DS 5220

Supervised Machine Learning and Learning Theory

Alina Oprea
Associate Professor, CCIS
Northeastern University

September 16 2019

Logistics

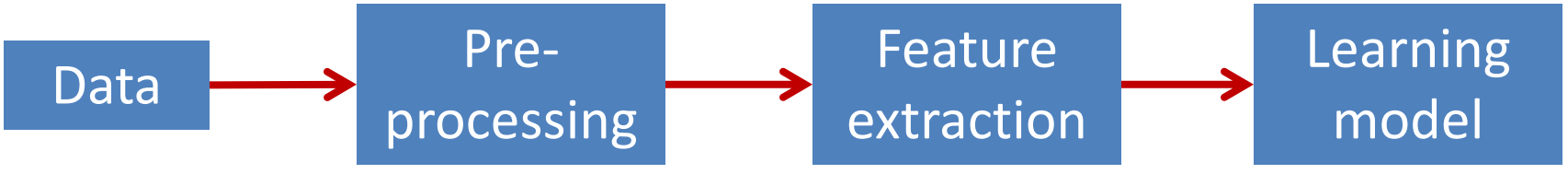
- HW1 is out, due date is Sept. 23 at midnight
- HW is posted on Piazza
- Submission on Gradescope – single PDF
  - Include response to math questions
  - Include results for coding questions
  - Submit zip file for code in Google form
- New resources posted on Piazza
- Lecture on Wed, Sept. 18 taught by Virgil Pavlu

Outline

- Multiple linear regression
  - Optimal closed-form solution
- Lab linear regression
- Gradient descent
  - Batch algorithm
  - Algorithm for linear regression
  - Line search

Supervised Learning: Regression

Training



Labeled

$x_i, y_i \in R$

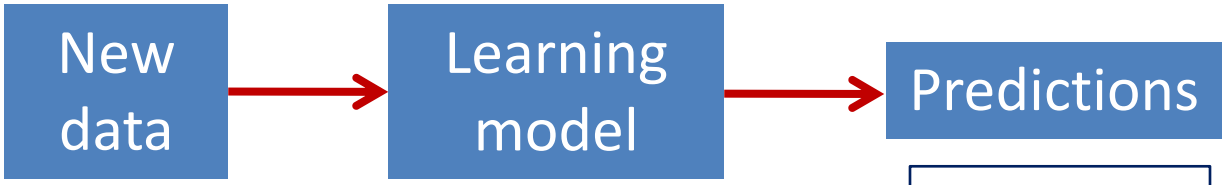
Normalization

Feature Selection

Regression

$f(x)$

Testing



$y' = f(x') \in R$

Unlabeled

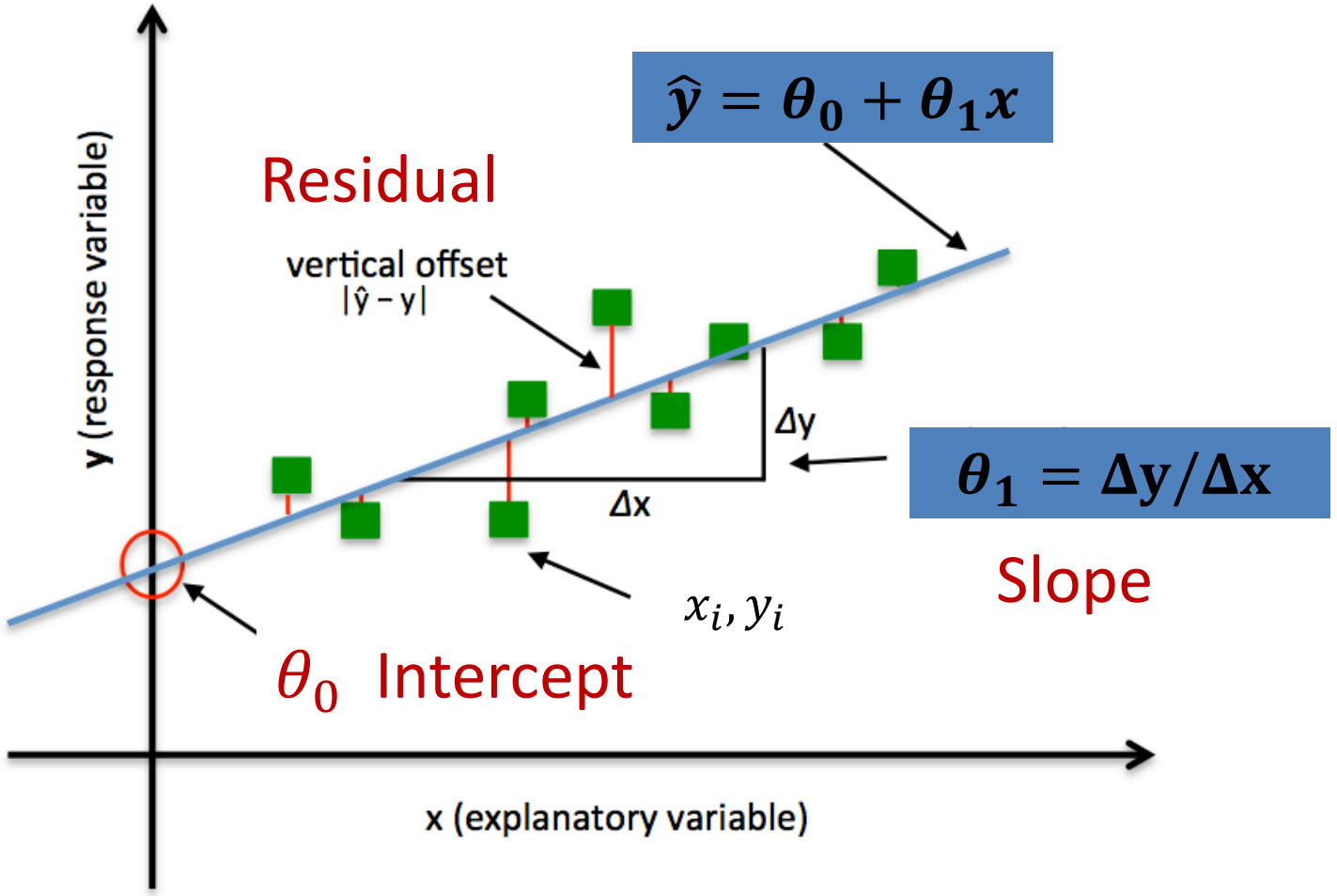
x'

$f(x)$

Response variable

Regression

Interpretation



$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}[h_\theta(x_i) - y_i]^2$$

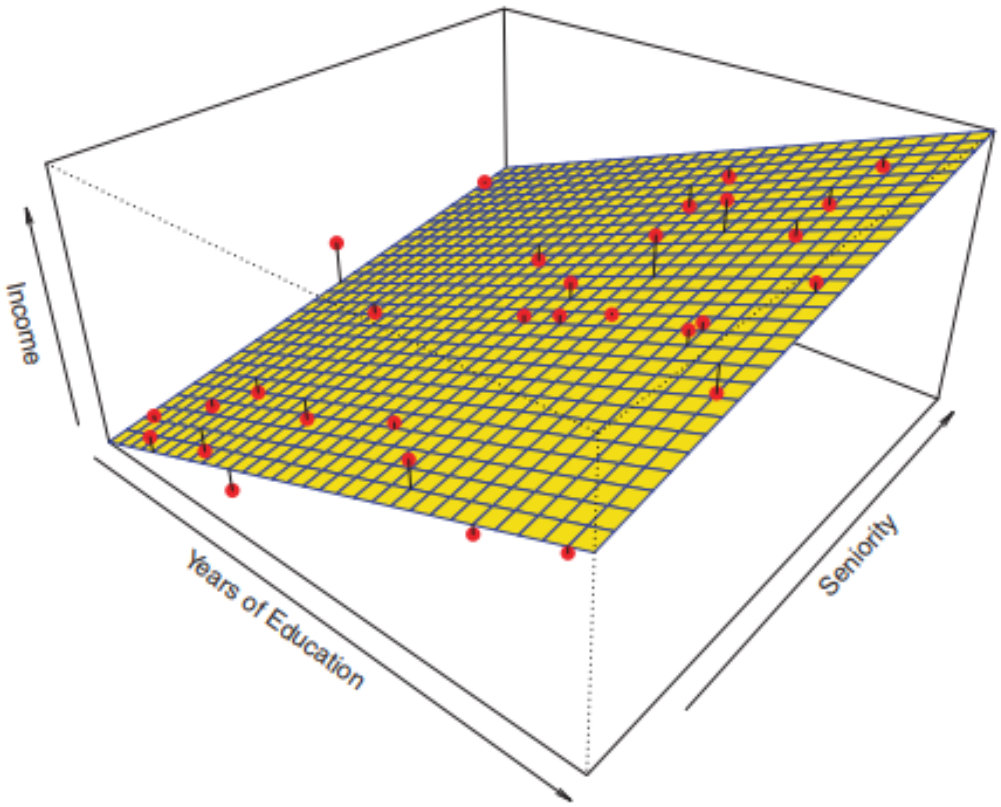
Review linear regression

- Simple linear regression: one dimension
- Multiple linear regression: multiple dimensions
- Minimize MSE is equivalent to MLE estimator
  - MSE: average of squared residuals
- Can derive closed-form solution for simple LR

$$-\theta_0 = \bar{y} - \theta_1 \bar{x}$$

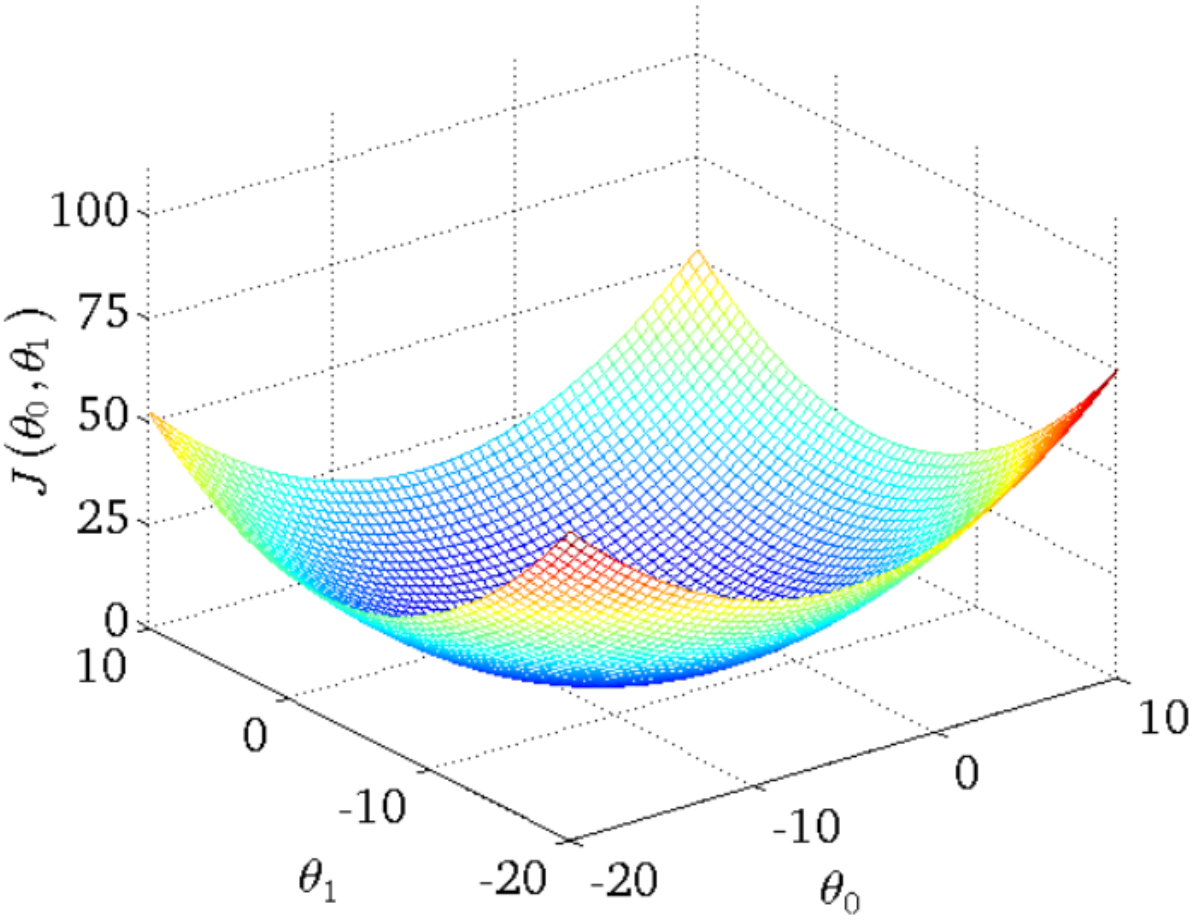
$$-\theta_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}$$

Multiple Linear Regression



- Linear Regression with 2 predictors
- Dataset: $x_i \in R^d, y_i \in R$

MSE function



Convex function implies unique minimum

Vector Norms

Vector norms: A norm of a vector ||x|| is informally a measure of the "length" of the vector.

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

- Common norms: L_1, L_2 (Euclidean)

$$\|x\|_1 = \sum_{i=1}^{n} |x_i| \qquad \|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

- L_∞

$$\|x\|_\infty = \max_i |x_i|$$

Vector products

We will use lower case letters for vectors
The elements are referred by x_i.

- Vector dot (inner) product:

$$x^T y \in \mathbb{R} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ x_2 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^{n} x_i y_i.$$

- Vector outer product:

$$xy^T \in \mathbb{R}^{m \times n} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix} = \begin{bmatrix} x_1 y_1 & x_1 y_2 & \cdots & x_1 y_n \\ x_2 y_1 & x_2 y_2 & \cdots & x_2 y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_m y_1 & x_m y_2 & \cdots & x_m y_n \end{bmatrix}$$

Hypothesis Multiple LR

- Linear Model

- Consider our model:

$$h(\boldsymbol{x}) = \sum_{j=0}^{d} \theta_j x_j$$

- Let

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \qquad \boldsymbol{x}^\intercal = \begin{bmatrix} 1 & x_1 & \dots & x_d \end{bmatrix}$$

- Can write the model in vectorized form as $h(\boldsymbol{x}) = \boldsymbol{\theta}^\intercal \boldsymbol{x}$

Vector inner product

Training data

Feature 1 Feature d

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{i1} & \cdots & x_{id} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & \cdots & x_{Nd} \end{bmatrix}$$

Training example i

- Total number of training example: *N*
- Dimension of training data point (number of features): *d*
- Dimension of matrix: *Nx(d+1)*

Use Vectorization

- Consider our model for n instances:

$$h(x_i) = \sum_{j=0}^{d} \theta_j x_{ij} = \theta^T x_i$$

- Let

Model parameter

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \quad \mathbb{R}^{(d+1)\times 1}$$

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{i1} & \cdots & x_{id} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & \cdots & x_{Nd} \end{bmatrix} \quad \mathbb{R}^{n\times(d+1)}$$

Training data

- Can write the model in vectorized form as $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{X}\boldsymbol{\theta}$

Model prediction vector $\hat{y}$

Loss function MSE

- For the linear regression cost function:

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}[h_\theta(x_i) - y_i]^2$$

$$= \frac{1}{N}\sum_{i=1}^{N}[\hat{y}_i - y_i]^2$$

$$y = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_N \end{bmatrix}$$

$$= \frac{1}{N}\left|\left|\hat{y} - y\right|\right|^2$$

$$= \frac{1}{N}\left|\left|\mathrm{X}\theta - y\right|\right|^2$$

$$\hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ . \\ . \\ . \\ \hat{y}_N \end{bmatrix}$$

Matrix and vector gradients

If $y = f(x), y \in R$ scalar , $x \in R^n$ vector

$$\frac{\partial y}{\partial x} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \frac{\partial y}{\partial x_2} & \cdots & \frac{\partial y}{\partial x_n} \end{bmatrix}$$

Vector gradient (row vector)

If $y = f(x), y \in R^m, x \in R^n$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

Jacobian matrix (Matrix gradient)

Properties

- If w, x are $(d \times 1)$ vectors, $\frac{\partial w^T x}{\partial x} = w^T$
- If A: $(n \times d)$ x: $(d \times 1)$, $\frac{\partial Ax}{\partial x} = A$

- If A: $(d \times d)$ x: $(d \times 1)$, $\frac{\partial x^T Ax}{\partial x} = (A + A^T)x$
- If A symmetric: $\frac{\partial x^T Ax}{\partial x} = 2Ax$
- If x: $(d \times 1)$, $\frac{\partial ||x||^2}{\partial x} = 2x^T$

Min loss function

– Notice that the solution is when $\frac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta}) = 0$

$$J(\theta) = \frac{1}{N} \left|\left|\mathrm{X}\theta - y\right|\right|^2$$

Using chain rule

$$f(\theta) = h\big(g(\theta)\big), \frac{\partial f(\theta)}{\partial \theta} = \frac{\partial h(g(\theta))}{\partial \theta} \frac{\partial g(\theta)}{\partial \theta}$$

$$h(x) = \left|\left|x\right|\right|^2, g(\theta) = X\theta - y$$

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{2}{N}[(\mathrm{X}\,\theta - y)^T X] = 0 \;\Rightarrow X^T\,(X\theta - y) = 0$$

$$(\boldsymbol{X}^\intercal \boldsymbol{X})\boldsymbol{\theta} = \boldsymbol{X}^\intercal \boldsymbol{y}$$

Closed Form Solution: $\boldsymbol{\theta} = (\boldsymbol{X}^\intercal \boldsymbol{X})^{-1} \boldsymbol{X}^\intercal \boldsymbol{y}$

Vectorization

- Two options for operations on training data
  - Matrix operations
  - For loops to update individual entries
- Most software packages are highly optimized for matrix operations
  - Python numpy
  - Preferred method!

- Matrix operations are much faster than loops!

Closed-form solution

- Can obtain $\boldsymbol{\theta}$ by simply plugging $\boldsymbol{X}$ and $\boldsymbol{y}$ into

$$\boldsymbol{\theta} = (\boldsymbol{X}^\intercal \boldsymbol{X})^{-1} \boldsymbol{X}^\intercal \boldsymbol{y}$$

$$X = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{i1} & \cdots & x_{id} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & \cdots & x_{Nd} \end{bmatrix} \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_N \end{bmatrix}$$

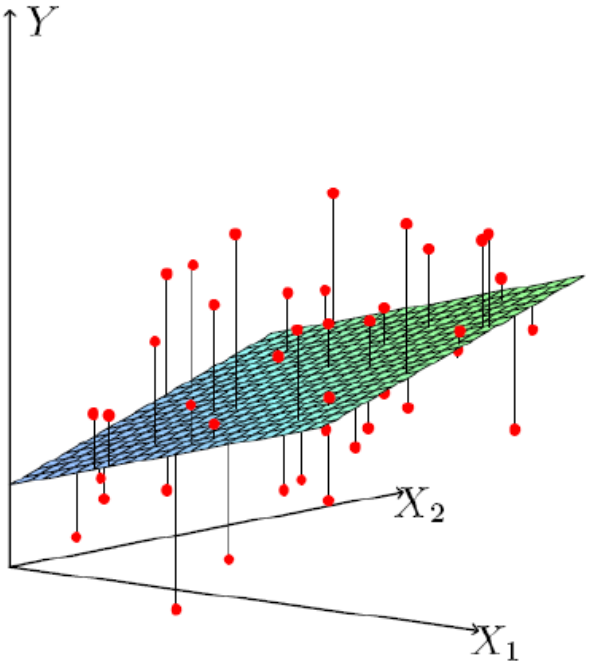
- If $\boldsymbol{X}^\intercal \boldsymbol{X}$ is not invertible (i.e., singular), may need to:
  - Use pseudo-inverse instead of the inverse
    - In python, `numpy.linalg.pinv(a)`
  - Remove redundant (not linearly independent) features
  - Remove extra features to ensure that $\mathrm{d} \leq \mathrm{n}$

$$AGA = A$$

Multiple Linear Regression

- Dataset: $x_i \in R^d, y_i \in R$
- Hypothesis $h_\theta(x) = \theta^T x$
- MSE $= \frac{1}{N}\sum (\theta^T x_i - y_i)^2$ Loss / cost

$$\theta = (X^\intercal X)^{-1} X^\intercal y$$



Practical issues: Feature Standardization

- Rescales features to have zero mean and unit variance
  - Let μ_j be the mean of feature j: $\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_j^{(i)}$
  - Replace each value with:

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$$

for j = 1...d
(not x_0!)

    - s_j is the standard deviation of feature j

- Must apply the same transformation to instances for both training and prediction
- Mean 0 and Standard Deviation 1

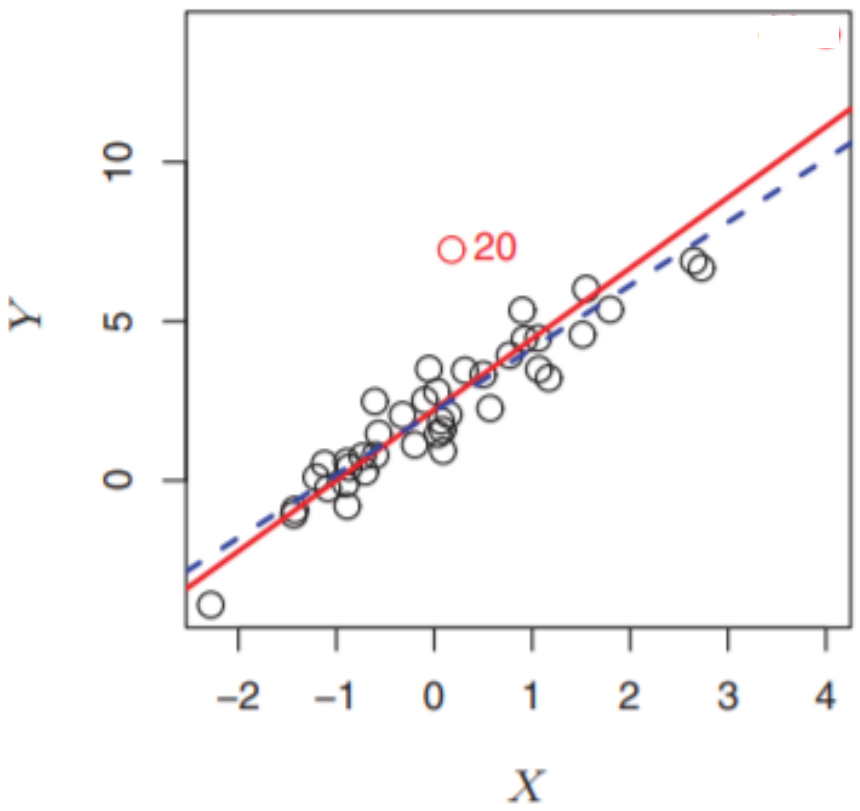
Other feature normalization

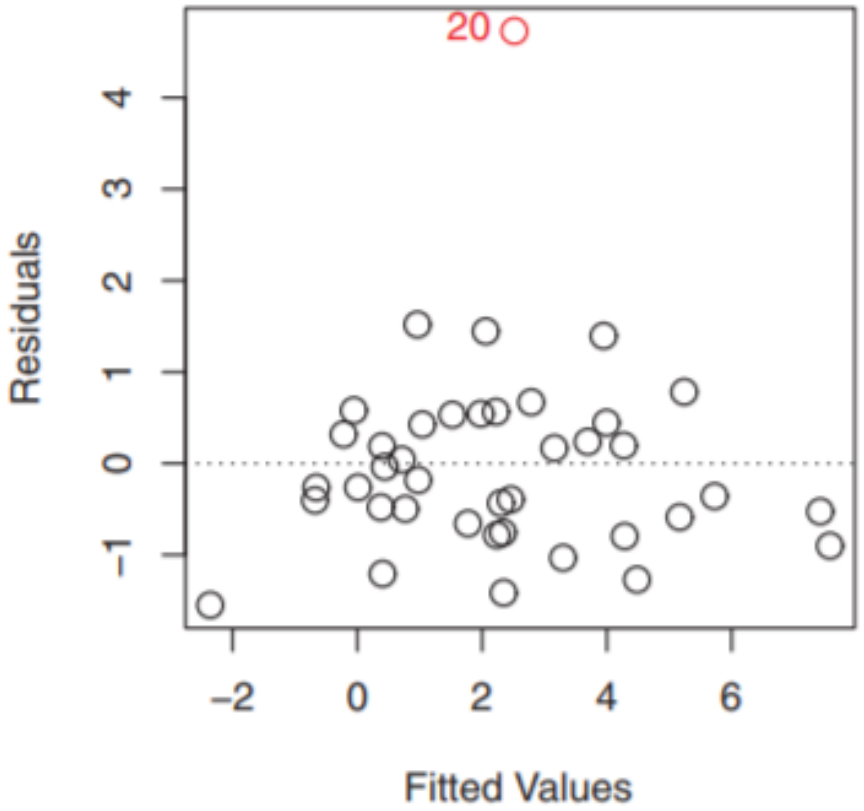
- Min-Max rescaling
  - $x_{ij} \leftarrow \frac{x_{ij} - min_j}{max_j - min_j} \in [0,1]$
  - min_j and max_j: min and max value of feature j
- Mean normalization
  - $x_{ij} \leftarrow \frac{x_{ij} - \mu_j}{max_j - min_j}$
  - Mean 0

Feature standardization/normalization

- Goal is to have individual features on the same scale
- Is a pre-processing step in most learning algorithms
- Necessary for linear models and Gradient Descent
- Different options:
  - Feature standardization
  - Feature min-max rescaling
  - Mean normalization

Practical issues: Outliers





- Dashed model is without outlier point
- Linear regression is not resilient to outliers!
- Outliers can be eliminated based on residual value
  - Use different loss functions (Huber loss)

Categorical variables

- Predict credit card balance
  - Age
  - Income
  - Number of cards
  - Credit limit
  - Credit rating
- Categorical variables
  - Student (Yes/No)
  - State (50 different levels)

Indicator Variables

- One-hot encoding
- Binary (two-level) variable
  - Add new feature $x_j = 1$ if student and 0 otherwise
- Multi-level variable
  - State: 50 values
  - $x_{MA} = 1$ if State $=$ MA and 0, otherwise
  - $x_{NY} = 1$ if State $=$ NY and 0, otherwise
  - …
  - How many indicator variables are needed?
- Disadvantages: data becomes too sparse for large number of levels
  - Will discuss feature selection later in class

Lab example

```
us_youtube = pd.read_csv('us_youtube_videos.csv')
```

```
us_youtube.head(15)
```

	views	likes	dislikes	comment_count
0	748374	57527	2966	15954
1	2418783	97185	6146	12703
2	3191434	146033	5339	8181
3	343168	10172	666	2146
4	2095731	132235	1989	17518
5	119180	9763	511	1434
6	2103417	15993	2445	1970
7	817732	23663	778	3432
8	826059	3543	119	340
9	256426	12654	1363	2368
10	81377	655	25	177
11	104578	1576	303	1279
12	687582	114188	1333	8371
13	544770	7848	1171	3981
14	207532	7473	246	2120

Lab example

```
us_youtube = pd.read_csv('us_youtube_videos.csv')
```

```
us_youtube.head(15)
```

Features min-max normalized

	views	likes	dislikes	published_in_morning	comment_count
0	0.003321	0.010247	0.001771	1	15954
1	0.010738	0.017312	0.003671	0	12703
2	0.014168	0.026013	0.003189	1	8181
3	0.001521	0.001812	0.000398	0	2146
4	0.009303	0.023555	0.001188	1	17518
5	0.000527	0.001739	0.000305	1	1434
6	0.009337	0.002849	0.001460	0	1970
7	0.003629	0.004215	0.000465	1	3432
8	0.003665	0.000631	0.000071	1	340
9	0.001136	0.002254	0.000814	1	2368
10	0.000359	0.000117	0.000015	0	177
11	0.000462	0.000281	0.000181	0	1279
12	0.003051	0.020340	0.000796	1	8371
13	0.002416	0.001398	0.000699	1	3981
14	0.000919	0.001331	0.000147	1	2120

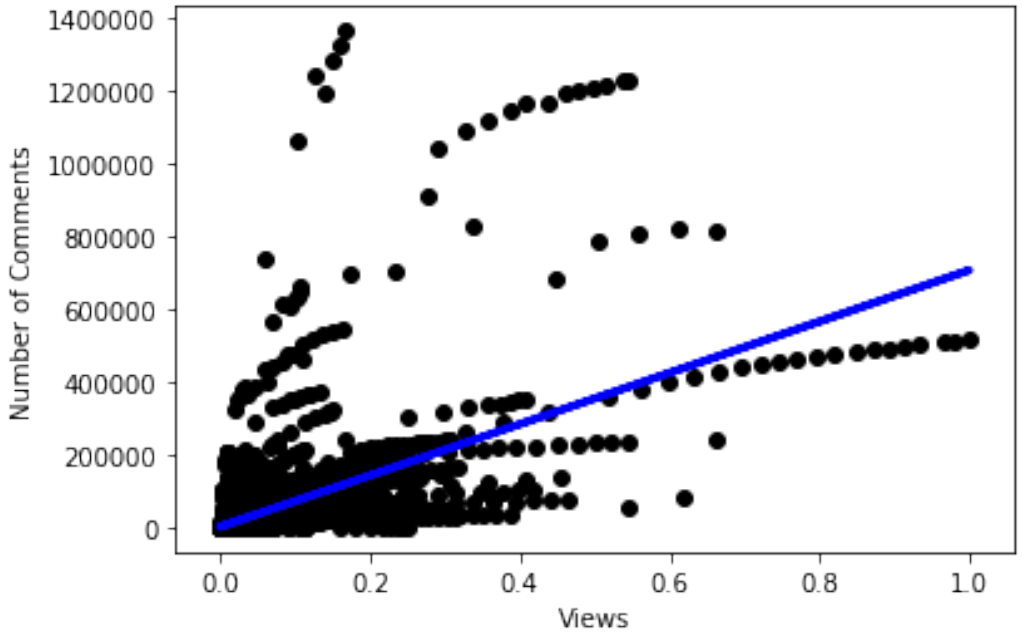
Simple LR

```
views_comments = us_youtube[['views', 'comment_count']]
```

```
reg = linear_model.LinearRegression()
reg.fit(views_comments.drop(columns='comment_count'), views_comments['comment_count'])

comments_pred = reg.predict(views_comments.drop(columns='comment_count'))

plt.scatter(views_comments['views'], views_comments['comment_count'], color='black')
plt.plot(views_comments['views'], comments_pred, color='blue', linewidth=3)
plt.xlabel('Views')
plt.ylabel('Number of Comments')
plt.show()
```



Simple LR

```
print("views coef: {}".format(reg.coef_[0]))
```

```
views coef: 704128.4873046515
```

```
mse_single = mean_squared_error(views_comments['comment_count'], comments_pred)
print("MSE: {}".format(mse_single))
print("RSE: {}".format(mse_single ** 0.5))
```

```
MSE: 866584512.2544774
RSE: 29437.807531378374
```

Without scaling response variable

```
views coef: 0.5171407389243756

MSE: 0.0004674386251650002
RSE: 0.021620328979111307
```

With scaling response variable

Residual Standard Error (RSE)

$$RSE = \sqrt{MSE}$$

Multiple LR

```
reg_multi = linear_model.LinearRegression()
reg_multi.fit(us_youtube.drop(columns='comment_count'), us_youtube['comment_count'])
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
comments_pred_multi = reg_multi.predict(us_youtube.drop(columns='comment_count'))
```

```
feature_names = us_youtube.drop(columns='comment_count').columns
for i in range(len(reg_multi.coef_)):
    print("{} coef: {}".format(feature_names[i], reg_multi.coef_[i]))
```

```
views coef: -441184.8341255368
likes coef: 845430.3290977236
dislikes coef: 1017058.2337598497
published_in_morning coef: -167.28387229563668
```

Without scaling response variable

```
mse_multi = mean_squared_error(us_youtube['comment_count'], comments_pred_multi)
print("MSE: {}".format(mse_multi))
print("RSE: {}".format(mse_multi ** 0.5))
```

```
MSE: 237564606.31275252
RSE: 15413.130970466465
```

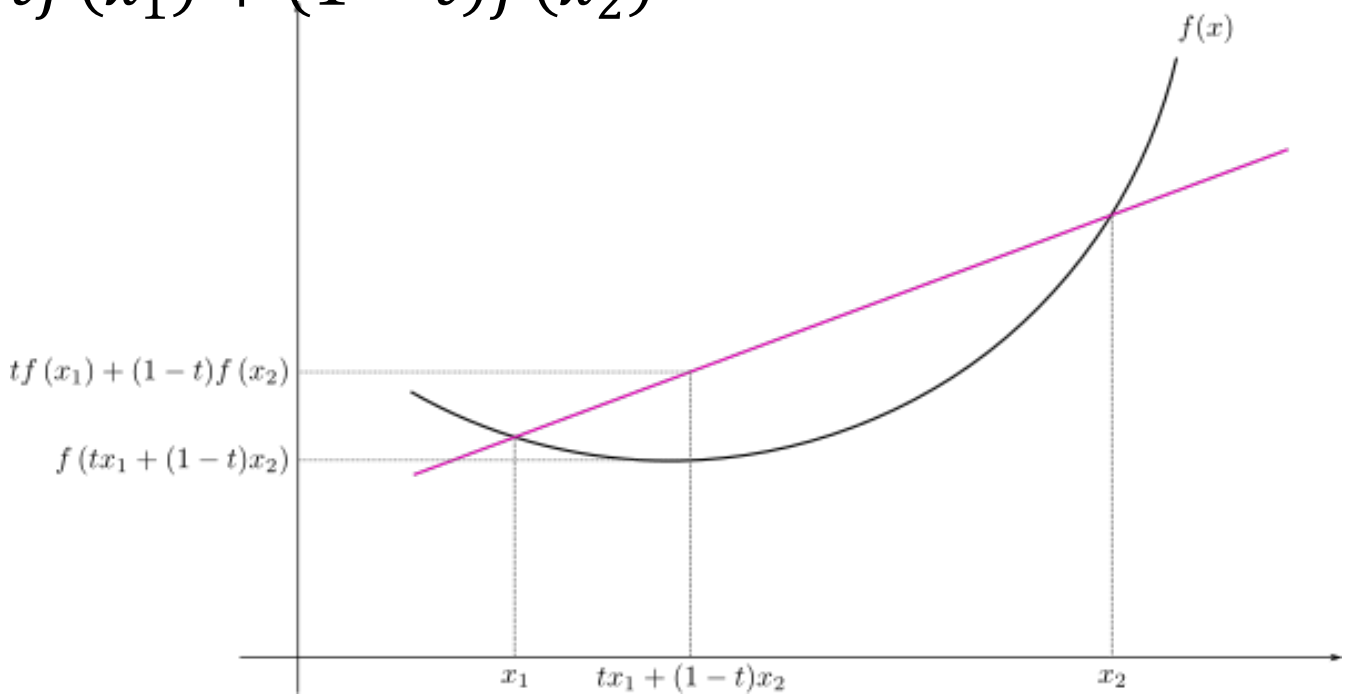
```
views coef: -0.32402417347899976
likes coef: 0.6209185865668729
dislikes coef: 0.7469691342116144
published_in_morning coef: -0.0001228601127334361


MSE: 0.0001281431544094894
RSE: 0.011320033321924869
```

With scaling response variable
Lower MSE/RSE with multiple features

How to optimize $J(\theta)$?

- (Strictly) Convex functions
  - $\forall x_1, x_2 \in \mathrm{X}, t \in [0,1], f(tx_1 + (1-t)x_2) < tf(x_1) + (1-t)f(x_2)$



  - Have single global minimum
  - Condition for differentiable functions: $f''(x) > 0$

Acknowledgements

- Slides made using resources from:
  - Andrew Ng
  - Eric Eaton
  - David Sontag
- Thanks!