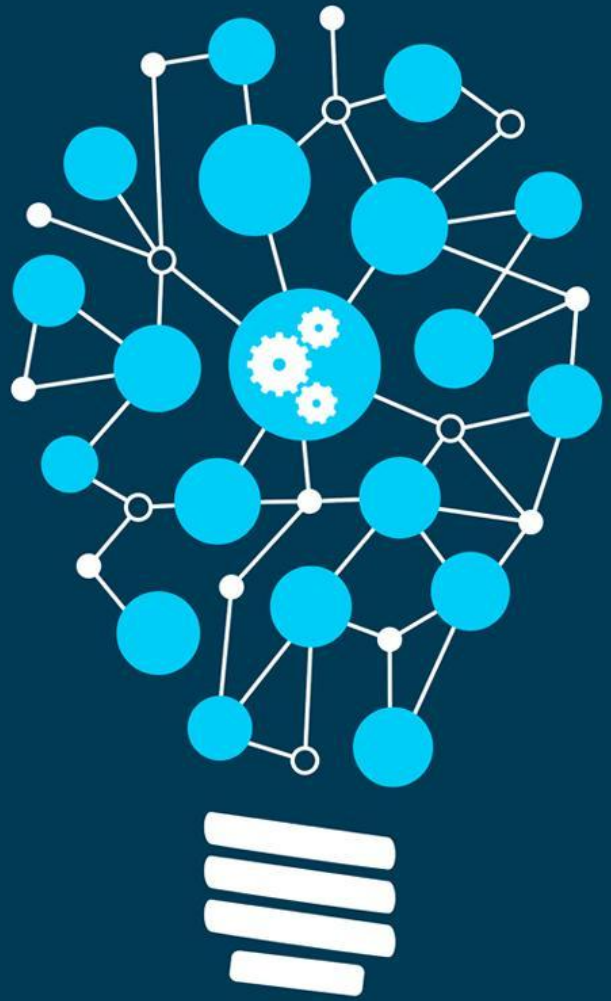
NYU

NYU Summer Machine Learning Program

Presenter Name Here
Date Here

Professor Rangan (with modification by Yao Wang)
Adapted by Nikola Janjusevic



NYU

Generalization Error

Day 4

Learning Objectives

- ❑Extend simple linear model to allow higher complexity modeling
- ❑Compute the model order for a given model class
- ❑Visually identify overfitting and underfitting of a model in a scatterplot
- ❑ Determine if there is under-modeling for a given true function and model class
- ❑Perform cross-validation for selecting an optimal order selection
- ❑ Formulate a linear regression problem with a regularizer
- ❑ Compute the optimal regularization level using cross-validation
- ❑ Interpret results from a LASSO path
- ❑ Understand Hyperparameters associated with Regularizers

Outline

- ❑ Transformed Linear Models
- ❑ Model Selection -- Polynomial Fitting
- ❑ Cross-Validation
- ❑ Lasso and Ridge Regularization

*

Transformed Linear Models

- Standard linear model: $\hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d$
- But, often it is useful to look at models in transformed form:

$$\hat{y} = \beta_1 \phi_1(\boldsymbol{x}) + \cdots + \beta_p \phi_p(\boldsymbol{x})$$

  - Each function $\phi_j(\boldsymbol{x}) = \phi_j(x_1, \ldots, x_d)$ is called a basis function
  - Each basis functions may be nonlinear and a function of multi-variables
- Can write in vector form: $\hat{y} = \boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{\beta}$
  - $\boldsymbol{\phi}(\boldsymbol{x}) = [\phi_1(\boldsymbol{x}), \ldots, \phi_b(\boldsymbol{x})]$, $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_p]$
- Enables a much richer class

Fitting Transformed Linear Models

- Consider transformed linear model

$$\hat{y} = \beta_1 \phi_1(\boldsymbol{x}) + \cdots + \beta_p \phi_p(\boldsymbol{x})$$

- We can fit this model exactly as before
  - Given data $(\boldsymbol{x}_i, y_i), i = 1, \ldots, N$
  - Then, fit the model from the transformed variables $\phi_j(\boldsymbol{x})$ to target y
  - Define the transformed matrix:

$$A = \begin{bmatrix} \phi_1(\boldsymbol{x}_1) & \cdots & \phi_p(\boldsymbol{x}_1) \\ \vdots & \vdots & \vdots \\ \phi_1(\boldsymbol{x}_N) & \cdots & \phi_p(\boldsymbol{x}_N) \end{bmatrix}$$

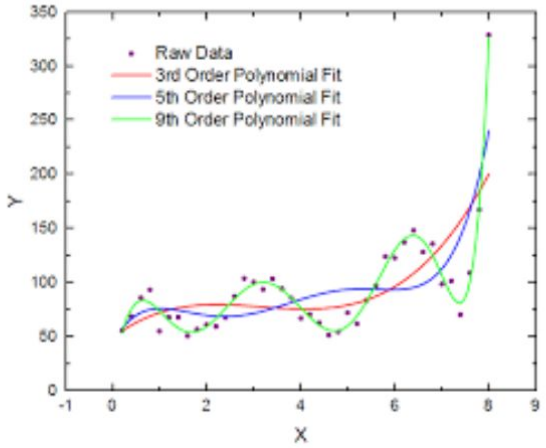
  - Least squares fit $\hat{\beta} = (A^T A)^{-1} A^T y$

Example: Polynomial Fitting

- Suppose y only depends on a single variable x,
- Want to fit a polynomial model
  - $y \approx \beta_0 + \beta_1 x + \cdots \beta_d x^d$
- Given data $(x_i, y_i), i = 1, \ldots, n$
- Take basis functions $\phi_j(x) = x^j, j = 0, \ldots, d$
- Transformed model: $\hat{y} = \beta_0 \phi_0(x) + \cdots + \beta_d \phi_d(x)$
- Transformed matrix is:

$$\boldsymbol{A} = \begin{bmatrix} 1 & x_1 & \cdots & x_1^d \\ \vdots & \vdots & \cdots & \vdots \\ 1 & x_n & \cdots & x_n^d \end{bmatrix}, \qquad \beta = \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_d \end{bmatrix}$$

  - $p = d + 1$ transformed features from 1 original feature
- Will discuss how to select d in the next lecture



Other Nonlinear Examples

- Multinomial model: $\hat{y} = a + b_1 x_1 + b_2 x_2 + c_1 x_1^2 + c_2 x_1 x_2 + c_3 x_2^2$
  - Contains all second order terms
  - Define parameter vector $\beta = [a, b_1, b_2, c_1, c_2, c_3]$
  - Transformed vector $\phi(x_1, x_2) = [1, x_1, x_2, x_1^2, x_1 x_2, x_2^2]$
  - Note that the features are nonlinear functions of $\boldsymbol{x} = [x_1, x_2]$

- Exponential model: $\hat{y} = a_1 \mathrm{e}^{-b_1 x} + \cdots + a_d e^{-b_d x}$
  - If the parameters $b_1, \ldots, b_d$ are fixed, then the model is linear in the parameters $a_1, \ldots, a_d$
  - Parameter vector $\beta = [a_1, \ldots, a_d]$
  - Transformed vector $\phi(x) = [e^{-b_1 x}, \ldots, e^{-b_d x}]$
  - But, if the parameters $b_1, \ldots, b_d$ are not fixed, the model is nonlinear in $b_1, \ldots, b_d$

Linear Models via Re-Parametrization

- Sometimes models can be made into a linear model via re-parametrization
- Example: Consider the model: $\hat{y} = Ax_1(1 + Be^{-x_2})$
  - Parameters (A, B)
- This is nonlinear in (A, B) due to the product AB: $\hat{y} = Ax_1 + ABx_1e^{-x_2}$
- But, we can define a new set of parameters:
  - $\beta_1 = A$ and $\beta_2 = AB$
- Then, $\hat{y} = \beta_1 x_1 + \beta_2 x_1 e^{-x_2}$
- Basis functions: $\phi(x_1, x_2) = [x_1, x_1 e^{-x_2}]$
- After we solve for β_1, β_2 we can recover A, B via inverting the equations:

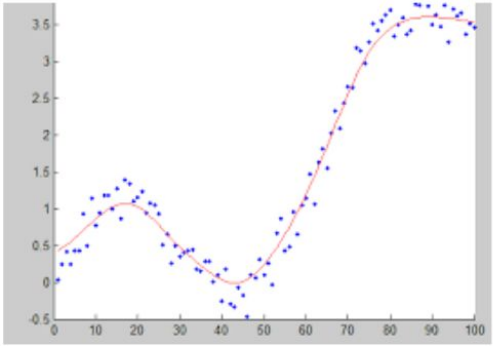
$$A = \beta_1, \qquad B = \frac{\beta_2}{A}$$

Polynomial Fitting

- Last lecture: polynomial regression
- Given data $(x_i, y_i), i = 1, \ldots, N$
- Learn a polynomial relationship:

$$y = \beta_0 + \beta_1 x + \cdots + \beta_d x^d + \epsilon$$

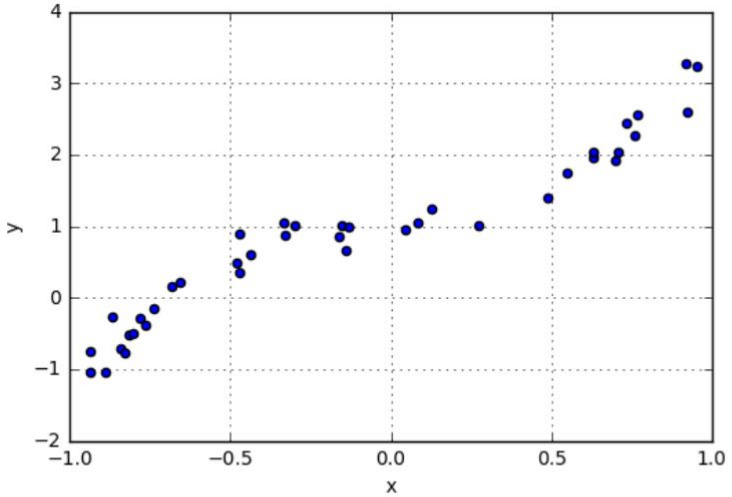
  - d = degree of polynomial. Called model order
  - $\boldsymbol{\beta} = (\beta_0, \cdots, \beta_d)$ = coefficient vector
- Given d, can find $\boldsymbol{\beta}$ via least squares
- How do we select d from data?
- This problem is called model order selection.



Example Question

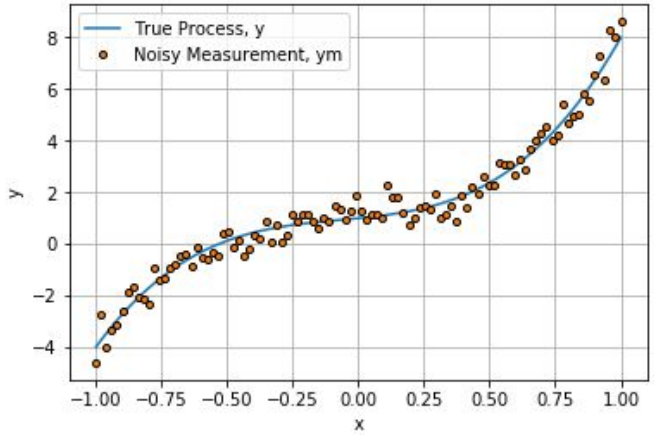
See Notebook!

- You are given some data.
- Want to fit a model: $y \approx f(x)$
- Decide to use a polynomial:
$$f(x) = \beta_0 + \beta_1 x + \cdots + \beta_d x^d$$
- What model order d should we use?
- Thoughts?



Synthetic Data

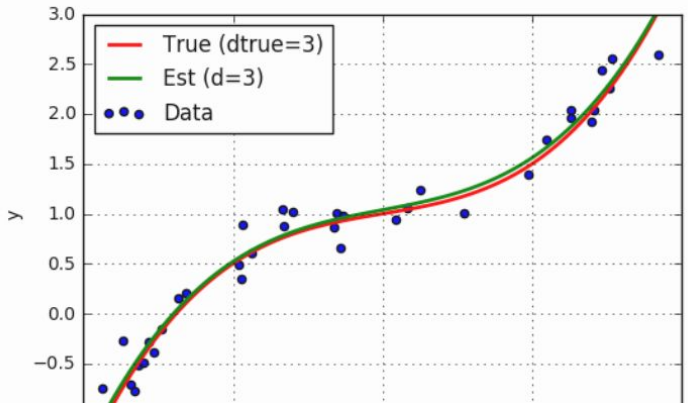
- Previous example is synthetic data
- x_i: 40 samples uniform in [-1,1]
- $y = f(x) + \epsilon$,
  - $f(x) = \beta_0 + \beta_1 x + \cdots + \beta_d x^d$ = "true relation"
  - $d = 3,\ \epsilon \sim N(0, \sigma^2)$
- Synthetic data useful for analysis
  - Know "ground truth"
  - Can measure performance of various estimators




```
nsamp = 100 # number of samples taken
c = np.array([1,1,1,5]) # true process coefficients
var = 0.2 # noise variance

# we'll take a set of measurements uniformly
x = np.linspace(-1,1,nsamp).reshape(-1,1)
# design matrix A
A = np.hstack((np.ones((nsamp,1)), x, x**2, x**3))
# true process output, y
y = np.matmul(A,c)
# noisy measurement, ym. use sqrt(var) as numpy normal standard deviation
ym = y + np.random.normal(0, np.sqrt(var), nsamp); ym = ym.reshape(-1,1)
```

Fitting with True Model Order

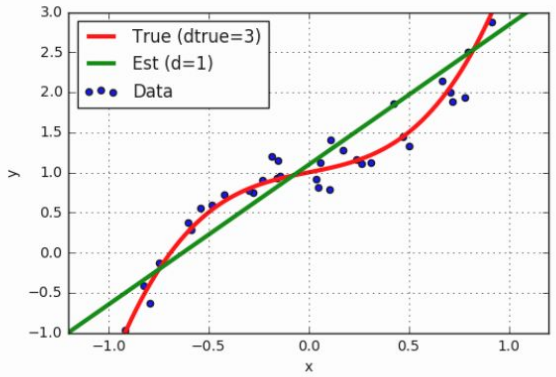
- Suppose true polynomial order, d=3, is known
- Use linear regression
  - numpy.polynomial package
- Get very good fit



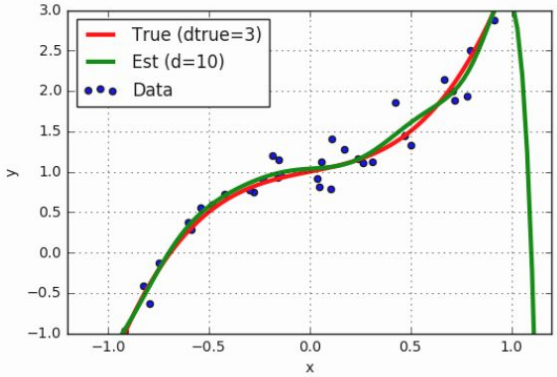

```
d = 3
beta_hat = poly.polyfit(xdat,ydat,d)

# Plot true and estimated function
xp = np.linspace(-1,1,100)
yp = poly.polyval(xp,beta)
yp_hat = poly.polyval(xp,beta_hat)
plt.xlim(-1,1)
plt.ylim(-1,3)
plt.plot(xp,yp,'r-',linewidth=2)
plt.plot(xp,yp_hat,'g-',linewidth=2)

# Plot data
plt.scatter(xdat,ydat)
plt.legend(['True (dtrue=3)', 'Est (d=3)', 'Data'], loc='upper left')
plt.grid()
plt.xlabel('x')
plt.ylabel('y')
```

But, True Model Order not Known

- Suppose we guess the wrong model order?

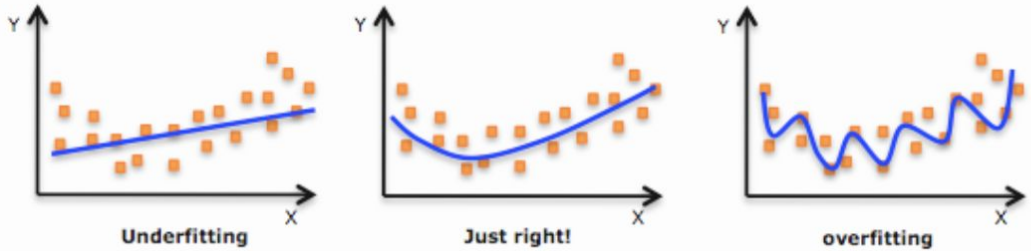


d=1 “Underfitting”



d=10 “Overfitting”

How Can You Tell from Data?



- Is there a way to tell what is the correct model order to use?
- Must use the data. Do not have access to the true d?
- What happens if we guess:
  - d too big?
  - d too small?

Using RSS on Training Data?

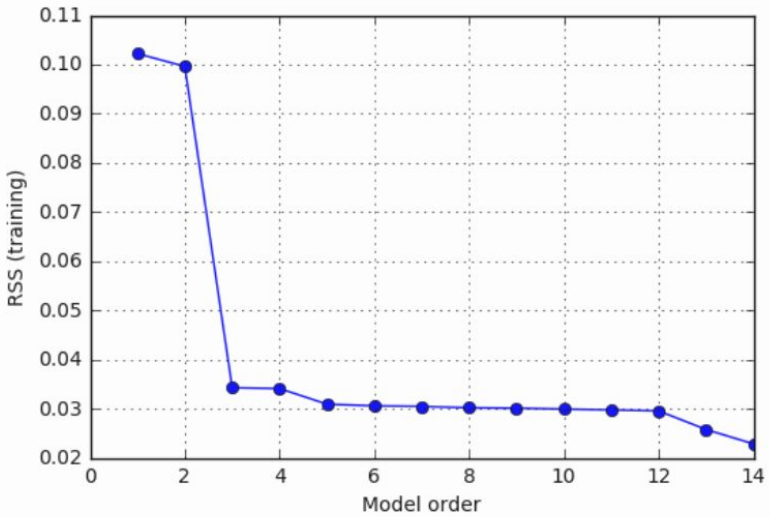
- Simple (but bad) idea:
  - For each model order, d, find estimate $\widehat{\boldsymbol{\beta}}$
  - Compute predicted values on training data

$$\hat{y}_i = \widehat{\boldsymbol{\beta}}^T \boldsymbol{x}_i$$

  - Compute RSS

$$RSS(d) = \sum_i (y_i - \hat{y}_i)^2$$

  - Find d with lowest RSS
- This doesn't work
  - $RSS(d)$ is always decreasing (Question: Why?)
  - Minimizing $RSS(d)$ will pick d as large as possible
  - Leads to overfitting
- What went wrong?
- How do we do better?



Cross Validation

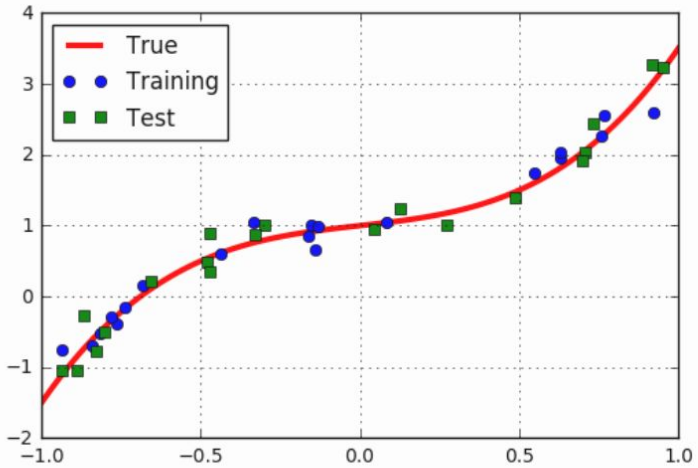
- Concept: Need to fit on test data independent of training data
- Divide data into two sets:
  - N_{train} training samples, N_{test} test samples
- For each model order, p, learn parameters $\hat{\beta}$ from training samples
- Measure RSS on test samples.

$$RSS_{test}(p) = \sum_{i \in \text{test}} (\hat{y}_i - y_i)^2$$

- Select model order p that minimizes $RSS_{test}(p)$

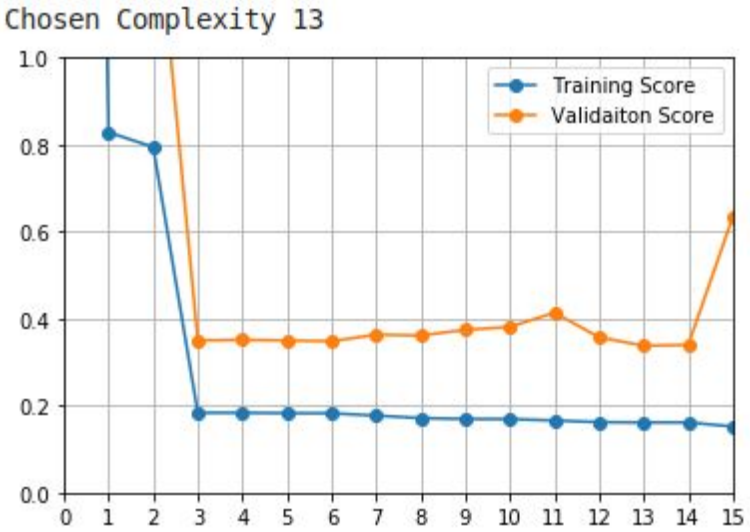
Polynomial Example: Training Test Split

- Example: Split data into 20 samples for training, 20 for test




```
ntrain = int(np.round(0.8*nsamp))
nval   = int(np.round(0.1*nsamp))
ntest  = nsamp - (ntrain + nval)

s = np.arange(nsamp); np.random.shuffle(s)
ind_train = s[:ntrain]
ind_val   = s[ntrain:(ntrain+nval)]
ind_test  = s[(ntrain+nval):]

x_train = x[ind_train]; y_train = ym[ind_train]
x_val   = x[ind_val];   y_val   = ym[ind_val]
x_test  = x[ind_test];  y_test  = ym[ind_test]
```

Finding the Model Order



RSS training always decreases

```
M = 15
train_score = np.zeros((M+1,1))
val_score = np.zeros((M+1,1))
coefficients = []

for m in range(M+1):
  # forming the design matrix
  A = np.ones((ntrain,1))
  for i in range(m):
    A = np.hstack((A,x_train**(i+1)))
  soln = np.linalg.lstsq(A,y_train,rcond=None)
  c_hat = soln[0]; coefficients.append(c_hat)
  trainRSS   = soln[1]
  train_score[m] = trainRSS/ntrain

  # validation score
  A = np.ones((nval,1))
  for i in range(m):
    A = np.hstack((A,x_val**(i+1)))
  y_hat = np.matmul(A,c_hat)
  valRSS = np.sum((y_val - y_hat)**2)
  val_score[m] = valRSS/nval

plt.plot(train_score,'o-')
plt.plot(val_score,'o-');
plt.xticks(np.arange(M+1))
plt.xlim(0,M)
plt.ylim(0,1)
plt.grid()
plt.legend(['Training Score','Validaiton Score']);
```

General Procedure

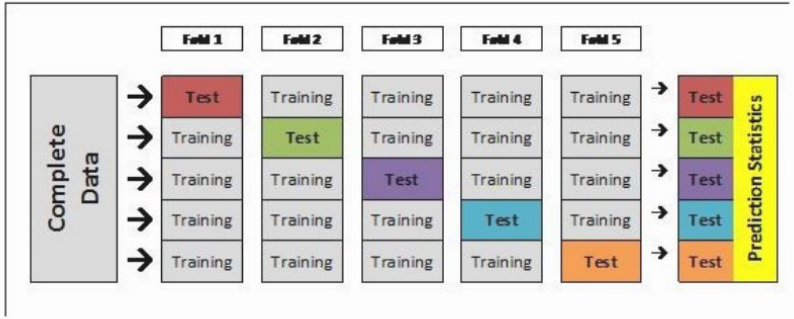
- Get data X, y
- Split into training X_{tr}, y_{tr} and test X_{ts}, y_{ts}
- For $p = 1$ to p_{max} // Loop over model order
  - Fit on training data with model order p : $\hat{\beta} = \text{fit}(X_{tr}, y_{tr}, p)$
  - Predict values on test data: $\hat{y}_{ts} = \text{predict}(X_{ts}, \hat{\beta})$
  - Score fit on test data: $S[p] = \text{score}(y_{ts}, \hat{y}_{ts})$
- Select model order with smallest score: $\hat{p} = \arg\min_p S[p]$
  - Or, use highest score if we want to maximize score instead of minimizing

Problems with Simple Train/Test Split

- Test error could vary significantly depending on samples selected
- Only use limited number of samples for training
- Problems particularly bad for data with limited number of samples

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/

K-Fold Pseudo-Code

- Get data X, y
- For $i = 1$ to K // Loop over folds
  - Split into training X_{tr}, y_{tr} and test X_{ts}, y_{ts} for fold i
  - For $p = 1$ to p_{max} // Loop over model order
    - Fit on training data with model order p : $\hat{\beta} = \text{fit}(X_{tr}, y_{tr}, p)$
    - Predict values on test data: $\hat{y}_{ts} = \text{predict}(X_{ts}, \hat{\beta})$
    - Score the fit on test data: $\text{S}[p, i] = \text{score}(y_{ts}, \hat{y}_{ts})$
- Find average score for each model order: $\bar{S}[\text{p}] = \frac{1}{K}\sum_{i=1}^{K} S[p, i]$
- Select model order with lowest average score: $\hat{p} = \arg\min_{p} \bar{S}[p]$

Polynomial Example

- Use sklearn Kfold object
- Loop
  - Outer loop: Over K folds
  - Inner loop: Over D model orders
  - Measure test error in each fold and order
  - Averaging test errors from K folds for each model order
  - Find the model order with the minimal average test errors
  - Can be time-consuming

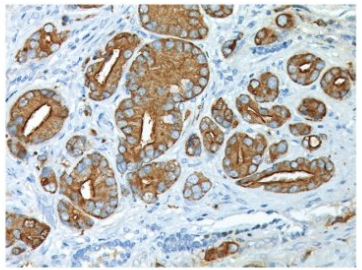
```
# Create a k-fold object
nfold = 20
kf = sklearn.model_selection.KFold(n_splits=nfold,shuffle=True)

# Model orders to be tested
dtest = np.arange(0,10)
nd = len(dtest)

# Loop over the folds
RSSts = np.zeros((nd,nfold))
for isplit, Ind in enumerate(kf.split(xdat)):

    # Get the training data in the split
    Itr, Its = Ind
    xtr = xdat[Itr]
    ytr = ydat[Itr]
    xts = xdat[Its]
    yts = ydat[Its]

    for it, d in enumerate(dtest):

        # Fit data on training data
        beta_hat = poly.polyfit(xtr,ytr,d)

        # Measure RSS on test data
        yhat = poly.polyval(xts,beta_hat)
        RSSts[it,isplit] = np.mean((yhat-yts)**2)
```

Learning Objectives

- ❑ Formulate a linear estimation problem with a regularization
- ❑ Compute an L1-regularized estimate (LASSO) using sklearn tools
- ❑ Compute the optimal regularization level using cross validation
- ❑ Interpret results from a LASSO path
- ❑ Determine final regression function from cross validation
- ❑ Set regularizer based on a probabilistic prior
- ❑ Perform other feature selection methods

Prostate Specific Antigen Testing

- PSA levels easily tested
- High PSA believed to be associated with prostate cancer
  - Potential tool for screening
- Classic 1989 study by Thomas et al:
  - Measured PSA level of 102 men prior to prostate removal
  - Measured characteristics of prostate from samples
  - Characteristics include cancer volume, weight, …
- Data analysis:
  - What characteristics predict PSA?



Stamey, Thomas A., et al. "Prostate specific antigen in the diagnosis and treatment of adenocarcinoma of the prostate. II. Radical prostatectomy treated patients." The Journal of urology 141.5 (1989): 1076-1083.

Data

- Prostate dataset widely-used in ML classes
- Can be downloaded from many sites
- Samples = 97 patients
- 8 features of the prostate
- Target variable = lpsa (log PSA)

```
# Get data
url = 'https://web.stanford.edu/~hastie/ElemStatLearn/datasets/prostate.data'
df = pd.read_csv(url, sep='\t', header=0)
df = df.drop('Unnamed: 0', axis=1)   # skip the column of indices
```

The data frame has the following components:

`lcavol`
: log(cancer volume)

`lweight`
: log(prostate weight)

`age`
: age

`lbph`
: log(benign prostatic hyperplasia amount)

`svi`
: seminal vesicle invasion

`lcp`
: log(capsular penetration)

`gleason`
: Gleason score

`pgg45`
: percentage Gleason scores 4 or 5

`lpsa`
: log(prostate specific antigen)

First Try: Linear Model

- Simple idea: Use linear regression

$$y \approx \hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d$$

  - y = lpsa (target PSA level)
  - $x_1, \ldots, x_d$ = prostate features (d = 8)

```
from sklearn import linear_model
from sklearn.model_selection import train_test_split
```

```
X_tr, X_ts, y_tr, y_ts = train_test_split(X,y,test_size=0.5,shuffle=True)
ntr = X_tr.shape[0]
nts = X_ts.shape[0]
print("num samples train = %d, test = %d" % (ntr, nts))
```

```
num samples train = 48, test = 49
```

- Why linear regression?
  - Easy to compute / interpret
  - Coefficients are easy to interpret
  - Larger coefficients ⇒ larger influence of feature on PSA

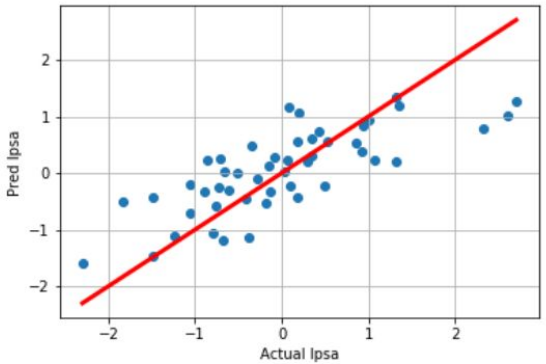
```
regr = linear_model.LinearRegression()
regr.fit(X_tr,y_tr)
```

Model Fit

- Evaluate model with cross validation
  - Train on 48 samples
  - Measure RSS on 49 samples
- We obtain reasonable fit on test data
  - $R^2 \approx 0.58$

```
y_ts_pred = regr.predict(X_ts)
rss_ts = np.mean((y_ts_pred-y_ts)**2)/(np.std(y_ts)**2)
rsq_ts = 1-rss_ts
print("Normalized test RSS = %f" % rss_ts)
print("Normalized test R^2 = %f" % rsq_ts)
```

```
Normalized test RSS = 0.419799
Normalized test R^2 = 0.580201
```

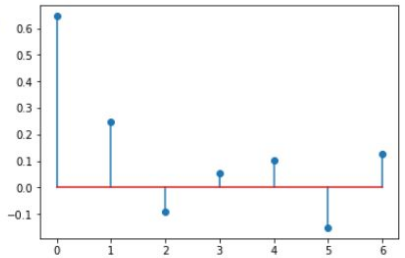


Looking at the Coefficients

- Recall that model is: $\hat{y} = b + w_1 x_1 + \cdots + w_d x_d$
- Weights w_i indicates dependence of feature i on target y
- For PSA test:
  - Highest weight on lcavol (log cancer volume)
  - But, weights on all features are non-zero
  - Hard to eliminate features
- How can we tell if some features are not significant?

```
w = regr.coef_
for name, wi in zip(names_x, w):
    print('%10s %9.4f' % (name, wi))
```

```
    lcavol    0.6457
   lweight    0.2466
       age   -0.0895
      lbph    0.0543
       svi    0.1034
       lcp   -0.1508
   gleason    0.1253
```



Outline

- Motivating Example: Predicting prostate cancer from a PSA test
- Model Selection
- Model Selection from LASSO regularization
- Probabilistic interpretation
- Other Model Selection Methods
- In-Class Exercise: Audio Pitch Detection

Model Selection

- Consider linear model: $y \approx \hat{y} = b + w_1 x_1 + \cdots + w_d x_d$
- Models target y as function of features $\boldsymbol{x} = (x_1, \ldots, x_d)$
- In many problems, we know only a few features are likely relevant
- This means we want $w_j = 0$ for most features
- But, we don't know a priori which features are relevant
- Model selection problem: Fit a model with a small number of features
- Mathematically:
  - Determine a subset of features $I \subseteq \{1, \ldots, d\}$ with $|I|$ small
  - Fit a model: $\hat{y} = b + w_1 x_1 + \cdots + w_d x_d$ with $w_j = 0$ for all $j \notin I$

Model Selection with Limited Data

- Model selection is particularly valuable when data is limited
- Ex: Consider linear model: $\hat{y} = b + w_1 x_1 + \cdots + w_d x_d$
  - Model has $d + 1$ parameters
- From previous lecture, we need $N > d + 1$ data points $(\boldsymbol{x}_i, y_i)$
- In many cases we have $N \ll d$
  - Examples below
  - Many few data points than features
  - Classic linear fit will not work
- But, suppose we can restrict to $K \ll N$ non-zero parameters
  - Then, we can find a good fit on those parameters
- Challenge: How do we find a small number K of relevant features

Example 1: Medical Modeling

- PSA test:
  - We have a number of features
- But, likely that only a small number of features are relevant
- Want a method that can determine which ones matter

The data frame has the following components:

`lcavol`
: log(cancer volume)

`lweight`
: log(prostate weight)

`age`
: age

`lbph`
: log(benign prostatic hyperplasia amount)

`svi`
: seminal vesicle invasion

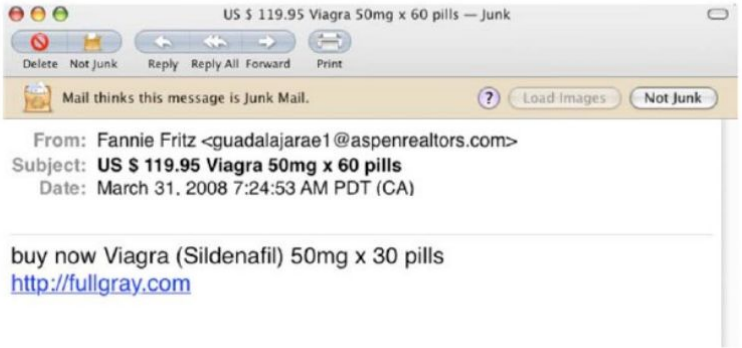
`lcp`
: log(capsular penetration)

`gleason`
: Gleason score

`pgg45`
: percentage Gleason scores 4 or 5

`lpsa`
: log(prostate specific antigen)

Example 2: Spam Detection



- Classification problem:
  - Is email junk or not junk?
- Typical bag-of-word model:
  - Enumerate all words, $i = 1, \ldots, d$
  - Represent email via word count x_i = num instances of word i
- Model selection:
  - d =vocabulary size is typically very large
  - But, only a few words are likely relevant
  - Want to find $K \ll d$ relevant words

Example 3: EEG

- EEG: Electroencephalography
- Measure brain activity from electrodes on scalp
- Source localization problem:
  - Find brain region responsible for evoked response
- Problem:
  - Many possible brain regions
    Typically use d >10,000 voxels
  - But, limited number of measurements:
    100s of electrodes
  - Cannot fit a model from all brain regions
- Model selection:
  - We know that responses are likely from a small brain region
  - Find a small number of voxels that explain response

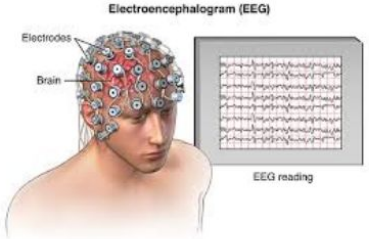
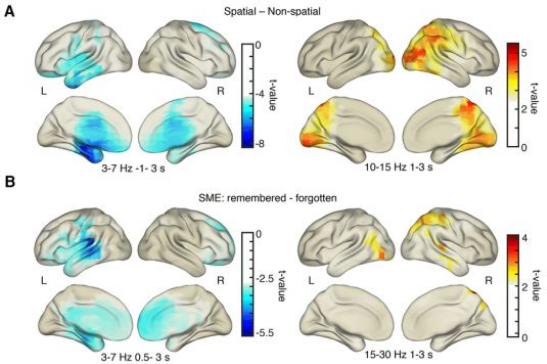
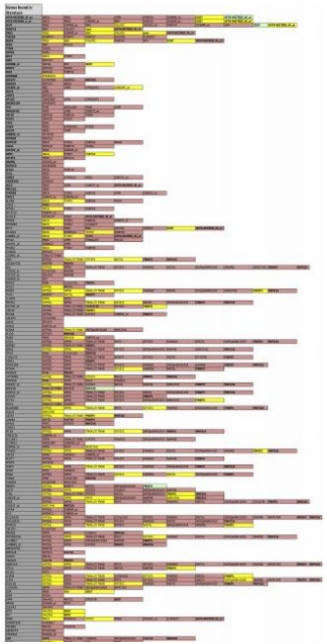


Image: mayoclinic.org



Example 4: DNA MicroArray Data

- Basic genetic problem
  - Which genes determine some characteristic (i.e. phenotype)?
- DNA microarrays:
  - Measure "expression" levels of large numbers of genes
  - Expression levels = amount of protein produced by gene
- Data modeling:
  - Fit phenotype to expression levels
  - Usually have large numbers of genes $(d \sim 1000)$
  - But, small number of data points $(n \sim 100)$
  - We know only a small number of genes are responsible
  - So, we can use model selection



Outline

- ❑ Motivating Example: Predicting prostate cancer from a PSA test
- ❑ Model Selection
- ❑ Model Selection from LASSO regularization
- ❑ Probabilistic interpretation
- ❑ Other Model Selection Methods
- ❑ In-Class Exercise: Audio Pitch Detection

Intuition

- We know from last lecture:
  - Too many parameters ⇒ Large generalization error
- In this data set, only a few factors are likely significant
- But, we don't know which one
- Can we automatically identify them?
  - Use correlation between features and target
    - Do not always work well
  - Exhaustive search can be expansive!
- Idea: Fit model under constraint:
  - Force only a few parameters to be non-zero
- General idea of regularization:
  - Constrain the parameters with prior knowledge

The data frame has the following components:

`lcavol`
: log(cancer volume)

`lweight`
: log(prostate weight)

`age`
: age

`lbph`
: log(benign prostatic hyperplasia amount)

`svi`
: seminal vesicle invasion

`lcp`
: log(capsular penetration)

`gleason`
: Gleason score

`pgg45`
: percentage Gleason scores 4 or 5

`lpsa`
: log(prostate specific antigen)

Regularized LS Estimation

- Standard least squares estimation (from Lecture 3):

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

- Regularized estimator:

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

  - $RSS(\beta)$ = 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

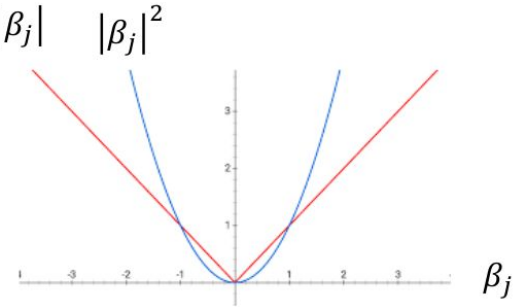
- Ridge regression (called L2)

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

- LASSO regression (called L1)

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

- Both penalize large β_j
- Level of regularization controlled by α
- Note the regularization sum
  - Does not include the intercept β_0,
  - This terms depends on the mean of the target
  - Should not be arbitrarily constrained to be small



Minimize $|\beta_j|^2$ do not penalize small non-zero coef., overly penalize large coef.
Minimize $|\beta_j|$ tend to make coefficients either 0 or large (SPARSE!)

Data Scaling

- Scaling:
  - Scale each feature and the target to have zero mean and unit variance (or STD)
  - $x_{i,j} \rightarrow (x_{i,j} - E(x_{i,j}))/\mathrm{STD}(x_{i,j})$
  - $y_i \rightarrow \big(y_i - \mathrm{E}(y_i)\big)/\mathrm{STD}(y_i)$
- After predictor for the scaled data are determined:
  - Derive the equivalent predictor on the original data (HW!)
- 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

L1 and L2 Norm

Advanced!

- Assuming the data have been scaled to have zero mean and unit variance
- Ridge cost function:

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

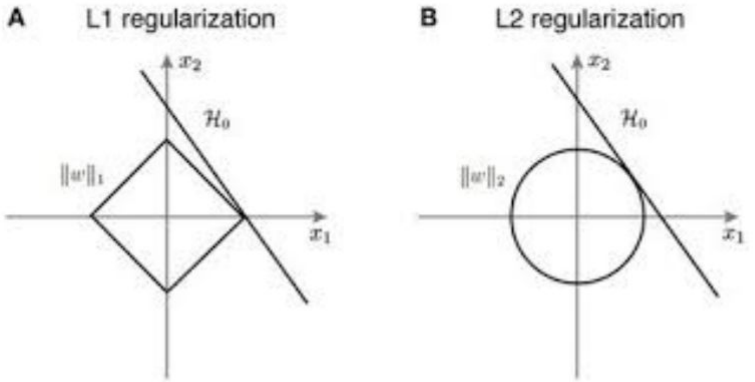
- LASSO cost function:

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

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

Ridge vs LASSO

- Optimization can be easily performed for L1 and L2 regularizers
  - Regularizer is convex
  - More on this later
- L2 tends to lead to many "small" coefficients
  - Not great for feature selection
  - Closed-form solution possible
- L1 tends to lead to more sparse solutions
  - Several coefficients are zero
  - No closed-form solution
  - Will focus this lecture on L1



Ridge Regression

- Loss function

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

- Why minimize $\|\beta\|^2$?
  - Tries to keep coefficients small
  - Only use coefficients when it reduces prediction error
- Without regularization:
  - Parameters are unrestricted
  - Models have high variance
  - Large positive and negative coefficients cancel each other for correlated features

Ridge Regression

- Solution for given regularization level
  - Easily obtainable by setting gradient to zero (HW!)

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

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

- How to determine the right regularization level α?
  - Through cross validation!

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

Coefficient path with ridge regression

Note that larger α does not lead to fewer non-zero coefficients, but only smaller (and mostly positive) coefficients!

Figure from [Hastie2008]: Hastie, Tibshirani, Friedman, The elements of statistical learning.

For more on this subject, see Sec. 3.4.1.

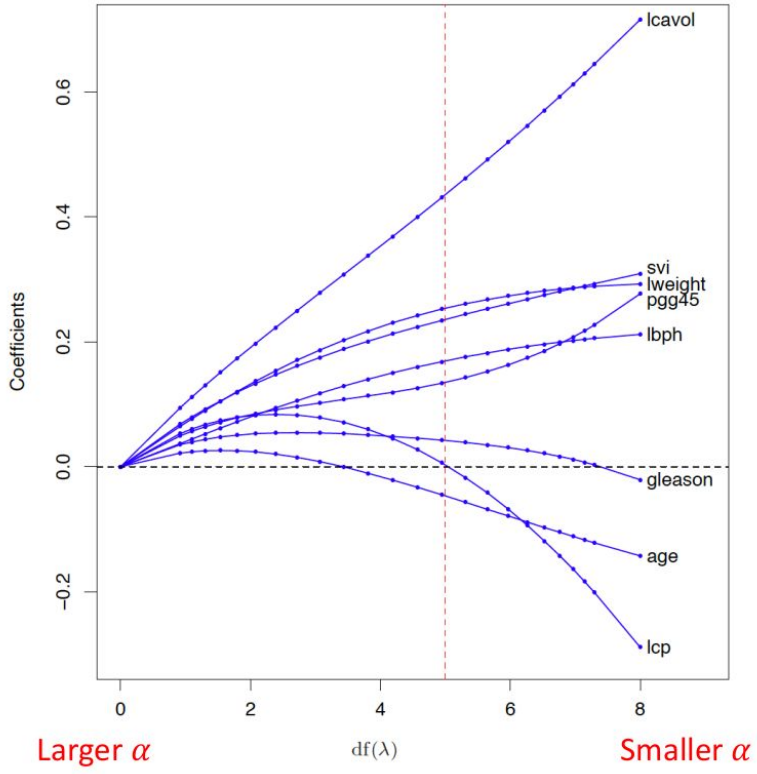


FIGURE 3.8. *Profiles of ridge coefficients for the prostate cancer example, as the tuning parameter λ is varied. Coefficients are plotted versus $\mathrm{df}(\lambda)$, the effective degrees of freedom. A vertical line is drawn at $\mathrm{df} = 5.0$, the value chosen by cross-validation.*

LASSO Regression

- LASSO cost function:

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

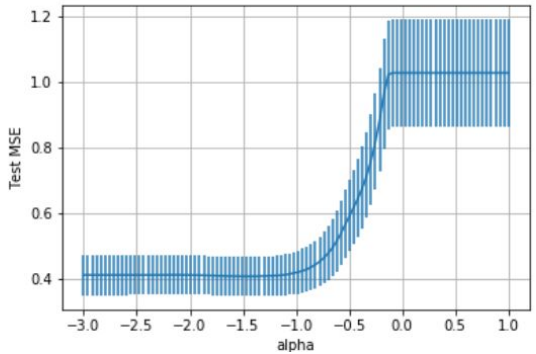
- Because derivative of $|\beta_j|$ is not continuous, there is no closed-form solution.
- However, there is a unique minimum because the cost function is convex.
- Many methods to solve iteratively
  - Least angle regression (LAR), coordinate descent, ADMM
  - Beyond the scope of this class
  - See textbook [Hastie2008] for LAR method

Selecting Regularization Level

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

Computing LASSO in python

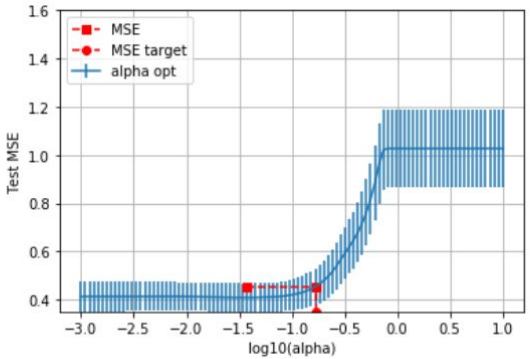
- Use sklearn Lasso method
  - Solve using coordinate descent
- Cross validation loop
  - Outer loop: Loop over folds
  - Inner loop: Loop over α
  - Measure mean and std deviation of MSE




```
# Create a k-fold cross validation object
nfold = 10
kf = sklearn.model_selection.KFold(n_splits=nfold,shuffle=True)

# Create the LASSO model.  We use the `warm start` parameter so
# This speeds up the fitting.
model = linear_model.Lasso(warm_start=True)

# Regularization values to test
nalpha = 100
alphas = np.logspace(-3,1,nalpha)

# MSE for each alpha and fold value
mse = np.zeros((nalpha,nfold))
for ifold, ind in enumerate(kf.split(X)):


    # Get the training data in the split
    Itr,Its = ind
    X_tr = X[Itr,:]
    y_tr = y[Itr]
    X_ts = X[Its,:]
    y_ts = y[Its]

    # Compute the lasso path for the split
    for ia, a in enumerate(alphas):

        # Fit the model on the training data
        model.alpha = a
        model.fit(X_tr,y_tr)

        # Compute the prediction error on the test data
        y_ts_pred = model.predict(X_ts)
        mse[ia,ifold] = np.mean((y_ts_pred-y_ts)**2)
```

Using One Standard Deviation Rule

- Use one standard deviation rule from before
  - Find α_0 with minimum mean MSE, mean_mean
  - Set $\text{mse\_tgt} = \text{mse\_mean}[\alpha_0] + \text{mse\_se}[\alpha_0]$
  - Find largest α where mse_mean[α] < mse_tgt




```
# Find the minimum MSE and MSE target
imin = np.argmin(mse_mean)
mse_tgt = mse_mean[imin] + mse_se[imin]
alpha_min = alphas[imin]

# Find the least complex model with mse_mean < mse_tgt
I = np.where(mse_mean < mse_tgt)[0]
iopt = I[-1]
alpha_opt = alphas[iopt]
print("Optimal alpha = %f" % alpha_opt)
```

Coefficients

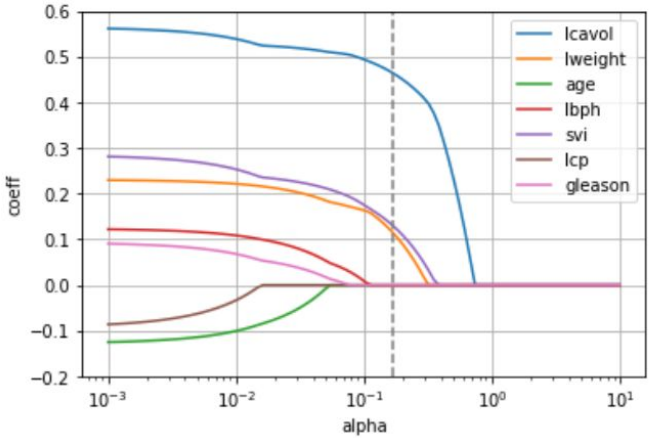
- Select α via cross-validation
- Then, find coefficients using all training data.
- Final coefficients are sparse:
  - Only three factors are non-zeros
  - Lcavol: log cancer volume
  - Lweight: Log weight
  - Svi: seminal vesicle invasion
- Use only features corresponding to non-zero coefficients for linear regression

```
model.alpha = alpha_opt
model.fit(X,y)

# Print the coefficients
for i, c in enumerate(model.coef_):
    print("%8s %f" % (names_x[i], c))
```

```
  lcavol 0.464526
 lweight 0.115832
     age 0.000000
    lbph 0.000000
     svi 0.131102
     lcp 0.000000
 gleason 0.000000
```

LASSO path

- Useful to plot coefficients as a function of α.
- Called the LASSO path
- Indicates relative importance of different factors
- For this data set:
  - lcavol most important
- Don't draw medical conclusions
  - Need more detailed significance testing
  - Complex subject for another class…



Finding the Final Regressor

- Select features from cross-validation
- Re-run ordinary (un-regularized) regression on reduced features
- Use K –fold validation
- K-folds yield K weights and biases
- Take mean of the weights and biases for the final parameter estimate
- Take mean of the test MSE for the estimate of the test MSE

Thank You!