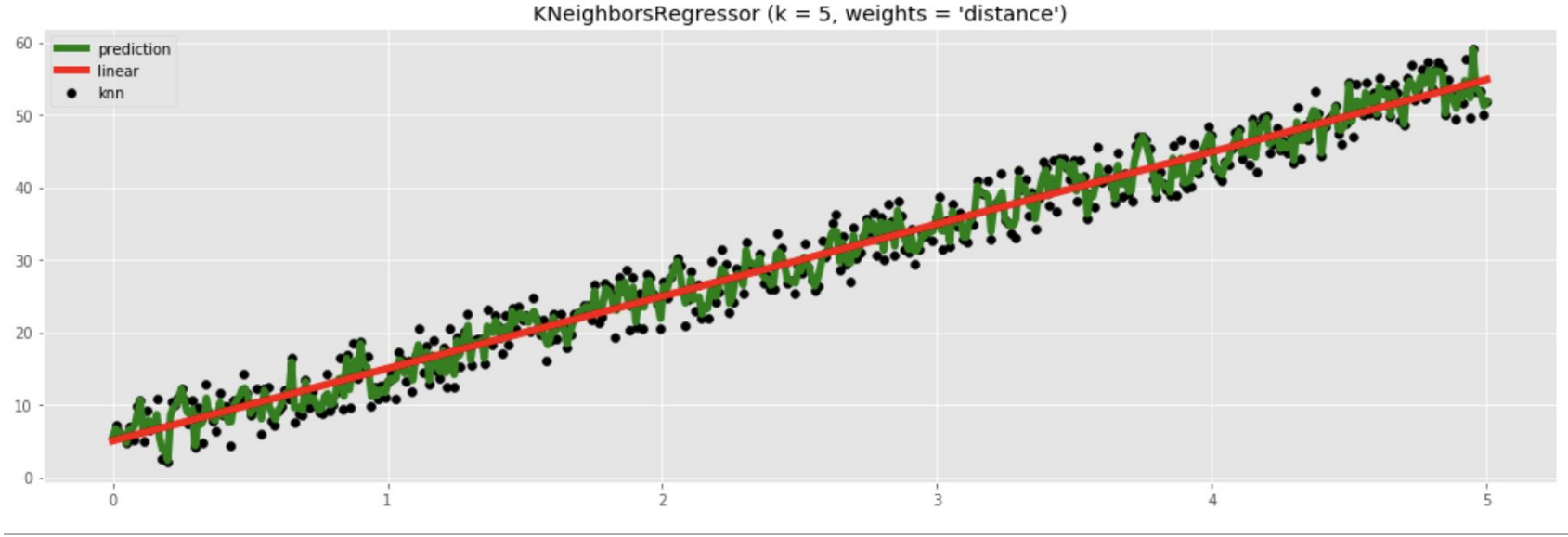
Model Evaluation, Complexity and Regularization

Lecture 3

Recall Lecture 2 Example 1

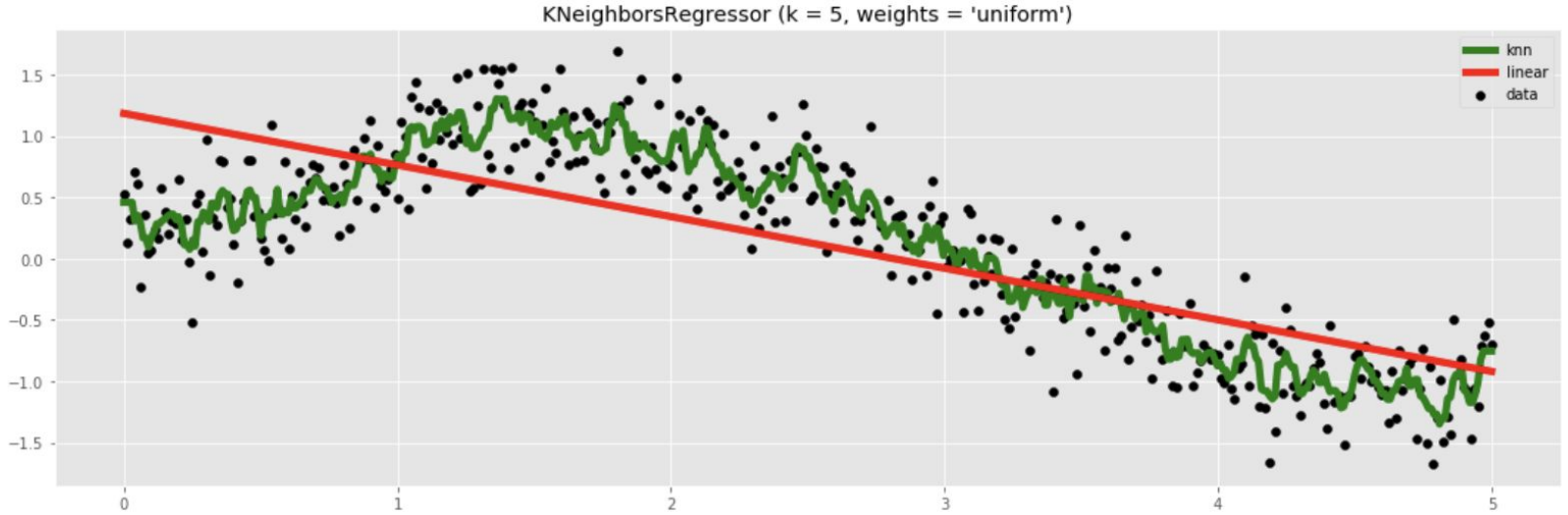


Nearest neighbour has very low bias but high variance.

Linear model has high bias, low variance and seem to model data well.

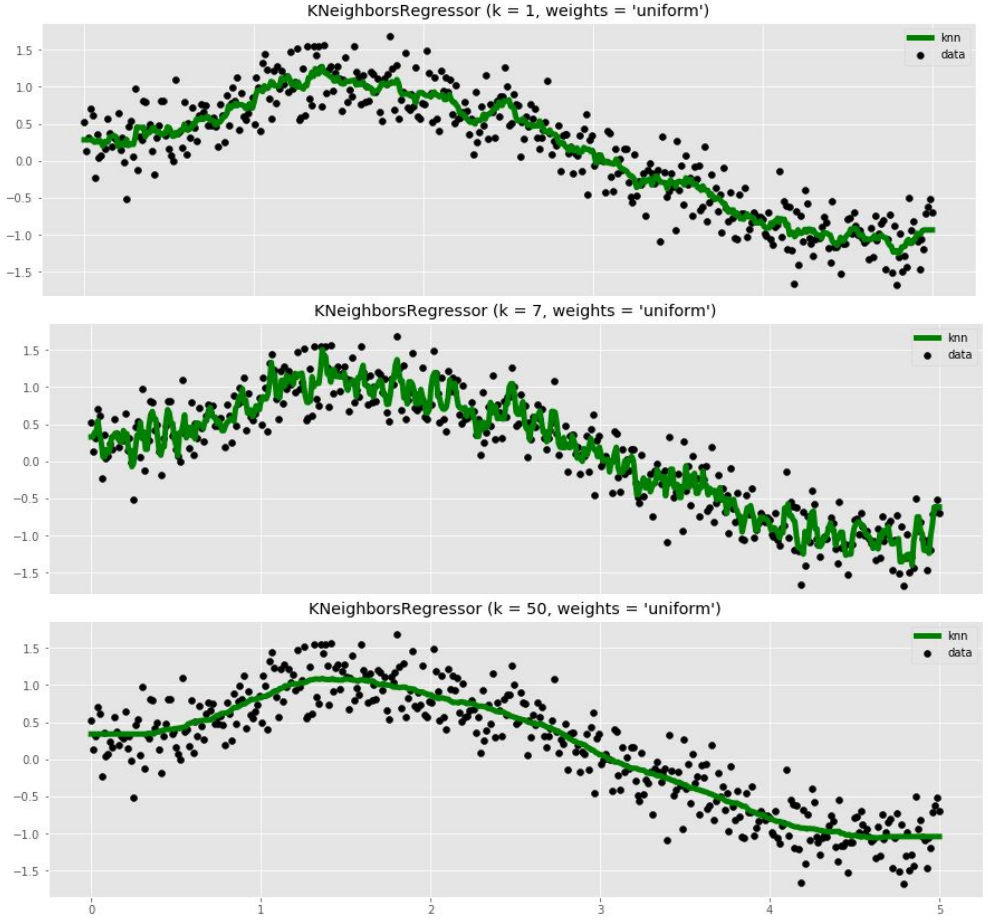
Which model is better?

Recall Lecture 2 Example 2



Which model is better here?

Which model looks better?



The last model looks best because it appears to not pick up all of the excess noise (irreducible error), while still capturing the trend of the conditional mean.

But how do we quantify this? We definitely have nothing to worry about with linear models then right? (Wrong!)

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.

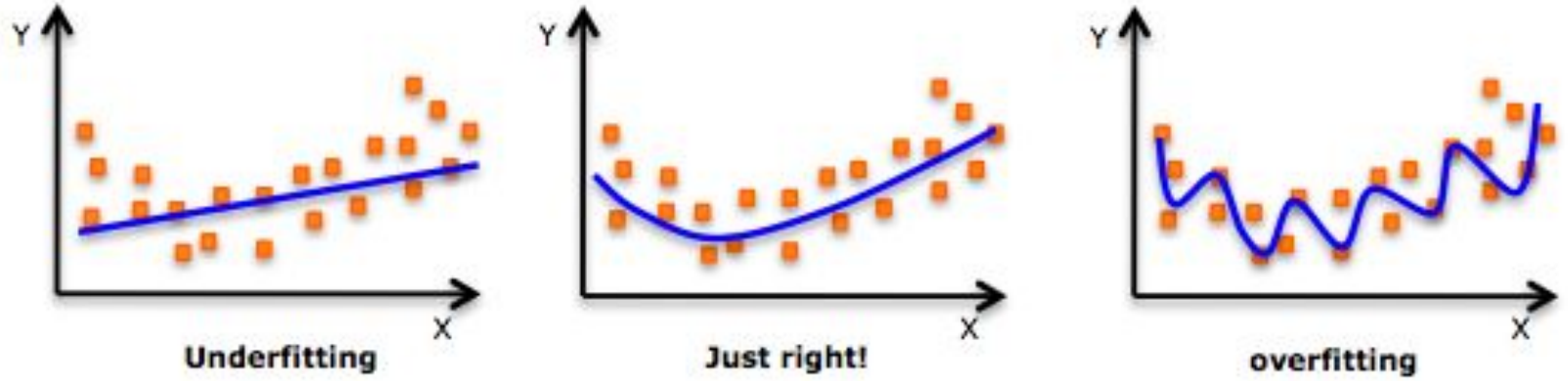
This leaves 4 questions:

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.

Complexity and Regularization

How less is more sometimes - the problem of overfitting

Model Complexity and Regularization

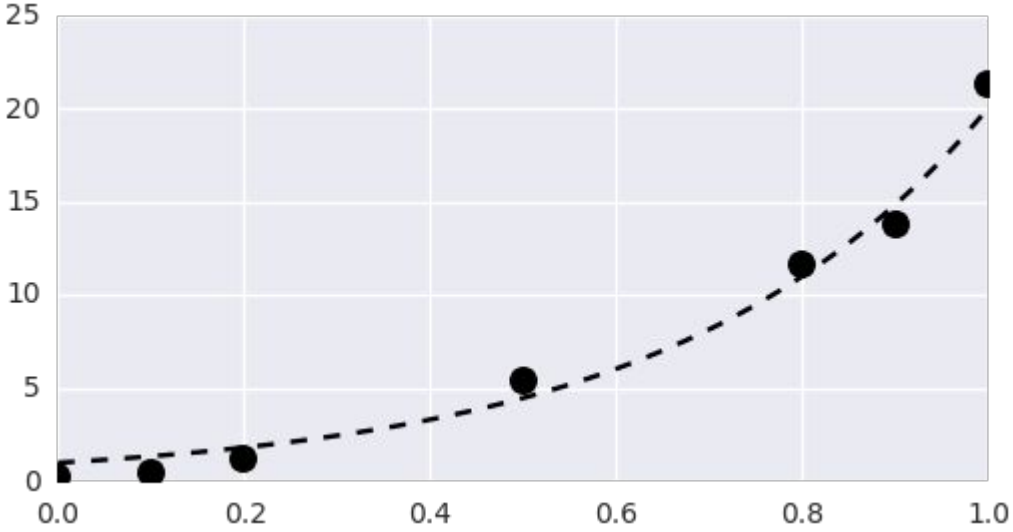


Overfitting occurs when the trained model picks up too much ‘variance’ , which isn’t reflective of its true behavior, and therefore will perform badly on new data.

Example - Polynomial Overfitting

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

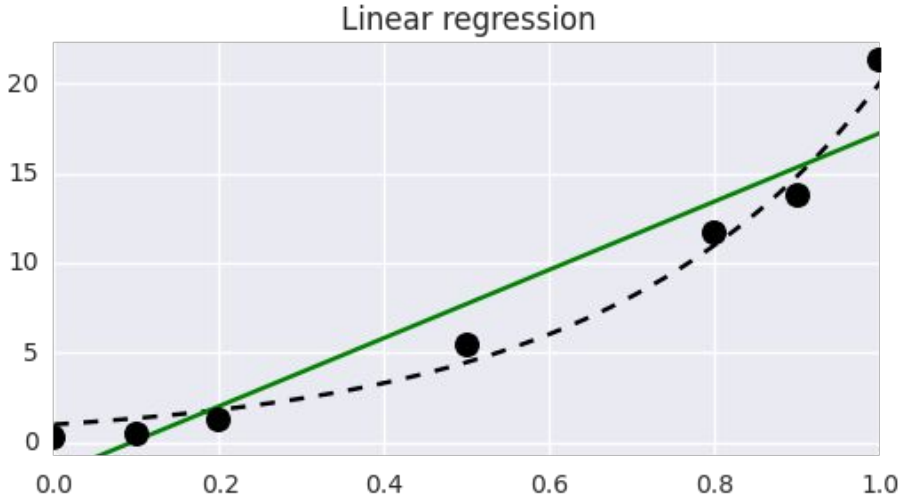
- What is the right degree of polynomial to fit this curve?
- We will see that despite the exponential technically having an infinite degree, we still can risk picking up too much variance if we don't limit the order.



Example - Polynomial Overfitting

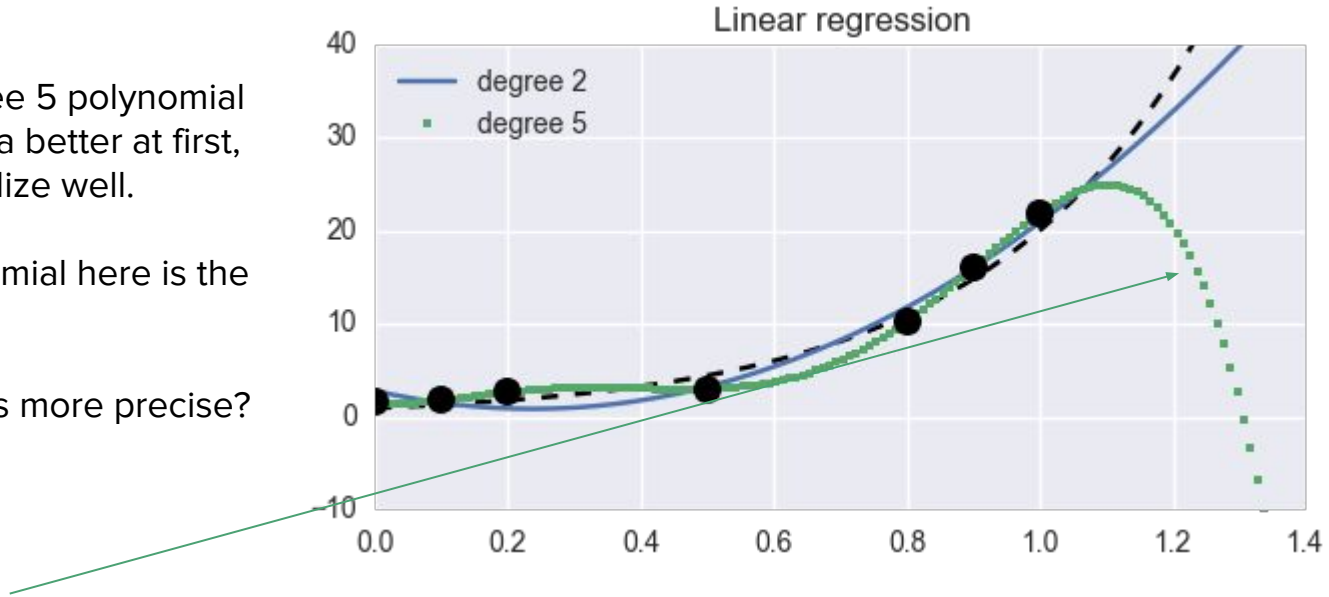
$$y = \exp(3x) + \text{noise}$$

- 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?

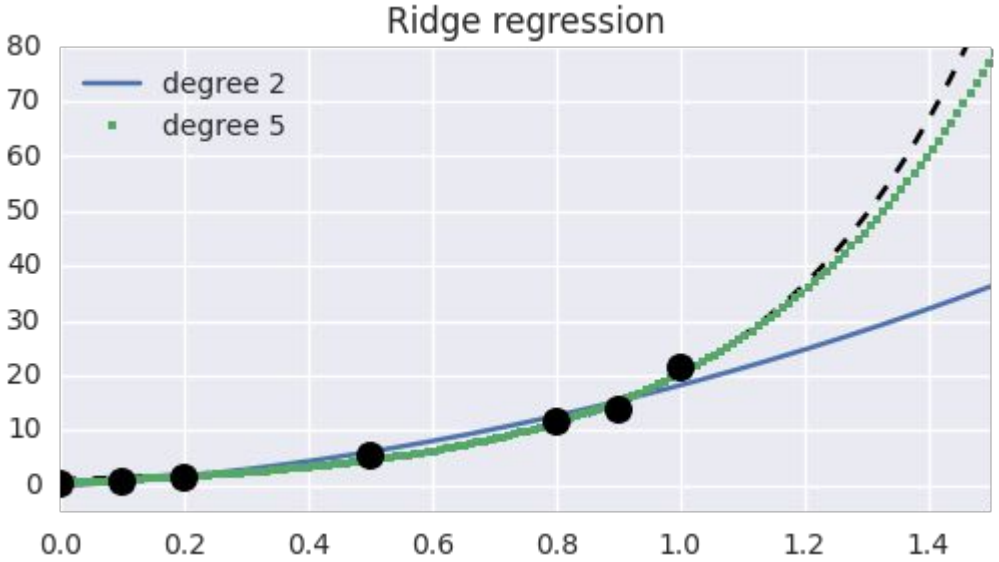


The model picks up variance, and blows up the coefficients, causing this behavior

Regularization- 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.

Recall from Lecture 1 - What is f?

In statistical learning, we only have access to samples from our distribution (x_i, y_i)

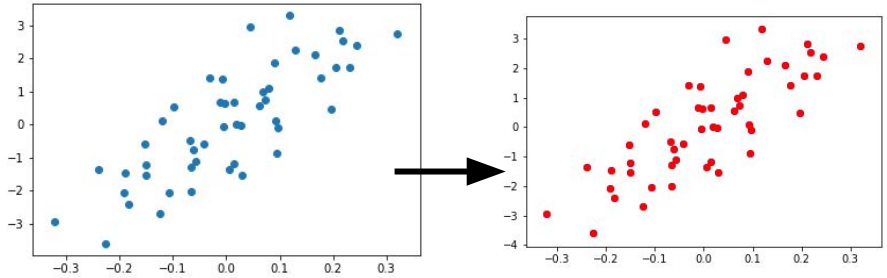
$$\hat{Y} = \hat{f}(\mathbf{X})$$

$$\mathbb{E}(y - \hat{y})^2 \sim \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$$

$$= \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{f}(\mathbf{x}_i))^2$$

Why not just choose $\hat{f}(x_i) = y_i$?

This was clearly a very strict and non-generalizable assumption.

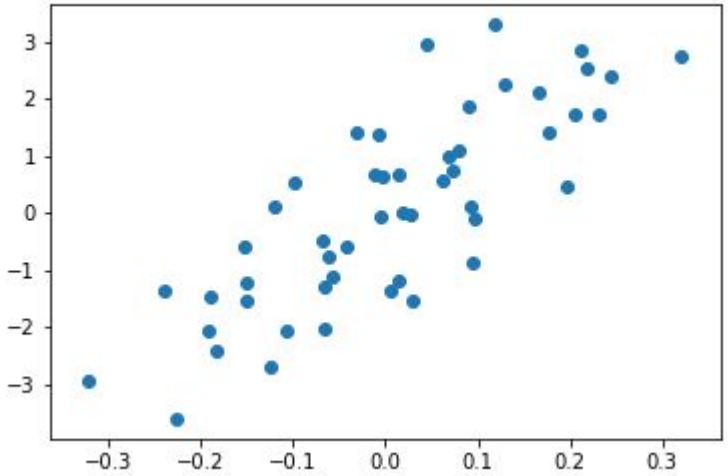


A linear example

$$\mathbf{y} = 10\mathbf{x_0} + \epsilon.$$

$$\mathbf{X} = \left(\mathbf{x_0}, \mathbf{x_1}, \quad \cdots, \quad \mathbf{x_{50}}\right)$$

- Let's assume our response variable y depends only on one of our 50 features. We plot the scatter plot to the right.
- A common situation in machine learning is that you have many other features, and you don't know which ones are important.



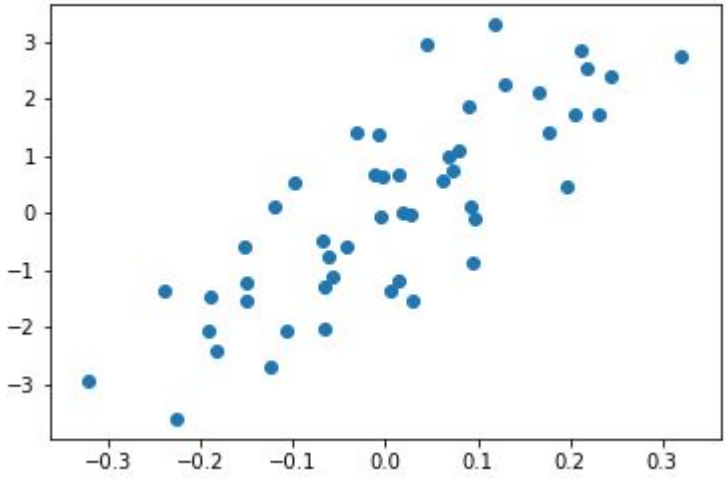
A more realistic example

$$\mathbf{y} = 10\mathbf{x_0} + \epsilon.$$

$$\mathbf{X} = \left(\mathbf{x_0}, \mathbf{x_1}, \quad \cdots, \quad \mathbf{x_{50}}\right)$$

Our goal is to solve:

$$\min_{\beta} \|Y - \mathbf{X}\beta\|^2$$

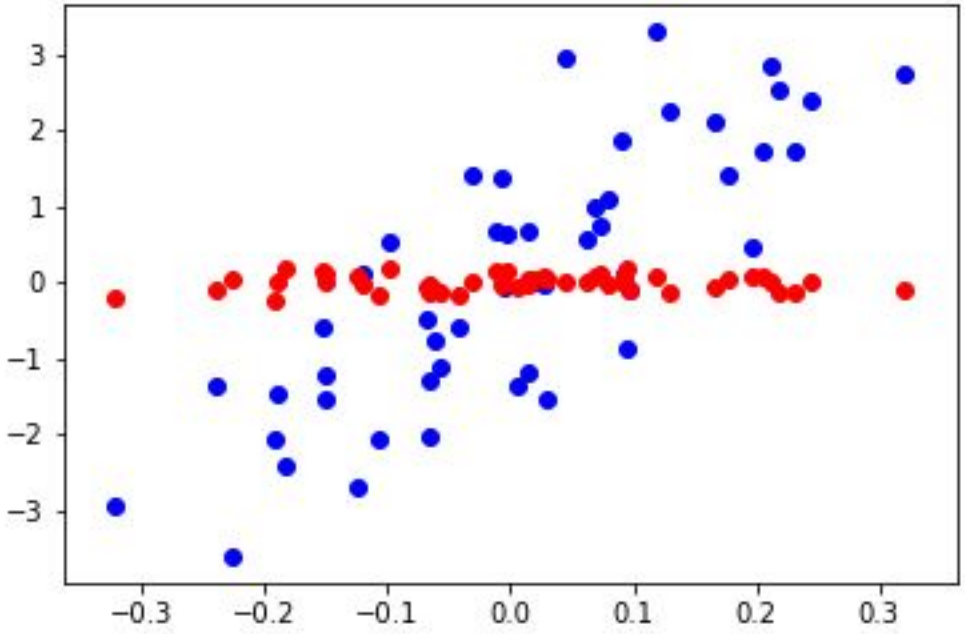


$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} - \begin{pmatrix} x_{11} & \cdots & x_{50} \\ x_{21} & \ddots & x_{50} \\ x_{31} & \cdots & x_{50} \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{50} \end{pmatrix}$$

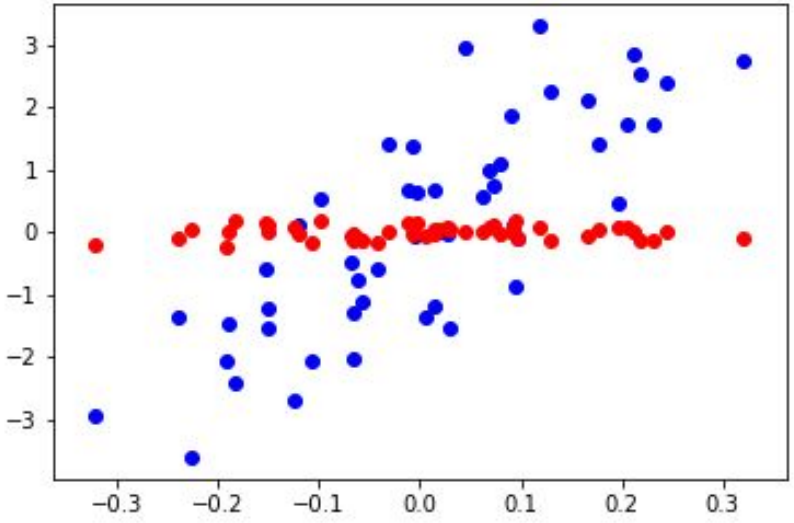
Python Code

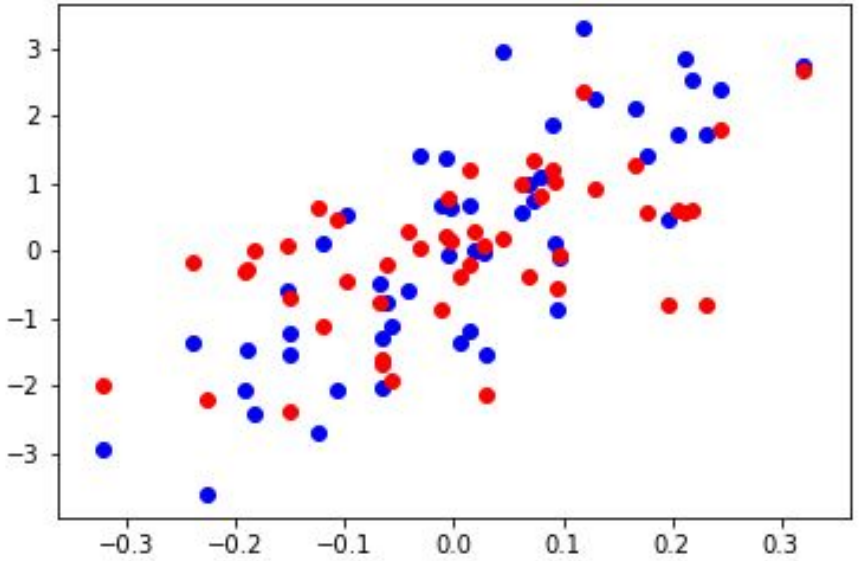
```
from sklearn import datasets, linear_model

for d in range(0,80,20):
    regr = linear_model.LinearRegression(fit_intercept=False)
    X=df_orth.loc[:,0:d]
    # Train the model using the training sets
    regr.fit(X,y)

    # Make predictions using the testing set
    y_pred = regr.predict(X)
    plt.scatter(x0,y,color='b')
    plt.scatter(x0,y_pred,color='r')
    plt.show()
```

Zero features

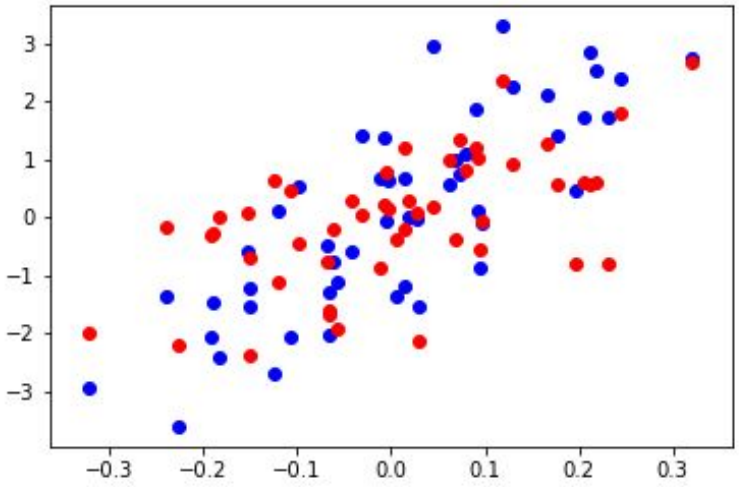


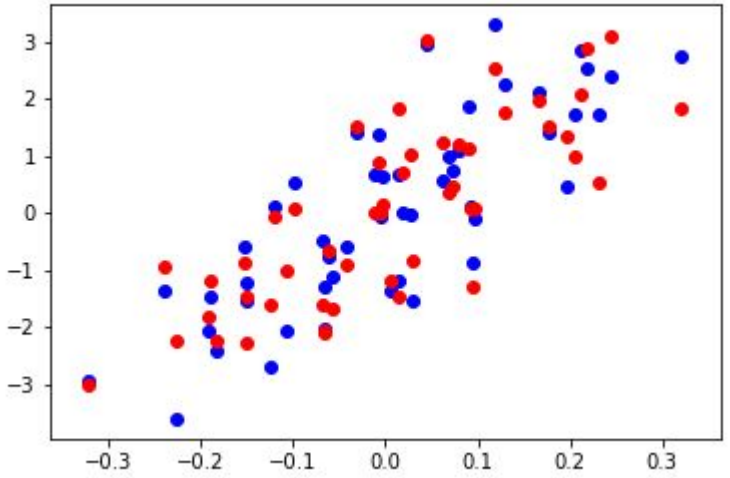
Two Features



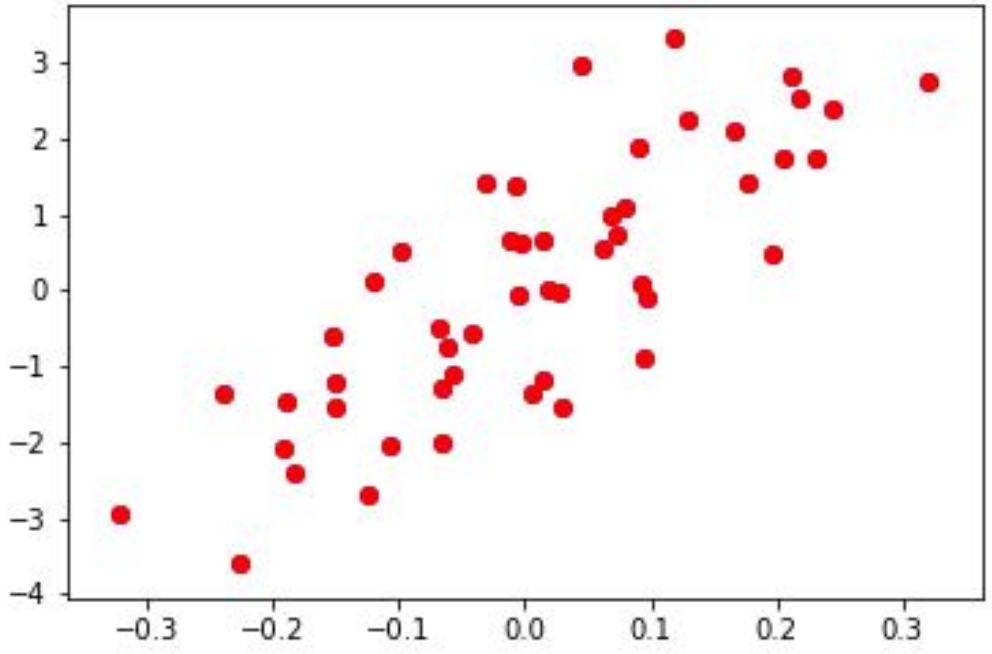


Three, Four Features





50 features



What happened here?

$$\mathbf{y} = 10\mathbf{x_0} + \epsilon.$$

$$\min_{\beta} \|Y - \mathbf{X}\beta\|^2$$

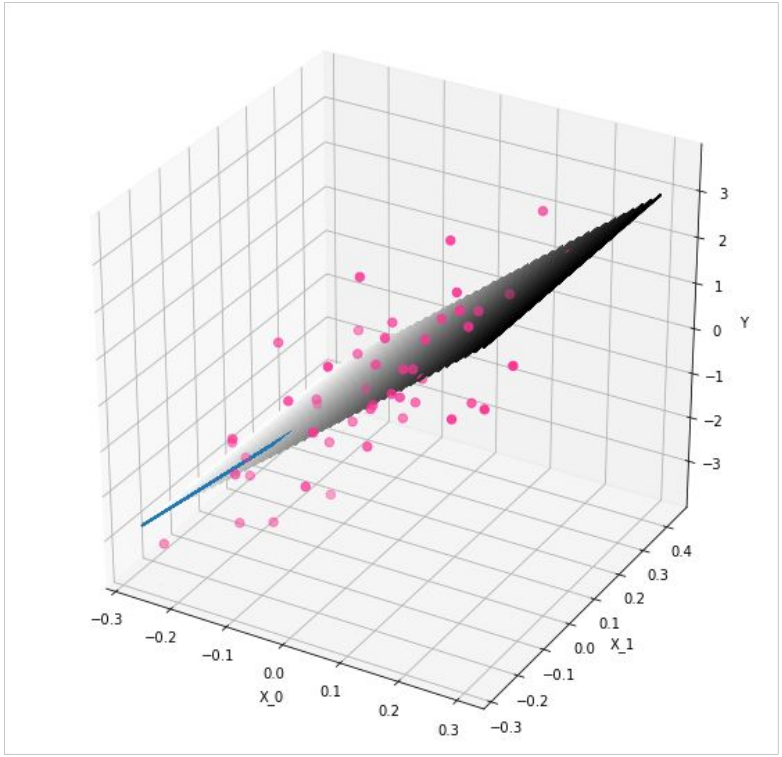
When the columns of X are independent, we have a unique non-trivial solution. So we can solve this exactly!

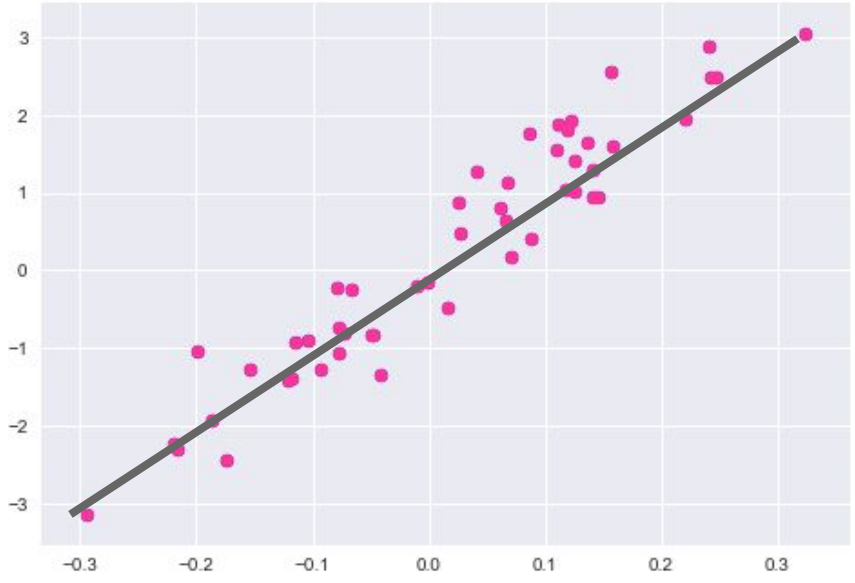
$$\beta = \mathbf{X}^{-1}y$$

$$Y = \mathbf{X}\beta$$

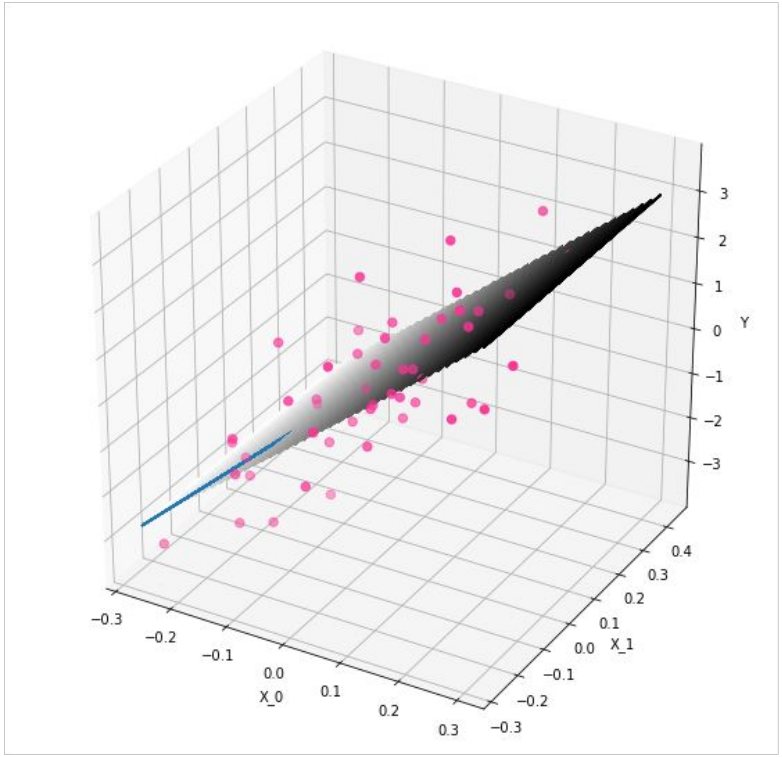
$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} x_{11} & \cdots & x_{50} \\ x_{21} & \ddots & x_{50} \\ x_{31} & \cdots & x_{50} \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{50} \end{pmatrix}$$

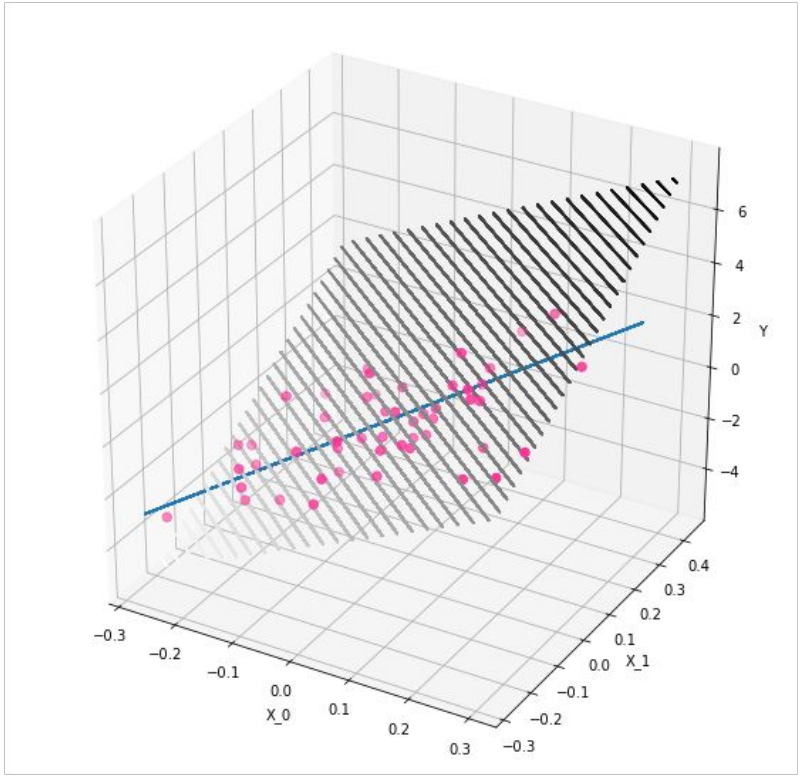
A 3d interpretation of overfitting





A 3d interpretation of overfitting





Model Complexity and Regularization

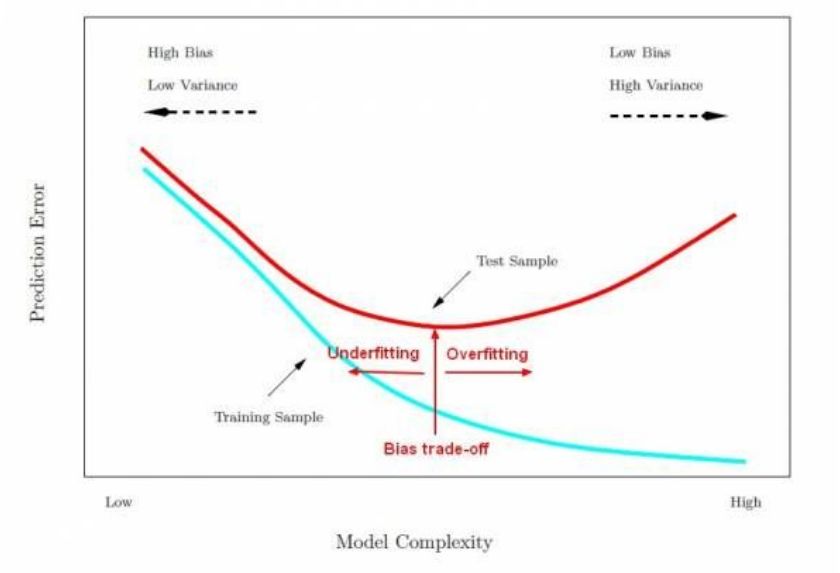
In fact, **adding more variables will *always* improve performance on training data**.

Adding in complexity in one dimension is the same as adding in more independent dimensions.

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 example, 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?

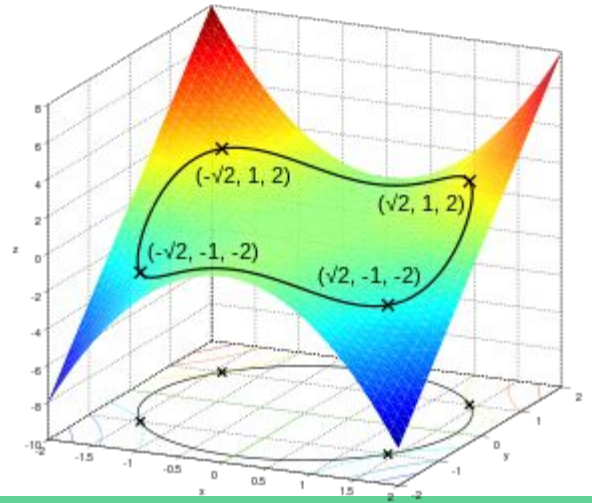
- The above is equivalent (from principles of the calculus of variations) to constraining the norm of the coefficients.
- We want to simply choose the lambda which performs best on randomly chosen held out data.
- Large lambda means large penalization on coefficients.

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 $\min_{x \in X} f(x) + \lambda g(x)$ is equivalent to

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

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

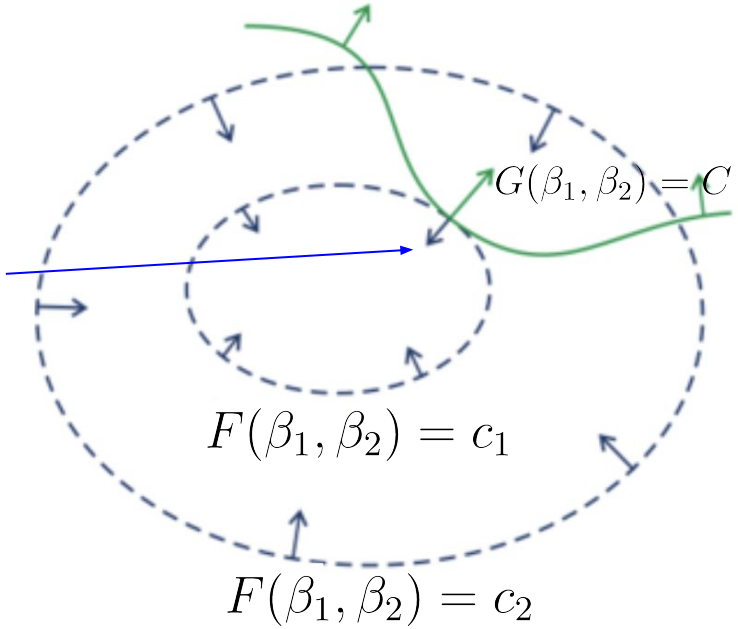


Constrained Optimization Review

$$\min_{\beta=(\beta_1,\beta_2)} F(\beta_1, \beta_2) + \lambda G(\beta_1, \beta_2)$$

$$\Rightarrow \nabla F(\beta_1, \beta_2) = -\lambda \nabla G(\beta_1, \beta_1)$$

Gradients of level sets need to be parallel at an optimal point.



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.

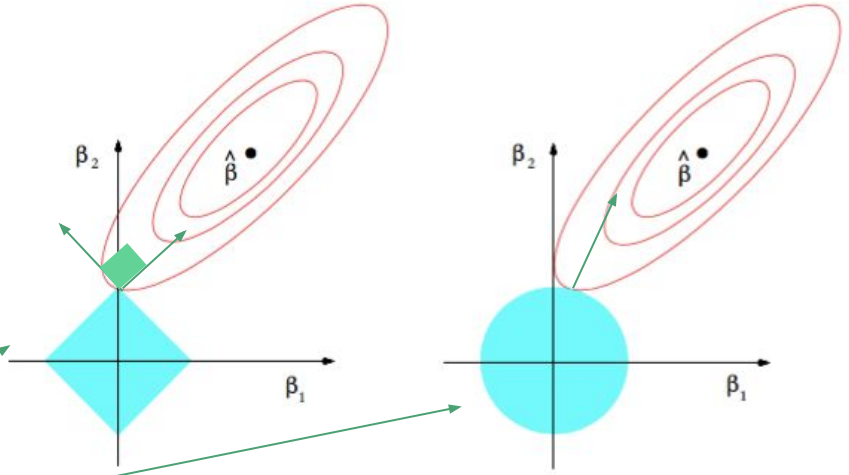
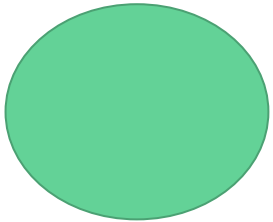


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are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

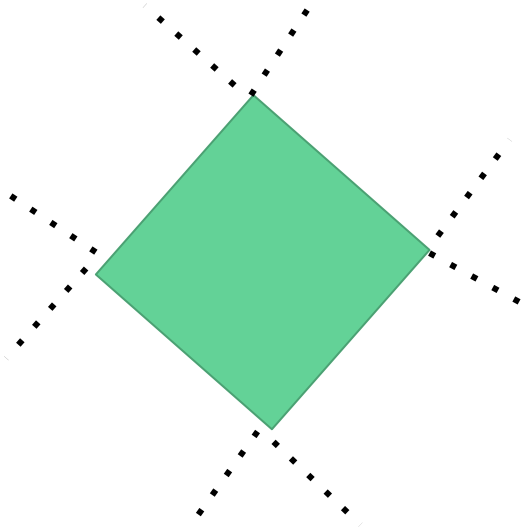
When p=1, the level sets are degenerate.

Example

$$\beta_1^2 + \beta_2^2 = 1 \qquad \nabla F(\beta_1, \beta_2) = (a, b) \qquad |\beta_1| + |\beta_2| = 1$$



Any point here has equal probability



Each corner has a range of $(\pi/4, -\pi/4)$ so it's more likely to hit the level set here if they intersect. .

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.

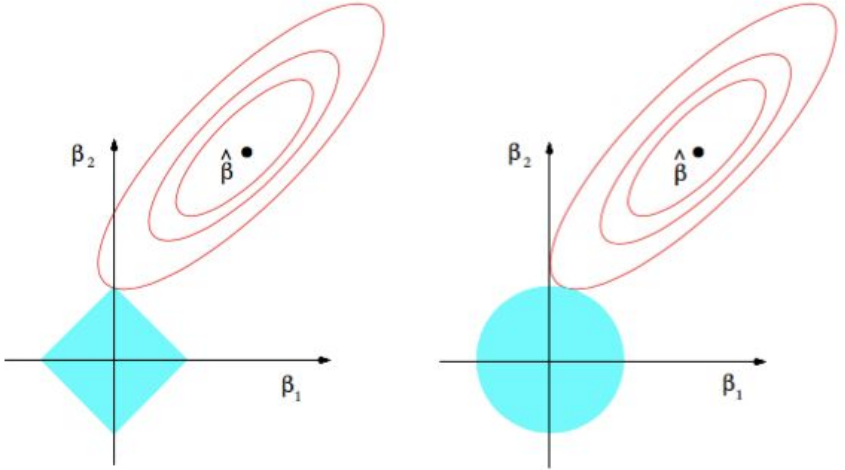
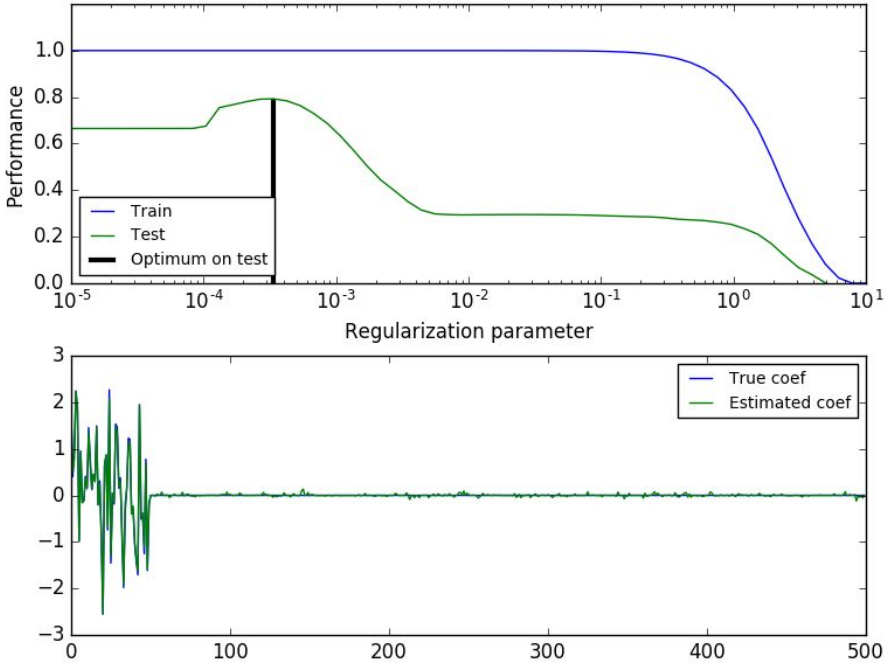


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are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

How do we use it? Concrete Example with L1

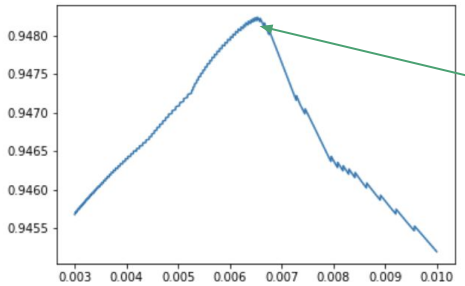
$$\mathcal{L}(y, f(x))_\lambda := \frac{1}{N}\sum_{i=1}^{N}(y_i - \beta \cdot x_i)^2 + \lambda\|\beta\|_{L^p}$$

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



Performance as we range alpha.

```
In [21]: scores = []

         alphas=np.linspace(0.003,0.01,1000)

         from sklearn.model_selection import train_test_split
         X_train, X_test,y_train,y_test= train_test_split(
             X_scaled, np.array(y), test_size=0.2, random_state=42)

         for d in alphas:
             regr = linear_model.Lasso(alpha=d)
             X=X_scaled

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

             # Make predictions using the testing set
             y_pred = regr.predict(X)
             scores.append(regr.score(X_test,y_test))
         plt.plot(alphas,scores)

Out[21]: [<matplotlib.lines.Line2D at 0x1c2c99e750>]
```



R^2

Alpha

Sweet spot

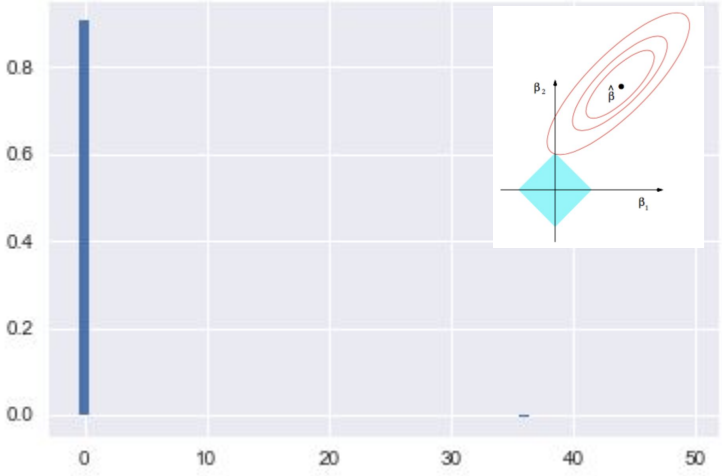
Using GridSearchCV

In [24]:
```
from sklearn.grid_search import GridSearchCV

alphas=np.linspace(0.003,0.01,1000)

model=linear_model.Lasso()
grid = GridSearchCV(estimator=model, param_grid=dict(alpha=alphas),cv=3)
grid.fit(X_train,y_train)

print(grid)
# summarize the results of the grid search
print(grid.best_score_)
print(grid.best_estimator_.alpha)
```

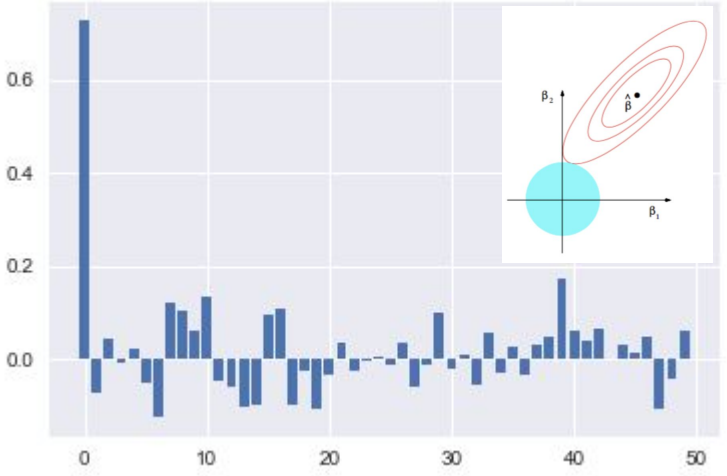
```
GridSearchCV(cv=3, error_score='raise',
       estimator=Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'alpha': array([0.003  , 0.00301, ..., 0.00999, 0.01   ])},
       pre_dispatch='2*n_jobs', refit=True, scoring=None, verbose=0)
0.908259563301
0.009985985985985987
```

Iterates optimally through the possible values of alpha using k-fold cross validation.

Lasso vs Ridge comparison



L^1



L^2

Lasso has done a better job of recovering the true model in this case.

Comparison of the two

- 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.

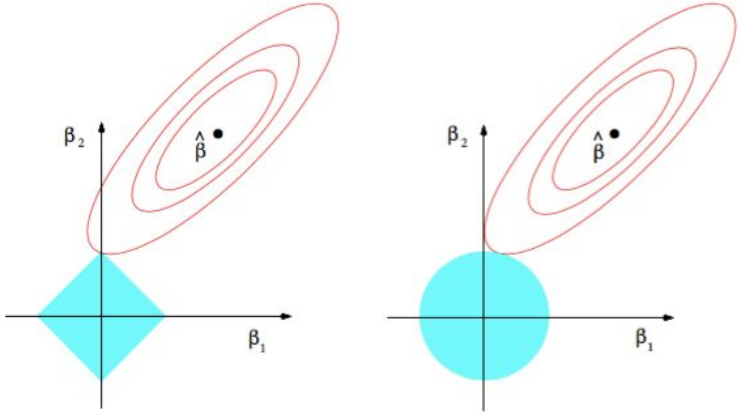


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are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

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).

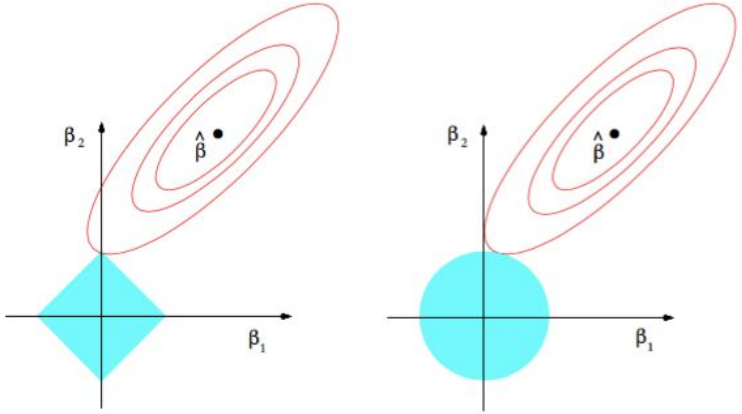
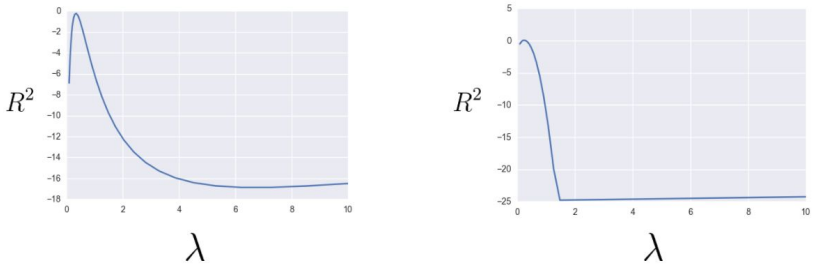


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are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

So which is better?



- 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.

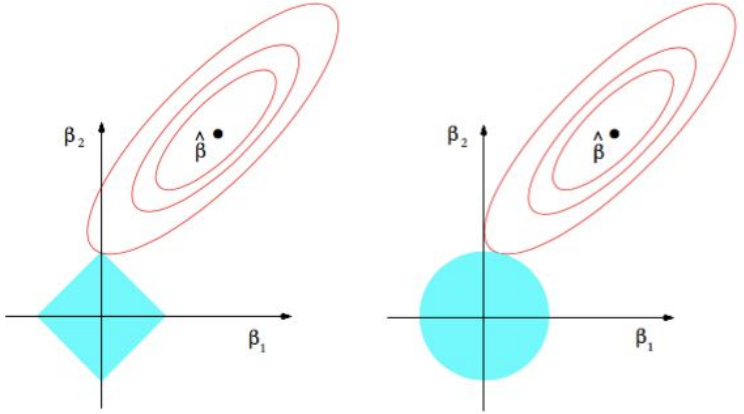
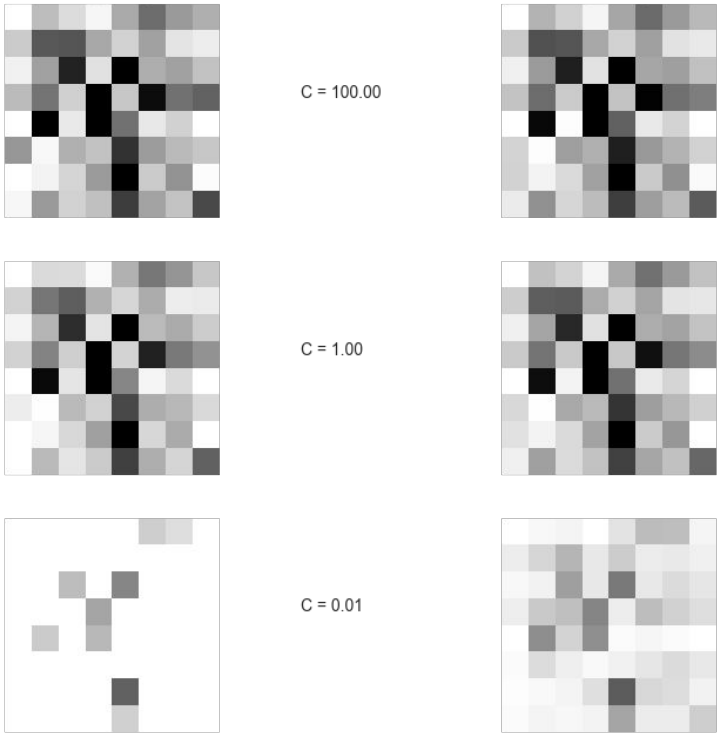


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are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

MNIST Digit Recognition

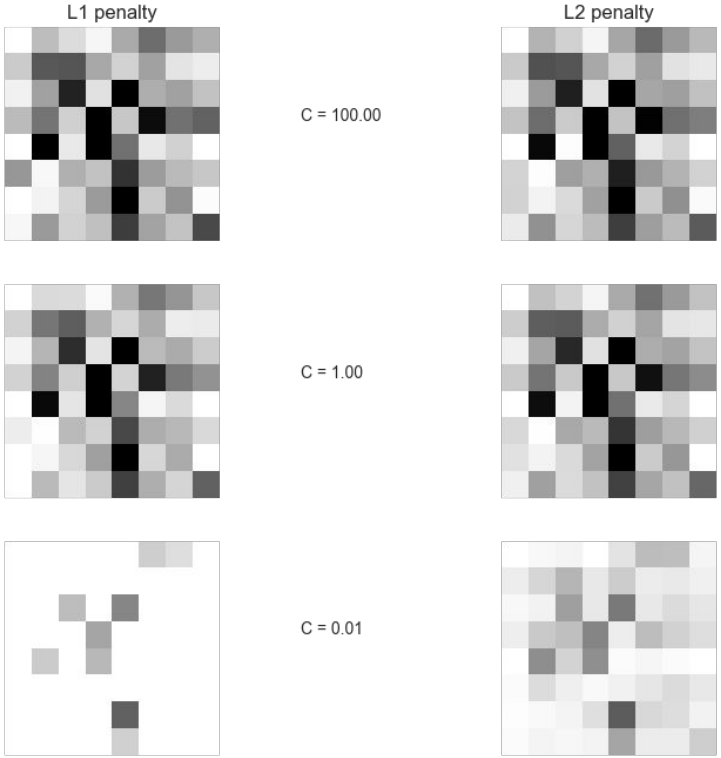
How less is more sometimes - the problem of overfitting

Digit Recognition with Regularization



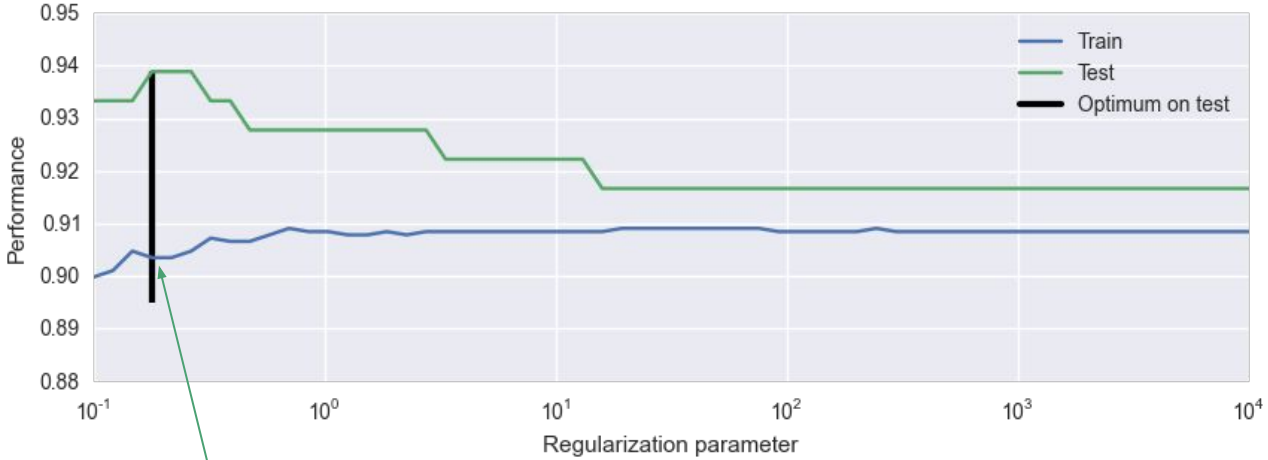
- Which one is Lasso and which one is Ridge?

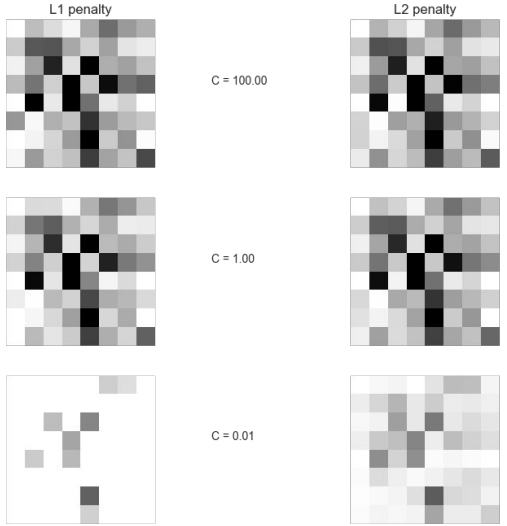
Digit Recognition with Regularization



- 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!

Suggested Quora Page

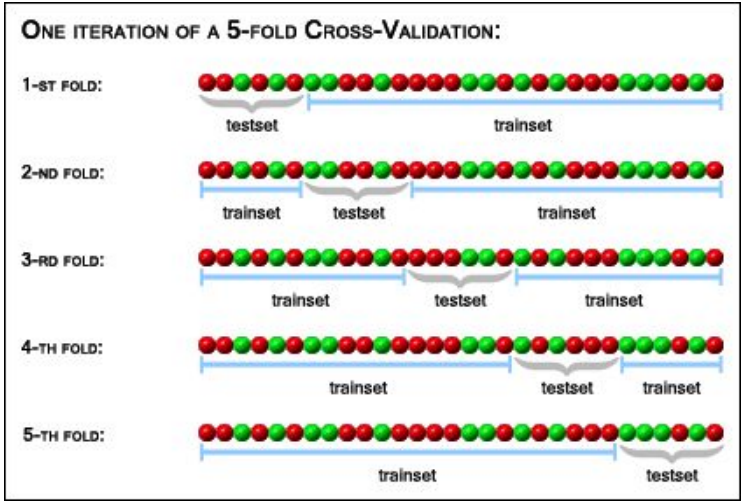
https://www.quora.com/What-is-the-difference-between-L1-and-L2-regularization

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

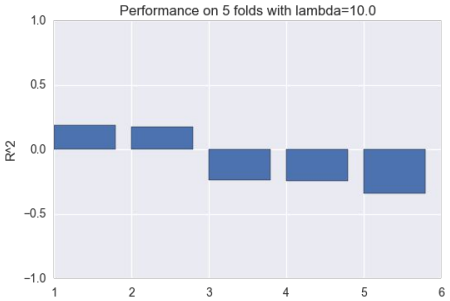
Cross Validation Revisited

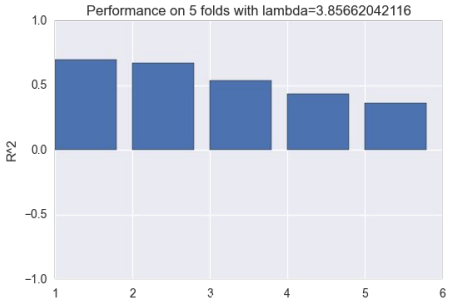


- Generally instead of just taking say **80% of your data for training**, **20% for testing** (for example), we randomly split the data into several **‘folds’.**

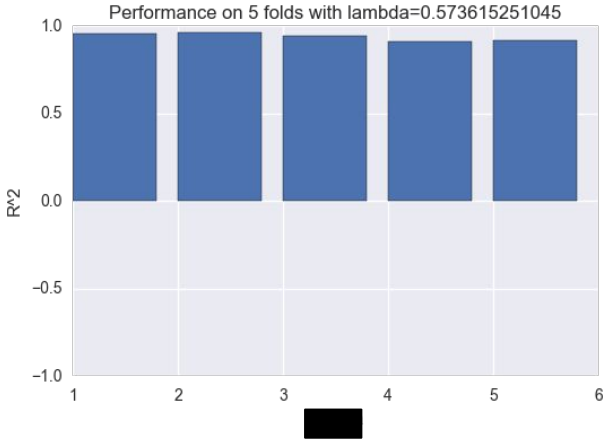
- In **k-fold cross-validation**, the original sample is randomly partitioned into **k equal sized subsamples**. Of the k subsamples, **a single subsample is retained as the validation data** for testing the model, and t**he remaining k − 1 subsamples are used as training data.**

Cross Validated Regularization

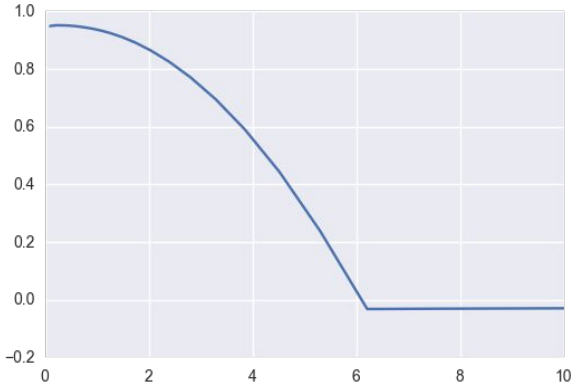




$$\frac{1}{N}\sum_{i=1}^{N}|y_i-\beta\cdot x_i|^2+\lambda\sum_{i=1}^{N}|\beta_i|$$



R^2



λ

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: Try to understand causal impact if possible.**