

Lecture 3

Multiple Linear Regression

EE-UY 4563/EL-GY 9123: INTRODUCTION TO MACHINE LEARNING

PROF. SUNDEEP RANGAN

(WITH MODIFICATION BY YAO WANG)

Learning Objectives

- ❑ Formulate a machine learning model as a multiple linear regression model.
 - Identify prediction vector and target for the problem.
- ❑ Write the regression model in matrix form. Write the feature matrix
- ❑ Compute the least-squares solution for the regression coefficients on training data.
- ❑ Derive the least-squares formula from minimization of the RSS

- ❑ Manipulate 2D arrays in python (indexing, stacking, computing shapes, ...)
- ❑ Compute the LS solution using python linear algebra and machine learning packages

Pre-Requisites for this Lecture

❑ Undergraduate students:

- We will cover Lecture 2 (Simple Linear Regression) in class first
- Some of the material in this lecture is a duplicate of Lecture 2
- I will go through this lecture more slowly, esp. for the linear algebra

❑ Graduate students:

- We can skip Lecture 2 and start this lecture directly after Lecture 1
- But, useful to read Lecture 2 and the corresponding demo on your own time.
- Will not review basic linear algebra in class. You should review this on your own.

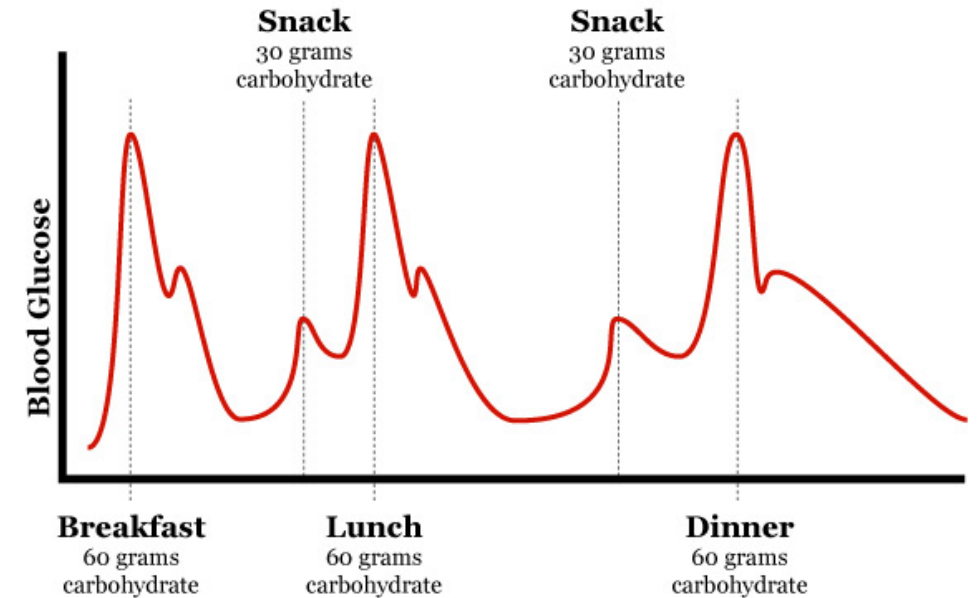
Outline

 Motivating Example: Understanding glucose levels in diabetes patients

- ☐ Multiple variable linear models
- ☐ Least squares solutions
- ☐ Computing the solutions in python
- ☐ Special case: Simple linear regression
- ☐ Extensions

Example: Blood Glucose Level

- ❑ Diabetes patients must monitor glucose level
- ❑ What causes blood glucose levels to rise and fall?
- ❑ Many factors
- ❑ We know mechanisms **qualitatively**
- ❑ But, **quantitative** models are difficult to obtain
 - Hard to derive from first principles
 - Difficult to model physiological process precisely
- ❑ Can machine learning help?



Data from AIM 94 Experiment

Data Set Information:

Diabetes patient records were obtained from two sources: an electronic clock to timestamp events, whereas the paper records only provided times assigned to breakfast (08:00), lunch (12:00), dinner (18:00), and bedtime. The electronic records have more realistic time stamps.

Diabetes files consist of four fields per record. Each field is separated by a space.

File Names and format:

- (1) Date in MM-DD-YYYY format
- (2) Time in XX:YY format
- (3) Code
- (4) Value

The Code field is deciphered as follows:

- 33 = Regular insulin dose
- 34 = NPH insulin dose
- 35 = UltraLente insulin dose
- 48 = Unspecified blood glucose measurement
- 57 = Unspecified blood glucose measurement
- 58 = Pre-breakfast blood glucose measurement
- 59 = Post-breakfast blood glucose measurement
- 60 = Pre-lunch blood glucose measurement
- 61 = Post-lunch blood glucose measurement
- 62 = Pre-supper blood glucose measurement
- 63 = Post-supper blood glucose measurement

❑ Data collected as series of events

- Eating
- Exercise
- Insulin dosage

❑ Target variable glucose level monitored



Diabetes Data Set

Download: [Data Folder](#), [Data Set Description](#)

Abstract: This diabetes dataset is from AIM '94

| | | | | | |
|-----------------------------------|---------------------------|------------------------------|-----|----------------------------|--------|
| Data Set Characteristics: | Multivariate, Time-Series | Number of Instances: | N/A | Area: | Life |
| Attribute Characteristics: | Categorical, Integer | Number of Attributes: | 20 | Date Donated | N/A |
| Associated Tasks: | N/A | Missing Values? | N/A | Number of Web Hits: | 161379 |

Demo on GitHub

☐ All code is available in github:

https://github.com/sdrangan/introml/blob/master/unit03_mult_lin_reg/demo2_glucose.ipynb

Demo: Predicting Glucose Levels using Multiple Linear Regression

In this demo, you will learn how to:

- Fit multiple linear regression models using python's sklearn package.
- Split data into training and test.
- Manipulating and visualizing multivariable arrays.

We first load the packages as usual.

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
```

Diabetes Data Example

To illustrate the concepts, we load the well-known diabetes data set. This dataset is included in the `sklearn.data` can be loaded as follows.

```
from sklearn import datasets, linear_model, preprocessing
```

Loading the Data

```
: from sklearn import datasets, linear_model, preprocessing  
  
# Load the diabetes dataset  
diabetes = datasets.load_diabetes()  
X = diabetes.data  
y = diabetes.target
```

```
nsamp, natt = X.shape  
print("num samples={0:d}  num attributes={1:d}".format(nsamp,natt))  
  
num samples=442  num attributes=10
```

- ❑ Sklearn package:
 - Many methods for machine learning
 - Datasets
 - Will use throughout this class
- ❑ Diabetes dataset is one example

Matrix Representation of Data

❑ Data is a **matrix**

❑ n samples:

- One sample per row

❑ k features / attributes / predictors:

- One feature per column

Attributes

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1k} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nk} \end{bmatrix}$$

Target vector

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

Samples

❑ This example:

- y_i = blood glucose measurement of i -th sample
- $x_{i,j}$: j -th feature of i -th sample
- $\mathbf{x}_i^T = [x_{i,1}, x_{i,2}, \dots, x_{i,k}]$: feature or predictor vector
- i -th sample contains \mathbf{x}_i, y_i

Outline

- ❑ Motivating Example: Understanding glucose levels in diabetes patients



- ❑ Multiple variable linear models

- ❑ Least squares solutions

- ❑ Computing the solutions in python

- ❑ Special case: Simple linear regression

- ❑ Extensions

Multiple Variable Linear Model

❑ Vector of **features**: $\mathbf{x} = [x_1, \dots, x_k]$

- k features (also known as predictors or independent variable attributes)

❑ Single **target variable** y

❑ Linear model:

$$y \approx \hat{y} = \beta_0 + \beta_1 x_1 + \dots + \beta_k x_k$$

- $p = k + 1$ terms in the model
- \hat{y} = predicted value

❑ Data for training

- Samples are (\mathbf{x}_i, y_i) , $i=1,2,\dots,n$.
- Each sample has a vector of features: $\mathbf{x}_i = [x_{i1}, \dots, x_{ik}]$ and scalar target y_i

❑ **Problem**: Learn the best coefficients $\boldsymbol{\beta} = [\beta_0, \beta_1, \dots, \beta_k]$ from the training data

Why Use a Linear Model?

- ❑ Many natural phenomena have linear relationship

- ❑ Predictor has small variation

- Suppose $y = f(x)$
- If variation of x is small around some value x_0 , then

$$y \approx f(x_0) + f'(x_0)(x - x_0) = \beta_0 + \beta_1 x,$$

$$\beta_0 = f(x_0) - f'(x_0)x_0, \quad \beta_1 = f'(x_0)$$

- ❑ Gaussian random variables:

- If two variables are jointly Gaussian, the optimal predictor of one from the other is linear predictor

- ❑ Simple to compute

- ❑ Easy to interpret relation

- Coefficient β_j indicates the importance of feature j for the target.

Matrix Review

□ Consider

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}, \quad B = \begin{bmatrix} 2 & 0 \\ 3 & 2 \end{bmatrix}, \quad x = \begin{bmatrix} 2 \\ 3 \end{bmatrix},$$

□ Compute (computations on the board):

- Matrix vector multiply: Ax
- Transpose: A^T
- Matrix multiply: AB
- Solution to linear equations: Solve for u : $x = Bu$
- Matrix inverse: B^{-1}

Matrix Form of Linear Regression

- ❑ Predicted value for i -th sample:

$$\hat{y}_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_k x_{ik}$$

- ❑ Define **feature matrix** and **regression vector**:

$$A = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1k} \\ \vdots & \vdots & \cdots & \vdots \\ 1 & x_{n1} & \cdots & x_{nk} \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix} \quad \left. \vphantom{\begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix}} \right\} p = k + 1 \text{ linear features}$$

- Feature matrix is data matrix + column of 1's

- ❑ Then, predicted vector for all training samples is: $\hat{\mathbf{y}} = A \boldsymbol{\beta}$

- ❑ Given a new sample with feature vector \mathbf{x} , the predicted value is $\hat{y}(\mathbf{x}) = [1, \mathbf{x}^T] \boldsymbol{\beta}$

Slopes and Intercept

- ❑ Model $\hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k$
- ❑ Sometimes use notation: $\hat{y} = b + w_1 x_1 + \cdots + w_k x_k$
- ❑ Components have two components:
 - $b = \beta_0$: Bias or intercept
 - $\mathbf{w} = \boldsymbol{\beta}_{1:k} = [\beta_1, \dots, \beta_k]$: Weights or slope vector
- ❑ Can write with inner product: $\hat{y}(\mathbf{x}) = \beta_0 + \boldsymbol{\beta}_{1:k} \cdot \mathbf{x} = b + \mathbf{w} \cdot \mathbf{x}$
- ❑ Inner product:
 - $\mathbf{w} \cdot \mathbf{x} = \sum_{j=1}^k w_j x_j$
 - Will use alternate notation: $\mathbf{w}^T \mathbf{x} = \langle \mathbf{w}, \mathbf{x} \rangle$

Arrays and Vector in Python and MATLAB

❑ There are some key differences between MATLAB and Python that you need to get used to

❑ MATLAB

- All arrays are at least 2 dimensions
- Vectors are $1 \times N$ (row vectors) or $N \times 1$ (column) vectors
- Matrix vector multiplication syntax depends if vector is on left or right: $x' * A$ or $A * x$


❑ Python:

- Arrays can have 1, 2, 3, ... dimension
- Vectors can be 1D arrays; matrices are generally 2D arrays
- Vectors that are 1D arrays are neither row nor column vectors
- If x is 1D and A is 2D, then left and right multiplication are the same: $x \cdot \text{dot}(A)$ and $A \cdot \text{dot}(x)$

❑ **Lecture notes:** We will generally treat x and x^T the same.

- Can write $x = (x_1, \dots, x_N)$ and still multiply by a matrix on left or right

Outline

- ❑ Motivating Example: Understanding glucose levels in diabetes patients
- ❑ Multiple variable linear models
- ❑ Least squares solutions
- ❑ Computing the solutions in python
- ❑ Special case: Simple linear regression
- ❑ Extensions

Least Squares Model Fitting

□ How do we select parameters $\boldsymbol{\beta} = (\beta_0, \dots, \beta_k)$?

□ Define $\hat{y}_i = \beta_0 + \beta_1 x_{i1} + \dots + \beta_k x_{ik}$

- Predicted value on sample i for parameters $\boldsymbol{\beta} = (\beta_0, \dots, \beta_k)$

□ Define average residual sum of squares:

$$\text{RSS}(\boldsymbol{\beta}) := \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- Note that \hat{y}_i is implicitly a function of $\boldsymbol{\beta} = (\beta_0, \dots, \beta_k)$
- Also called the sum of squared residuals (SSR) and sum of squared errors (SSE)

□ Least squares solution: Find $\boldsymbol{\beta}$ to minimize RSS.

- Geometrically, minimizes squared distances of samples to regression line





Finding Parameters via Optimization

A general ML recipe

General ML problem

- Pick a **model** with **parameters**
- Get data
- Pick a **loss function**
 - Measures goodness of fit model to data
 - Function of the parameters

Multiple linear regression

- Pick a **model** with **parameters**  Linear model: $\hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k$
- Get data  Data: $(x_i, y_i), i = 1, 2, \dots, n$
- Pick a **loss function**  Loss function:
$$RSS(\beta_0, \dots, \beta_k) := \sum (y_i - \hat{y}_i)^2$$
- Find parameters that **minimizes** loss  Select $\boldsymbol{\beta} = (\beta_0, \dots, \beta_k)$ to minimize $RSS(\boldsymbol{\beta})$

RSS as a Vector Norm

□ RSS is given by sum:

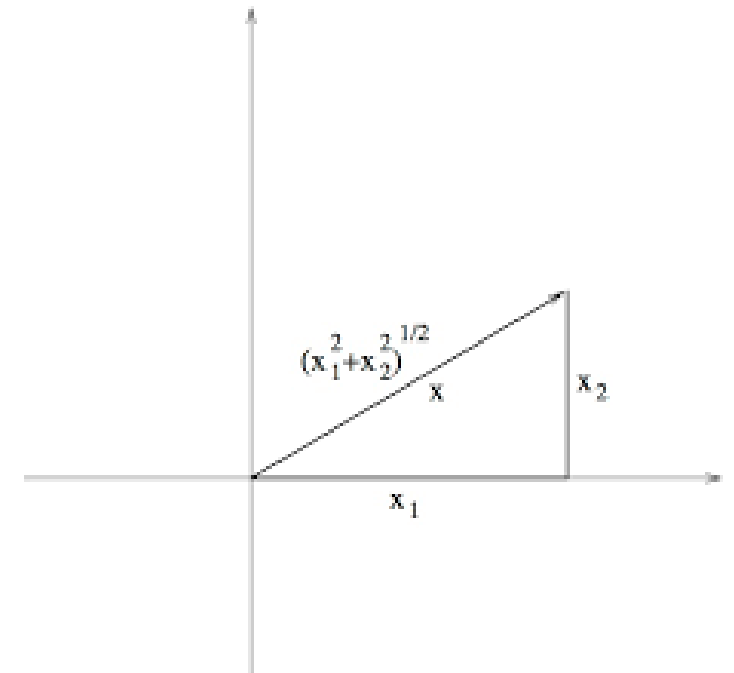
$$\text{RSS} = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

□ Define **norm** of a vector:

- $\|\mathbf{x}\| = (x_1^2 + \dots + x_r^2)^{1/2}$
- Standard Euclidean norm.
- Sometimes called ℓ -2 norm. ℓ is for Lebesgue

□ Write RSS in vector form:

$$\text{RSS} = \|\mathbf{y} - \hat{\mathbf{y}}\|^2$$



Gradients and Multi-Variable Functions

□ Consider scalar valued function of a vector: $f(\mathbf{x}) = f(x_1, \dots, x_n)$

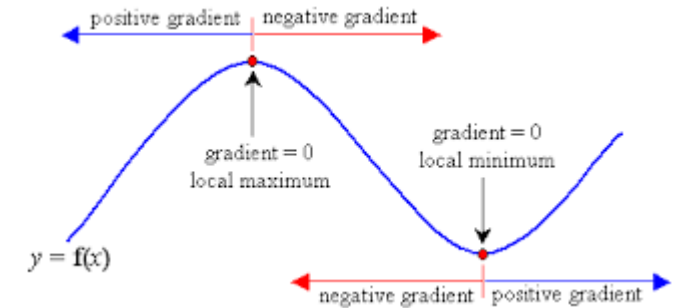
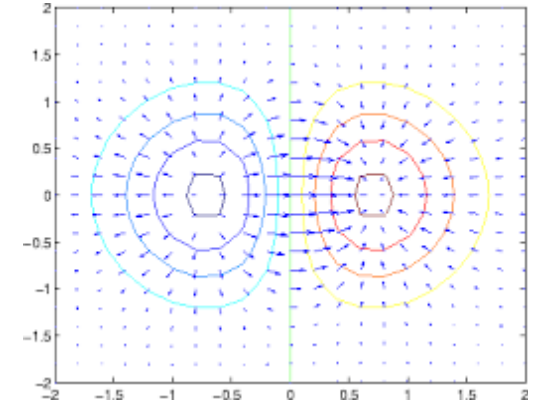
□ **Gradient** is the column vector:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \partial f(\mathbf{x}) / \partial x_1 \\ \vdots \\ \partial f(\mathbf{x}) / \partial x_n \end{bmatrix}$$

□ Ex: $f(x_1, x_2) = x_1 \sin x_2 + x_1^2 x_2$.
◦ Compute $\nabla f(\mathbf{x})$. Solution on board

□ Represents direction of maximum increase

□ At a local minima or maxima: $\nabla f(\mathbf{x}) = 0$
◦ Solve n equations and n unknowns



Least Squares Solution

- Consider cost function of the RSS:

$$RSS = \sum_{i=1}^n (y_i - \hat{y}_i)^2, \quad \hat{y}_i = \sum_{j=0}^p A_{ij}\beta_j$$

- Vector $\boldsymbol{\beta}$ that minimizes RSS called the **least-squares** solution

- Compute partial derivatives via chain rule: $\frac{\partial RSS}{\partial \beta_j} = 2 \sum_{i=1}^n (y_i - \hat{y}_i) A_{ij}, j = 1, 2, \dots, k$

- Matrix form: $RSS = \|A\boldsymbol{\beta} - \mathbf{y}\|^2, \nabla RSS = 2A^T(\mathbf{y} - A\boldsymbol{\beta})$

- Solution: $A^T(\mathbf{y} - A\boldsymbol{\beta}) = 0 \rightarrow \boldsymbol{\beta} = (A^T A)^{-1} A^T \mathbf{y}$ (least squares solution of equation $A\boldsymbol{\beta} = \mathbf{y}$)

- Minimum RSS: $RSS = \mathbf{y}^T [I - A(A^T A)^{-1} A^T] \mathbf{y}$

- Proof on the board

LS Solution via Auto-Correlation Functions

- Each data sample has a linear feature vector:

$$A_i = (A_{i0}, \dots, A_{ik}) = (1, x_{i1}, \dots, x_{ik})$$

- Define sample **auto-correlation** matrix and **cross-correlation** vector:

- $R_{AA} = \frac{1}{n} A^T A$, $R_{AA}(\ell, m) = \frac{1}{n} \sum_{i=1}^n A_{i\ell} A_{im}$ (correlation of feature ℓ and feature m)
- $R_{Ay} = \frac{1}{n} A^T y$, $R_{yA}(\ell) = \frac{1}{n} \sum_{i=1}^n A_{i\ell} y_i$ (correlation of feature ℓ and target)

- Least squares solution is: $\beta = R_{AA}^{-1} R_{Ay}$

R^2 : Goodness of Fit

- Define **target sample mean** and **variance**:

$$\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i, \quad s_y^2 = \frac{1}{n} \sum_{i=1}^n (y_i - \bar{y})^2$$

- Consider minimum prediction error per sample

$$\frac{RSS}{n} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- Multiple variable **coefficient of determination**:

$$R^2 = 1 - \frac{RSS/n}{s_y^2} = 1 - \frac{\text{avg error with linear model}}{\text{avg error with } \textit{prediction by mean}}$$

- $R^2 \in [0,1]$ always
- $R^2 \approx 1 \Rightarrow$ linear model provides a good fit
- $R^2 \approx 0 \Rightarrow$ linear model provides a poor fit

Notation

- ❑ Often, RSS is quoted in some relative form
- ❑ We will use the following terminology
 - Note: these are not standard
- ❑ Residual sum of squares: $RSS = \sum_{i=1}^n (y_i - \hat{y}_i)^2$
- ❑ RSS per sample: RSS/n
- ❑ Normalized RSS:

$$\frac{RSS/n}{s_y^2} = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$


Mean Removed Form of the LS Solution

- Often useful to remove mean from data before fitting
- Sample mean: $\bar{y} = \frac{1}{N} \sum_{i=1}^N y_i$, $\bar{x}_j = \frac{1}{N} \sum_{i=1}^N x_{ij}$, $\bar{\mathbf{x}} = [\bar{x}_1, \dots, \bar{x}_k]$
- Defined **mean removed data**: $\tilde{X}_{ij} = x_{ij} - \bar{x}_j$, $\tilde{y}_i = y_i - \bar{y}$
- Sample **covariance** matrix and **cross-covariance** vector:
 - $S_{xx}(\ell, m) = \frac{1}{N} \sum_{i=1}^N (x_{i\ell} - \bar{x}_\ell)(x_{im} - \bar{x}_m)$, $S_{xx} = \frac{1}{N} \tilde{\mathbf{X}}^T \tilde{\mathbf{X}}$
 - $S_{xy}(\ell) = \frac{1}{N} \sum_{i=1}^N (x_{i\ell} - \bar{x}_\ell)(y_i - \bar{y})$, $S_{xy} = \frac{1}{N} \tilde{\mathbf{X}}^T \tilde{\mathbf{y}}$
- **Mean-Removed form** of the least squares solution:

$$\hat{y} = \boldsymbol{\beta}_{1:k} \cdot \mathbf{x} + \beta_0, \quad \boldsymbol{\beta}_{1:k} = S_{xx}^{-1} S_{xy}, \quad \beta_0 = \bar{y} - \boldsymbol{\beta}_{1:k} \cdot \bar{\mathbf{x}}$$

- Proof: On board

Outline

- ❑ Motivating Example: Understanding glucose levels in diabetes patients
- ❑ Multiple variable linear models
- ❑ Least squares solutions
- ❑ Computing the solutions in python
- ❑ Special case: Simple linear regression
- ❑ Extensions

Fitting Using sklearn

```
: ns_train = 300
  ns_test = nsamp - ns_train
  X_tr = X[:ns_train,:]
  y_tr = y[:ns_train]
```

- ❑ Return to diabetes data example
- ❑ All code in demo
- ❑ Divide data into two portions:
 - Training data: First 300 samples
 - Test data: Remaining 142 samples
- ❑ Train model on training data.
- ❑ Test model (i.e. measure RSS) on test data
- ❑ Reason for splitting data discussed next lecture.

Manually Computing the Solution

```
ones = np.ones((ns_train,1))  
A = np.hstack((ones,X_tr))
```

```
out = np.linalg.lstsq(A,y_tr)  
beta = out[0]
```

❑ Use numpy linear algebra routine to solve

$$\beta = (A^T A)^{-1} A^T y$$

❑ Common mistake:

- Compute matrix inverse $P = (A^T A)^{-1}$,
- Then compute $\beta = P A^T y$
- Full matrix inverse is VERY slow. Not needed.
- Can directly solve linear system: $A \beta = y$
- Numpy has routines to solve this directly

Calling the sklearn Linear Regression method

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

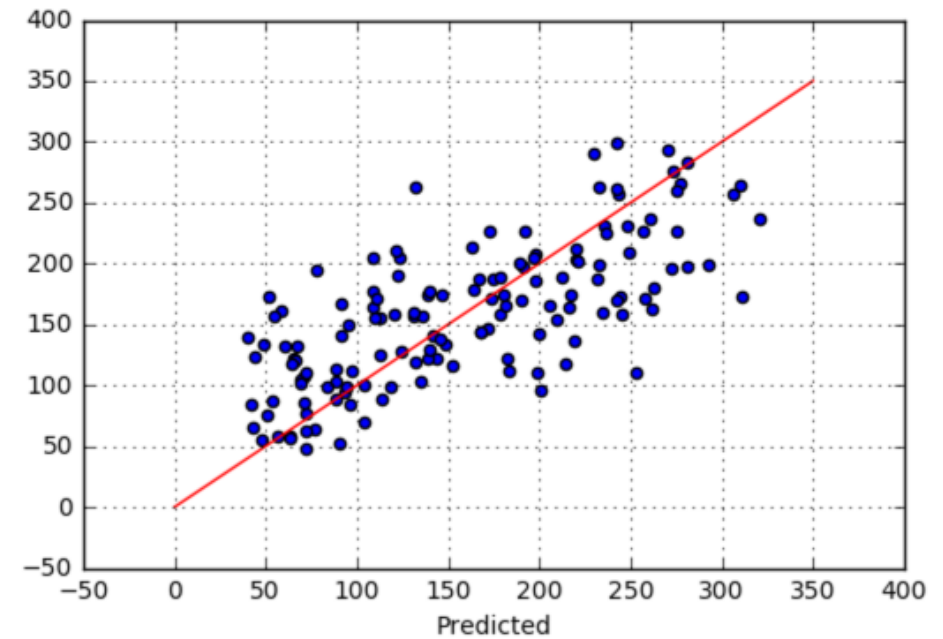
```
X_test = X[ns_train:,:]
y_test = y[ns_train:]
y_test_pred = regr.predict(X_test)
RSS_test = np.mean((y_test_pred-y_test)**2)/(np.std(y_test)**2)
Rsquared_test = 1-RSS_test
print("RSS per sample = {0:f}".format(RSS_test))
print("R^2 = {0:f}".format(Rsquared_test))
```

```
RSS per sample = 0.492801
R^2 = 0.507199
```


We see that the model predicts new samples almost as well as it did the training :

```
plt.scatter(y_test,y_test_pred)
plt.plot([0,350],[0,350], 'r')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.grid()
```

- ❑ Construct a linear regression object
- ❑ Run it on the training data
- ❑ Predict values on the test data



Outline

- ❑ Motivating Example: Understanding glucose levels in diabetes patients
- ❑ Multiple variable linear models
- ❑ Least squares solutions
- ❑ Computing the solutions in python
-  ❑ Special case: Simple linear regression
- ❑ Extensions

Simple vs. Multiple Regression

❑ Simple linear regression: One predictor (feature)

- Scalar predictor x
- Linear model: $\hat{y} = \beta_0 + \beta_1 x$
- Can only account for one variable

❑ Multiple linear regression: Multiple predictors (features)

- Vector predictor $\mathbf{x} = (x_1, \dots, x_k)$
- Linear model: $\hat{y} = \beta_0 + \beta_1 x_1 + \dots + \beta_k x_k$
- Can account for multiple predictors
- Turns into simple linear regression when $k = 1$

Comparison to Single Variable Models

- ❑ We could compute models for each variable separately:

$$\begin{aligned}y &= a_1 + b_1x_1 \\y &= a_2 + b_2x_2 \\&\vdots\end{aligned}$$

- ❑ But, doesn't provide a way to account for joint effects
- ❑ Example: Consider three linear models to predicting longevity:
 - A: Longevity vs. some factor in diet (e.g. amount of fiber consumed)
 - B: Longevity vs. exercise
 - C: Longevity vs. diet AND exercise
 - What does C tell you that A and B do not?

Special Case: Single Variable

□ Suppose $k = 1$ predictor.

□ Feature matrix and coefficient vector:

$$A = \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

□ LS soln: $\beta = \left(\frac{1}{N} A^T A\right)^{-1} \left(\frac{1}{N} A^T y\right) = P^{-1} r$

$$P = \begin{bmatrix} 1 & \bar{x} \\ \bar{x} & \bar{x}^2 \end{bmatrix}, \quad r = \begin{bmatrix} \bar{y} \\ \overline{xy} \end{bmatrix}$$

□ Obtain single variable solutions for coefficients (after some algebra):

$$\beta_1 = \frac{s_{xy}}{s_x^2}, \quad \beta_0 = \bar{y} - \beta_1 \bar{x}, \quad R^2 = \frac{s_{xy}^2}{s_x^2 s_y^2}$$

Simple Linear Regression for Diabetes Data

```
ym = np.mean(y)
syy = np.mean((y-ym)**2)
Rsq = np.zeros(natt)
for k in range(natt):
    xm = np.mean(X[:,k])
    sxy = np.mean((X[:,k]-xm)*(y-ym))
    sxx = np.mean((X[:,k]-xm)**2)
    Rsq[k] = (sxy)**2/sxx/syy

print("{0:2d} Rsq={1:f}".format(k, Rsq[k]))
```

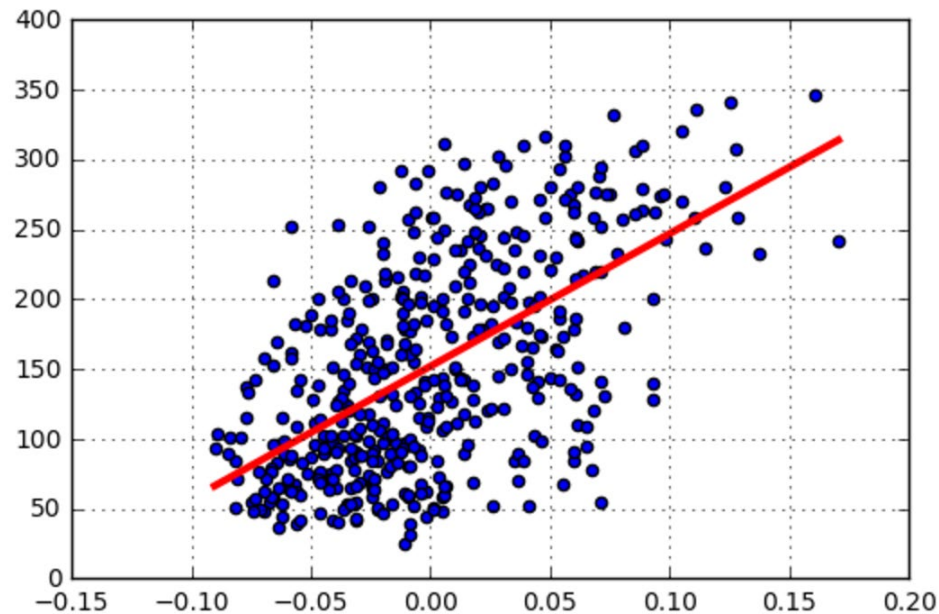
```
0 Rsq=0.035302
1 Rsq=0.001854
2 Rsq=0.343924
3 Rsq=0.194908
4 Rsq=0.044954
5 Rsq=0.030295
6 Rsq=0.155859
7 Rsq=0.185290
8 Rsq=0.320224
9 Rsq=0.146294
```

← Best individual variable

- ❑ Try a fit of each variable individually
- ❑ Compute R_k^2 coefficient for each variable
- ❑ Use formula on previous slide
- ❑ “Best” individual variable is a poor fit
 - $R_k^2 \approx 0.34$

Scatter Plot

- ❑ No one variable explains glucose well
- ❑ Multiple linear regression is much better

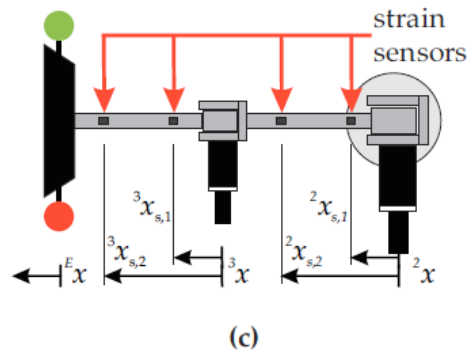
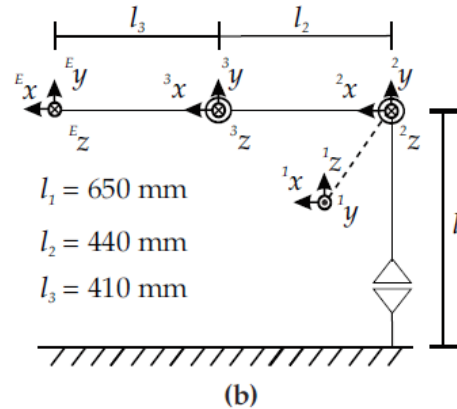
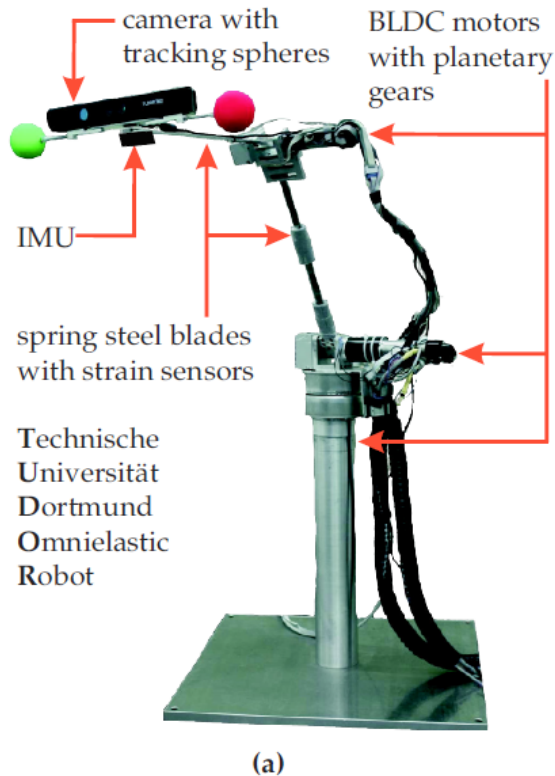


```
# Find the index of the single variable with the best R^2
imax = np.argmax(Rsq)

# Regression line over the range of x values
xmin = np.min(X[:,imax])
xmax = np.max(X[:,imax])
ymin = beta0[imax] + beta1[imax]*xmin
ymax = beta0[imax] + beta1[imax]*xmax
plt.plot([xmin,xmax], [ymin,ymax], 'r-', linewidth=3)

# Scatter plot of points
plt.scatter(X[:,imax],y)
plt.grid()
```

Lab: Robot Calibration



□ Predict the current draw

- Needed to predict power consumption


□ Predictors:

- Joint angles, velocity and acceleration
- Strain gauge readings (measure of load)

□ Full website at TU Dortmund, Germany

- http://www.rst.e-technik.tu-dortmund.de/cms/en/research/robotics/TUDOR_engl/index.html

Outline

- ❑ Motivating Example: Understanding glucose levels in diabetes patients
- ❑ Multiple variable linear models
- ❑ Least squares solutions
- ❑ Computing in python
- ❑ Extensions

Transformed Linear Models

□ Standard linear model: $\hat{y} = \beta_0 + \beta_1 x_1 + \dots + \beta_d x_d$

□ But, often it is useful to look at models in **transformed form**:

$$\hat{y} = \beta_1 \phi_1(\mathbf{x}) + \dots + \beta_p \phi_p(\mathbf{x})$$

- Each function $\phi_j(\mathbf{x}) = \phi_j(x_1, \dots, x_d)$ is called a **basis function**
- Each basis functions may be nonlinear and a function of multi-variables

□ Can write in vector form: $\hat{y} = \boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\beta}$

- $\boldsymbol{\phi}(\mathbf{x}) = [\phi_1(\mathbf{x}), \dots, \phi_b(\mathbf{x})]$, $\boldsymbol{\beta} = [\beta_1, \dots, \beta_p]$

□ Enables a much richer class

Fitting Transformed Linear Models

- Consider transformed linear model

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

- We can fit this model exactly as before

- Given data $(\mathbf{x}_i, y_i), i = 1, \dots, N$
- Then, fit the model from the transformed variables $\phi_j(\mathbf{x})$ to target y
- Define the transformed matrix:

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

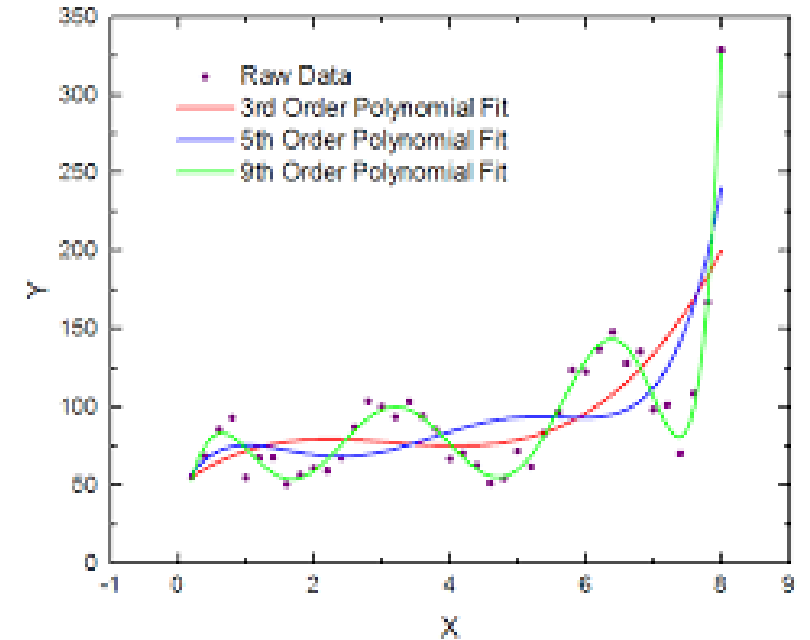
- Least squares fit $\hat{\beta} = (A^T A)^{-1} A^T y$

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 $(x_i, y_i), i = 1, \dots, n$
- ❑ Take basis functions $\phi_j(x) = x^j, j = 0, \dots, d$
- ❑ Transformed model: $\hat{y} = \beta_0 \phi_0(x) + \cdots + \beta_d \phi_d(x)$
- ❑ Transformed matrix is:

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

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



Other Nonlinear Examples

❑ **Multinomial model:** $\hat{y} = a + b_1x_1 + b_2x_2 + c_1x_1^2 + c_2x_1x_2 + c_3x_2^2$

- Contains all second order terms
- Define parameter vector $\beta = [a, b_1, b_2, c_1, c_2, c_3]$
- Transformed vector $\phi(x_1, x_2) = [1, x_1, x_2, x_1^2, x_1x_2, x_2^2]$
- Note that the features are nonlinear functions of $\mathbf{x} = [x_1, x_2]$

❑ **Exponential model:** $\hat{y} = a_1e^{-b_1x} + \dots + a_de^{-b_dx}$

- If the parameters b_1, \dots, b_d are fixed, then the model is linear in the parameters a_1, \dots, a_d
- Parameter vector $\beta = [a_1, \dots, a_d]$
- Transformed vector $\phi(x) = [e^{-b_1x}, \dots, e^{-b_dx}]$
- But, if the parameters b_1, \dots, b_d are not fixed, the model is nonlinear in b_1, \dots, 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, $\hat{y} = \beta_1x_1 + \beta_2x_1e^{-x_2}$
- Basis functions: $\phi(x_1, x_2) = [x_1, x_1e^{-x_2}]$
- After we solve for β_1, β_2 we can recover A, B via inverting the equations:

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

Example: Learning Linear Systems

Linear system: $y_k = a_1 y_{k-1} + \dots + a_m y_{k-m} + b_0 x_k + \dots + b_n x_{k-n} + w_k$

Transfer function: $H(z) = \frac{b_0 + \dots + b_n z^{-n}}{1 - a_1 z^{-1} - \dots - a_m z^{-m}}$

Given input sequence and output sequence for T samples,

How do we determine $\beta = (a_1, \dots, a_m, b_0, \dots, b_n)^T$.

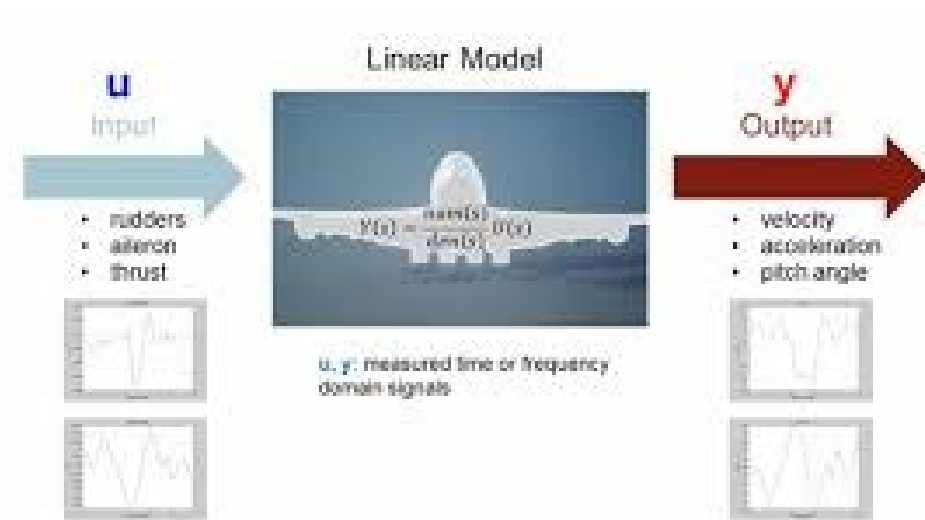
Can be solved using linear regression!

Write $y = A\beta + w$ and define A, y

- See homework problem

Many applications

- Learning dynamics in robots / mechanical systems
- Modeling responses in neural systems
- Stock market time series
- Speech modeling. Fit a model each 25 ms.



One Hot Coding

- ❑ Suppose that one feature x_j is a **categorical** variable
- ❑ Ex: Predict the price of a car, y , given model x_1 and interior space x_2
 - Suppose there are 3 different models of a car (Ford, BMW, GM)
 - Bad idea: Arbitrarily assign an index to each possible car model
 - Can give unreasonable relations
- ❑ One-hot coding example:
 - With 3 possible categories, represent x_1 using 3 binary features (ϕ_1, ϕ_2, ϕ_3)
 - Model: $y = \beta_0 + \beta_1\phi_1 + \beta_2\phi_2 + \beta_3\phi_3 + \beta_4x_2$
 - Essentially obtain 3 different models:
 - Ford: $y = \beta_0 + \beta_1 + \beta_4x_2$
 - BMW: $y = \beta_0 + \beta_2 + \beta_4x_2$
 - GM: $y = \beta_0 + \beta_3 + \beta_4x_2$
 - Allows different intercepts (or mean values) for different categories!

| Model | ϕ_1 | ϕ_2 | ϕ_3 |
|-------|----------|----------|----------|
| Ford | 1 | 0 | 0 |
| BMW | 0 | 1 | 0 |
| GM | 0 | 0 | 1 |