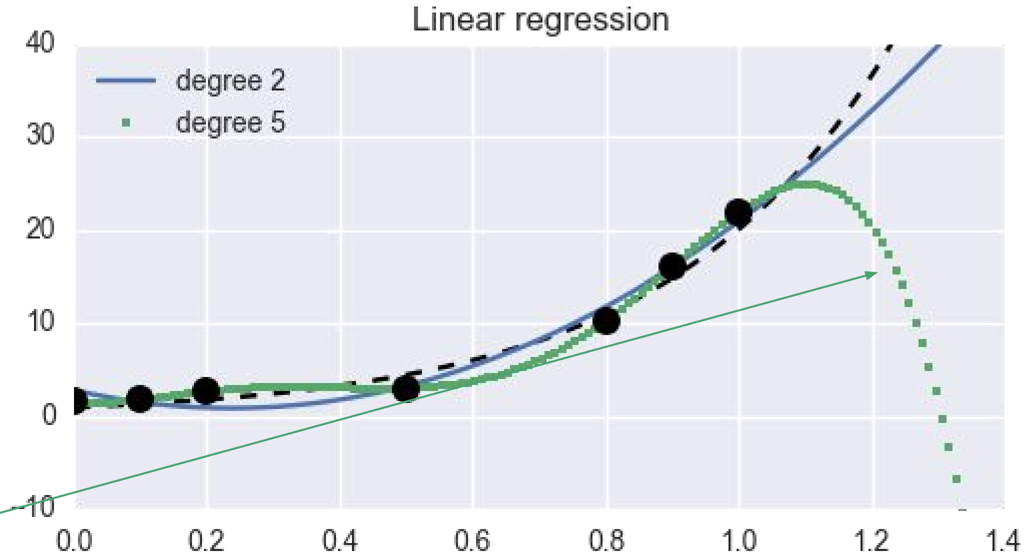


What is the right degree of polynomial?

- Notice that the degree 5 polynomial appears to fit the data better at first, but it doesn't generalize well.

- The degree 2 polynomial here is the better fit ultimately.

- How do we make this more precise?

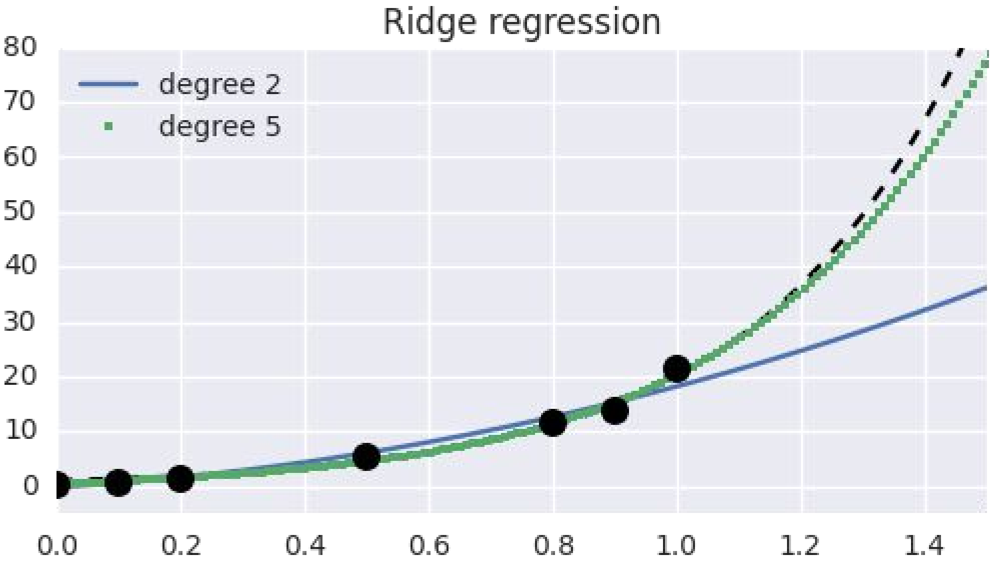


The linear model picks up variance, and blows up the coefficients, causing this behavior

Ridge Regression - Penalize coefficients

- Rather than manually choosing the degree, what we actually do in practice is to penalize the size of the coefficients.

- This allows us to have more control in how we fit the data in a way which we can generalize.



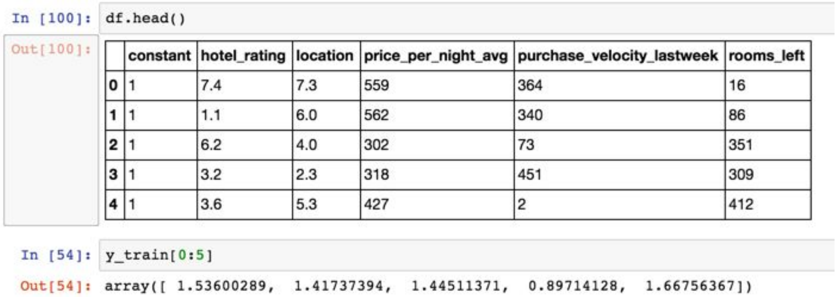
$$\beta \cdot x_i = \sum_{k=1}^{n} \beta_k \theta_i^k$$

$$\frac{1}{N}\sum_{i=1}^{N} |y_i - \beta \cdot x_i|^2 + \lambda \sum_{k=1}^{n} |\beta_k|$$

Penalize the size of the coefficients, rather than the degree of the polynomial itself.

A more realistic example from

Let’s try to predict the number of rooms that will be booked tomorrow from attributes about the hotel.



```
In [100]: df.head()
```

Out[100]:

	constant	hotel_rating	location	price_per_night_avg	purchase_velocity_lastweek	rooms_left
0	1	7.4	7.3	559	364	16
1	1	1.1	6.0	562	340	86
2	1	6.2	4.0	302	73	351
3	1	3.2	2.3	318	451	309
4	1	3.6	5.3	427	2	412

```
In [54]: y_train[0:5]
Out[54]: array([ 1.53600289,  1.41737394,  1.44511371,  0.89714128,  1.66756367])
```

Actual Rule:

$$y = 0.1 \cdot (\text{hotel rating}) + \text{noise}$$

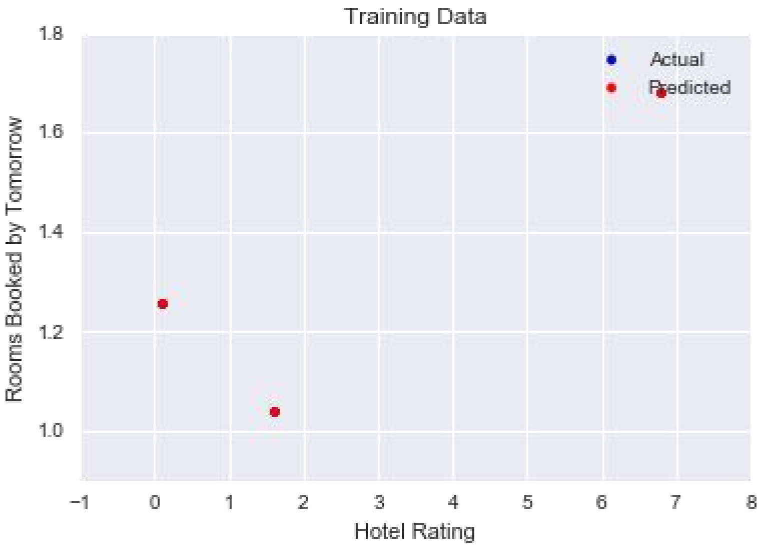
Model Complexity and Regularization

Let's take a concrete example and show what can go wrong when we include too much information in the model.

$$\begin{aligned} y_1 &= \alpha x_{11} + \beta x_{12} + \gamma x_{13} \\ y_2 &= \alpha x_{21} + \beta x_{22} + \gamma x_{23} \\ y_3 &= \alpha x_{31} + \beta x_{32} + \gamma x_{33} \end{aligned}$$

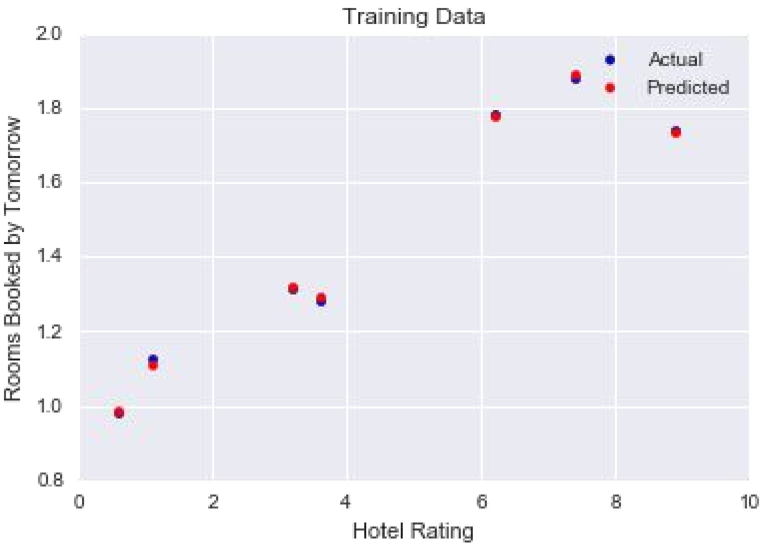
Here X is the matrix of features from the previous slide: price, location, etc.

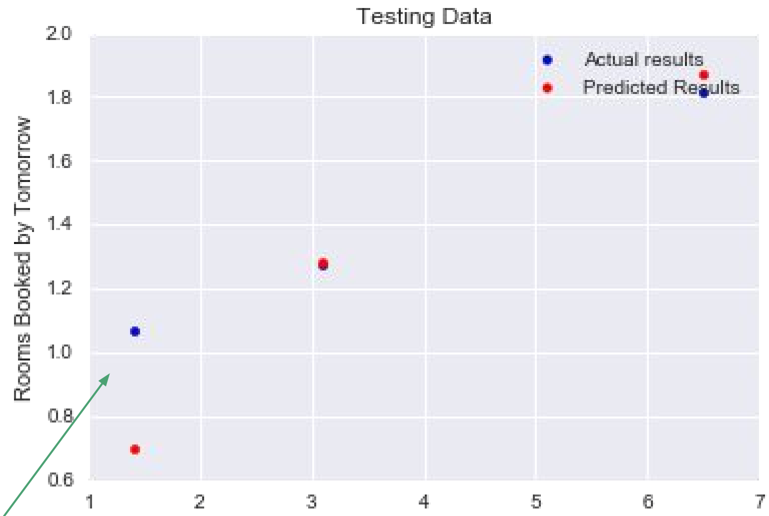
All features being used



Model Complexity and Regularization

All features being used - variance is picked up!



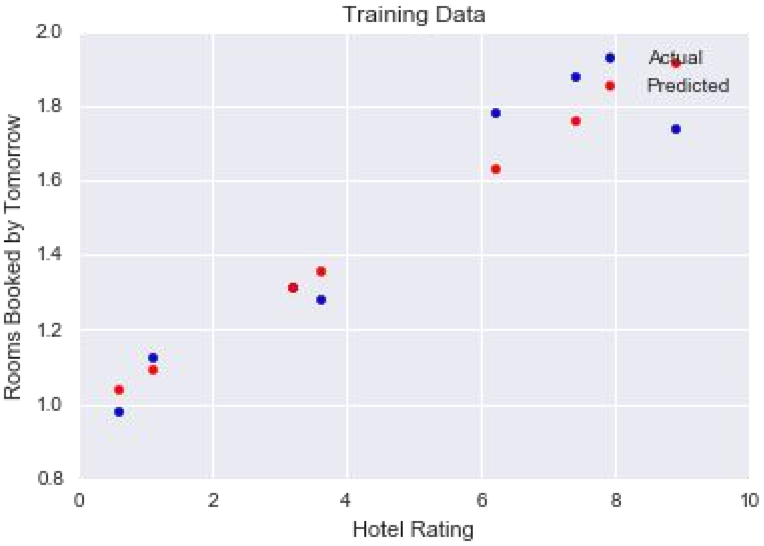


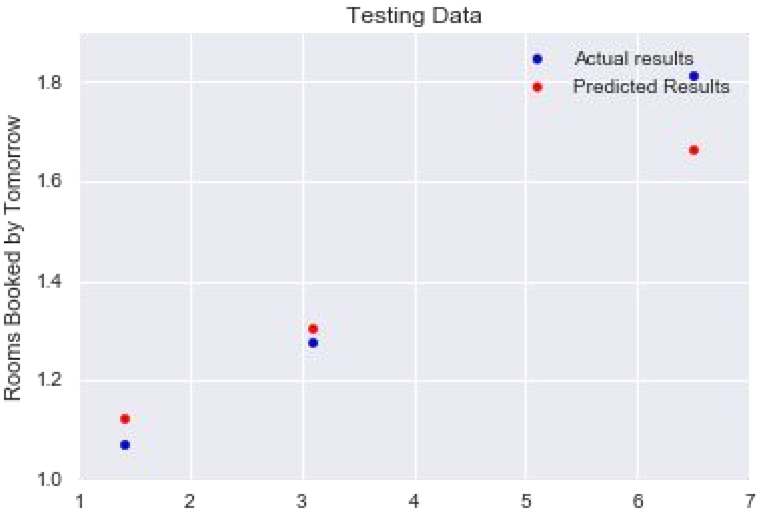
RMSE: 0.51

The variance picks up too much noise.

Model Complexity and Regularization

One feature being used - model generalizes much better!

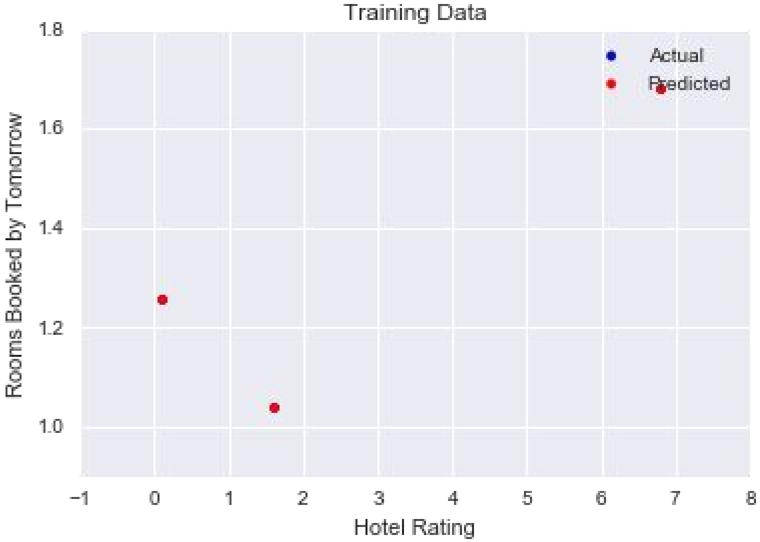




RMSE: 0.07

Model Complexity and Regularization

All features being used



How can we understand this?

$$\begin{aligned} y_1 &= \alpha x_{11} + \beta x_{12} + \gamma x_{13} \\ y_2 &= \alpha x_{21} + \beta x_{22} + \gamma x_{23} \\ y_3 &= \alpha x_{31} + \beta x_{32} + \gamma x_{33} \end{aligned}$$

The above system has a unique collection of features (alpha, beta, gamma) whenever X is invertible - this means that **we can solve exactly for each data point!** This may not be reflective of the model.

Question: How do we find the ‘correct’ complexity for a model?

Model Complexity and Regularization

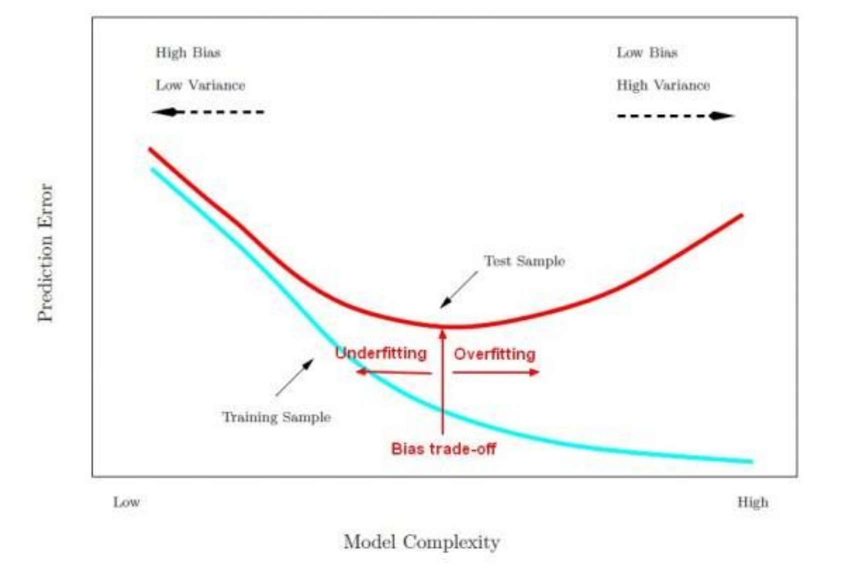
In fact, **adding more variables will *always* improve performance on training data**.

So if somebody says that you should just throw as many variables as possible in, you know they don't know what they're doing.

Variance/Bias Tradeoff and Learning Curves

How less is more sometimes - the problem of overfitting

We want to minimize the prediction error



How do we define model complexity though?

- In the first polynomial examle, we simply restricted the number of features in the model.
- Can we be more precise than this?

Constrained Minimization

$$\mathcal{L}_\lambda(y, f(x)) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \beta \cdot x_i)^2 + \lambda||\beta||_{L^p}$$

How do we penalize coefficients to optimize for performance on testing data?

- The above is equivalent (from principles of the calculus of variations) to constraining the norm of the coefficients.
- We want to simply choose the lambda which performs best on randomly chosen held out data.

Convex Analysis (Calculus)

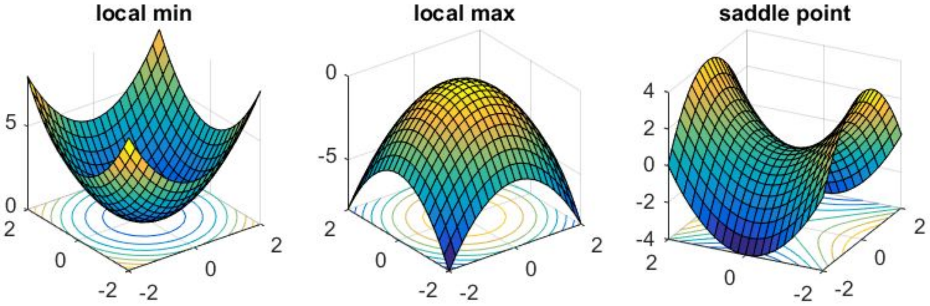
- Assume $\mathcal{L} : \mathbf{S} \to \mathbb{R}$ is continuously differentiable and convex, ie.

$$\mathcal{L}''(\beta) \geq 0 \text{ for all } \beta \in \mathbf{S}$$

$$\exists \beta_0 \text{ such that } \mathcal{L}(\beta_0) \leq \mathcal{L}(\beta) \text{ for all } \beta \in S$$

- There f has a minimum value. Moreover, the minimum is unique when f is strictly convex, ie.

$$\mathcal{L}''(\beta) \geq c > 0 \text{ for all } \beta \in \mathbf{S}$$



local min

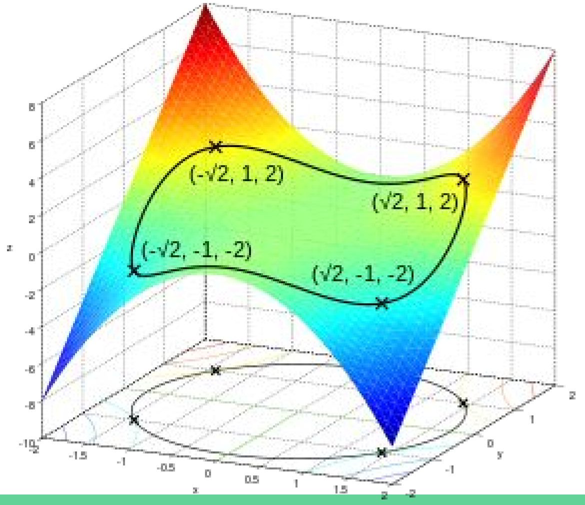
local max

saddle point

Review of Convex Analysis (Congrainted Calculus) - Part 2

- Assume $f : \mathbf{X} \to \mathbb{R}$ and $g : X \to \mathbb{R}$ are continuously differentiable, and f is convex.

- Then $\min\limits_{x \in X} f(x) + \lambda g(x)$ is equivalent to



$$\min_{x \in X} f(x)$$

$$\text{for } x \text{ s.t } g(x) = c$$

L2 vs L1 - when and why?

$$\mathcal{L}(y, f(x))_\lambda := \frac{1}{N}\sum_{i=1}^{N}(y_i - \beta \cdot x_i)^2 + \lambda \|\beta\|_{L^p}$$

- Constraining the coefficients by different norms produces different results.

- When p=1, the level sets are boxes.

- When p=2, the level sets are spheres.

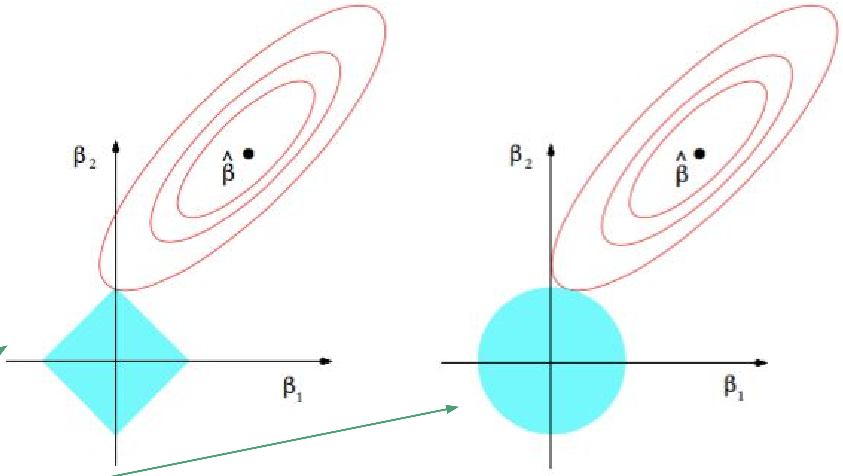


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \le t$ *and* $\beta_1^2 + \beta_2^2 \le t^2$*, respectively, while the red ellipses are the contours of the least squares error function.*

L2 vs L1 - when and why?

$$\mathcal{L}(y, f(x))_\lambda := \frac{1}{N}\sum_{i=1}^{N}(y_i - \beta \cdot x_i)^2 + \lambda\|\beta\|_{L^p}$$

- This is why the L1 norm results in more zero-valued features.

- The L2 norm spreads out errors more evenly amongst the features.

- L1 is better for features selection.

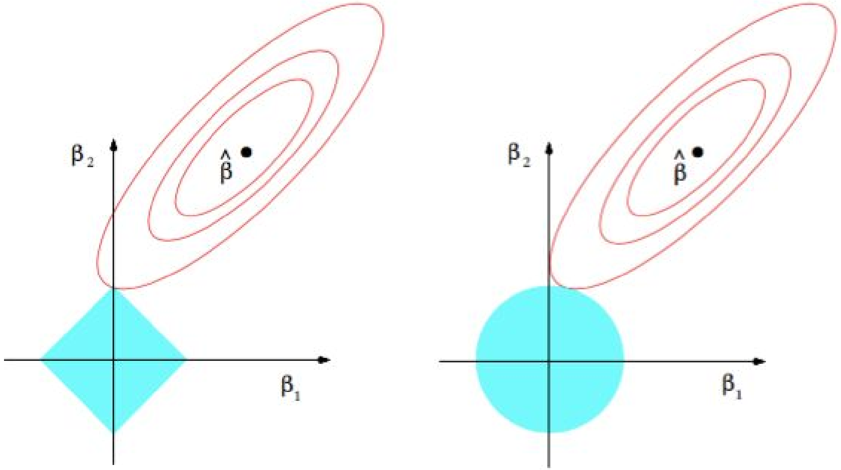
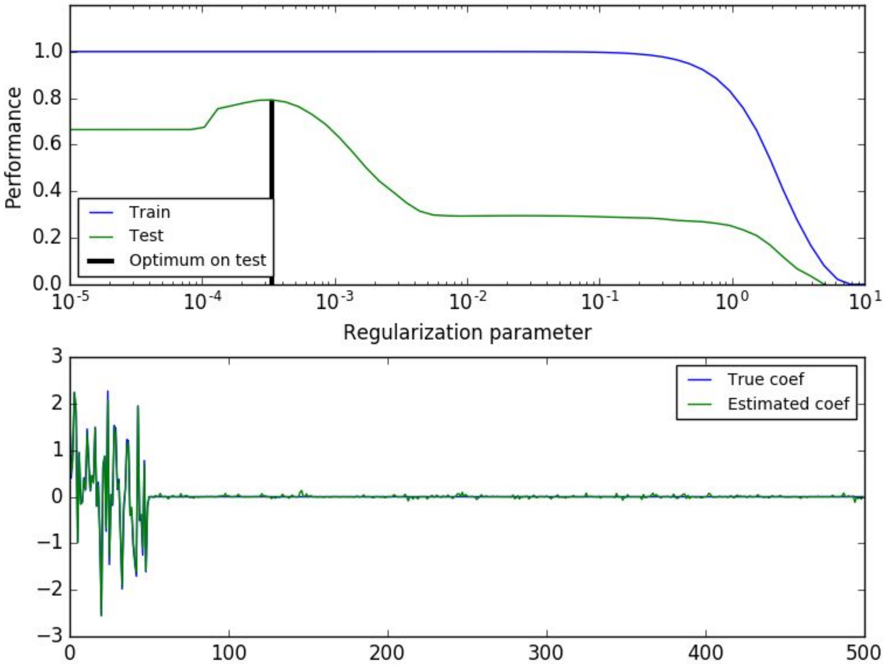


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions $|\beta_1| + |\beta_2| \leq t$ and $\beta_1^2 + \beta_2^2 \leq t^2$, respectively, while the red ellipses are the contours of the least squares error function.*

How do we use it? Concrete Example with L1

$$\mathcal{L}(y, f(x))_\lambda := \frac{1}{N}\sum_{i=1}^{N}(y_i - \beta \cdot x_i)^2 + \lambda\|\beta\|_{L^p}$$

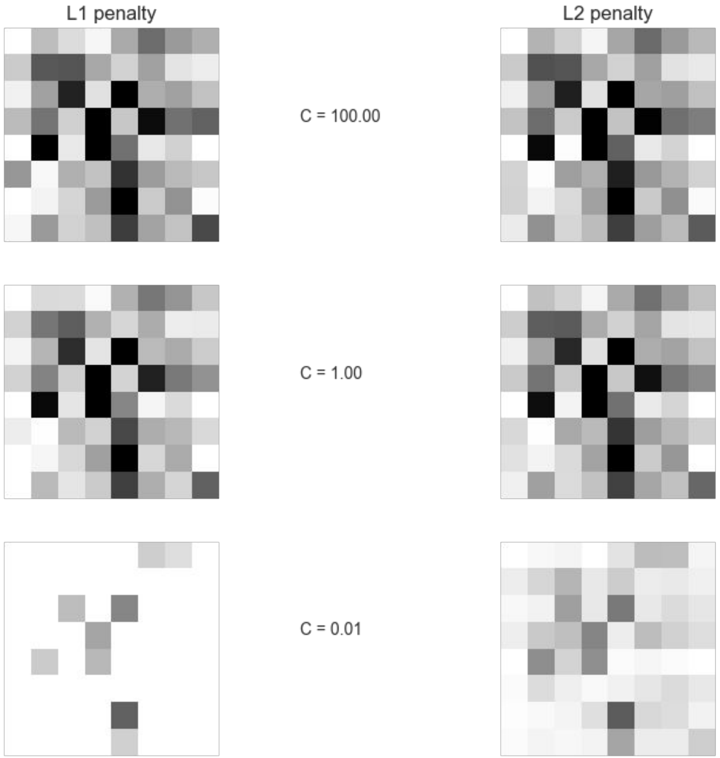
Let’s choose p=1, and see what happens as we range lambda.



MNIST Digit Recognition

How less is more sometimes - the problem of overfitting

Digit Recognition with Regularization

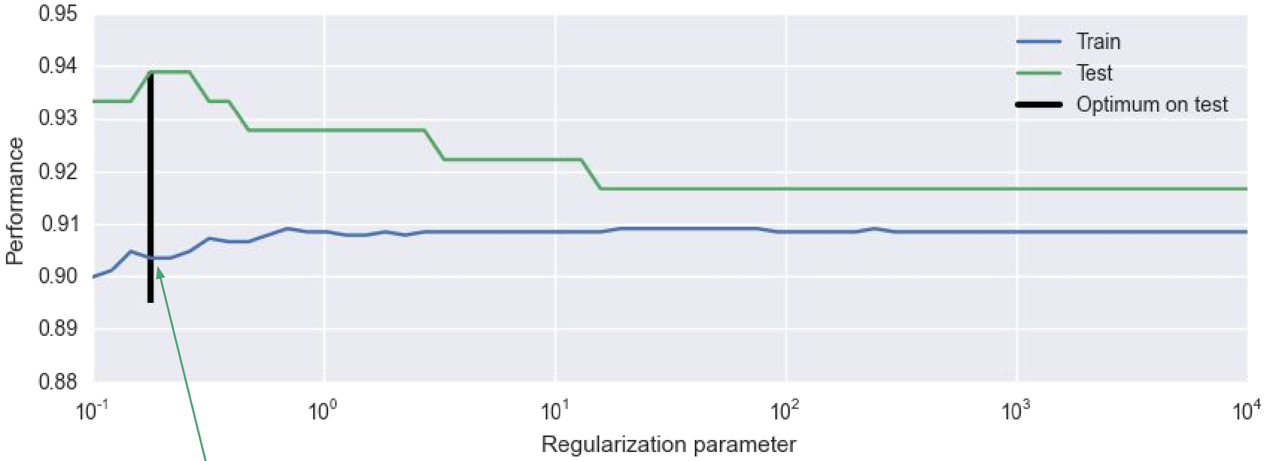


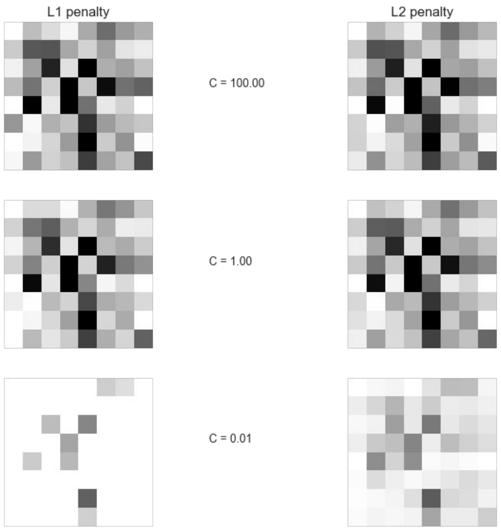
- Here we see an example of digit classification using L1 and L2 penalties.

- Notice how when our constraint is large, we have more freedom, and when the constraint is small we have less freedom.

- L1 is much more sparse than L2 when C is small - related to the visualizations we saw before with the lagrange multipliers.

MNIST Regularized Performance





The sweet spot!

Class Announcements

- Homework 1 will be returned next week. Grading and solutions available here: https://github.com/Columbia-Intro-Data-Science/APMAE4990-/tree/master/homework/hw1grading

- Homework 2 had some problems/ambiguities - apologies! Rewrote a version which we will start in class today.

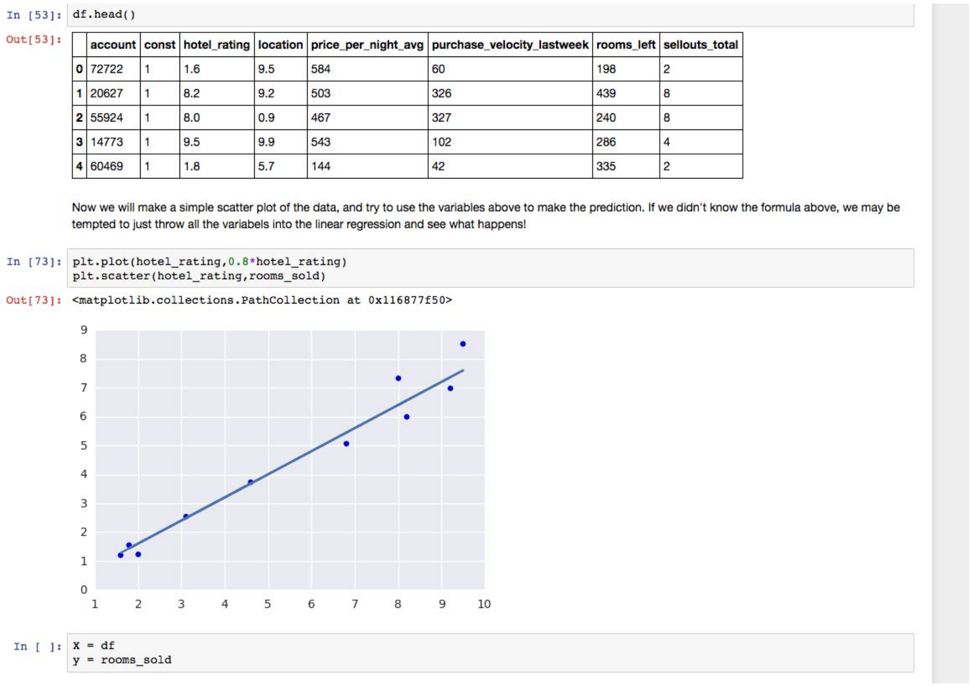
- Two weeks from today I'd like a written proposal for your project. Please chat with me if you'd like some ideas.

- Today we'll finish regularization and start classification problems.

Predicting Rooms Booked Revisited

How less is more sometimes - the problem of overfitting

How too many variables cause problems.



```
In [53]: df.head()
```

Out[53]:

	account	const	hotel_rating	location	price_per_night_avg	purchase_velocity_lastweek	rooms_left	sellouts_total
0	72722	1	1.6	9.5	584	60	198	2
1	20627	1	8.2	9.2	503	326	439	8
2	55924	1	8.0	0.9	467	327	240	8
3	14773	1	9.5	9.9	543	102	286	4
4	60469	1	1.8	5.7	144	42	335	2

Now we will make a simple scatter plot of the data, and try to use the variables above to make the prediction. If we didn't know the formula above, we may be tempted to just throw all the variabels into the linear regression and see what happens!

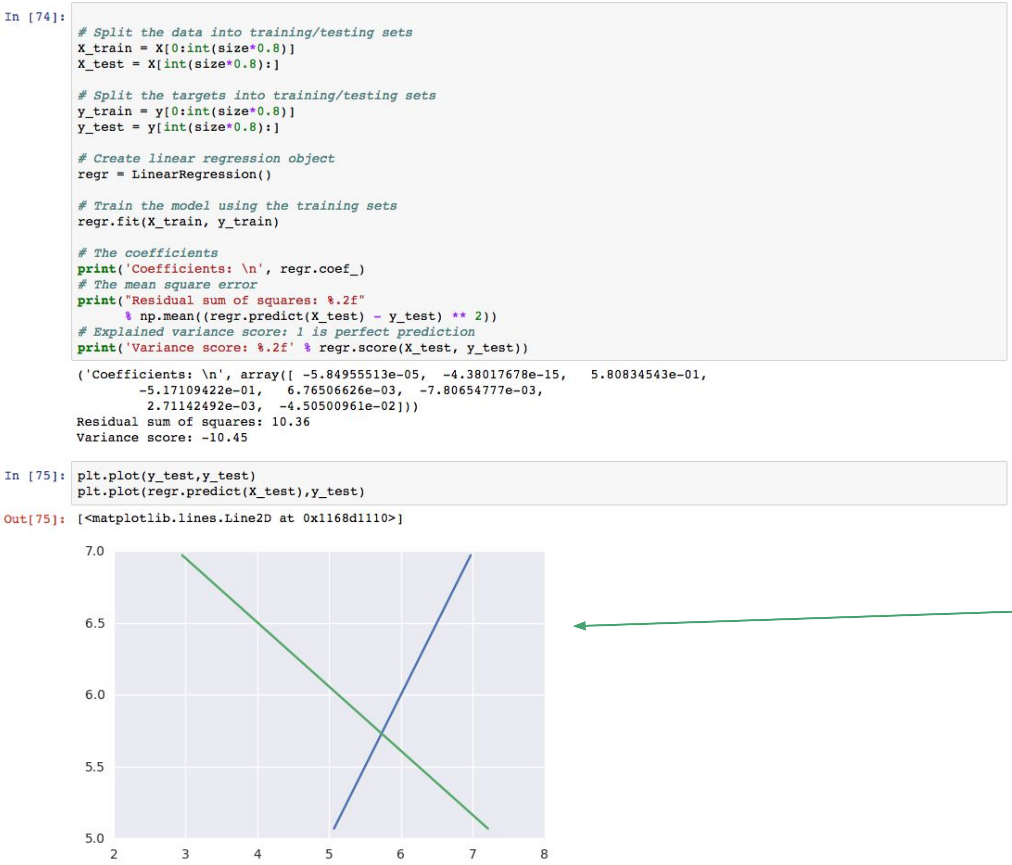
```
In [73]: plt.plot(hotel_rating,0.8*hotel_rating)
         plt.scatter(hotel_rating,rooms_sold)
Out[73]: <matplotlib.collections.PathCollection at 0x116877f50>
```

```
In [ ]: X = df
        y = rooms_sold
```

- A very common thing for people to do is to simply throw as many features as possible into a linear model.

- The assumption is that more is better - this is generally false!

Including all of the variables



```
In [74]:
# Split the data into training/testing sets
X_train = X[0:int(size*0.8)]
X_test = X[int(size*0.8):]

# Split the targets into training/testing sets
y_train = y[0:int(size*0.8)]
y_test = y[int(size*0.8):]

# Create linear regression object
regr = LinearRegression()

# Train the model using the training sets
regr.fit(X_train, y_train)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean square error
print("Residual sum of squares: %.2f"
      % np.mean((regr.predict(X_test) - y_test) ** 2))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % regr.score(X_test, y_test))
```

```
('Coefficients: \n', array([ -5.84955513e-05,  -4.38017678e-15,   5.80834543e-01,
        -5.17109422e-01,   6.76506626e-03,  -7.80654777e-03,
         2.71142492e-03,  -4.50500961e-02]))
Residual sum of squares: 10.36
Variance score: -10.45
```

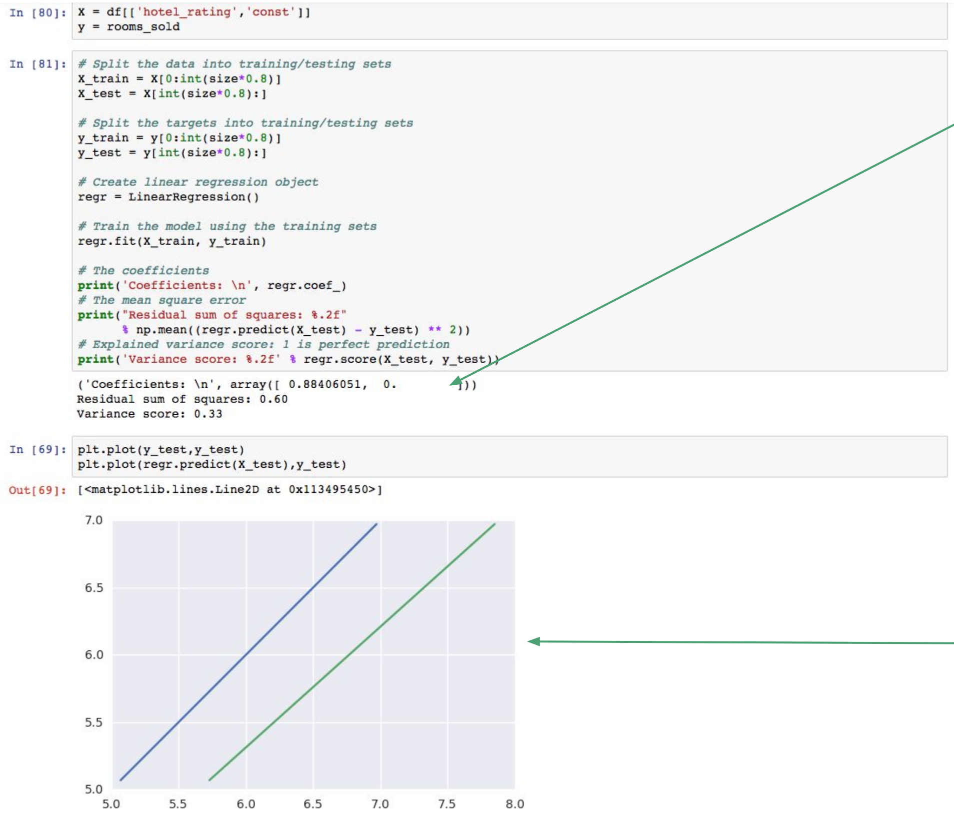
```
In [75]: plt.plot(y_test,y_test)
         plt.plot(regr.predict(X_test),y_test)
```

```
Out[75]: [<matplotlib.lines.Line2D at 0x1168d1110>]
```

- Here we split our data into testing/training.
- We create an instance of the Linear Regression model from scikit-learn.
- We fit it to our training data.
- We compute the R^2 and RMSE.
- We plot our predictions against the actual data.

Terrible! Sad!

Including one variable



```
In [80]: X = df[['hotel_rating','const']]
         y = rooms_sold
```

```
In [81]: # Split the data into training/testing sets
         X_train = X[0:int(size*0.8)]
         X_test = X[int(size*0.8):]

         # Split the targets into training/testing sets
         y_train = y[0:int(size*0.8)]
         y_test = y[int(size*0.8):]

         # Create linear regression object
         regr = LinearRegression()

         # Train the model using the training sets
         regr.fit(X_train, y_train)

         # The coefficients
         print('Coefficients: \n', regr.coef_)
         # The mean square error
         print("Residual sum of squares: %.2f"
               % np.mean((regr.predict(X_test) - y_test) ** 2))
         # Explained variance score: 1 is perfect prediction
         print('Variance score: %.2f' % regr.score(X_test, y_test))
```

```
('Coefficients: \n', array([ 0.88406051,  0.          ]))
Residual sum of squares: 0.60
Variance score: 0.33
```

```
In [69]: plt.plot(y_test,y_test)
         plt.plot(regr.predict(X_test),y_test)
```

```
Out[69]: [<matplotlib.lines.Line2D at 0x113495450>]
```

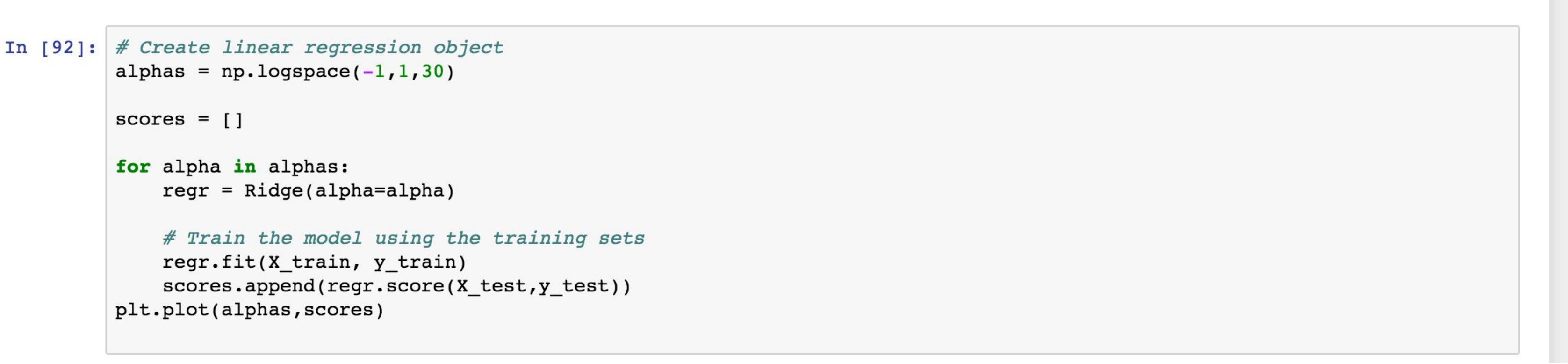
- By reducing to one variable, we’ve dramatically improved performance.

Much better!

How do we generalize this?

$$\mathcal{L}(y, f(x))_\lambda := \frac{1}{N}\sum_{i=1}^{N}(y_i - \beta \cdot x_i)^2 + \lambda||\beta||_{L^p}$$

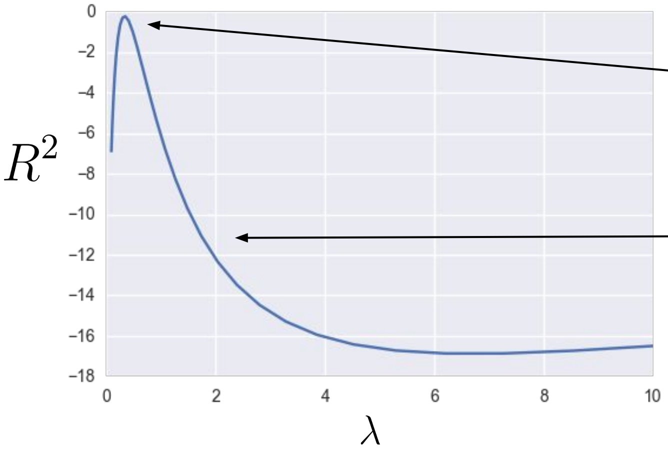
- Our goal is to choose the optimal restriction on the size of the coefficients to generalize to the testing set.



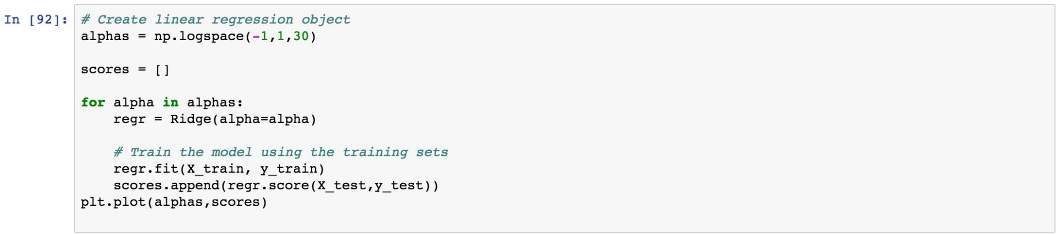
```
In [92]: # Create linear regression object
         alphas = np.logspace(-1,1,30)

         scores = []

         for alpha in alphas:
             regr = Ridge(alpha=alpha)

             # Train the model using the training sets
             regr.fit(X_train, y_train)
             scores.append(regr.score(X_test,y_test))
         plt.plot(alphas,scores)
```

Ridge Regression - L^2

$$\frac{1}{N}\sum_{i=1}^{N}|y_i-\beta\cdot x_i|^2+\lambda\sum_{i=1}^{N}|\beta_i|^2$$



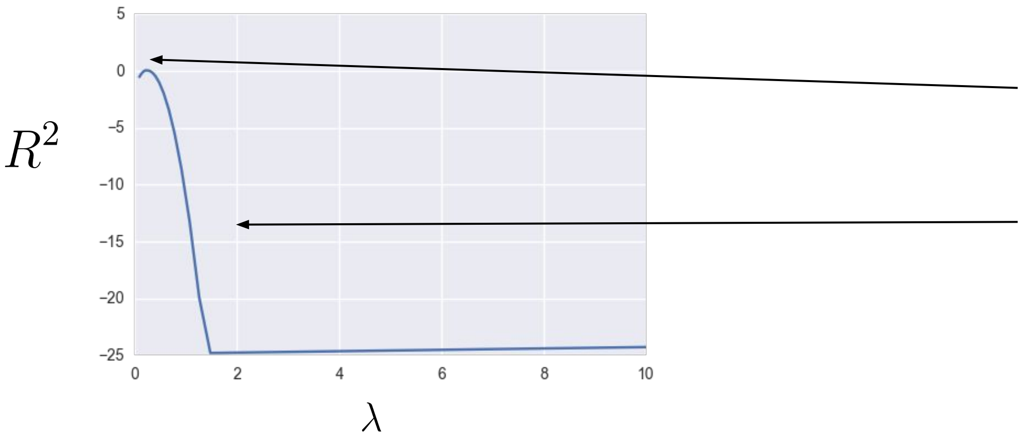
- Our goal is always to find the size of the constraint which maximizes the performance on held out data.
- Notice the smooth decay of performance as the parameter grows.



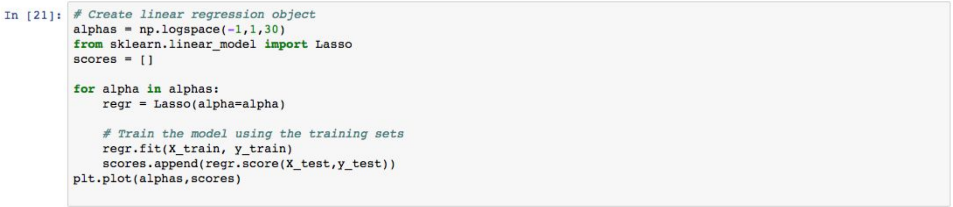
```
In [92]: # Create linear regression object
         alphas = np.logspace(-1,1,30)

         scores = []

         for alpha in alphas:
             regr = Ridge(alpha=alpha)

             # Train the model using the training sets
             regr.fit(X_train, y_train)
             scores.append(regr.score(X_test,y_test))
         plt.plot(alphas,scores)
```

Lasso Regression - L^1

$$\frac{1}{N}\sum_{i=1}^{N}|y_i - \beta \cdot x_i|^2 + \lambda \sum_{i=1}^{N}|\beta_i|^2$$



- Our goal as before is to find the size of the constraint which maximizes the performance on held out data.
- Notice how the L^1 norm decays much faster. Why?

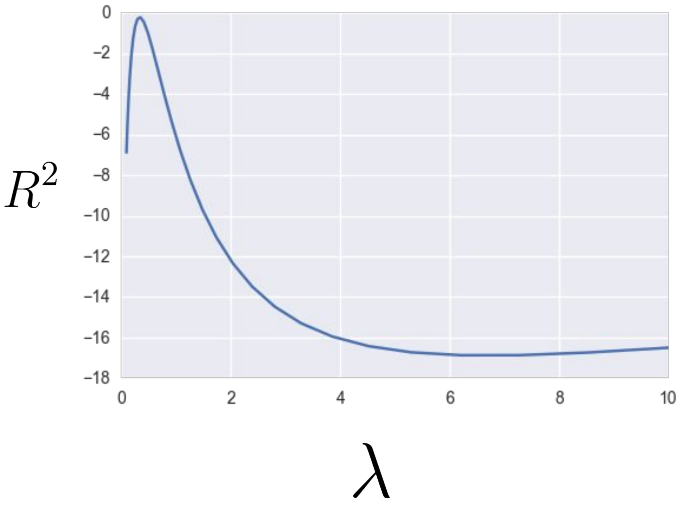


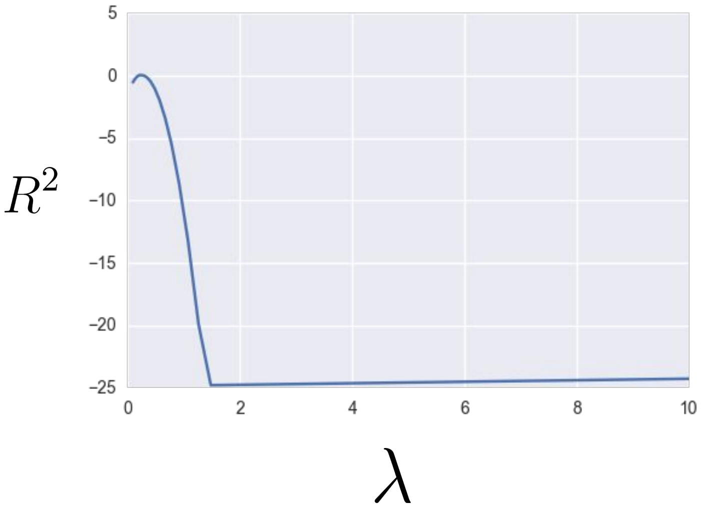
```
In [21]: # Create linear regression object
         alphas = np.logspace(-1,1,30)
         from sklearn.linear_model import Lasso
         scores = []

         for alpha in alphas:
             regr = Lasso(alpha=alpha)

             # Train the model using the training sets
             regr.fit(X_train, y_train)
             scores.append(regr.score(X_test,y_test))
         plt.plot(alphas,scores)
```

Comparison of the two

$$\frac{1}{N}\sum_{i=1}^{N}|y_i - \beta \cdot x_i|^2 + \lambda \sum_{i=1}^{N}|\beta_i|^2$$

$$\frac{1}{N}\sum_{i=1}^{N}|y_i - \beta \cdot x_i|^2 + \lambda \sum_{i=1}^{N}|\beta_i|$$





Another Interpretation

$$\frac{1}{N}\sum_{i=1}^{N}|y_i - \beta \cdot x_i|^2 + \lambda \sum_{i=1}^{N}|\beta_i|^2$$

$$\frac{1}{N}\sum_{i=1}^{N}|y_i - \beta \cdot x_i|^2 + \lambda \sum_{i=1}^{N}|\beta_i|$$

$$\begin{aligned} y_1 &= \alpha x_{11} + \beta x_{12} + \gamma x_{13} \\ y_2 &= \alpha x_{21} + \beta x_{22} + \gamma x_{23} \\ y_3 &= \alpha x_{31} + \beta x_{32} + \gamma x_{33} \end{aligned}$$

$$\mathbf{y} = \mathbf{X}\tilde{\beta}$$

$$\tilde{\beta} = (\alpha, \beta, \gamma)$$

- Recall that when **we have more equations than data points**, we can **have too many solutions.**
- Hence we pick up **too much variance!**

Let’s choose the one that minimizes some norm:

$$\min_{\tilde{\beta}} ||\tilde{\beta}||_{L^p}$$

Comparison of the two

$$\begin{aligned} y_1 &= \alpha x_{11} + \beta x_{12} + \gamma x_{13} \\ y_2 &= \alpha x_{21} + \beta x_{22} + \gamma x_{23} \\ y_3 &= \alpha x_{31} + \beta x_{32} + \gamma x_{33} \end{aligned}$$

$$\tilde{\beta} = (\alpha, \beta, \gamma)$$

$$\mathbf{y} = \mathbf{X}\tilde{\beta}$$

$$\min_{\tilde{\beta}} \|\tilde{\beta}\|_{L^p}$$

- We see that the answer is different - the L2 norm tends to penalize large coefficients more for instance.

- L2 wants to spread the error out (make each coefficient as small as possible).

- L1 is ok with sparse (zero valued) coefficients.

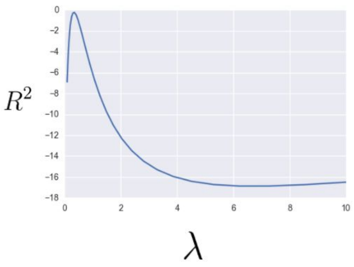
$$\tilde{\beta}_1 = (0.5, 0.5, 0)$$

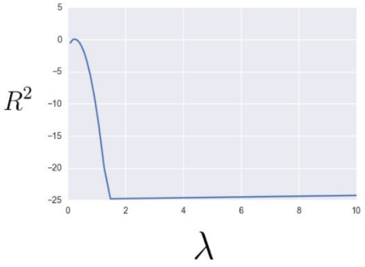
$$\tilde{\beta}_0 = (-1, 0, 0)$$

$$\|\tilde{\beta}_0\|_{L^1} = \|\tilde{\beta}_1\|_{L^1} = 1$$

$$\|\tilde{\beta}_0\|_{L^2} = \sqrt{0.25 + 0.25} = \sqrt{0.5}$$

Comparison of the two





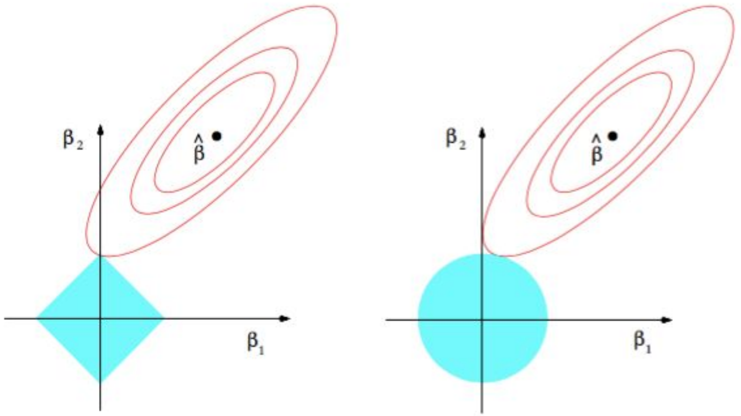


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \le t$ *and* $\beta_1^2 + \beta_2^2 \le t^2$*, respectively, while the red ellipses are the contours of the least squares error function.*

- The L1 constrained norm has level sets which are boxes, meaning it sets coefficients to zero usually - hence the immediate drop off.

- For L2 the constraint is distributed more evenly amongst the remaining coefficients, causing smoother decay.

Why not other Lp spaces?

- L2 enjoys a lot of nice properties since it is a **Hilbert space** (the norm is defined by an inner product). This makes many operations such as taking derivatives or applying operators easier.

- L1 is badly behaved (not uniformly convex) but is great for feature selection since it sends many coefficients to zero for the reasons mentioned.

- L2 is consistent with a Gaussian prior on the initial coefficients (more on this later).

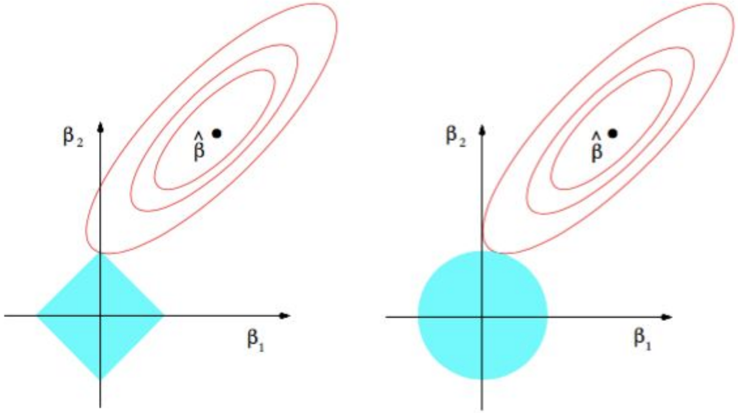
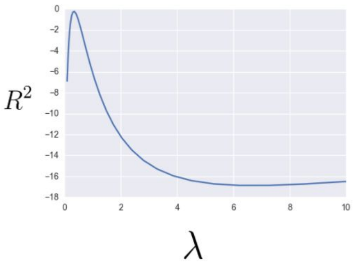
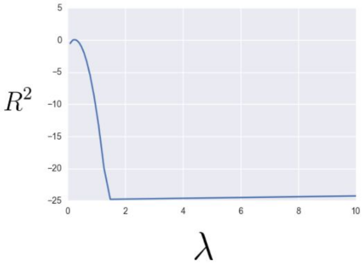


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \le t$ *and* $\beta_1^2 + \beta_2^2 \le t^2$*, respectively, while the red ellipses are the contours of the least squares error function.*

So which is better?





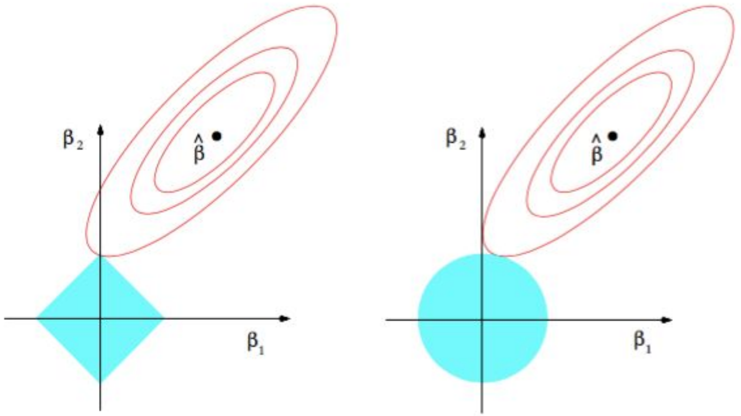


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \le t$ *and* $\beta_1^2 + \beta_2^2 \le t^2$*, respectively, while the red ellipses are the contours of the least squares error function.*

- L1 is better for eliminating collinear features in general.
- Many believe L2 to perform better for performance, since it distributes the errors more evenly amongst the remaining features.
- L1 is better for feature selection in high dimensional data.
- In the end, try both ! Whichever performs better is what you should use.

Suggested Quora Page

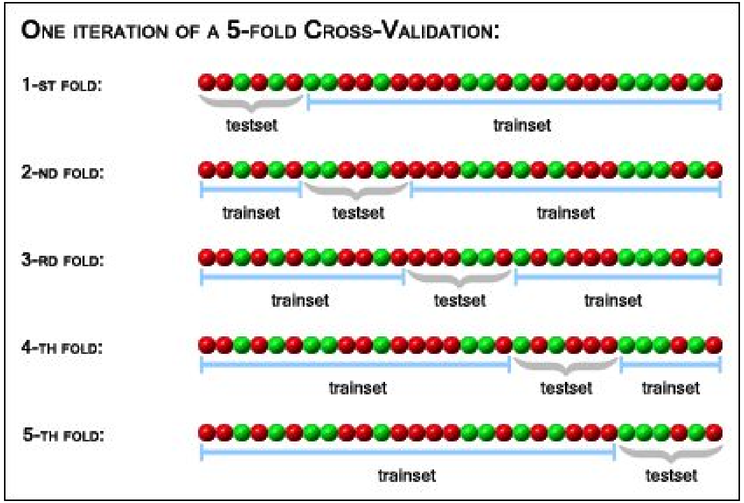
https://www.quora.com/What-is-the-difference-between-L1-and-L2-regularization

This page has a large number of fantastic answers by world leaders in machine learning. Try reading them all and see if one makes more sense to you than the others.

Cross Validation and Regularization

Both should be combined together

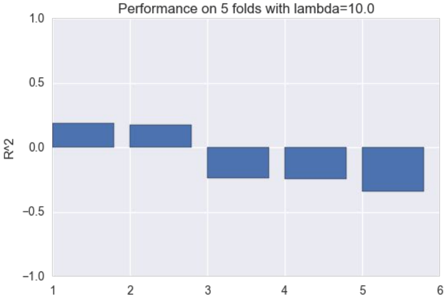
Cross Validation Revisited

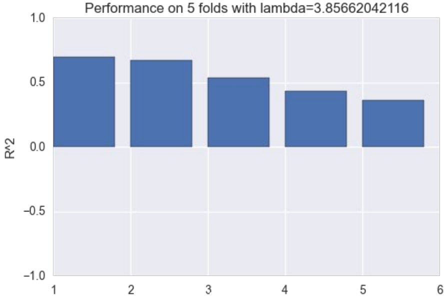


- Generally instead of just taking say **80% of your data for training**, **20% for testing** (for example), we randomly split the data into several **‘folds’.**

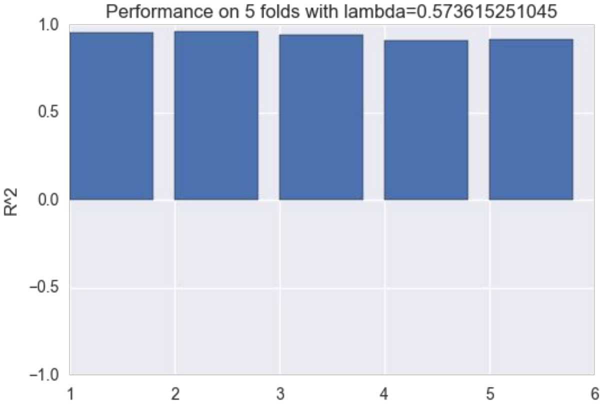
- In **k-fold cross**-**validation**, the original sample is randomly partitioned into **k equal sized subsamples**. Of the k subsamples, **a single subsample is retained as the validation data** for testing the model, and t**he remaining k** − **1 subsamples are used as training data.**

Cross Validated Regularization

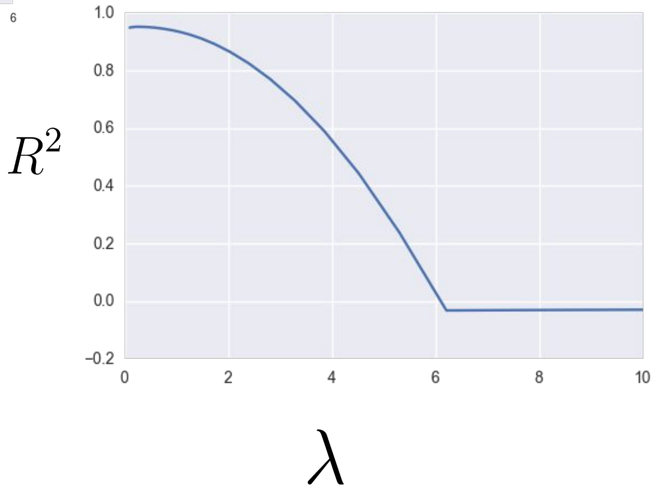

Performance on 5 folds with lambda=10.0


Performance on 5 folds with lambda=3.85662042116

$$\frac{1}{N}\sum_{i=1}^{N}|y_i-\beta\cdot x_i|^2+\lambda\sum_{i=1}^{N}|\beta_i|$$


Performance on 5 folds with lambda=0.573615251045

[redacted]



Summary of Model Building

- We have covered **all of the essential ingredients for constructing a regression model from data** (not including data cleansing/manipulation).

- **Step 1:** Plot your data and understand the r**elationship between features and the dependent variable** (correlation, eda).

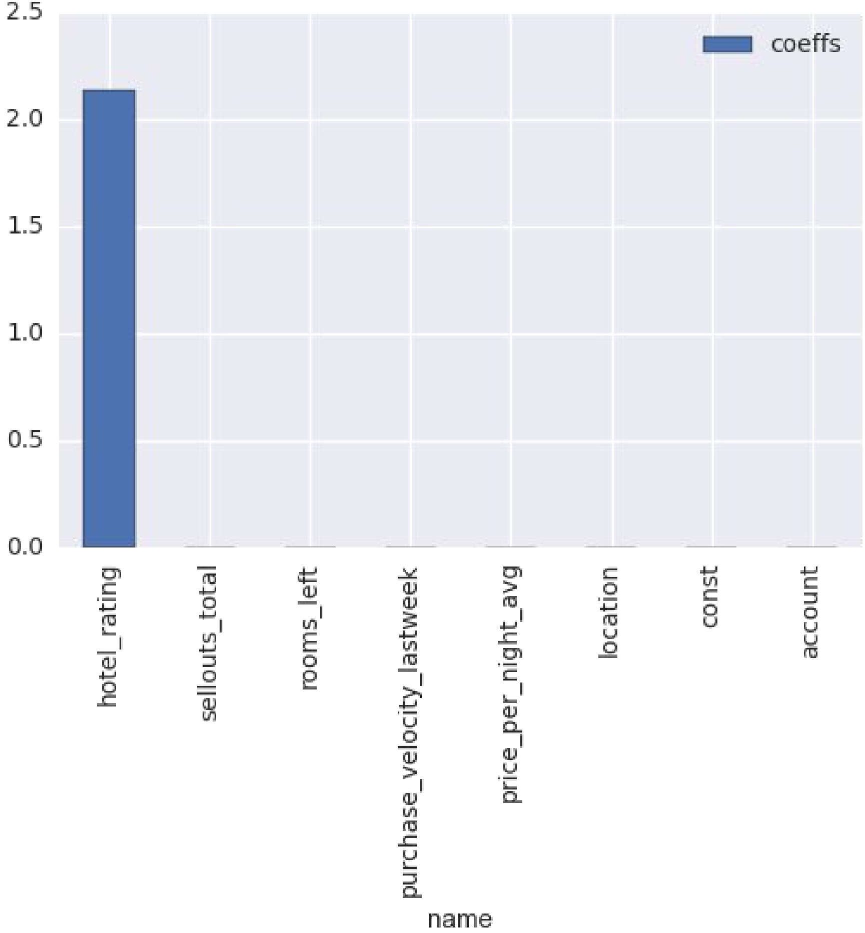
- **Step 2:** Train your model and optimize it by using regularization and cross validation. Choose the coefficient and norm which results in the **best performance on held out data (over many folds, take the averaged coefficients).**

- **Step 3: Normalize your data and plot feature importances** with ranges specified by cross validation.

Our final model

$$\mathbf{y} = \beta \cdot \mathbf{x}$$

$$\beta = [2.1, 0, 0, 0, 0, 0, 0, 0]$$


coeffs
hotel_rating
sellouts_total
rooms_left
purchase_velocity_lastweek
price_per_night_avg
location
const
account
name