Lecture 1: Machine Learning Basics

Siddharth Garg

sg175@nyu.edu

This Course…

Growing use of ML techniques in cyber-security application

Spam filtering

Biometrics

Malware detection

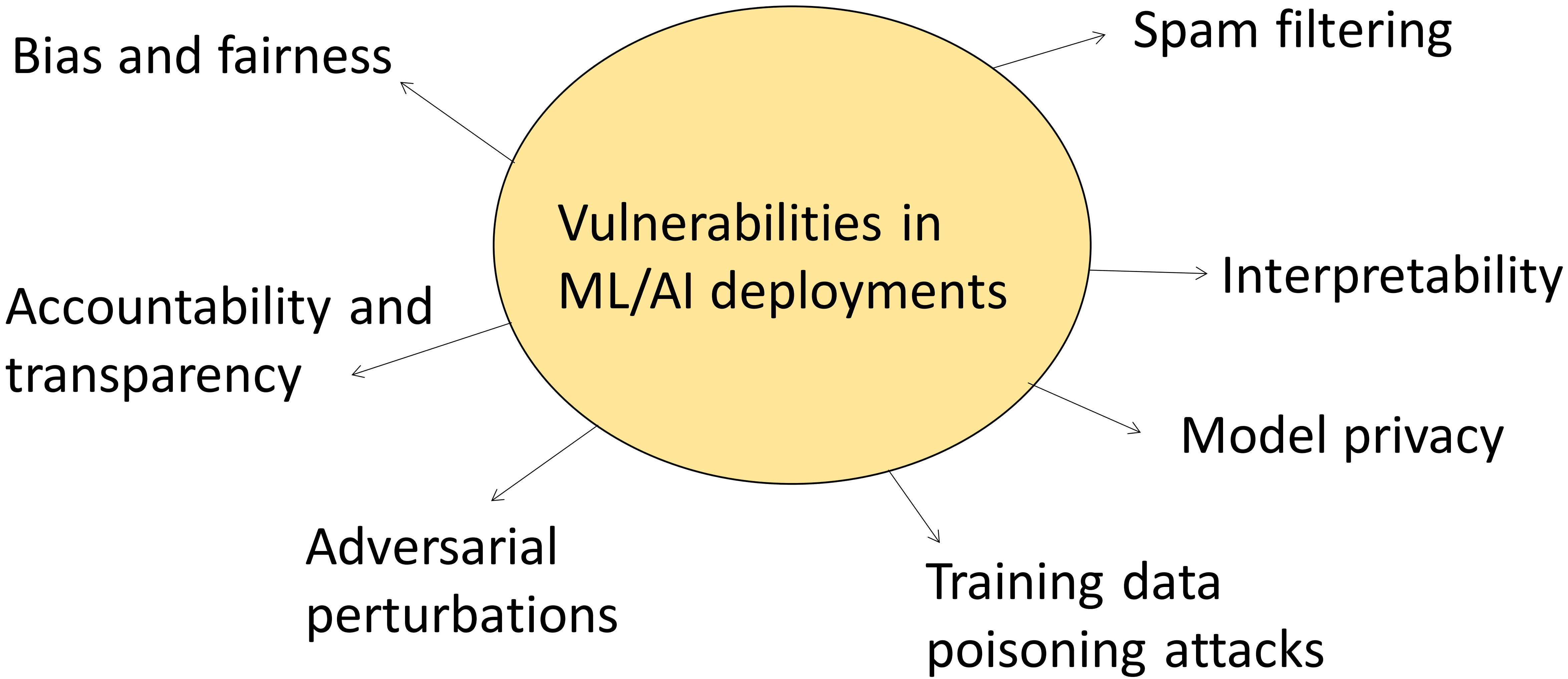
Network intrusion detection

Automated Evasion

Browser fingerprinting

Social network deanonymization

This Course…


Vulnerabilities in ML/AI deployments
Bias and fairness
Spam filtering
Interpretability
Accountability and transparency
Model privacy
Adversarial perturbations
Training data poisoning attacks

What is Machine Learning?

- Ability for machines to learn without being *explicitly* programmed

> "A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P* if its performance at tasks in *T*, as measured by *P*, improves with experience *E*." --- Mitchell, T. (1997). Machine Learning. McGraw Hill. p. 2.

- Why not use user knowledge, experience or expertise?
  - Are humans always able to explain their expertise?
  - Can machines *outperform* humans?

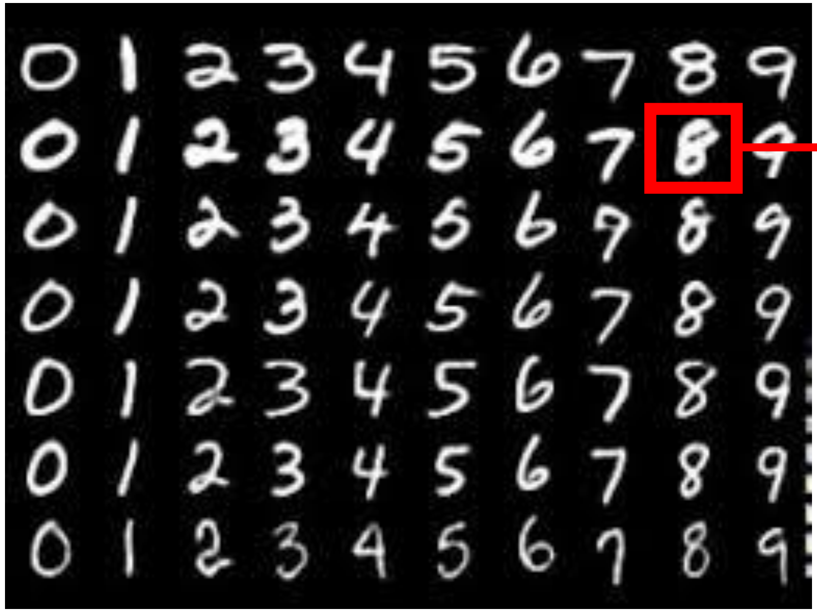
- What kinds of experiences (E), tasks (T) and performance measures (P)?

Example: MNIST Digit Recognition

Task (T):

- Given gray-scale images $x \in [0,255]^{28\times 28}$ and y$\in$ $[0,9]$ find a function

$$f : x \rightarrow y$$



8

https://www.npmjs.com/package/mnist

Experience (E):

- A ”training dataset” a set of correctly labeled images

Performance (P):

- Accuracy on a “test dataset”

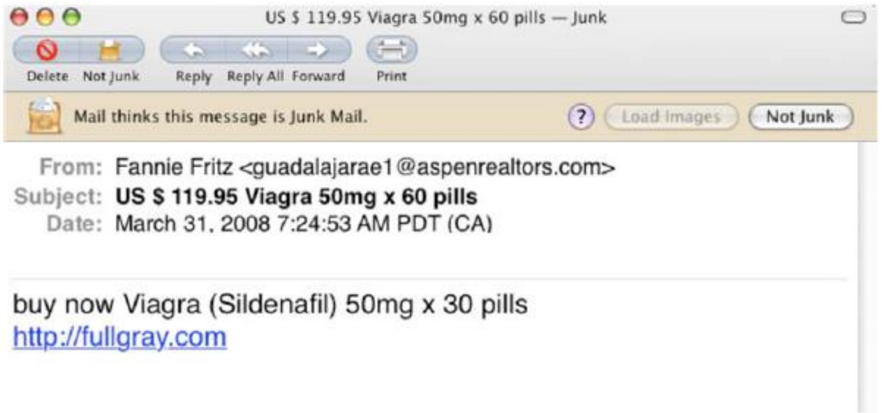
“*Supervised* Learning (Classification)”c

Example: Spam Classification

Task (T):

- Emails $x \in \text{all possible emails}$ and y $\in \{spam, non\_spam\}$ find

$$f : x \rightarrow y$$



SPAM

Experience (E):

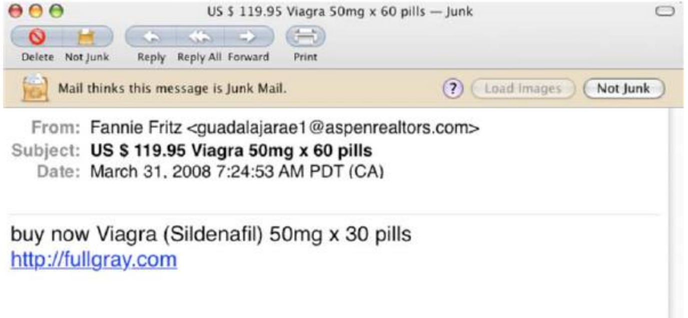
- A ”training dataset” a emails marked as “spam” or “non_spam”

Performance (P):

- Spam detection accuracy

“*Supervised* Learning (Classification)”

Some Challenges



Representing Data (or Feature Extraction)

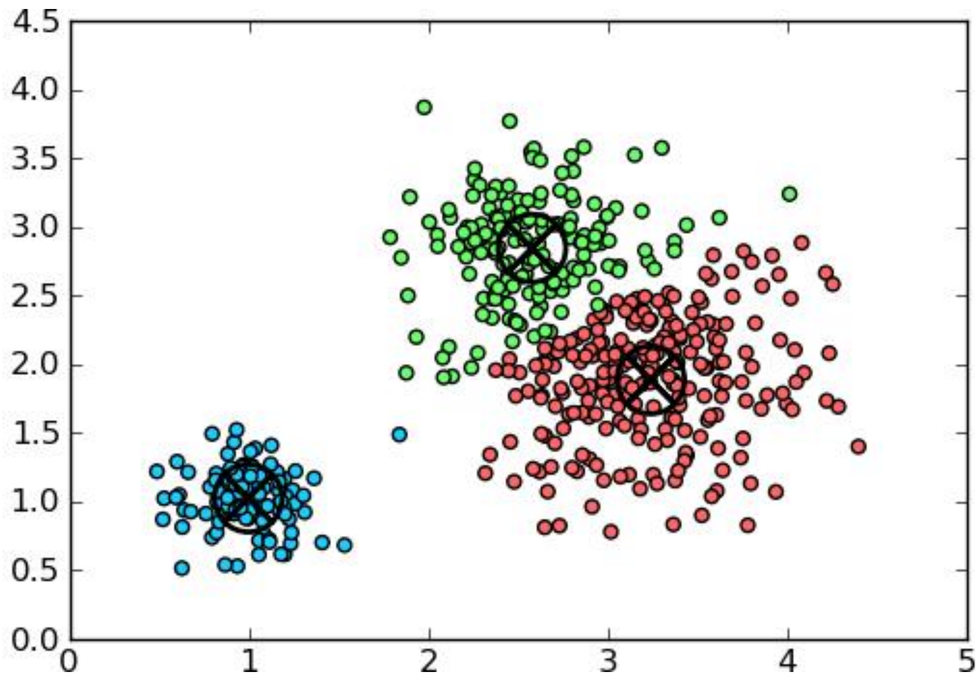
- How to represent $x \in \text{all possible emails}$ *mathematically*
- One example is "bag of words" representation: # times each word in the dictionary occurs
  - What do you lose?
  - What do you gain?
  - How can we compress this representation further?

What kind of classifier?

- What does the function *f* look like?
- And how do we learn it's parameters?

Example: Clustering

Task (T): “Cluster” a set of documents into k or groups such that “similar” documents appear in the same group



Experience (E):

- A ”training dataset” of documents without “labels”

Performance (P):

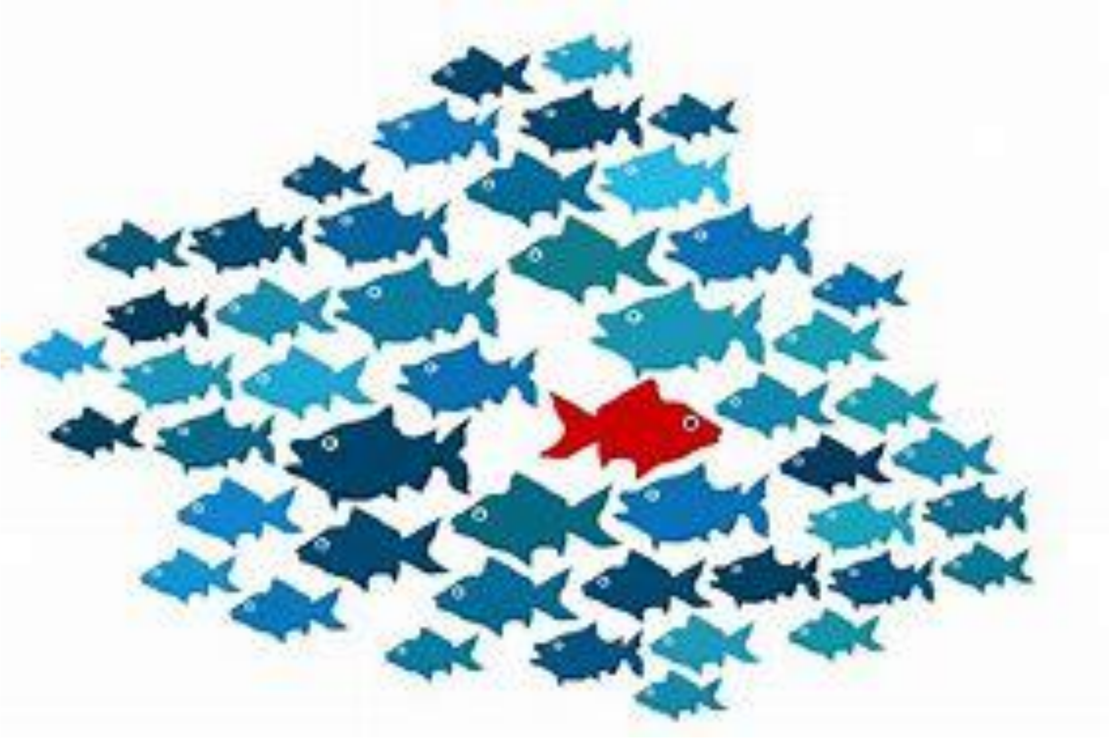
- Average distance to cluster center

“Un*supervised* Learning”

Example: Anomaly Detection

Task (T):

- Which of these is like the others?



Experience (E):

- Unlabeled samples

Performance (P):

- Anomaly detection accuracy

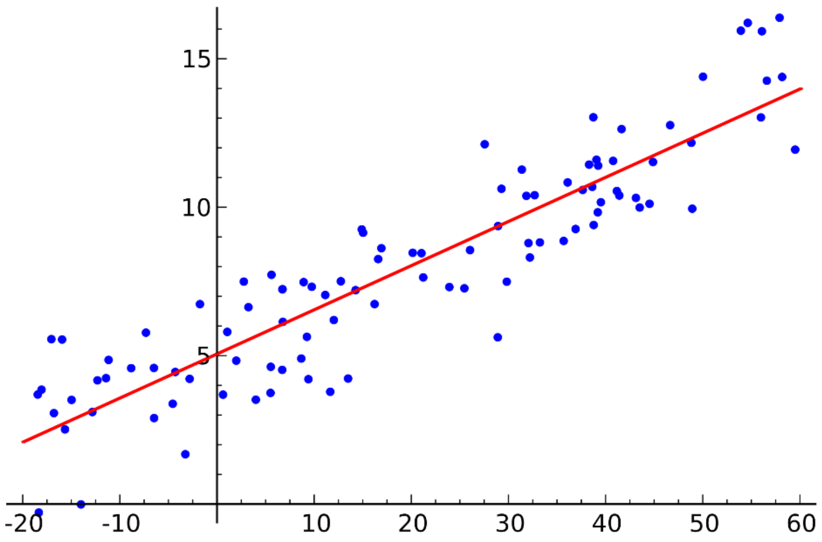
***"Unsupervised* Learning"**

Regression

Task (T):

- Given $x \in \mathbb{R}$ and y $\in \mathbb{R}$ find a ***linear*** function

$$\boxed{f: x \to y}$$



[S. Rangan, EL-GY-9123 Lec 2]

Experience (E):

- Training data: Points $(x_i, y_i), i \in [1, N]$

Performance (P):

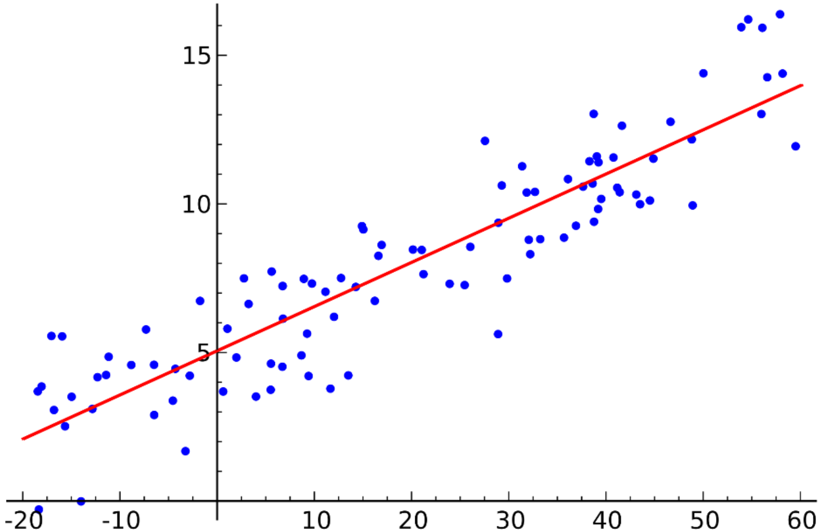
- Least squares fit: minimize mean square error between prediction and ground-truth

"*Supervised* Learning (Regression)"

Linear Least Squares Regression

$$y = f(x) = \beta_1 x + \beta_0$$

- How do we find the values (β_1, β_0)?



$$\min_{\beta_1,\beta_0} \sum_{i=1}^{N} (y_i - \widehat{y_i})^2$$

$$\widehat{y_i} = \beta_1 x_i + \beta_0 \quad \forall i \in [1, N]$$

Linear Least Squares Regression

$$y = f(x) = \beta_1 x + \beta_0$$

- How do we find the values (β_1, β_0)?

$$\min_{\beta_1,\beta_0} \sum_{i=1}^{N} (y_i - \widehat{y_i})^2$$

$$\widehat{y_i} = \beta_1 x_i + \beta_0 \quad \forall i \in [1, N]$$

$$g(\beta_1, \beta_0)$$

$$\min_{\beta_1,\beta_0} \sum_{i=1}^{N} (y_i - \beta_1 x_i - \beta_0)^2$$

$$\frac{\partial g}{\partial \beta_1} = 0 \qquad \frac{\partial g}{\partial \beta_0} = 0$$

Linear Least Squares Regression

$$y = f(x) = \beta_1 x + \beta_0$$

- How do we find the values (β_1, β_0)?

Residual Sum Squares (RSS) $g(\beta_1, \beta_0)$

$$\min_{\beta_1,\beta_0} \sum_{i=1}^{N} (y_i - \beta_1 x_i - \beta_0)^2$$

$$\frac{\partial g}{\partial \beta_1} = 0 \qquad \frac{\partial g}{\partial \beta_0} = 0$$

$$\frac{\partial g}{\partial \beta_0} = \sum_{i=1}^{N} -2(y_i - \beta_1 x_i - \beta_0) = 0$$

Sample mean

$$\beta_0 = \frac{\sum_{i=1}^{N}(y_i - \beta_1 x_i)}{N} = \bar{y} - \beta_1 \bar{x}$$

Are you surprised?

Linear Least Squares Regression

- How do we find the values (β_1, β_0)?

$$g(\beta_1, \beta_0)$$

$$\min_{\beta_1,\beta_0} \sum_{i=1}^{N} (y_i - \beta_1 x_i - \beta_0)^2$$

$$\frac{\partial g}{\partial \beta_1} = 0 \qquad \frac{\partial g}{\partial \beta_0} = 0$$

$$\frac{\partial g}{\partial \beta_1} = \sum_{i=1}^{N} -2x_i(y_i - \beta_1 x_i - \beta_0) = 0$$

$$\sum_{i=1}^{N} x_i((y_i - \bar{y}) - \beta_1(x_i - \bar{x})) = 0$$

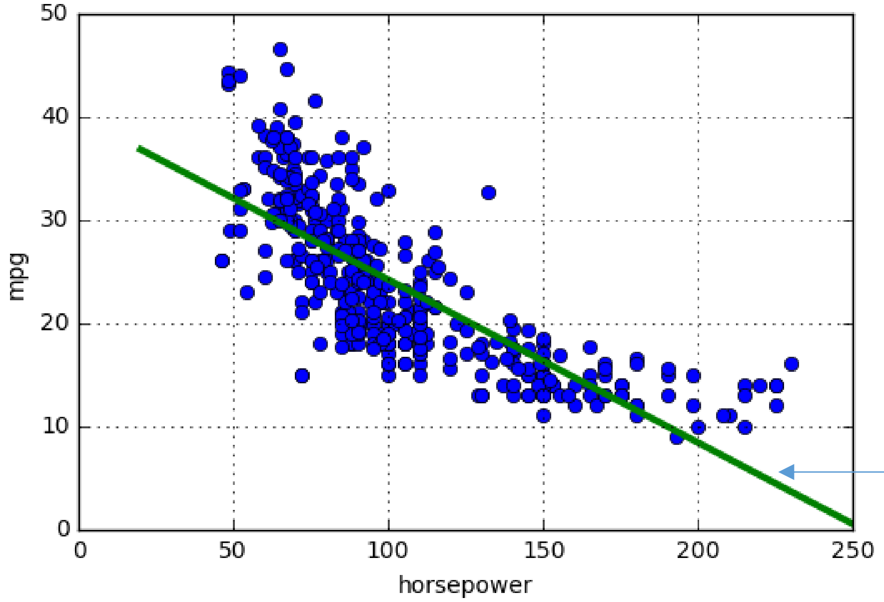
Sample covariance

$$\beta_1 = \frac{\overline{xy} - \bar{x}\bar{y}}{\overline{x^2} - \bar{x}^2}$$

Sample variance

Auto Example

- Python code




```
xm = np.mean(x)
ym = np.mean(y)
syy = np.mean((y-ym)**2)
syx = np.mean((y-ym)*(x-xm))
sxx = np.mean((x-xm)**2)
beta1 = syx/sxx
beta0 = ym - beta1*xm
```

```
beta0=  39.94, beta1=  -0.16
```

Regression line:

$$\text{mpg} = \beta_0 + \beta_1 \text{ horsepower}$$

Linear Least Squares (Multivariate)

- Now consider input: $x \in \Re^{\mathrm{M}}$ and output $y \in \Re$ the goal is to learn

$$y = f(x) = \beta_M x_M + \ldots + \beta_1 x_1 + \beta_0$$

- Given training dataset $X \in \Re^{\mathrm{N \times M}}$ and $Y \in \Re^{N}$

Training sample ➡

$$\begin{pmatrix} 1 & x_{01} & x_{02} \ldots & x_{0M} \\ 1 & x_{11} & x_{12} \ldots & x_{1M} \\ \\ 1 & x_{N1} & x_{N2} \ldots & x_{NM} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \\ \beta_M \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \\ y_N \end{pmatrix}$$

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

Note: for simplicity we will assume that X includes a column of 1s

Linear Least Squares (Multivariate)

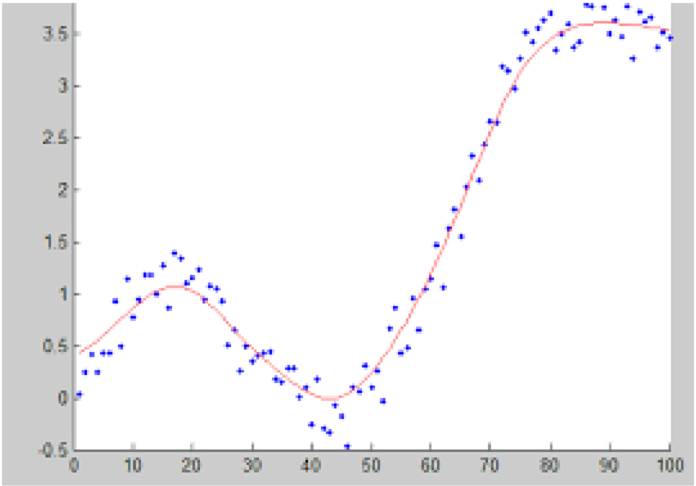
$$RSS = \sum (y - \hat{y})^2 = (Y - \hat{Y})^T \times (Y - \hat{Y}) = (Y - X\beta)^T \times (Y - X\beta)$$

Objective: $\boxed{\min_{\beta} (Y - X\beta)^T \times (Y - X\beta)}$

Solution: $\boxed{\beta^* = (X^T X)^{-1} X^T Y}$

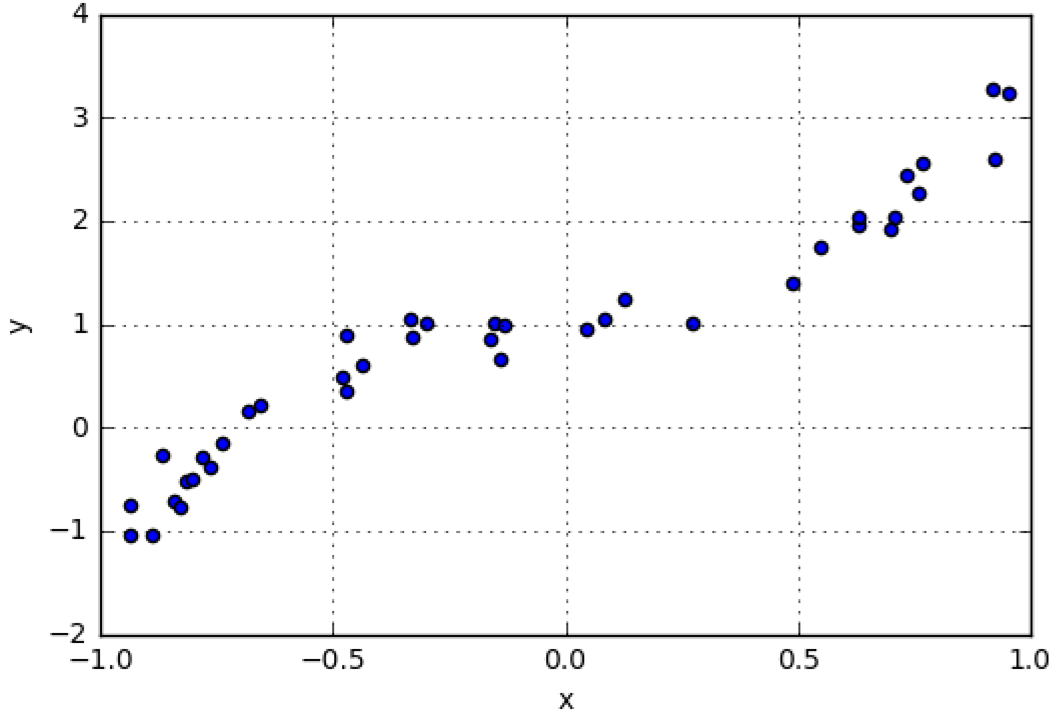
Polynomial Fitting

- Last lecture: polynomial regression
- Given data $(x_i, y_i), i = 1, \dots, N$
- 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.



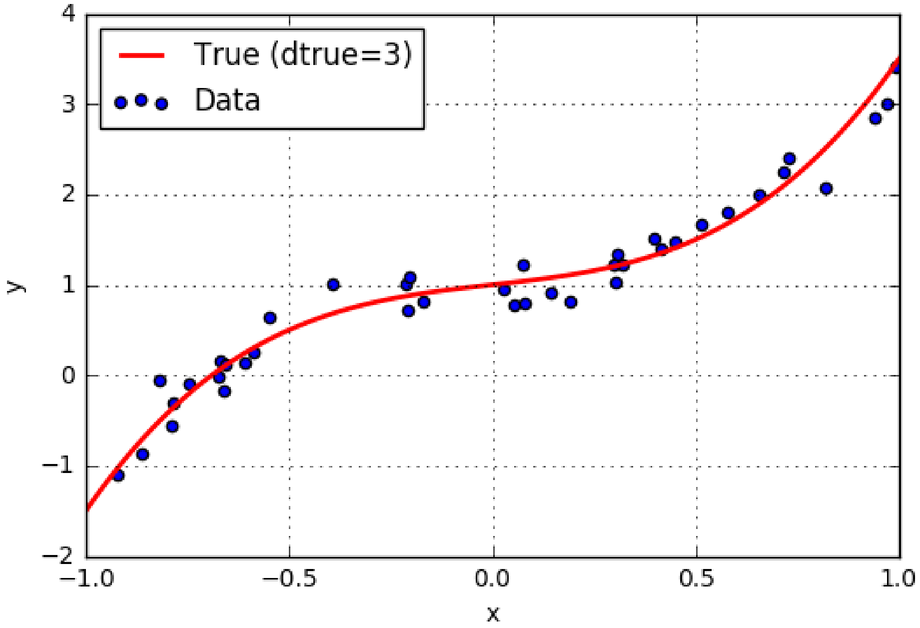
Example Question

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



Synthetic Data

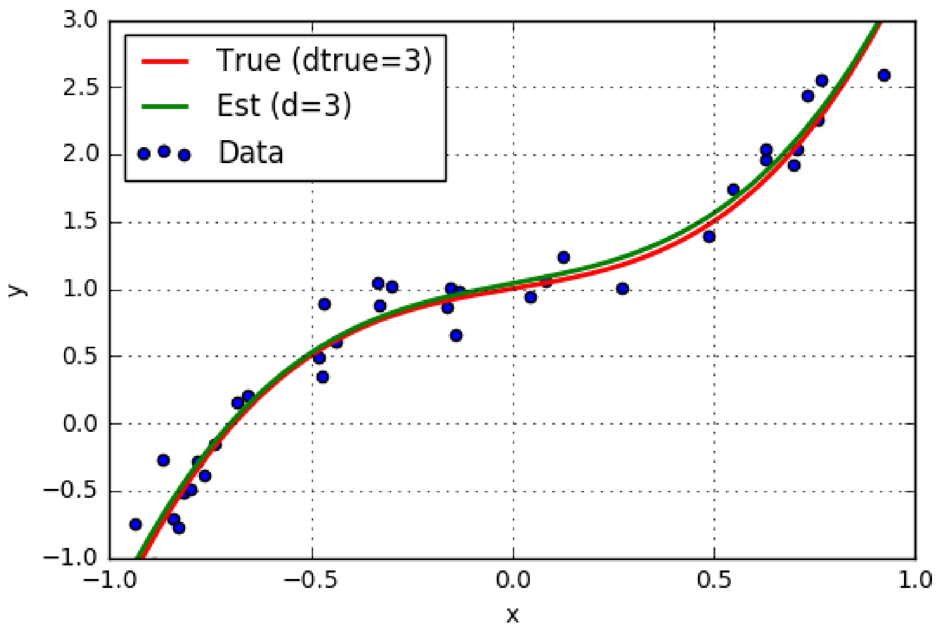
- Previous example is synthetic data
  - x_i: 40 samples uniform in [-1,1]
  - $y = f(x) + \epsilon,$
    - $f(x) = \beta_0 + \beta_1 x + \cdots + \beta_d x^d$ = “true relation”
    - $d = 3,\ \epsilon \sim N(0, \sigma^2)$
- Synthetic data useful for analysis
  - Know “ground truth”
  - Can measure performance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




```python
# Import useful polynomial library
import numpy.polynomial.polynomial as poly

# True model parameters
beta = np.array([1,0.5,0,2])   # coefficients
wstd = 0.2                     # noise
dtrue = len(beta)-1            # true poly degree

# Independent data
nsamp = 40
xdat = np.random.uniform(-1,1,nsamp)

# Polynomial
y0 = poly.polyval(xdat,beta)
ydat = y0 + np.random.normal(0,wstd,nsamp)
```

Fitting with True Model Order

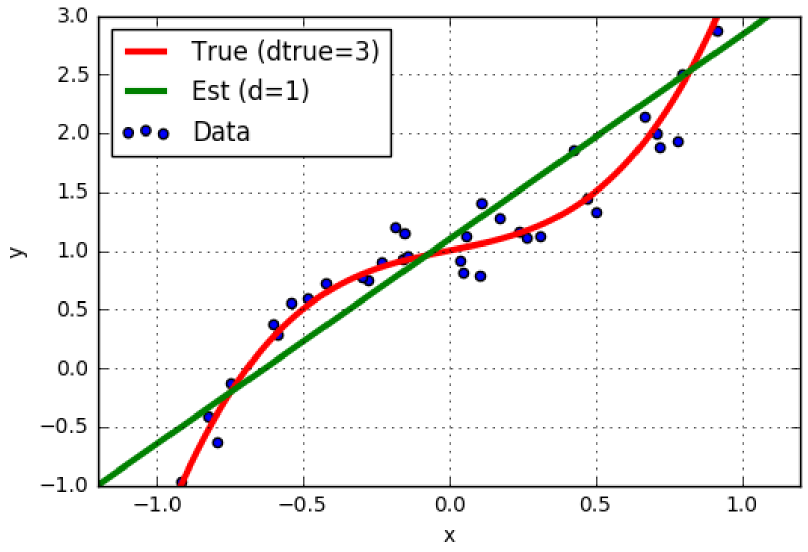
- Suppose true polynomial order, d=3, is known
- Use linear regression
  - numpy.polynomial package



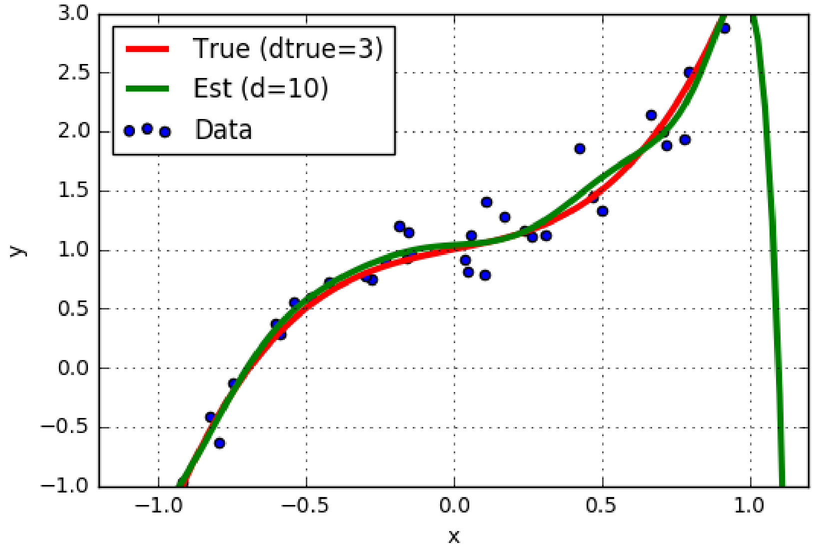

```
d = 3
beta_hat = poly.polyfit(xdat,ydat,d)

# 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?

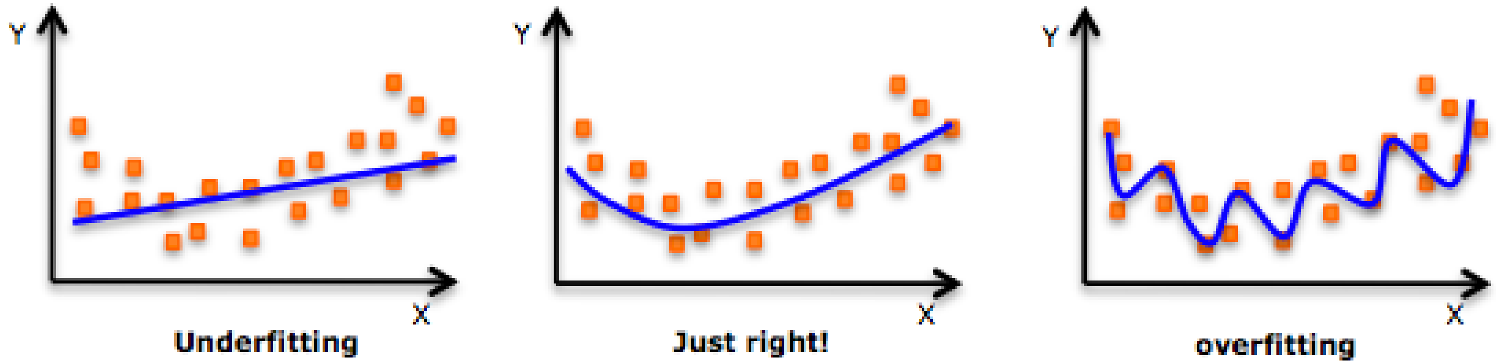


d=1 “Underfitting”



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 ?

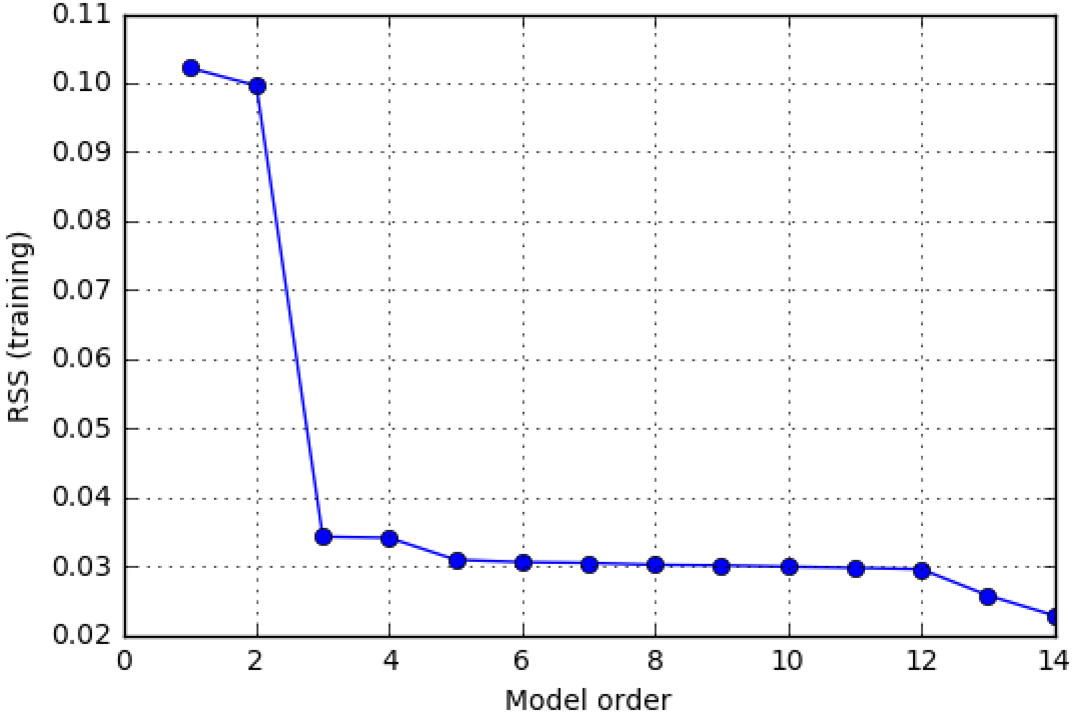
- Simple (but bad) idea:
  - For each model order, d, find estimate
  - Compute predicted values on training

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

  - 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 (Questio Why?)
  - Minimizing $RSS(d)$ will pick d as large possible
  - Leads to overfitting
- What went wrong?
- How do we do better?



Model Class and True Function

- Analysis set-up:
  - Learning algorithm assumes a model class: $\hat{y} = f(\boldsymbol{x}, \boldsymbol{\beta})$
  - But, data has true relation: $y = f_0(x) + \epsilon,\ \ \epsilon \sim N(0, \sigma_\epsilon^2)$

- Will quantify three key effects:
  - Irreducible error
  - Under-modeling
  - Over-fitting

Output Mean Squared Error

- To evaluate prediction error suppose we are given:
  - A parameter estimate $\widehat{\boldsymbol{\beta}}$ (computed from the learning algorithm)
  - A test point $\boldsymbol{x}_{test}$
  - Test point is generally different from training samples.
- Predicted value: $\hat{y} = f\left(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}\right)$
- Actual value: $y = f_0(\boldsymbol{x}_{test}) + \boldsymbol{\epsilon}$
- Output mean squared error:

$$MSE_y\left(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}\right) \coloneqq E[y - \hat{y}]^2$$

  - Expectation is over noise $\boldsymbol{\epsilon}$ on the test sample.

Irreducible Error

- Rewrite output MSE:

$$MSE_y(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}) \coloneqq E[y-\hat{y}]^2 = E\left[f_0(\boldsymbol{x}_{test}) + \epsilon - f(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}})\right]^2$$

- Since noise on test sample is independent of $\widehat{\boldsymbol{\beta}}$ and $\boldsymbol{x}_{test}$:

$$MSE_y(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}) \coloneqq \left[f_0(\boldsymbol{x}_{test}) - f(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}})\right]^2 + \mathrm{E}(\epsilon^2)$$
$$= \left[f_0(\boldsymbol{x}_{test}) - f(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}})\right]^2 + \sigma_\epsilon^2$$

- Define irreducible error: σ_ϵ^2
  - Lower bound on $MSE_y(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}) \geq \sigma_\epsilon^2$
  - Fundamental limit on ability to predict y
  - Occurs since y is influenced by other factors than $\boldsymbol{x}$

Under-Modeling

- Definition: A true function $f_0(\boldsymbol{x})$ is in the model class $\hat{y} = f(\boldsymbol{x}, \boldsymbol{\beta})$ if:

$$f_0(\boldsymbol{x}) = f(\boldsymbol{x}, \boldsymbol{\beta}_0) \text{ for all } \boldsymbol{x}$$

  for some parameter $\boldsymbol{\beta}_0$.
  - $\boldsymbol{\beta}_0$ called the true parameter

- Under-modeling: When $f_0(\boldsymbol{x})$ is not in the model class

Sample Question

- For each pair, state if the true function is in the model class or not
  - That is, is there under-modeling or not?
  - If true function is in the model class, state the true parameter

- Examples:
  - True function: $f_0(x) = 2 + 3x$ Model class: $f(x, \beta) = \beta_0 + \beta_1 x + \beta_2 x^2$
  - True function: $f_0(x) = 2 + 3x + 4x^2$ Model class: $f(x, \beta) = \beta_0 + \beta_1 x$
  - True function: $f_0(x) = \sin(2\pi(5)x + 7)$ Model class: $f(x, \beta) = \beta_0 \sin(2\pi(5)x) + \beta_1 \cos(2\pi(5)x)$
  - True function: $f_0(x) = \sin(2\pi(8)x + 7)$ Model class: $f(x, \beta) = \beta_0 \sin(2\pi(5)x) + \beta_1 \cos(2\pi(5)x)$

- Solutions in class

Analysis of Under-Modeling: Noise-Free Case

- Assume true relation has no noise: $y = f_0(\boldsymbol{x})$
  - Can model noise, but requires more probability theory
- Get training data: $(\boldsymbol{x}_i, y_i), i = 1, \dots, n$
- Fit model parameter from least-squares:

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} \left(y_i - f(\boldsymbol{x}_i, \boldsymbol{\beta})\right)^2$$
$$= \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} \left(f_0(\boldsymbol{x}_i) - f(\boldsymbol{x}_i, \boldsymbol{\beta})\right)^2$$

- Conclusions: With no noise
  - Fitting finds best least squares fit of the true functions in the model class
  - If there is a unique true parameter, then $\hat{\boldsymbol{\beta}}=\boldsymbol{\beta}_0$. Estimator identifies correct parameter

Bias: Noise-Free Case

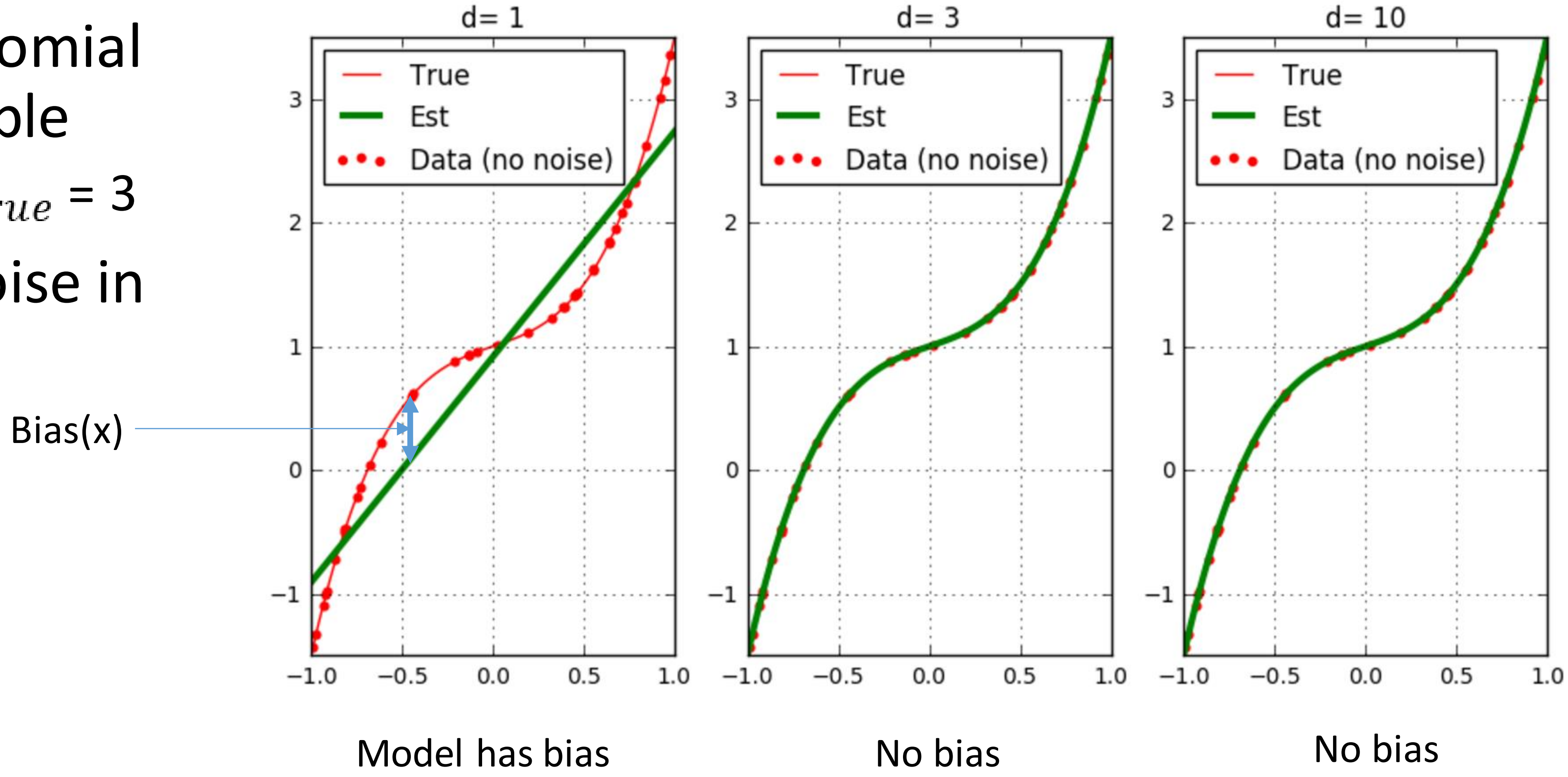
- Let $\boldsymbol{x}_{test}$ = some test point
  - Can be different from the training data set
- Definition: When there is no noise, the bias at a test point $\boldsymbol{x}_{test}$ is:

$$Bias(\boldsymbol{x}_{test}) := f_0(\boldsymbol{x}_{test}) - f\left(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}\right)$$

- Measures difference true and estimated relation in absence of noise
- Previous analysis shows:
  - Bias is small when true function is close to model class
  - When there is no under-modeling, $Bias(\boldsymbol{x}_{test}) = 0$ and true parameter found.

Bias Visualized

- Polynomial example
  - d_{true} = 3
- No noise in data



Analysis with Noise (Advanced)

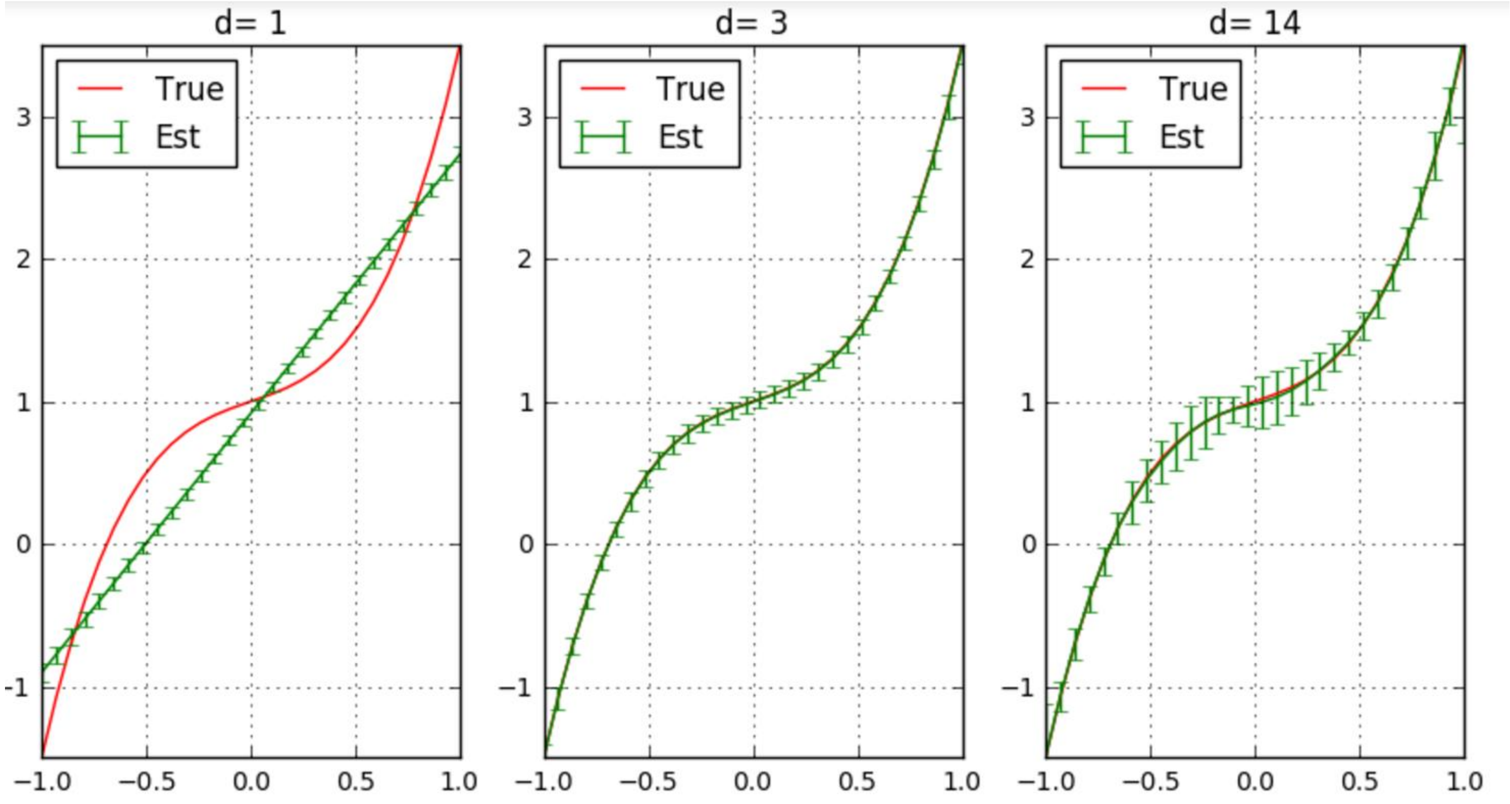
- Now assume noise: $y = f_0(\boldsymbol{x}) + \epsilon, \epsilon \sim N(0, \sigma_\epsilon^2)$
- Get training data: $(\boldsymbol{x}_i, y_i), i = 1, \dots, n$
- Fit a parameter:

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} \left(y_i - f(\boldsymbol{x}_i, \boldsymbol{\beta})\right)^2$$

  - $\widehat{\boldsymbol{\beta}}$ will be random.
  - Depends on particular noise realization.
- Take a new test point $\boldsymbol{x}_{test}$ (not random)
- Compute mean and variance of estimated function $f\left(\boldsymbol{x}_{test}, \widehat{\boldsymbol{\beta}}\right)$
- Define:
  - Bias: Difference of true function from mean estimate
  - Variance: Variance of estimate around its mean

Bias and Variance Illustrated

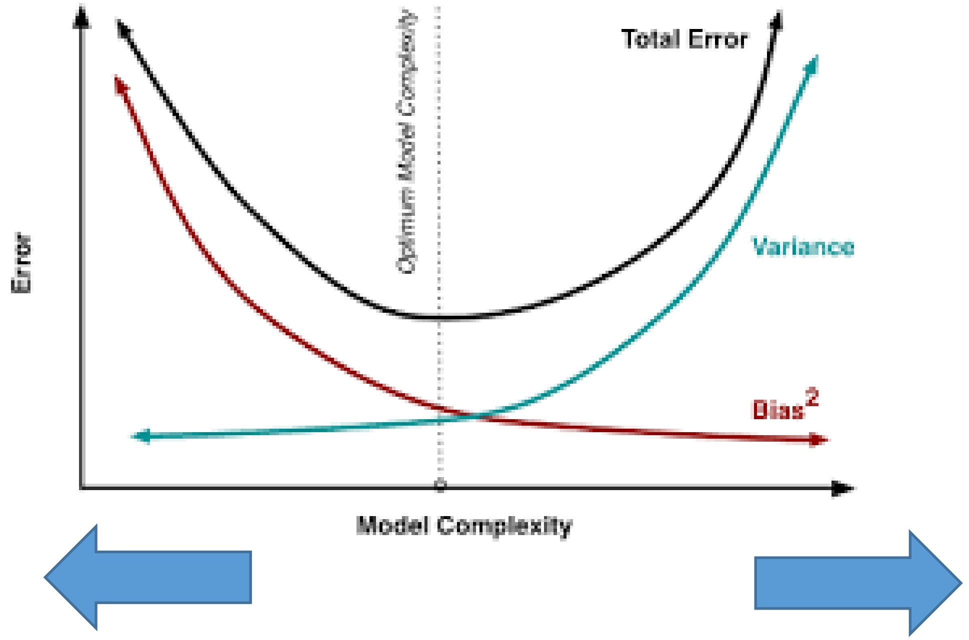
- Polynomial ex
- Mean and std dev of estimated functions
- 100 trials



Low variance,
High bias

High variance,
Zero bias

Bias-Variance Tradeoff



Simpler models
Less parameters
Under-fitting

Richer models
More parameters
Over-fitting

- Optimal model order depends on:
  - Amount of samples available
  - Underlying complexity of the relation

Cross Validation

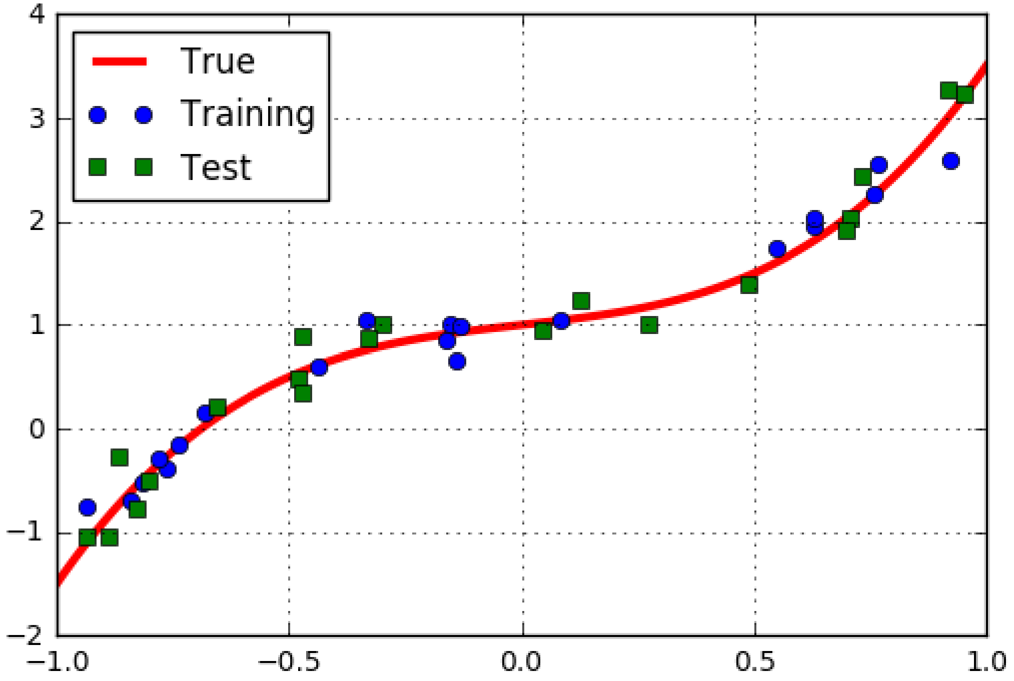
- Concept: Need to test fit on data independent of training data
- Divide data into two sets:
  - N_{train} training samples, N_{test} test samples
- For each model order, p, learn parameters $\hat{\beta}$ from training samples
- Measure RSS on test samples.

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

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

Polynomial Example: Training Test Split

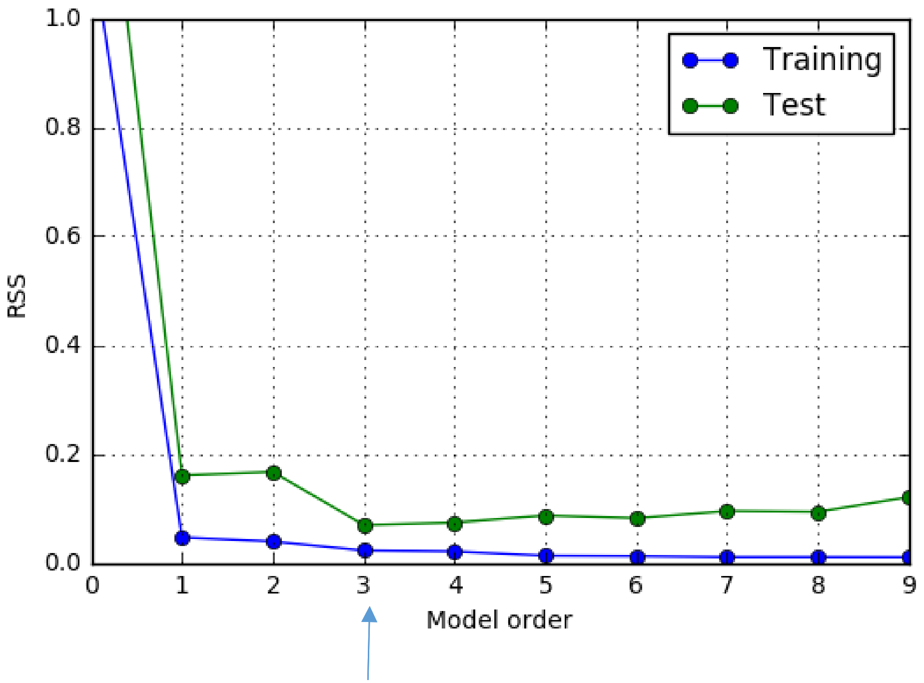
- Example: Split data into 20 samples for training, 20 for test




```
# Number of samples for training and test
ntr = nsamp // 2
nts = nsamp - ntr

# Training
xtr = xdat[:ntr]
ytr = ydat[:ntr]

# Test
xts = xdat[ntr:]
yts = ydat[ntr:]
```

Finding the Model Order

- Estimated optimal model order = 3



RSS test minimized at $d = 3$
RSS training always decreases

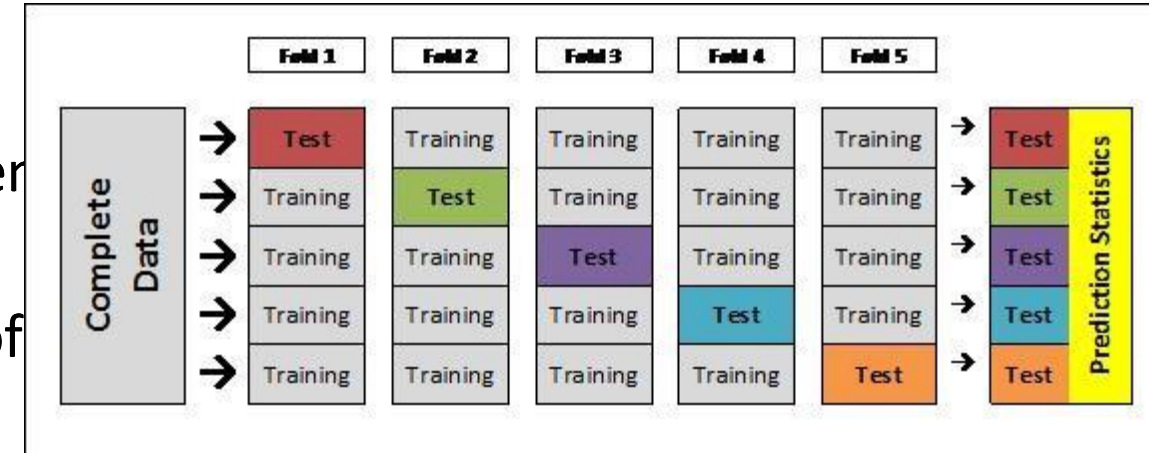
```
dtest = np.array(range(0,10))
RSStest = []
RSStr = []
for d in dtest:

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

    # Measure RSS on training data
    # This is not necessary, but we do it just to show the training error
    yhat = poly.polyval(xtr,beta_hat)
    RSSd = np.mean((yhat-ytr)**2)
    RSStr.append(RSSd)

    # Measure RSS on test data
    yhat = poly.polyval(xts,beta_hat)
    RSSd = np.mean((yhat-yts)**2)
    RSStest.append(RSSd)

plt.plot(dtest,RSStr,'bo-')
plt.plot(dtest,RSStest,'go-')
plt.xlabel('Model order')
plt.ylabel('RSS')
plt.grid()
plt.ylim(0,1)
plt.legend(['Training','Test'],loc='upper right')
```

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 rer
  - Average over the K test choices
  - More accurate, but requires K fits of



- Leave one out cross validation (LOOCV)
  - Take $K = N$ so one sample is left out.
  - Most accurate, but requires N model fittings

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

Polynomial Example

- Use sklearn Kfold object
- Loop
  - Outer loop: Over K folds
  - Inner loop: Over model order
  - Measure test error in each fold
  - Can be time-consuming

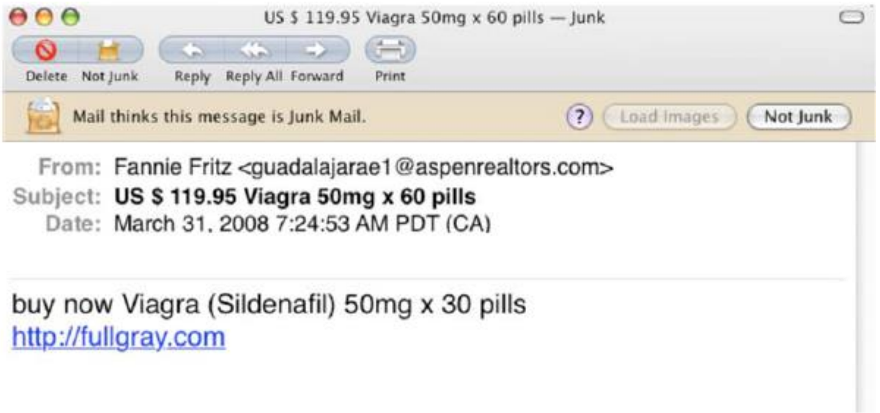
```python
# Create a k-fold object
nfold = 20
kf = sklearn.model_selection.KFold(n_splits=nfold,shuffle=True)

# 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)
```

Classification

Task (T):

- Emails $x \in \text{all possible emails}$ and y $\in \{spam, non\_spam\}$ find

$$f : x \rightarrow y$$



SPAM

Experience (E):

- A "training dataset" a emails marked as "spam" or "non_spam"

Performance (P):

- Spam detection accuracy

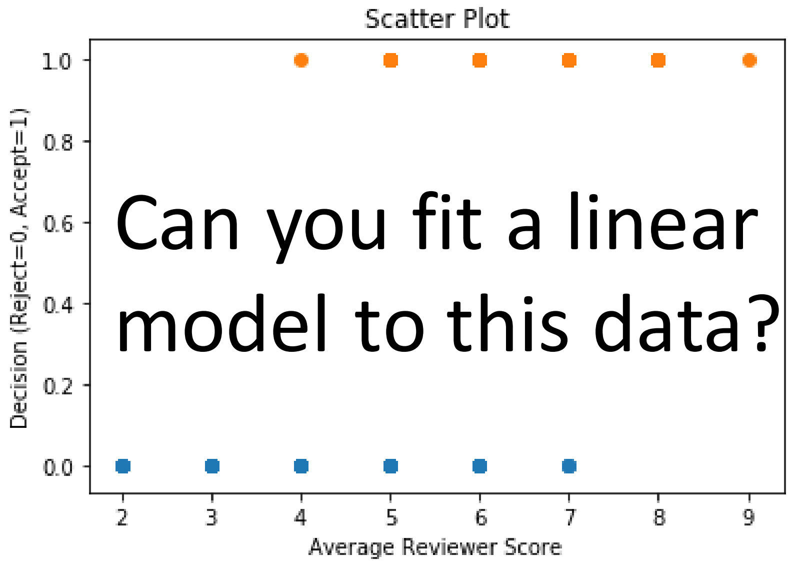
"*Supervised* Learning (Classification)"

Binary Classification

Binary Classification Task (T):

“Categorical variable”

- Simplest example where $x \in \Re$ and $y \in \{0,1\}$



Can you fit a linear model to this data?

- Dataset of ICLR’18 review scores vs. accept/reject decisions

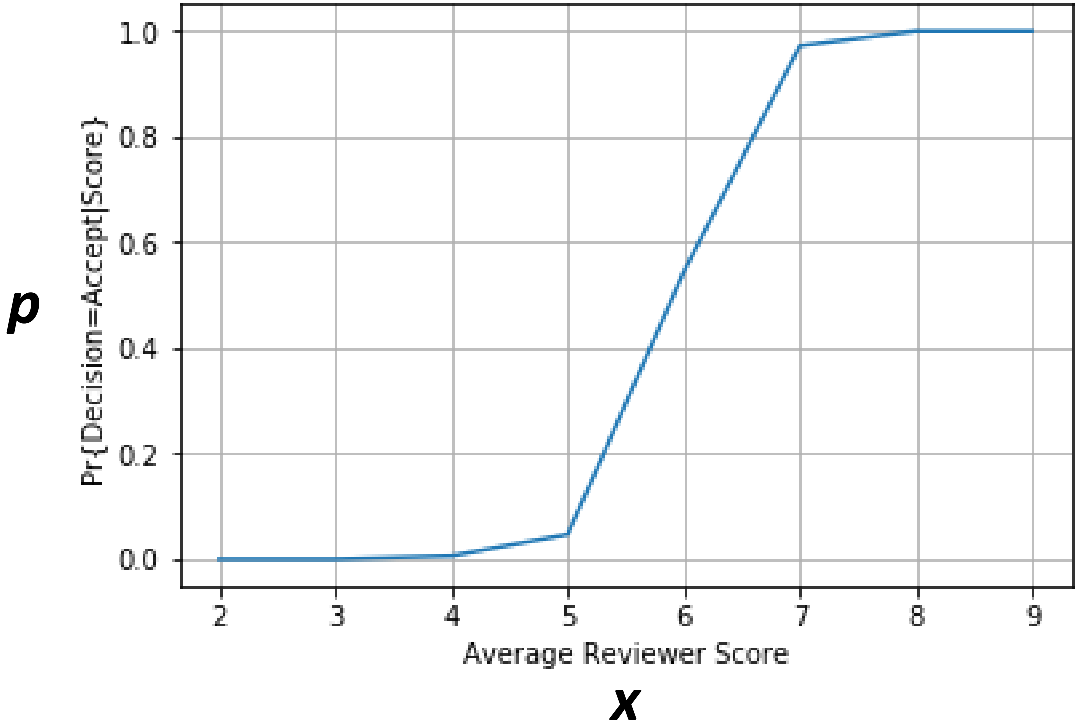
TL;DR	_bibtex	abstract	authorids	authors	conf_1	conf_2	conf_3	decision	review	review_1	review_2	review_3	title
None	@article{\nsharma2018hyperedge2vec:,\ntitle={H...	Data structured in form of overlapping or non-...	[sharm170@umn.edu, srjoty@ntu.edu.sg, himanshu...	[Ankit Sharma, Shafiq Joty, Himanshu Kharkwal,...	3.0	3.0	4.0	Reject	5.000000	5.0	5.0	5.0	Hyperedge2vec: Distributed Representations for...
Query-based black-box attacks on deep neural n...	@article{\nnitin2018exploring,\ntitle={Explori...	Existing black-box attacks on deep neural netw...	[abhagoji@princeton.edu, _w@eecs.berkeley.edu,...	[Arjun Nitin Bhagoji, Warren He, Bo Li, Dawn S...	4.0	3.0	4.0	Reject	6.000000	5.0	6.0	7.0	Exploring the Space of Black-box Attacks on De...
A theory and algorithmic framework for predict...	@article{\nd.2018learning,\ntitle={Learning We...	Predictive models that generalize well under d...	[fredrikj@mit.edu, kallus@cornell.edu, urish22...	[Fredrik D. Johansson, Nathan Kallus, Uri Shal...	3.0	3.0	4.0	Reject	6.666667	5.0	8.0	7.0	Learning Weighted Representations for Generali...
We prove that DNN is a recursively approximate...	@article{\nzheng2018understanding,\ntitle={Und...	Deep learning achieves remarkable generalizati...	[zhenggh@mail.ustc.edu.cn, jtsang@bjtu.edu.cn,...	[Guanhua Zheng, Jitao Sang, Changsheng Xu]	3.0	3.0	2.0	Reject	3.666667	2.0	3.0	6.0	Understanding Deep Learning Generalization by

Logistic Regression

Binary Classification Task (T):

Pr{Decision=Accept|Score}

- Instead, let's compute and plot $p = \Pr\{y = 1 | x\}$



- Idea: Linear regression to fit *p* as a function of *x*

$$p = \beta_1 x + \beta_0$$

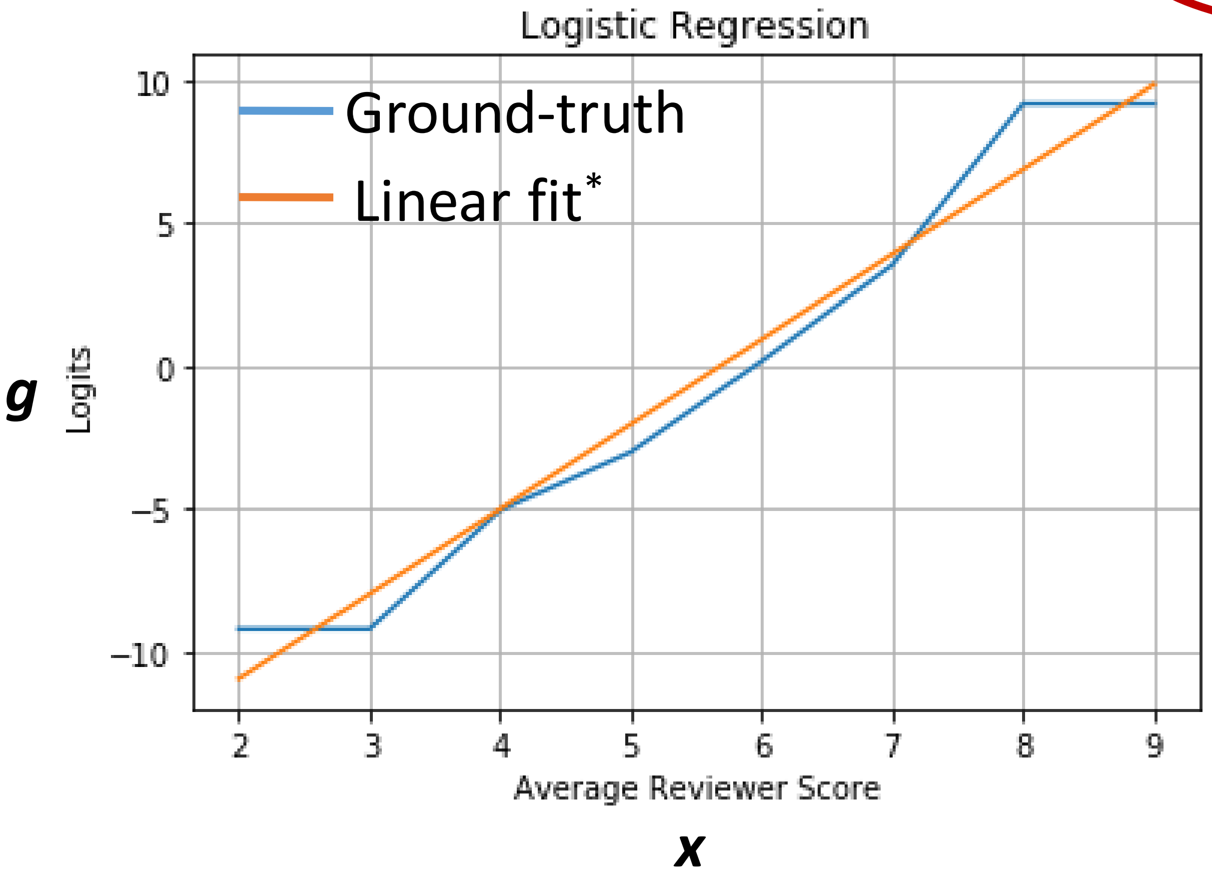
- Is this a good idea?
  - Probability *p* is always bounded between [0,1]

Logistic Regression

Binary Classification Task (T):

"Logits" Function

- Consider the following function: $g = \log(\frac{p}{1-p})$



- What is the range of g?

$$g \in [-\infty, \infty]$$

- Logistic Regression: fit logits function using a linear model!

$$g = \log(\frac{p}{1-p}) = \beta_1 x + \beta_0$$

Note: the linear fit is illustrative only. How to determine the best linear fit will be discussed next!

Logistic Regression

Pr{Decision=Accept|Score}

$$g = \log(\frac{p}{1-p}) = \beta_1 x + \beta_0 \Rightarrow p = \frac{1}{1+e^{-(\beta_1 x + \beta_0)}}$$

- What is Pr{Decision=Reject|Score}

$$1 - p = \frac{e^{-(\beta_1 x + \beta_0)}}{1+e^{-(\beta_1 x + \beta_0)}}$$

How do we find the model parameters β_1 and β_0?

Model Estimation

- We will use an approach referred to as Maximum Likelihood Estimation (MLE)
  - Let's assume that the model(i.e., β_1 and β_0) is magically known. Consider the training dataset below. What is the likelihood that the dataset came from our model?

#	X	Y
1	$x_1 = 3$	$y_1 = 0$
2	$x_2 = 8$	$y_2 = 1$
..		
..		
N	$x_N = 6$	$y_N = 1$

$$Likelihood = \frac{e^{-(\beta_1 x_1 + \beta_0)}}{1 + e^{-(\beta_1 x_1 + \beta_0)}} * \frac{1}{1 + e^{-(\beta_1 x_2 + \beta_0)}} * \ldots \frac{1}{1 + e^{-(\beta_1 x_N + \beta_0)}}$$

Model Estimation

- We will use an approach referred to as Maximum Likelihood Estimation (MLE)
  - Let's assume that the model(i.e., β_1 and β_0) is magically known. Consider the training dataset below. What is the likelihood that the dataset came from our model?

#	X	Y
1	$x_1 = 3$	$y_1 = 0$
2	$x_2 = 8$	$y_2 = 1$
..		
..		
N	$x_N = 6$	$y_N = 1$

$$Likelihood = \frac{e^{-(3\beta_1+\beta_0)}}{1+e^{-(3\beta_1+\beta_0)}} * \frac{1}{1+e^{-(8\beta_1+\beta_0)}} * \ldots \frac{1}{1+e^{-(6\beta_1+\beta_0)}}$$

Model Estimation

- We will use an approach referred to as Maximum Likelihood Estimation (MLE)
  - Let's assume that the model(i.e., β_1 and β_0) is magically known. Consider the training dataset below. What is the likelihood that the dataset came from our model?

#	X	Y
1	$x_1 = 3$	$y_1 = 0$
2	$x_2 = 8$	$y_2 = 1$
..		
..		
N	$x_N = 6$	$y_N = 1$

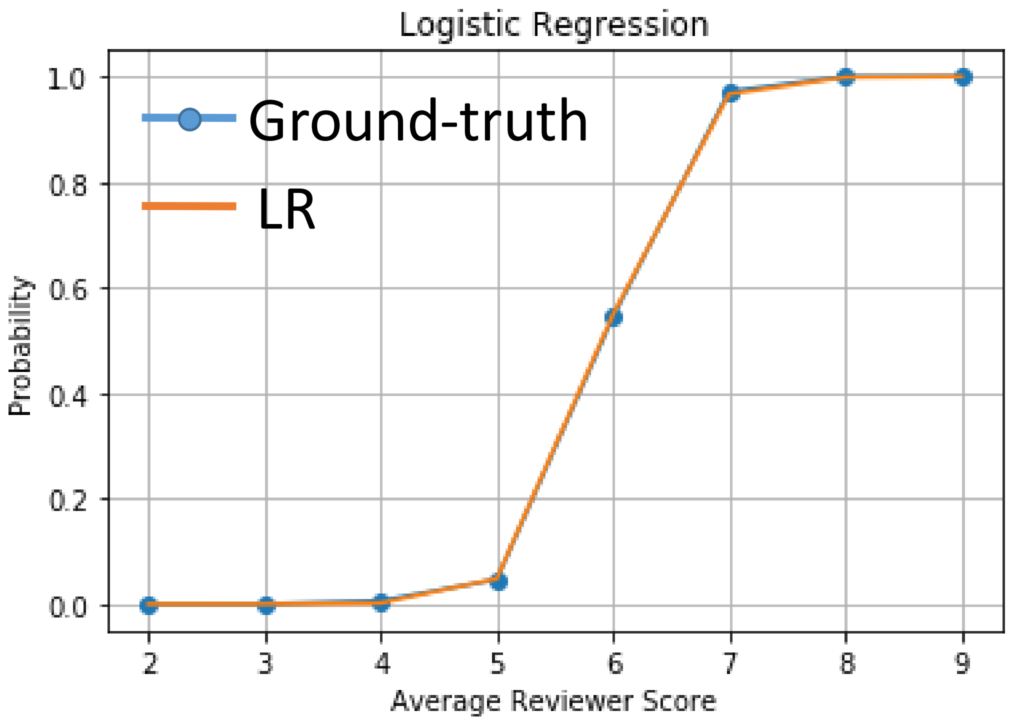
$$Log-Likelihood = \log\left(\frac{e^{-(3\beta_1+\beta_0)}}{1+e^{-(3\beta_1+\beta_0)}}\right) + \log\left(\frac{1}{1+e^{-(8\beta_1+\beta_0)}}\right) + \ldots \log\left(\frac{1}{1+e^{-(6\beta_1+\beta_0)}}\right)$$

$g(\beta_1, \beta_0)$ Function of model parameters only

$$\boxed{\text{Find } \beta_1 \text{ and } \beta_0 \text{ that maximize g (or minimize the “loss” } -\text{g)}}$$

$$Loss(\beta_1, \beta_0) = -g(\beta_1, \beta_0)$$

We Won't Worry About How (Phew!)




```python
from sklearn import linear_model

#Instantiate an LR object
logreg = sklearn.linear_model.LogisticRegression(C=1e5);

#Recall: your training data must have a column of ones for the constant term
xd = np.ones((numPapers,2));
xd[:,0] = np.append(rscores,ascores)

yd = np.append(rlabels,alabels);

logreg.fit(xd,yd);

#Plot Pr{Accept|Score}
rv = np.ones((len(revRange),2));
rv[:,0] = revRange;
prpredict=logreg.predict_proba(rv)
```

From regression to classification: if probability of Accept > 0.5, then output Accept.

Logistic Regression: Multi-Variate Case

UCI Spam Dataset:

https://archive.ics.uci.edu/ml/datasets/Spambase

Attribute Information:

The last column of 'spambase.data' denotes whether the e-mail was considered spam (1) or not (0), i.e. unsolicited commercial e-mail. Most of the attributes indicate whether a particular word or character was frequently occuring in the e-mail. The run-length attributes (55-57) measure the length of sequences of consecutive capital letters. For the statistical measures of each attribute, see the end of this file. Here are the definitions of the attributes:

48 continuous real [0,100] attributes of type word_freq_WORD
= percentage of words in the e-mail that match WORD, i.e. 100 * (number of times the WORD appears in the e-mail) / total number of words in e-mail. A "word" in this case is any string of alphanumeric characters bounded by non-alphanumeric characters or end-of-string.

6 continuous real [0,100] attributes of type char_freq_CHAR]
= percentage of characters in the e-mail that match CHAR, i.e. 100 * (number of CHAR occurences) / total characters in e-mail

1 continuous real [1,...] attribute of type capital_run_length_average
= average length of uninterrupted sequences of capital letters

1 continuous integer [1,...] attribute of type capital_run_length_longest
= length of longest uninterrupted sequence of capital letters

1 continuous integer [1,...] attribute of type capital_run_length_total
= sum of length of uninterrupted sequences of capital letters
= total number of capital letters in the e-mail

1 nominal {0,1} class attribute of type spam
= denotes whether the e-mail was considered spam (1) or not (0), i.e. unsolicited commercial e-mail.

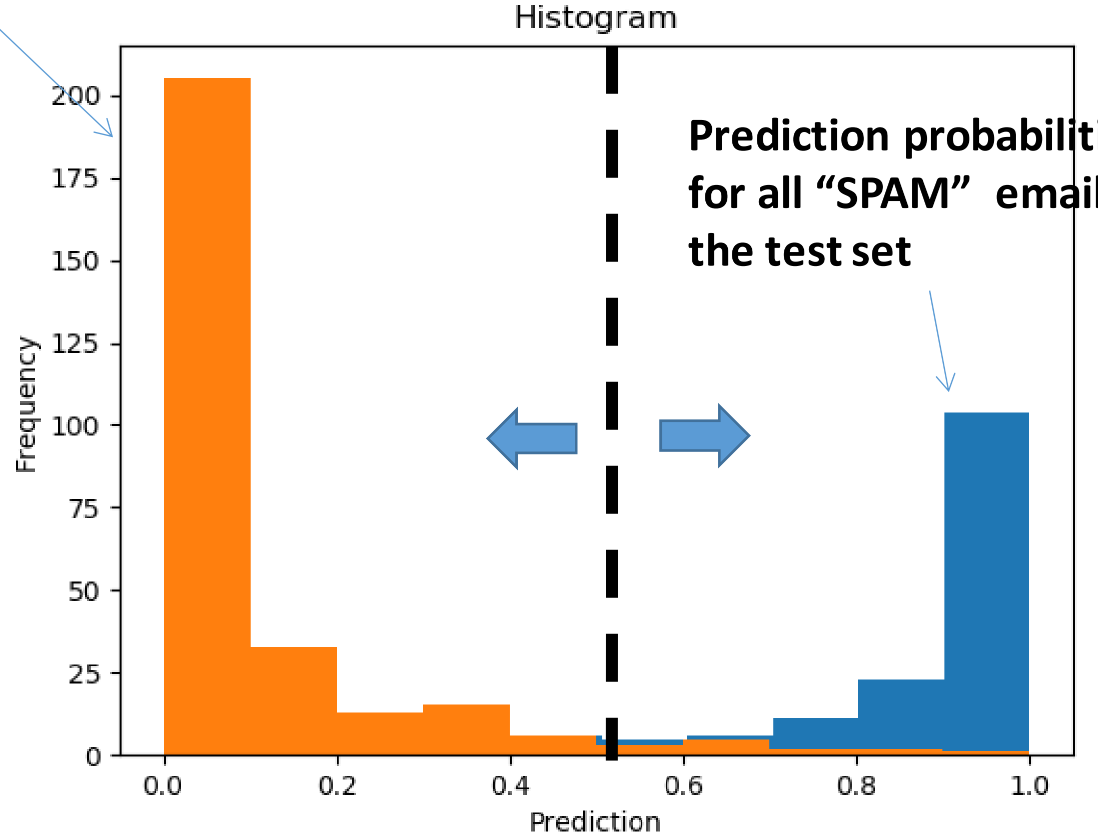
- **57 Real or integer valued features**
- **Binary output class**

$$p_{spam} = \frac{1}{1 + e^{-(\sum_{i=1}^{M} \beta_i x_i + \beta_0)}}$$

LR on Spam Database: Results

90% of samples used for training, remaining 10% used for test

Prediction probabilities for all "SPAM" emails in the test set



Prediction probabilities for all "SPAM" emails in the test set

```
#Instantiate an LR object
logreg = sklearn.linear_model.LogisticRegression(C=1e5);

#Recall: your training data must have a column of ones for the constant term
xd = np.ones((numPapers,2));
xd[:,0] = np.append(rscores,ascores)

yd = np.append(rlabels,alabels);

logreg.fit(xd,yd);

#Plot Pr{Accept|Score}
rv = np.ones((len(revRange),2));
rv[:,0] = revRange;
prpredict=logreg.predict_proba(rv)
```

Which emails are mis-predicted?

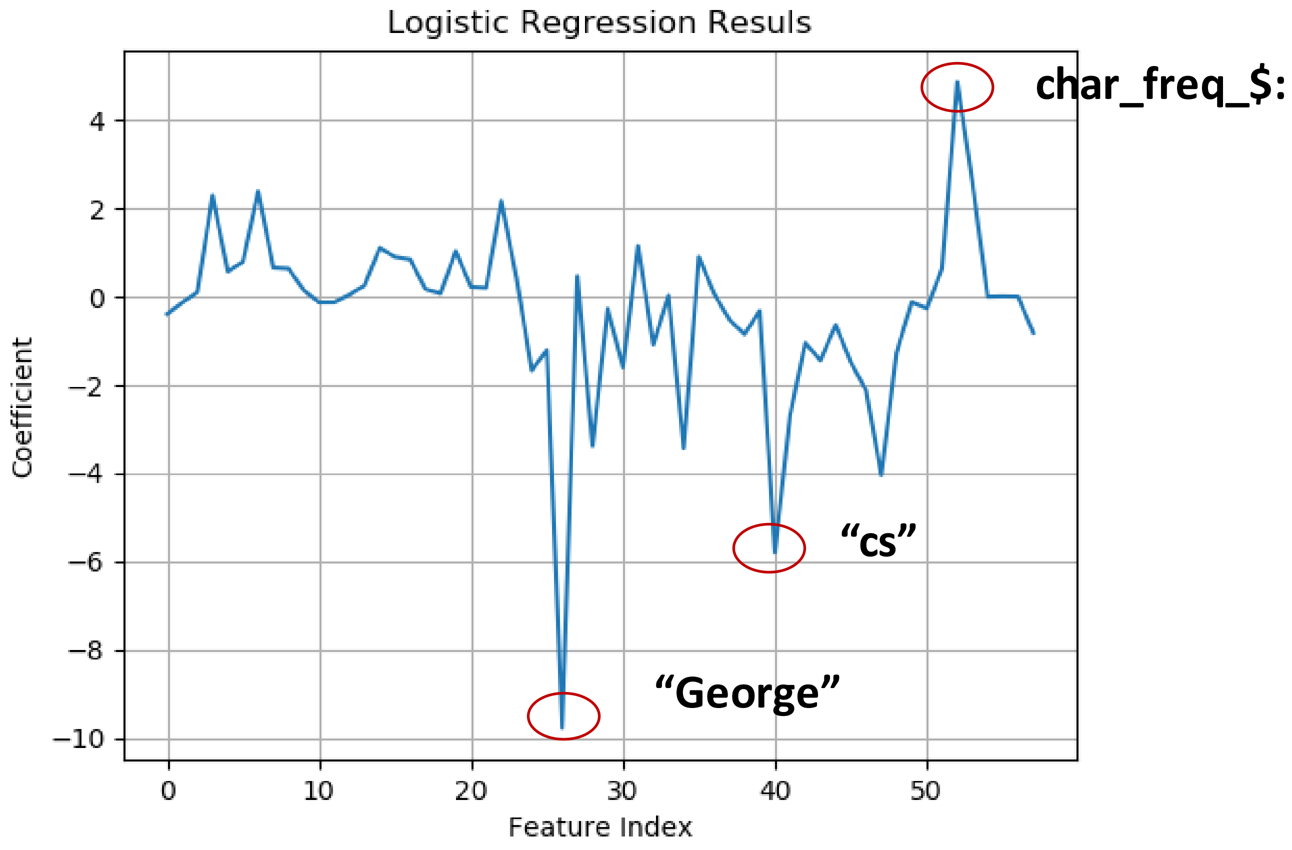
Accuracy on test set: ~92%

Which Features Matter?

Our Model:

$$p_{spam} = \frac{1}{1 + e^{-(\sum_{i=1}^{M} \beta_i x_i + \beta_0)}}$$

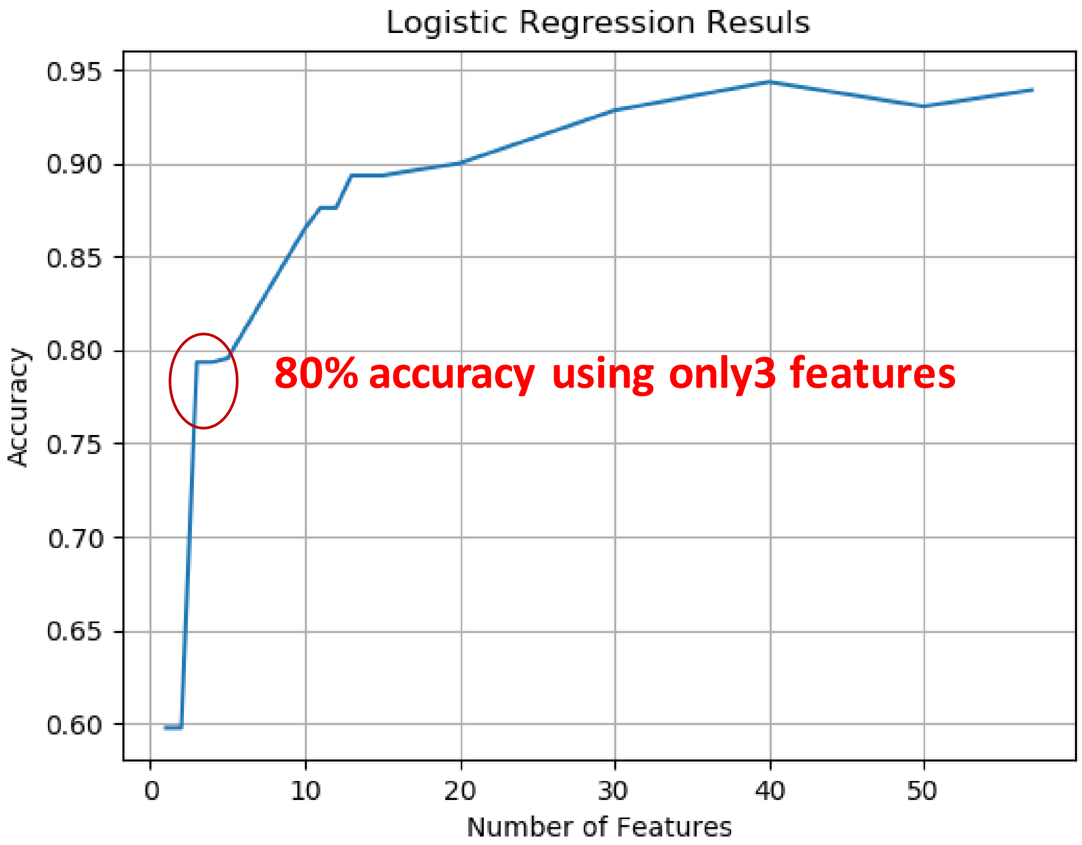
What does β_i=0 imply about feature *i*?



Reasonable hypothesis: features with larger absolute values of β matter more.

Feature Selection

Retrain and predict using only the top-k features



Can we explicitly train the parameters so as to prioritize a "sparser" model?

Why?

Low model complexity prevents overfitting!!

Recall that during training we were seeking to minimize:

$$\hat{\beta} = \min_{\beta} Loss(\beta)$$

How should this objective function change?

Regularization

L_p Norm of a vector *x* $\|x\|_p = (\sum |x_i|^p)^{1/p}$

p	L_p Norm	Interpretation
0	$\|x\|_0 = (\sum |x_i|^0)^{1/0}$	Number of Non-zero Entries
1	$\|x\|_1 = (\sum |x_i|)$	Sum of absolute values
2	$\|x\|_2 = (\sum |x_i|^2)^{0.5}$	Root mean square
∞	$\|x\|_\infty = (\sum |x_i|^\infty)^0$	Max. value

"Regularized" loss

$$\hat{\beta} = \min_{\beta}\{Loss(\beta) + c\|\beta\|_0\}$$

c controls the relative importance of the regularization penalty

Regularization In Practice

L0 Regularization

$$\hat{\beta} = \min_{\beta}\{Loss(\beta) + c\|\beta\|_0\}$$

Hard "combinatorial" optimization problem!

Instead, the following regularization functions are commonly used:

L1 Regularization (LASSO)

$$\hat{\beta} = \min_{\beta}\{Loss(\beta) + c\|\beta\|_1\}$$

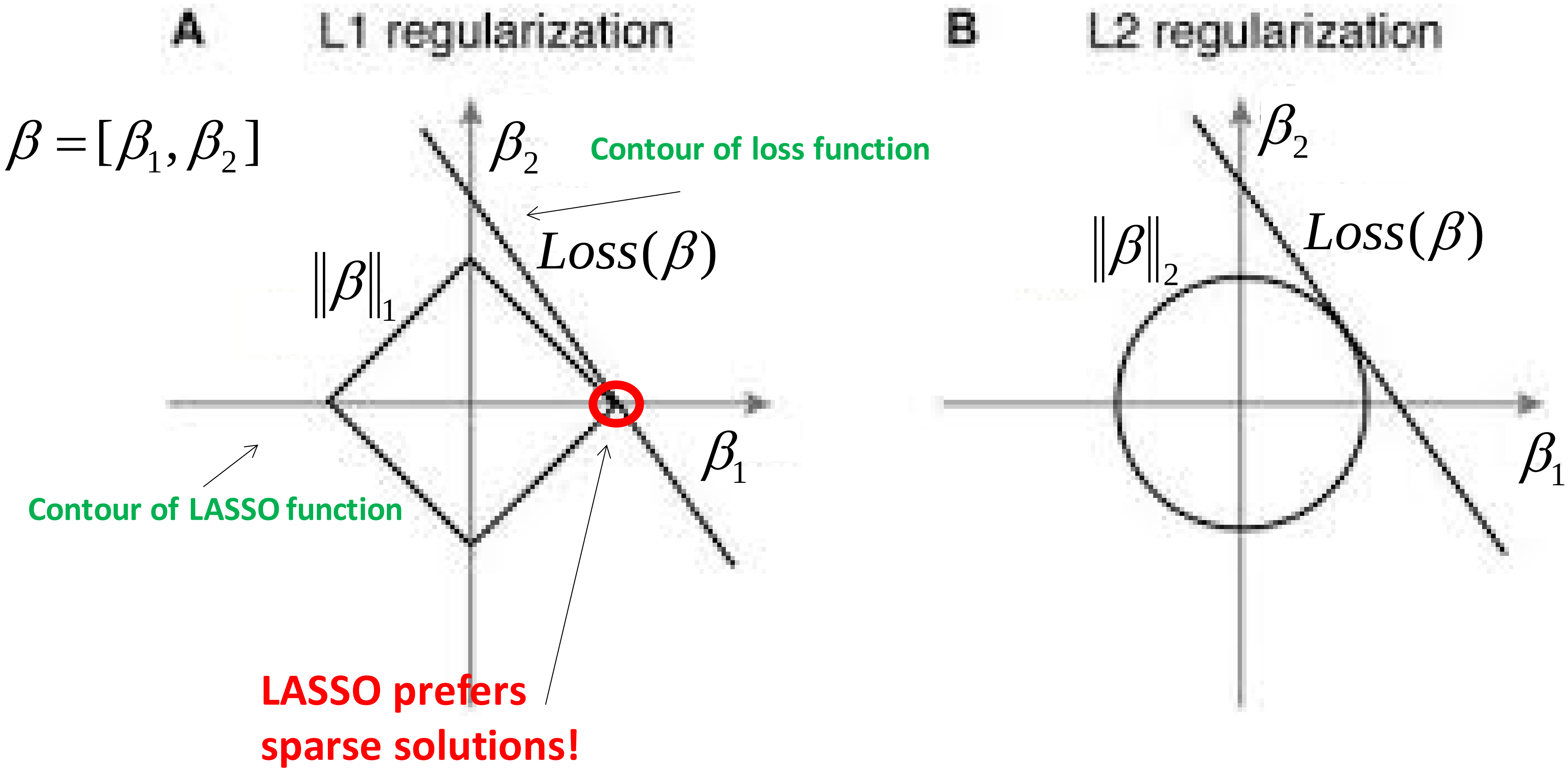
L2 Regularization (Ridge)

$$\hat{\beta} = \min_{\beta}\{Loss(\beta) + c\|\beta\|_2\}$$

We are penalizing "large" coefficients.

But why?

LASSO and Ridge Regularization



Regularization for Spam Classification

Logistic Regression (aka logit, MaxEnt) classifier.

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi_class' option is set to 'ovr', and uses the cross- entropy loss if the 'multi_class' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs', 'sag' and 'newton-cg' solvers.)

This class implements regularized logistic regression using the 'liblinear' library, 'newton-cg', 'sag' and 'lbfgs' solvers. It can handle both dense and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted (and copied).

The 'newton-cg', 'sag', and 'lbfgs' solvers support only L2 regularization with primal formulation. The 'liblinear' solver supports both L1 and L2 regularization, with a dual formulation only for the L2 penalty.

Read more in the User Guide.

Parameters: **penalty** : str, 'l1' or 'l2', default: 'l2'

Used to specify the norm used in the penalization. The 'newton-cg', 'sag' and 'lbfgs' solvers support only l2 penalties.

New in version 0.19: l1 penalty with SAGA solver (allowing 'multinomial' + L1)

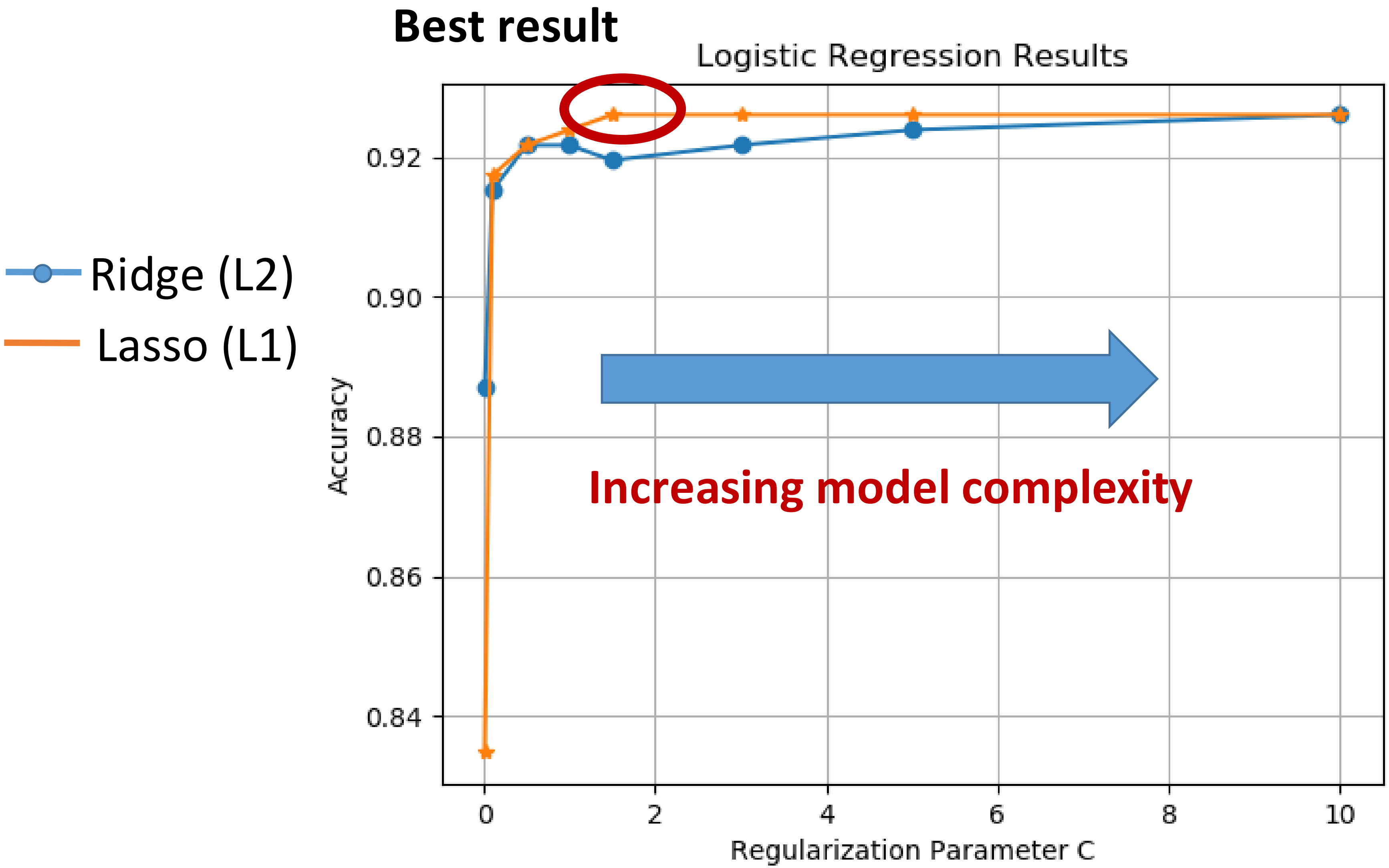
C : float, default: 1.0

Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

Which regularization function to use?

How should we select *c*?

Impact of C



Best result
Logistic Regression Results
Ridge (L2)
Lasso (L1)
Accuracy
0.92
0.90
0.88
0.86
0.84
0
2
4
6
8
10
Regularization Parameter C
Increasing model complexity

Errors in Binary Classification

- Two types of errors:
  - Type I error (False positive / false alarm): Decide $\hat{y} = 1$ when $y = 0$
  - Type II error (False negative / missed detection): Decide $\hat{y} = 0$ when $y = 1$
- Implication of these errors may be different
  - Think of breast cancer diagnosis
- Accuracy of classifier can be measured by:
  - $TPR = P(\hat{y} = 1 | y = 1)$
  - $FPR = P(\hat{y} = 1 | y = 0)$

predicted→ real↓	*Class_pos*	*Class_neg*
Class_pos	TP	FN
Class_neg	FP	TN

$$\text{TPR (sensitivity)} = \frac{\text{TP}}{\text{TP}+\text{FN}}$$

$$\text{FPR (1-specificity)} = \frac{\text{FP}}{\text{TN}+\text{FP}}$$

[Remaining Slides from Prof. Rangan's Intro to ML Class]

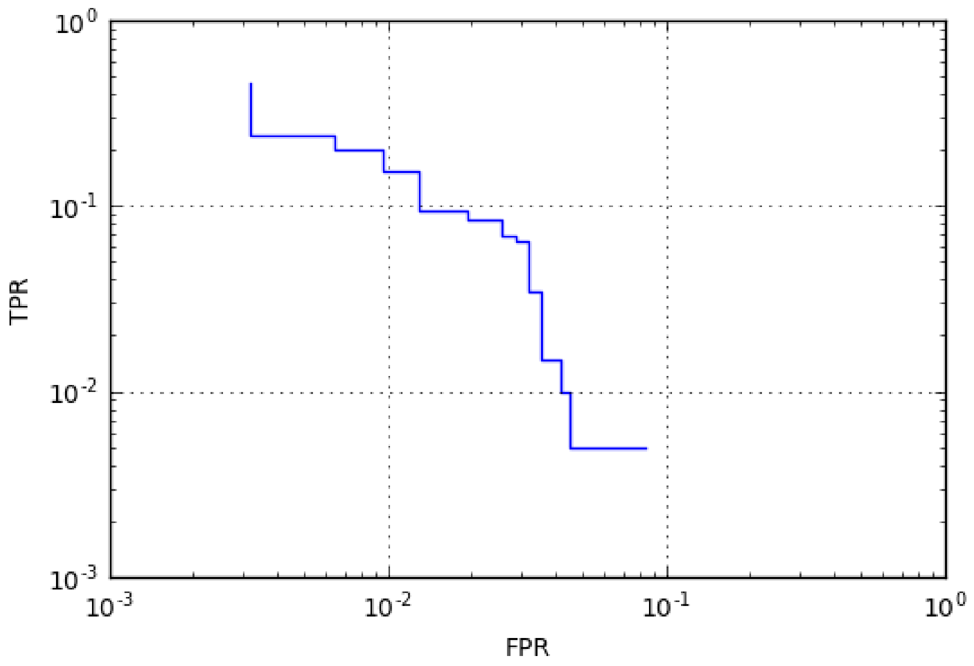
Hard Decisions

- Logistic classifier outputs a soft label: $P(y = 1|x) \in [0,1]$
  - $P(y = 1|x) \approx 1 \Rightarrow y = 1$ more likely
  - $P(y = 0|x) \approx 1 \Rightarrow y = 0$ more likely
- Can obtain a hard label by thresholding:
  - Set $\hat{y} = 1 \Leftrightarrow \quad P(y = 1|x) > t$
  - t = Threshold
- How to set threshold?
  - Set $t = \frac{1}{2} \Rightarrow$ Minimizes overall error rate
  - Increasing $t \Rightarrow$ Decreases false positives
  - Decreasing $t \Rightarrow$ Decreases missed detections

ROC Curve

- Varying threshold obtains a set of cla
- Trades off FPR and TPR
- Can visualize with ROC curve
  - Receiver operating curve
  - Term from digital communications

```
from sklearn import metrics
yprob = logreg.predict_log_proba(Xtr)
fpr, tpr, thresholds = metrics.roc_curve(ytr,yprob[:,1])

plt.loglog(fpr,1-tpr)
plt.grid()
plt.xlabel('FPR')
plt.ylabel('TPR')
```



Multi-Class Logistic Regression

- Suppose $y \in 1, \ldots, K$
  - K possible classes (e.g. digits, letters, spoken words, …)
- Multi-class regression:
  - $\boldsymbol{W} \in R^{K \times d}, \boldsymbol{w}_0 \in R^M$ Slope matrix and bias
  - $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{w}_0$: Creates M linear functions

- Then, class probabilities given by:

$$P(y = k|\boldsymbol{x}) = \frac{\mathrm{e}^{z_k}}{\sum_{\ell=1}^{K} e^{z_\ell}}$$

Softmax Operation

- Consider soft-max function:

$$g_k(\boldsymbol{z}) = \frac{\mathrm{e}^{z_k}}{\sum_{\ell=1}^{K} e^{z_\ell}}$$

  - K inputs $\mathbf{z} = (\mathrm{z}_1, \ldots, z_K)$, K outputs $f(\mathbf{z}) = (f(\mathbf{z})_1, \ldots, f(\mathbf{z})_K)$
- Properties: $f(\mathbf{z})$ is like a PMF on the labels $[0, 1, \ldots, K-1]$
  - $g_k(\mathbf{z}) \in [0,1]$ for each component k
  - $\sum_{k=1}^{K} g_k(\mathbf{z}) = 1$
- Softmax property: When $z_k \gg z_\ell$ for all $\ell \neq k$:
  - $g_k(\mathbf{z}) \approx 1$
  - $g_\ell(\mathbf{z}) \approx 0$ for all $\ell \neq k$
- Multi-class logistic regression: Assigns highest probability to class k when z_k is largest

$$z_k = \boldsymbol{w}_k^T \boldsymbol{x} + w_{0k}$$