Lecture 3
Multiple Linear Regression

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

PROF. SUNDEEP RANGAN

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:
  - Go through Lecture 2 (Simple Linear Regression) 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

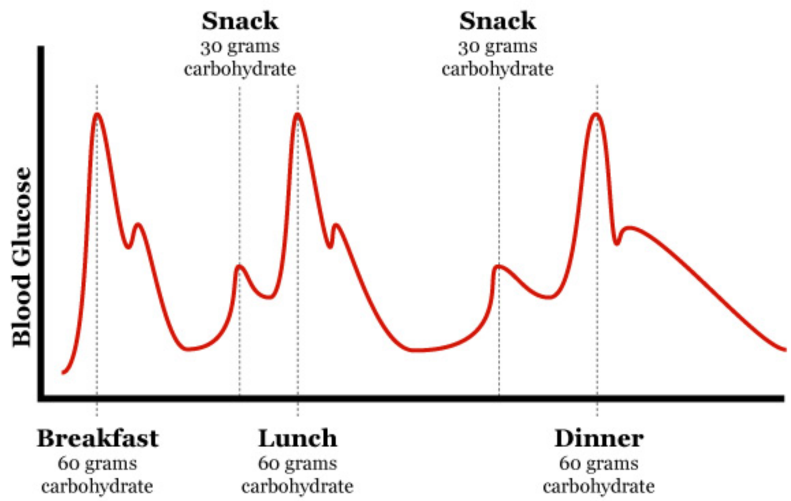
- Graduate students:
  - You can skip Lecture 2 and start this after Lecture 1
  - But, useful to read Lecture 2 and the corresponding demo on your own time.
  - Will not review basic linear algebra in class. You should review this on your own.

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: ar
clock to timestamp events, whereas the paper records only p
assigned to breakfast (08:00), lunch (12:00), dinner (18:00), :
records have more realistic time stamps.

Diabetes files consist of four fields per record. Each field is s

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

About Citation Policy

UCI

Machine Learning Repository

Center for Machine Learning and Intelligent Systems

Reposito

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/mult_lin_reg/glucose.ipynb

Demo: Predicting Glucose Levels using Mulitple 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.da`
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:
  - One feature per column
- This example:
  - y_i = blood glucose measurements

$$X = \overbrace{\begin{bmatrix} x_{11} & \cdots & x_{1k} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nk} \end{bmatrix}}^{\text{Attributes}} \qquad \overset{\text{Target vector}}{y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}} \left.\vphantom{\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}}\right\} \text{Samples}$$

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 predictors: $\boldsymbol{x} = (x_1, \ldots, x_k)$
  - k predictors / independent variable attributes
- Single target variable y
- Linear model:

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

  - $p = k + 1$ terms in the model
  - $\hat{y}$ = predicted value
- Data for training
  - Samples are $(\boldsymbol{x}_i, y_i)$.
  - Each sample has a vector of predictors: $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{ik})$ and scalar target y_i

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

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

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

- Gaussian random variables
- Simple to compute
- Easy to interpret relation

Matrix Review

- Consider

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

- 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:

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

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

- Then, predicted value is: $\hat{y} = A\beta$

Slopes and Intercept

- Model $\hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k$

- Divide coefficients into two parts: $\boldsymbol{\beta} = (\beta_0, \boldsymbol{\beta}_{1:k})$
  - β_0 : Intercept
  - $\boldsymbol{\beta}_{1:k} = (\beta_1, \ldots, \beta_k)$: Slope coefficients

- Then, can rewrite model as: $\hat{y} = \beta_0 + \boldsymbol{\beta}_{1:k}^T \boldsymbol{x}$

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 $\hat{y}_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_k x_{ik}$
  - Predicted value on sample i for parameters $\boldsymbol{\beta} = (\beta_0, \dots, \beta_k)$
- Define average residual sum of squares:

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

  - 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

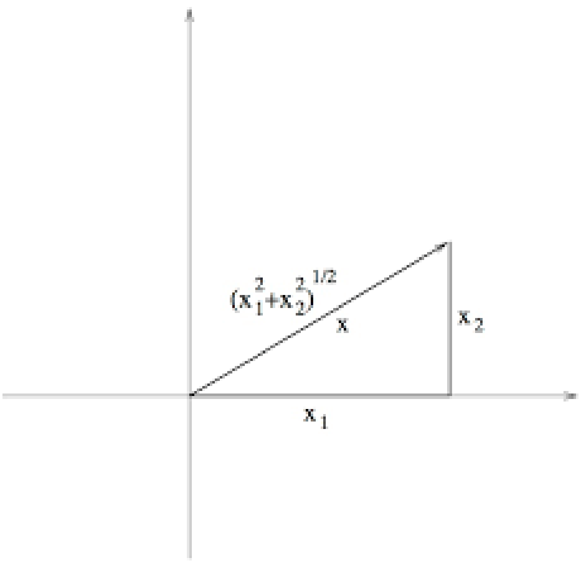
General ML problem		Multiple linear regression
❑Pick a model with parameters	→	Linear model: $\hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k$
❑Get data	→	Data: $(\boldsymbol{x}_i, y_i), i = 1,2, \dots, N$
❑Pick a loss function ◦ Measures goodness of fit model to data ◦ Function of the parameters	→	Loss function: $RSS(\beta_0, \dots, \beta_k) \coloneqq \sum (y_i - \hat{y}_i)^2$
❑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:
  - $\|\boldsymbol{x}\| = \left(x_1^2 + \cdots + x_r^2\right)^{1/2}$
  - Standard Euclidean norm.
  - Sometimes called ℓ-2 norm. ℓ is for Lebesque



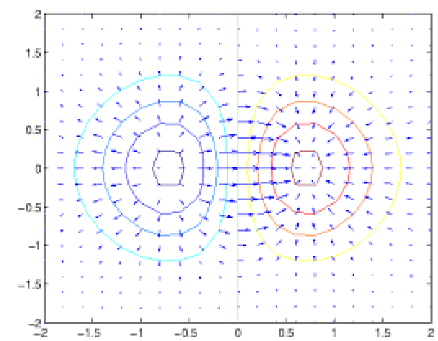
- Write RSS in vector form:

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

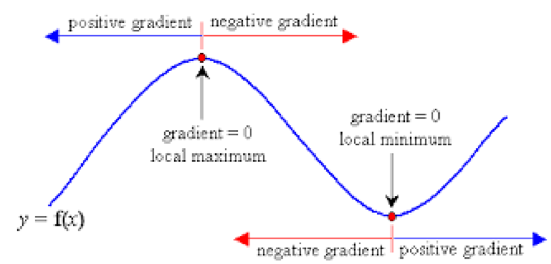
Gradients and Multi-Variable Functions

- Consider scalar valued function of a vector: $f(\boldsymbol{x}) = f(x_1, \ldots, x_n)$
- Gradient is the column vector:

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



- Ex: $f(x_1, x_2) = x_1 \sin x_2 + x_1^2\, x_2$.
  - Compute $\nabla f(\boldsymbol{x})$. Solution on board
- Represents direction of maximum increase
- At a local minima or maxima: $\nabla f(\boldsymbol{x}) = 0$
  - Solve n equations and n unknowns



Least Squares Solution

- Consider cost function of the RSS:

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

  - Vector β that minimizes RSS called the least-squares solution

- Compute partial derivatives via chain rule: $\frac{\partial RSS}{\partial \beta_j} = 2\sum_{i=1}^{n}(y_i - \hat{y}_i)A_{ij}$
- Matrix form of the gradient: $\nabla RSS = 2A^T(y - A\beta)$
- Least squares solution: $\beta = \left(A^T A\right)^{-1} A^T y$
- Minimum RSS: $RSS = y^T \left[I - A\left(A^T A\right)^{-1} A^T\right] y$

LS Solution via Auto-Correlation Functions

- Each data sample has a linear feature vector:
$$A_i = (A_{10}, \cdots, A_{ik}) = (1, x_{i1}, \cdots, x_{ik})$$

- Define sample auto-correlation matrix and cross-correlation vector:
  - $R_{AA} = \frac{1}{n} A^T A,\ R_{AA}(\ell, m) = \frac{1}{n} \sum_{i=1}^{n} A_{i\ell} A_{im}$
  - $R_{Ay} = \frac{1}{n} A^T y,\ R_{yA}(\ell) = \frac{1}{n} \sum_{i=1}^{n} A_{i\ell} y_i$

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

R^2: Goodness of Fit

- Define output sample mean and variance:

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

- 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:

$$R^2 = 1 - \frac{RSS/n}{s_y^2} = 1 - \frac{\text{avg error with linear model}}{\text{avg error with a constant model}}$$

  - $R^2 \in [0,1]$ 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: $\mathrm{RSS} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- RSS per sample: $\mathrm{RSS}/_n$
- Normalized RSS:

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

LS Solution via Covariance Matrices

- Define sample means $\bar{\boldsymbol{x}} = (\bar{x}_1, \cdots, \bar{x}_k)^T, \quad \bar{y}$
- Define sample covariance matrix and cross-covariance vector:
  - $S_{xx}(\ell, m) = \frac{1}{n}\sum_{i=1}^{n}(x_{i\ell} - \bar{x}_\ell)(x_{im} - \bar{x}_m)$
  - $S_{xy}(\ell) = \frac{1}{n}\sum_{i=1}^{n}(x_{i\ell} - \bar{x}_\ell)(y_i - \bar{y})$
- Write parameters as $\boldsymbol{\beta} = (\beta_0, \boldsymbol{\beta}_{1:k})$
  - $\boldsymbol{\beta}_{1:k} = (\beta_1, \dots, \beta_k)^T$ = coefficients for the values x_j
  - β_0 = constant term
- With some long algebraic manipulations (not in this class):

$$\boldsymbol{\beta}_{1:k} = S_{xx}^{-1} S_{xy}, \qquad \beta_0 = \bar{y} - \boldsymbol{\beta}_{1:k}^T \bar{\boldsymbol{x}}$$

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.

Calling the sklearn method

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

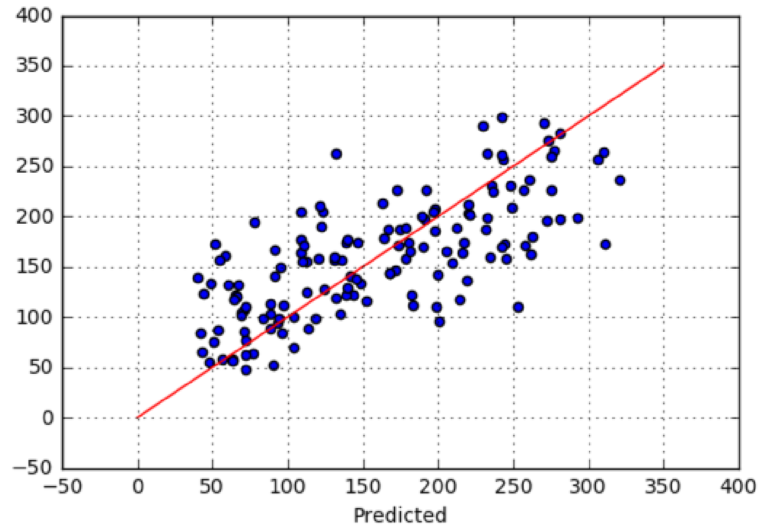
- ❑Construct a linear regression object
- ❑Run it on the training data
- ❑Predict values on the test data

```
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)
Rsq_test = 1-RSS_test
print("RSS per sample = {0:f}".format(RSS_test))
print("R^2 =            {0:f}".format(Rsq_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 s

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



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]
```

- Can also use numpy linear algebra routine to solve

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

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

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

- **Multiple linear regression**: Multiple predictors
  - Vector predictor $\boldsymbol{x} = (x_1, \dots, x_k)$
  - Linear model: $\hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k$
  - 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:

$$y = a_1 + b_1 x_1$$
$$y = a_2 + b_2 x_2$$
$$\vdots$$

- 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:

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

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

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

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

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

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]))
```

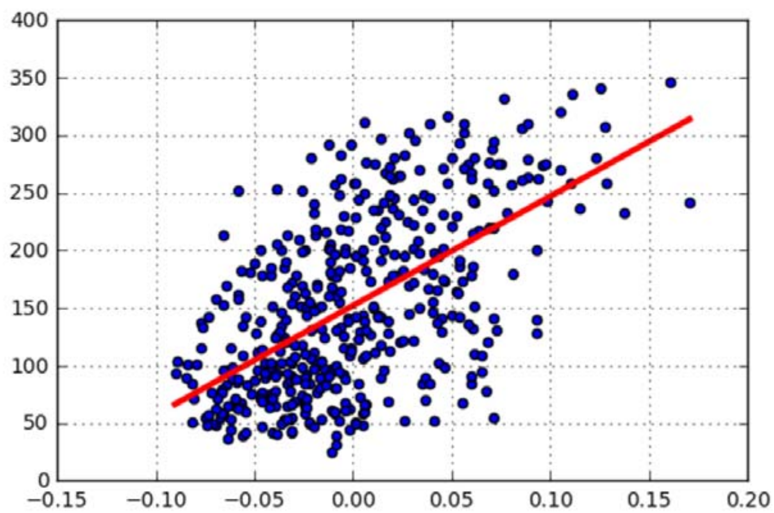
```
 0  Rsq=0.035302
 1  Rsq=0.001854
 2  Rsq=0.343924
 3  Rsq=0.194908
 4  Rsq=0.044954
 5  Rsq=0.030295
 6  Rsq=0.155859
 7  Rsq=0.185290
 8  Rsq=0.320224
 9  Rsq=0.146294
```

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

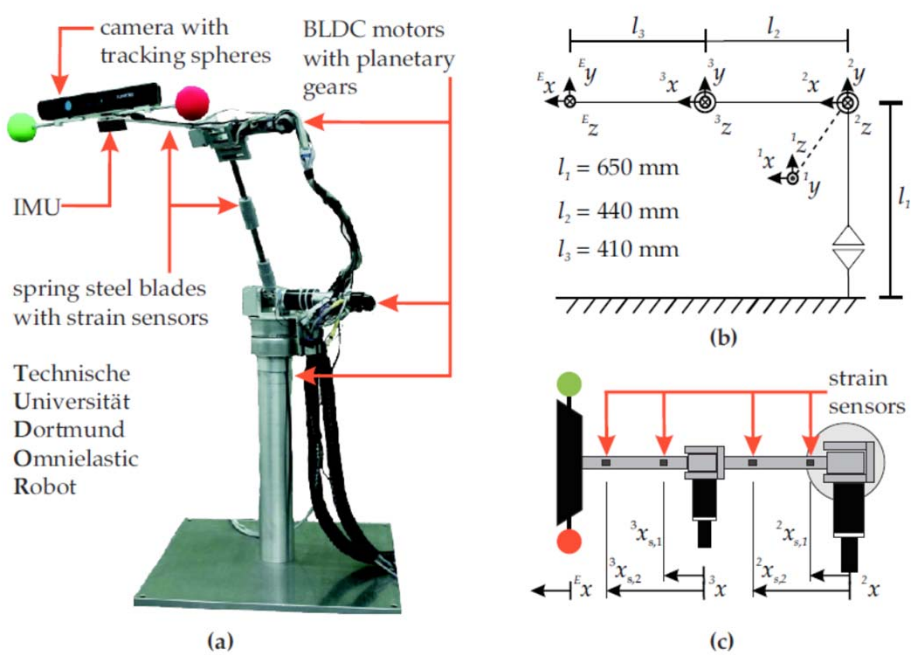



```python
# Find the index of the single variable with the best R^2
imax = np.argmax(Rsq)

# 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

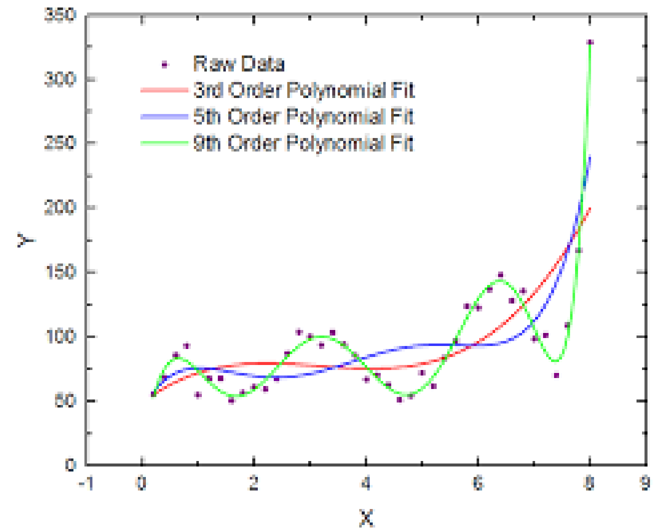
- Motivating Example: Understanding glucose levels in diabetes patients
- Multiple variable linear models
- Least squares solutions
- Computing in python
- Extensions

Polynomial Fitting

- Learn a polynomial model $y \approx \beta_0 + \beta_1 x + \cdots \beta_d x^d$
- Given data $(x_i, y_i), i = 1, \dots, n$
- Form feature matrix and coefficient vector

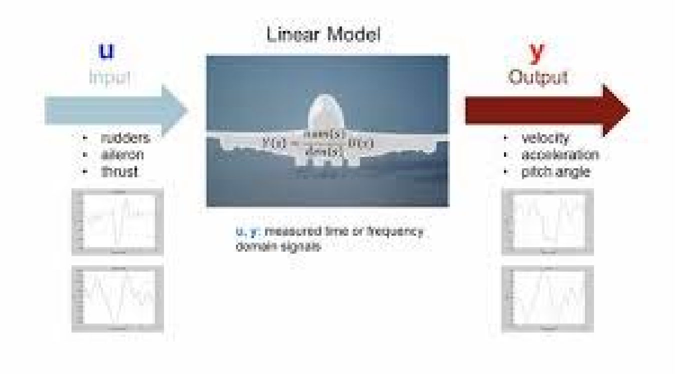
$$\boldsymbol{A} = \begin{bmatrix} 1 & x_1 & \cdots & x_1^d \\ \vdots & \vdots & \cdots & \vdots \\ 1 & x_n & \cdots & x_n^d \end{bmatrix}, \qquad \beta = \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_d \end{bmatrix}$$

  - $p = d + 1$ transformed features from 1 original feature
- Will discuss model order selection in next year
- Extensions to other nonlinear transforms



Learning Linear Systems

- Linear system: $y_k = a_1 y_{k-1} + \cdots + a_m y_{k-m} + b_0 x_k + \cdots + b_n x_{k-n} + w_k$
- Transfer function: $H(z) = \frac{b_0 + \cdots + b_n z^{-n}}{1 + a_1 z^{-1} + \cdots + a_m z^{-m}}$
- Learn transfer function from data $(x_k, y_k), k = 0, \ldots, T-1$
- Unknown coefficients $\beta = (a_1, \cdots, a_m, b_0, \cdots, b_n)^T$
- Write $y = A\beta + w$ for matrix A
  - See homework problem
- Many applications
  - Learning dynamics in robots / mechanical systems
  - Modeling responses in neural systems
  - Stock market time series
  - Speech modeling. Fit model each 25 ms.



One Hot Coding

- Suppose that x is a categorical variable
  - One of a finite number of choices $x \in \{1, \ldots, M\}$
  - Example: male or female, model of a car, …

Model	u_1	u_2	u_3
Ford	0	0	0
BMW	1	0	0
GM	0	1	0
VW	0	0	1

- One-hot coding example: Car model
  - Model: $y = \beta_0 + \beta_1 u_1 + \cdots + \beta_{M-1} u_{M-1} + \cdots$
  - Obtain M different models:
    - Ford: $y = \beta_0 + \cdots$
    - BMW: $y = \beta_0 + \beta_1 + \cdots$
    - GM: $y = \beta_0 + \beta_2 + \cdots$
    - …