DS 4400

Machine Learning and Data Mining I
Spring 2022

Alina Oprea
Associate Professor
Khoury College of Computer Science
Northeastern University

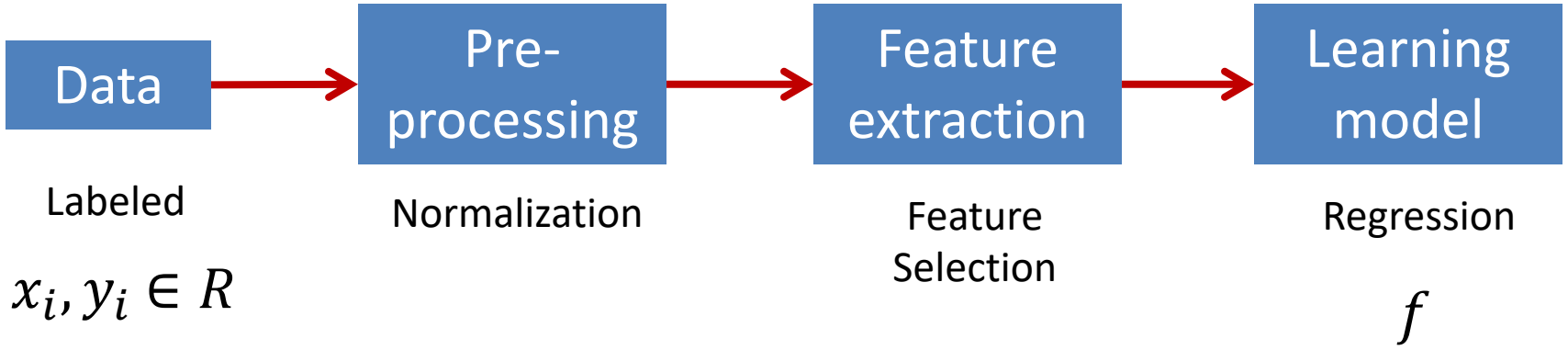
February 2 2022

Today's Outline

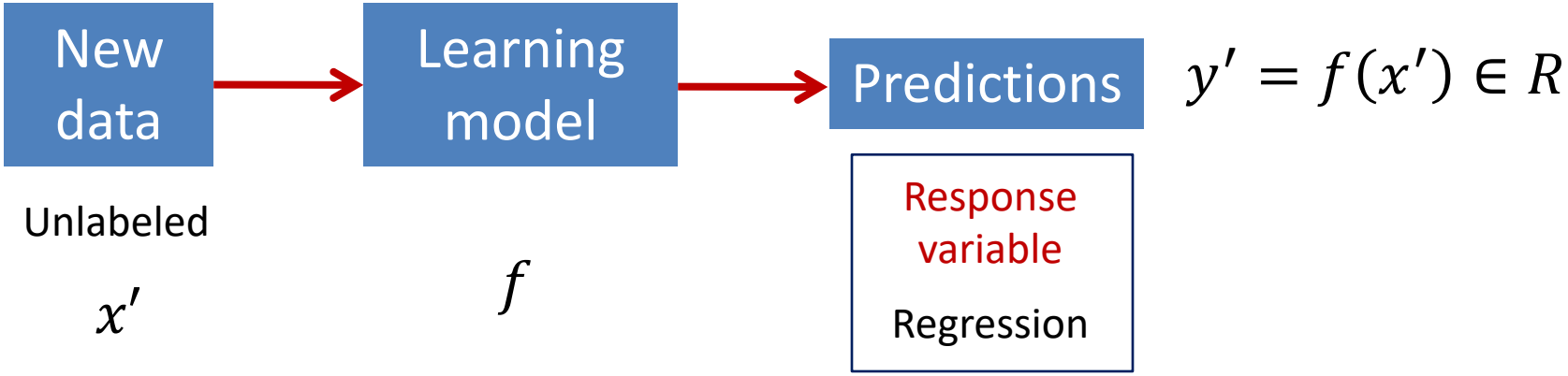
- Multiple Linear Regression
  - Vector and matrix gradients
  - Closed-form solution derivation
- Lab in Python
  - Simple LR
  - Multiple LR
- Practical issues when training LR models

Supervised Learning: Regression

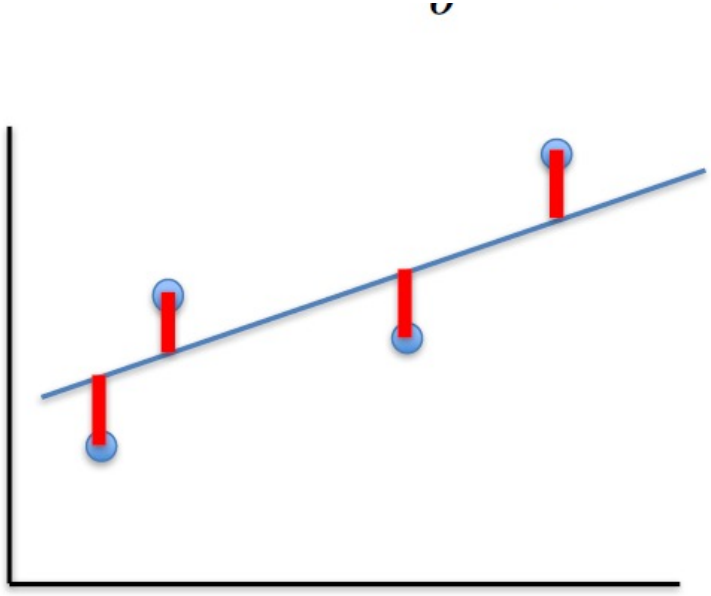
Training

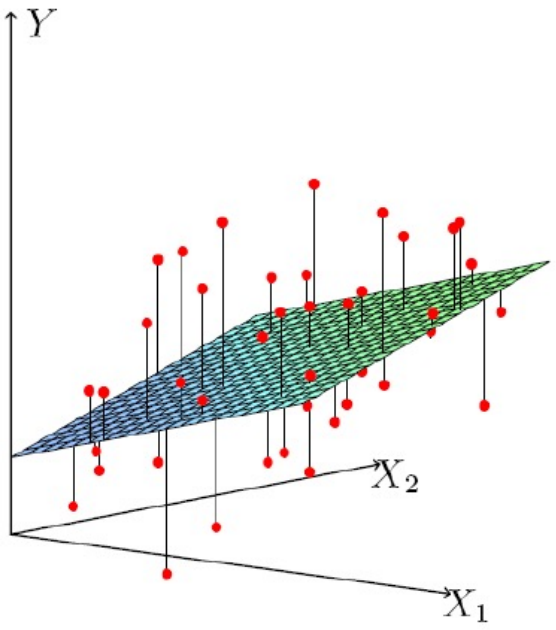


Testing

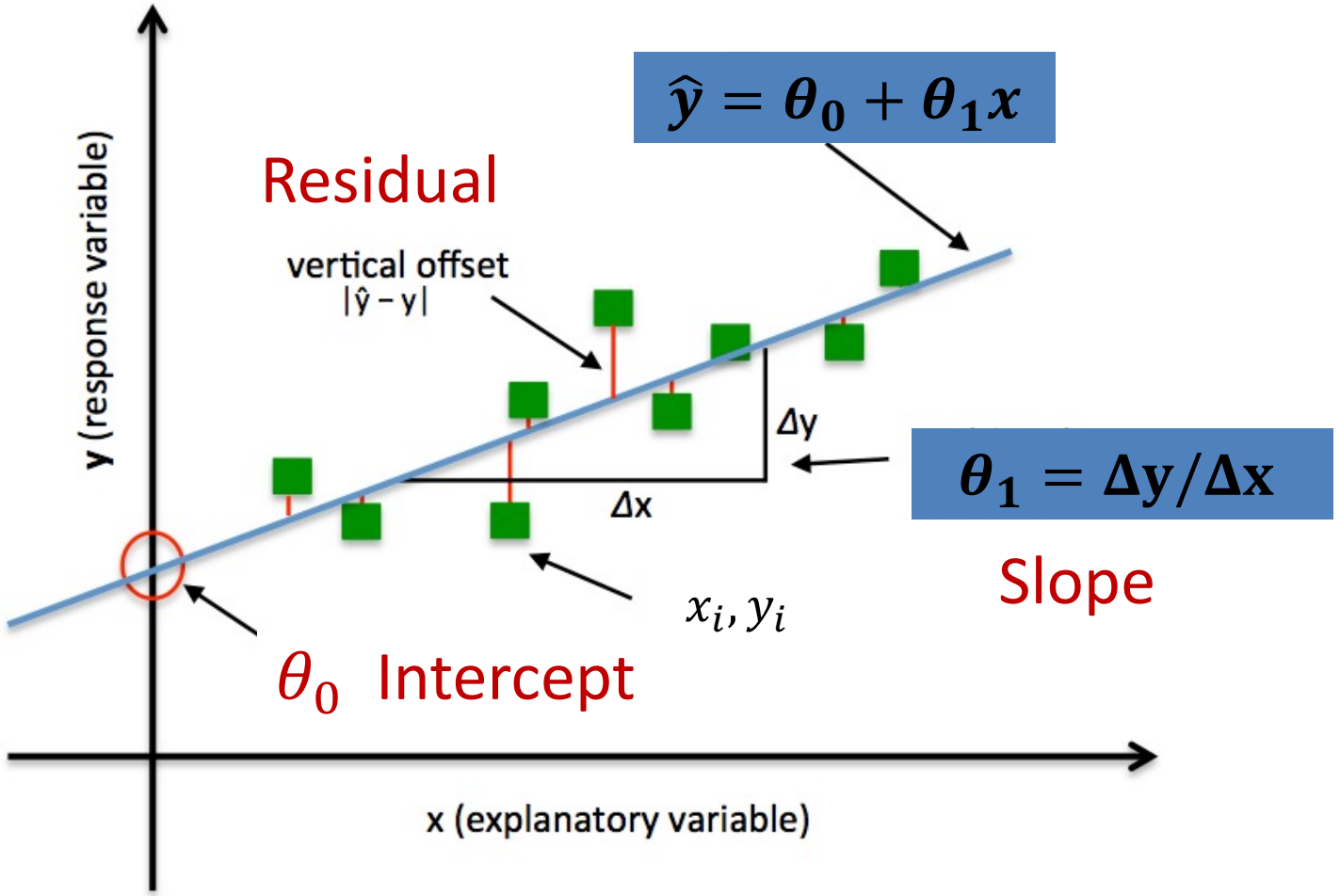


Least-Squares Linear Regression





Interpretation



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

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

Solution for simple linear regression

- Training data $x_i \in R, y_i \in R, h_\theta(x) = \theta_0 + \theta_1 x$
- $J(\theta) = \frac{1}{N}\sum_{i=1}^{N}(\theta_0 + \theta_1 x_i - y_i)^2$ MSE / Loss

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{2}{N}\sum_{i=1N}(\theta_0 + \theta_1 x_i - y_i) \quad = 0$$

$$\frac{\partial J(\theta)}{\partial \theta_1} = \frac{2}{N}\sum_{i=1}^{N} x_i(\theta_0 + \theta_1 x_i - y_i) \quad = 0$$

- Solution of min loss

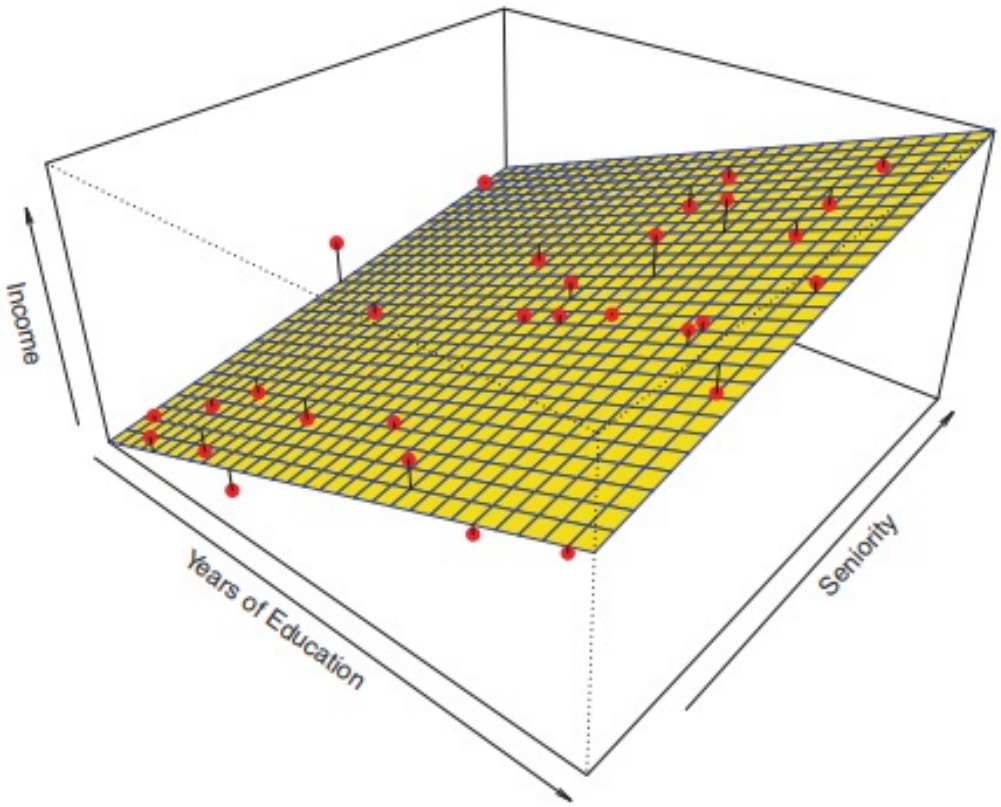
$$-\theta_0 = \bar{y} - \theta_1 \bar{x}$$

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

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N}$$

$$\bar{y} = \frac{\sum_{i=1}^{N} y_i}{N}$$

Multiple Linear Regression



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

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_∞

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

Norms and Distances

- Any norm will induce
  a distance metric
  between 2 vectors

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:

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

- Let

Model parameter

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \quad \boldsymbol{X} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{i1} & \cdots & x_{id} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & \cdots & x_{Nd} \end{bmatrix}$$

Training data

$\mathbb{R}^{(d+1)\times 1}$ $\mathbb{R}^{n\times(d+1)}$

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

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 d ≤ n

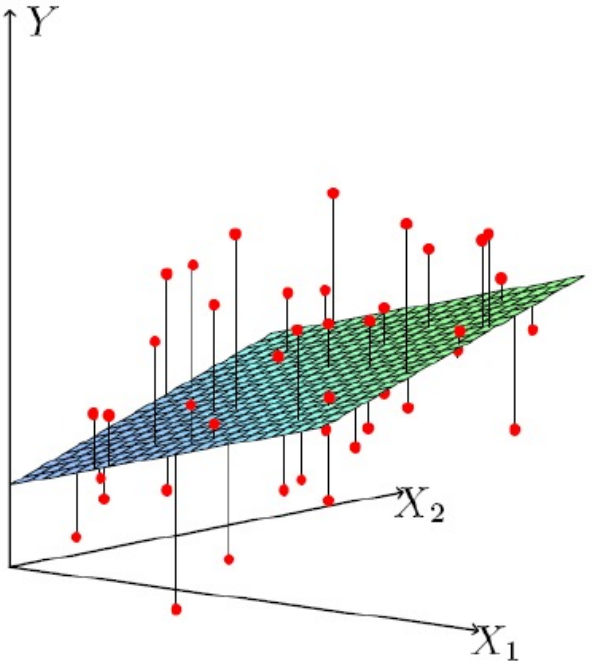
$$AGA = A$$

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

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

Closed-form optimum solution for linear regression



Lab Simple Linear Regression

```
#!/usr/bin/env python

import numpy as np
import matplotlib.pyplot as plt

import pandas as pd
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

from sklearn.linear_model import Ridge, RidgeCV, Lasso, LassoCV

from sklearn.datasets import load_boston
boston_dataset = load_boston()
```

Boston house prediction dataset

Lab

```
boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)
boston['MEDV'] = boston_dataset.target
boston.head(5)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

```
print(len(boston))
boston.shape
```

```
506
```

```
(506, 14)
```

```
correlation_matrix = boston.corr().round(2)
# annot = True to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True)
```

```
<AxesSubplot:>
```

Lab

```
# Simple LR
X = pd.DataFrame(np.c_[boston['RM']], columns = ['RM'])

Y = boston['MEDV']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=5)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(404, 1)
(102, 1)
(404,)
(102,)
```

```
slr = LinearRegression()
slr.fit(X_train, Y_train)
```

Lab

```
print(slr.intercept_)
print(slr.coef_)
```

```
-32.839129906011266
[8.82345634]
```

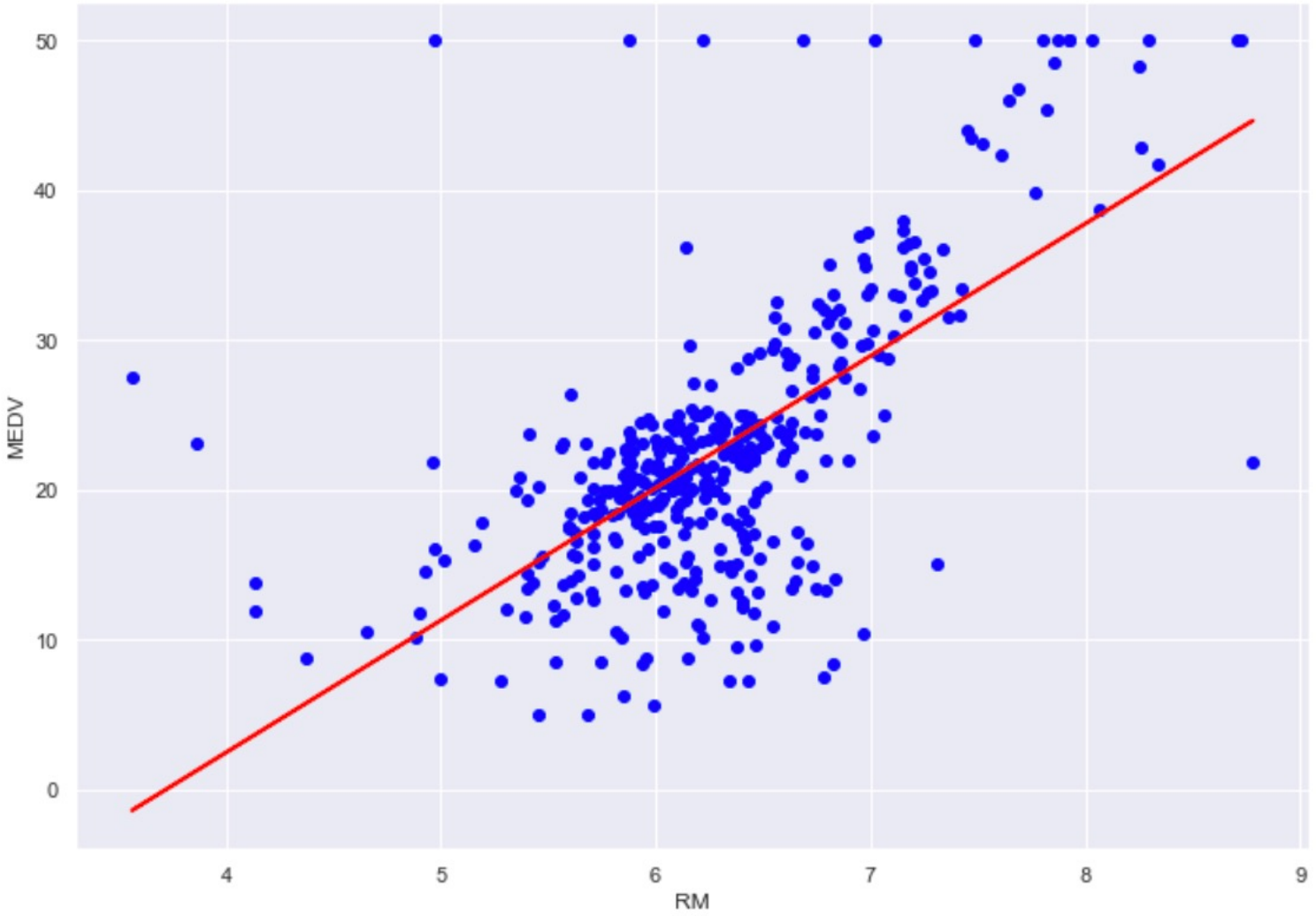
```
Y_train_predict = slr.predict(X_train)
mse = mean_squared_error(Y_train, Y_train_predict)

print("The model performance for training set")
print('MSE is {}'.format(mse))
```

```
The model performance for training set
MSE is 48.612648648611334
```

```
df = pd.DataFrame({'Actual': Y_train, 'Predicted': Y_train_predict})
df.head()
```

	Actual	Predicted
33	13.1	17.463395
283	50.0	37.069115
418	8.8	19.722200
502	20.6	21.160423
402	12.1	23.666285

```
plt.scatter(X_train, Y_train,  color='blue')
plt.plot(X_train, Y_train_predict, color='red', linewidth=2)
plt.xlabel('RM')
plt.ylabel('MEDV')
plt.show()
```



Multiple LR Lab

```
# Multiple LR

#X_multi = pd.DataFrame(np.c_[boston['LSTAT'], boston['RM']], columns = ['LSTAT','RM'])
X_multi = boston.loc[:, boston.columns != 'MEDV']
Y = boston['MEDV']

X_m_train, X_m_test, Y_m_train, Y_m_test = train_test_split(X_multi, Y, test_size = 0.2, random_state=5)
print(X_m_train.shape)
print(X_m_test.shape)
print(Y_m_train.shape)
print(Y_m_test.shape)
```

```
(404, 13)
(102, 13)
(404,)
(102,)
```

```
mlr = LinearRegression()
mlr.fit(X_m_train, Y_m_train)
```

```
LinearRegression()
```

Multiple LR Lab

```
coeff_df = pd.DataFrame(mlr.coef_, X_m_train.columns, columns=['Coefficient'])
coeff_df
```

	Coefficient
CRIM	-0.130800
ZN	0.049403
INDUS	0.001095
CHAS	2.705366
NOX	-15.957050
RM	3.413973
AGE	0.001119
DIS	-1.493081
RAD	0.364422
TAX	-0.013172
PTRATIO	-0.952370
B	0.011749
LSTAT	-0.594076

Simple vs Multiple LR

```
print(slr.intercept_)
print(slr.coef_)
```

```
-32.839129906011266
[8.82345634]
```

```
Y_train_predict = slr.predict(X_train)
mse = mean_squared_error(Y_train, Y_train_predict)

print("The model performance for training set")
print('MSE is {}'.format(mse))
```

```
The model performance for training set
MSE is 48.612648648611334
```

```
Y_m_train_predict = mlr.predict(X_m_train)
mse = mean_squared_error(Y_m_train, Y_m_train_predict)

print("The model performance for training set")
print("-------------------------------------")
print('MSE is {}'.format(mse))
print("\n")
```

```
The model performance for training set
-------------------------------------
MSE is 22.477090408387635
```

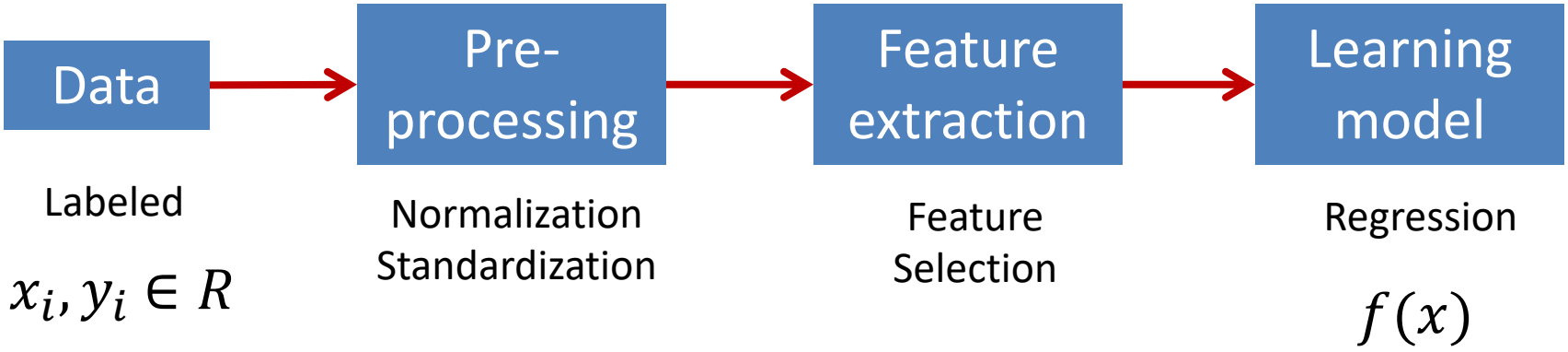
Simple vs Multiple LR

```
df_m = pd.DataFrame({'Actual': Y_train, 'Predicted simple': Y_train_predict, 'Predicted multi': Y_m_train_predict})
df_m.head(10)
```

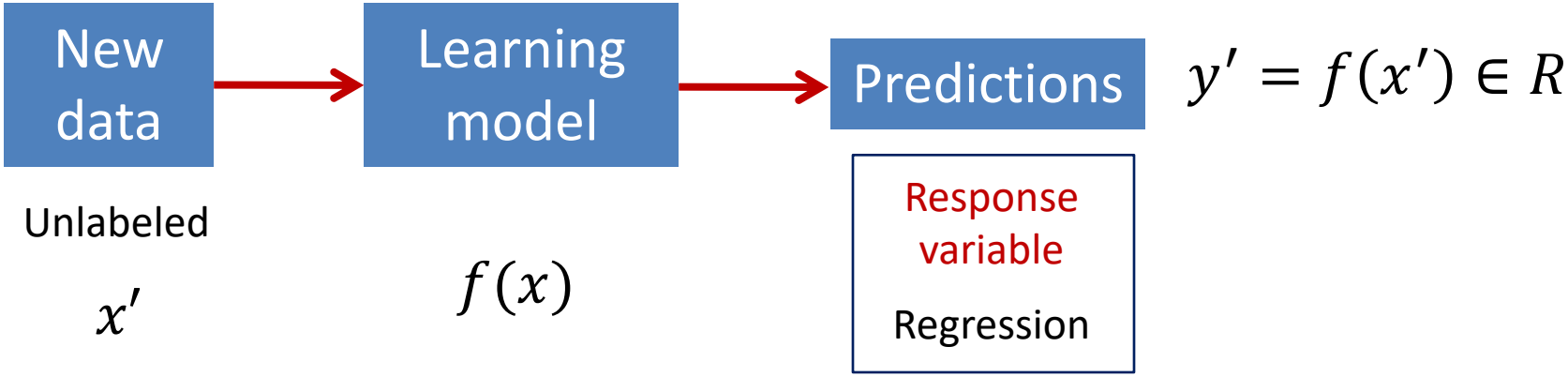
	Actual	Predicted simple	Predicted multi
33	13.1	17.463395	13.828770
283	50.0	37.069115	44.528528
418	8.8	19.722200	3.915991
502	20.6	21.160423	22.377959
402	12.1	23.666285	18.235923
368	50.0	11.013448	25.523748
201	24.1	21.531008	29.439747
310	16.1	11.039918	18.694533
343	23.9	26.242734	27.856463
230	24.3	19.933962	24.644734

Supervised Learning: Regression

Training



Testing



Practical issues: Feature Standardization

- Rescales features to have zero mean and unit variance
  - Let μ_j be the mean of feature j: $\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{ij}$
  - Replace each value with: $x_{ij} \leftarrow \frac{x_{ij}-\mu_j}{s_j}$ for j = 1...d (not x_0!)
    - s_j is the standard deviation of feature j

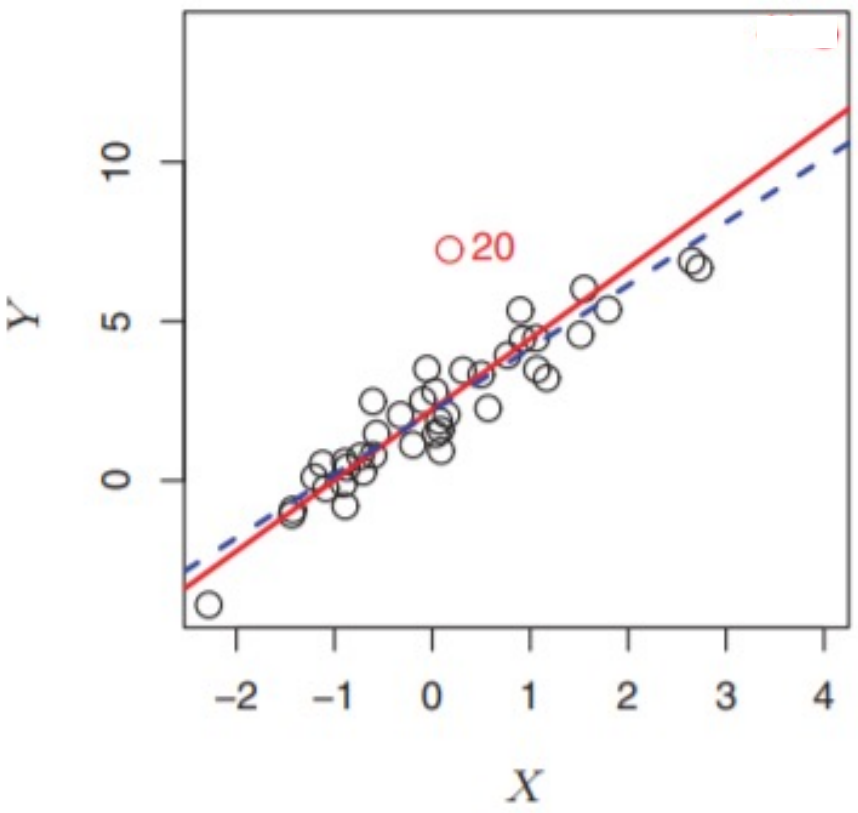
Other feature normalization

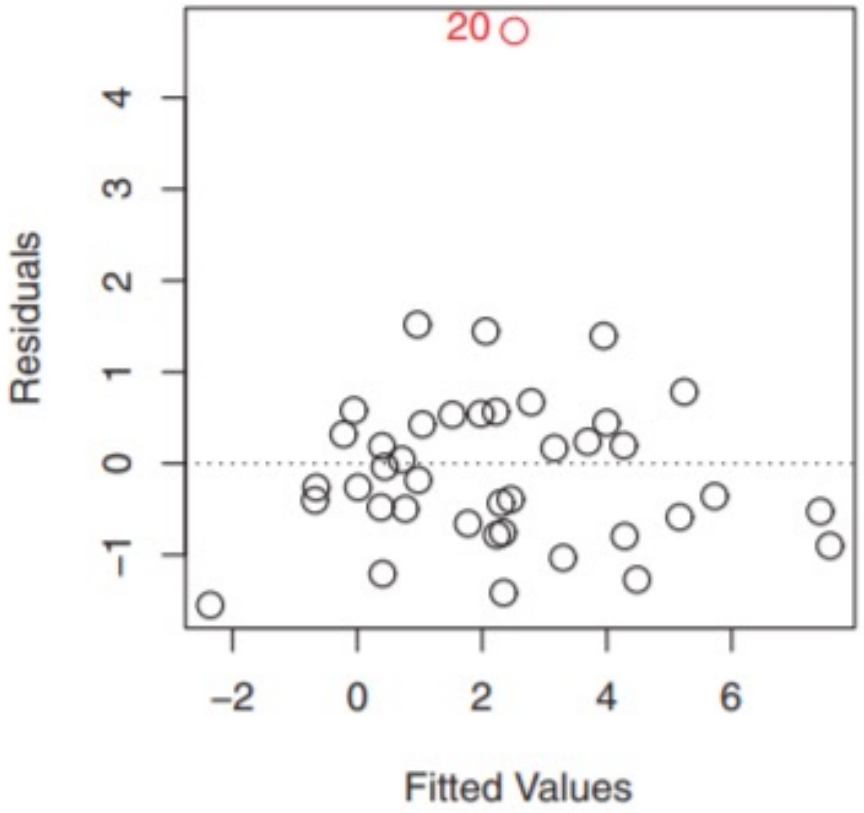
- Min-Max rescaling
  - $x_{ij} \leftarrow \frac{x_{ij} - min_j}{max_j - min_j} \in [0,1]$
  - min_j and max_j: min and max value of feature j
- Mean normalization
  - $x_{ij} \leftarrow \frac{x_{ij} - \mu_j}{max_j - min_j}$
  - Mean 0

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)

Recap Linear Regression

- Optimal solution to min MSE
  - Use vectorization for compact representation
- Advantages of linear regression
  - Simplicity and interpretability
  - Closed-form optimal solution (depends uniquely on training data)
- Limitations of linear regression
  - Small capacity in number of parameters (d+1)
  - Does not fit well non-linear data
- Practical issues
  - Feature standardization
  - Outliers
  - Categorical features

Acknowledgements

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