Artificial Intelligence

University of East Anglia

Seminar Topics of CW2 :

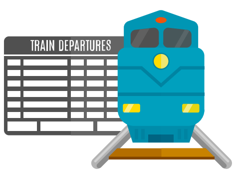
1. **Web Scraping**
2. **Train Delay Prediction**
3. **Regression Analysis**
4. **Train Service Data**
5. **Knowledge Engines**

Douglas Fraser
& Mas Golchehreh

Images source: https://www.freepik.com

Developing An Intelligent Chatbot

1. Finding the cheapest train ticket

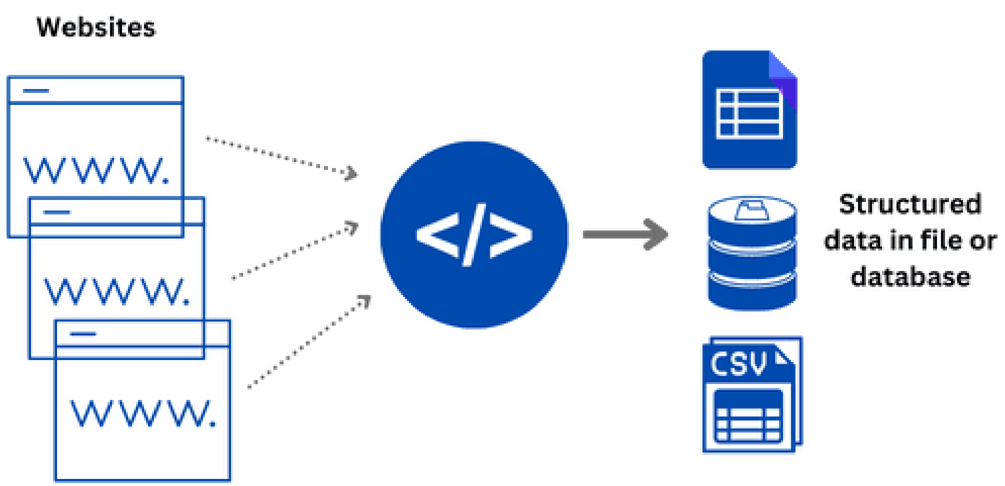


2. Improving Customer Service:
Train Delay Predictor

3. Providing advice for dealing with contingencies (**PGT student**)

1.**Web Scraping and Fare Information**

- Two alternative sources of fare information
- Quick overview of web scraping
- Useful Links



Sources of Fare Information

Access to the Realtime Journey Planner costs money – NR profits from fare information

Option 1: brfares.com

HTML easy to parse (simple tables) and also has JSON based API

- static information on tickets available for a trip from station A to station B
- easier to use, but only provides generic information on ticketing options
- does not indicate if a ticket actually available or options like reserved seating
- does not provide time of train information
- have to use Network Rail documentation to understand the data

LONDON OVERGROUND, WEST MIDLANDS TRAINS

SINGLE FARES

Ticket	Route	Validity	Adult	Child
ANYTIME 1S FOS	VIA PETERBOROUGH	TWO DAYS	£304.20	£152.10
Ticket issued to: LONDON TERMINALS			Fare Setter: LONDON NORTH EASTERN RAILWAY	
ANYTIME 1S GFS	LNER ONLY	TWO DAYS	£303.20	£151.60
			Fare Setter: LONDON NORTH EASTERN RAILWAY	
CARNET 5 1ST CN1	LNER ONLY	THREE MONTHS Restrictions **1Z** IEC CARNETS	£1,212.80	
VALID ANYTIME. NO REFUNDS			Fare Setter: LONDON NORTH EASTERN RAILWAY	
Transfer Fee 1C 1TF	LNER ONLY	BOOKDTRAINONLY (no break of journey) Restrictions **GC** IEC ADVANCE	£152.10	
VALID ON DATE&TRAIN SHOWN ONLY.LTD CHANGE.NO RFND			Fare Setter: LONDON NORTH EASTERN RAILWAY	

Origin	7728	**NEWCASTLE**	NCL
Destination	6121	**LONDON KINGS X**	KGX
Railcard / Discount	PUBLIC		
Fares Period	3 Mar 24—31 Mar 24		

New | Amend | Expert Mode | Reverse

Sources of Fare Information

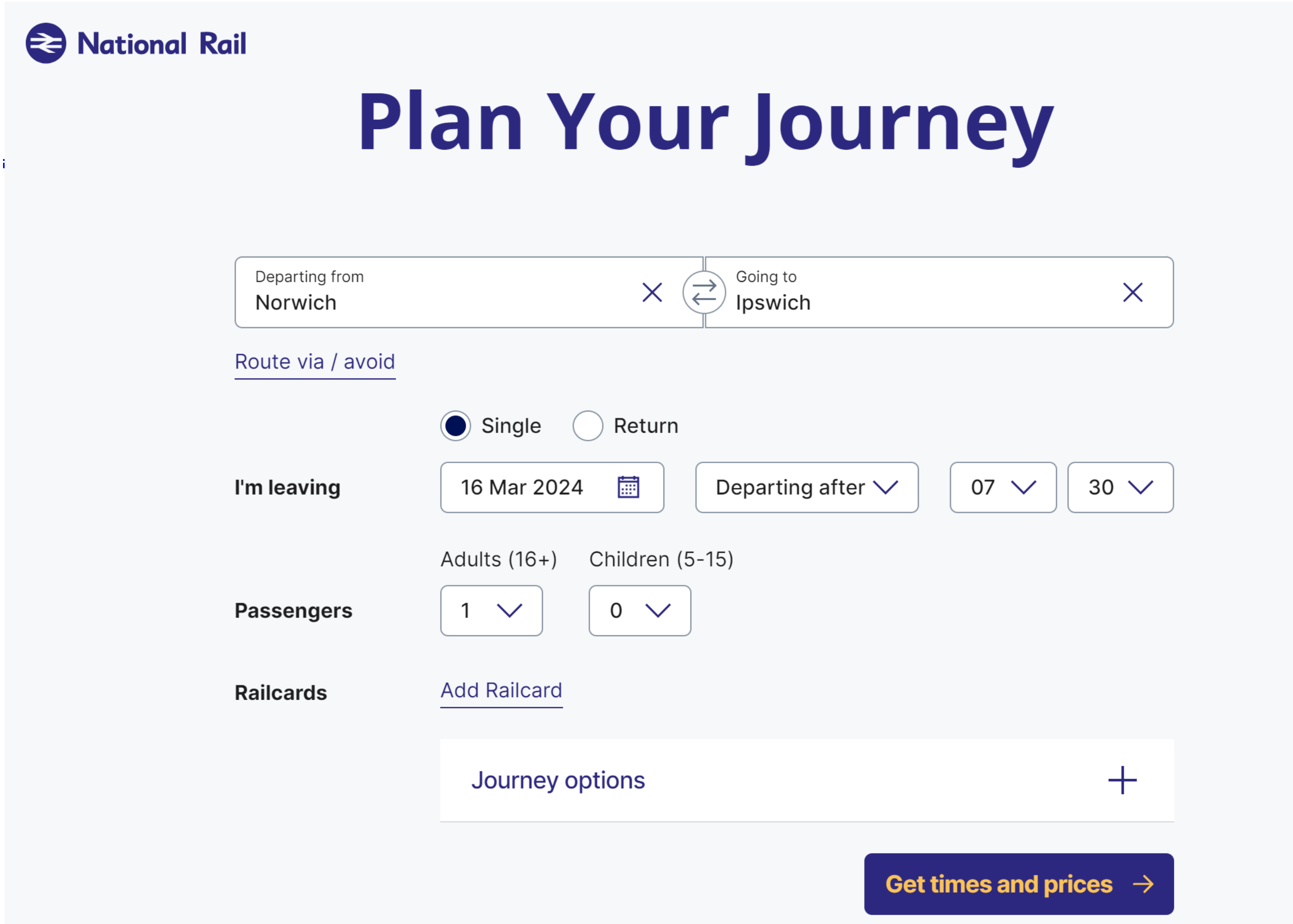
Option 2: web scraping of ticketing sites

- any number of websites could be used
  nationalrail.co.uk, greateranglia.co.uk, trainsplit.com, etc.

- slightly more work, but lots of tutorials / information on the web

- more flexible in type of data that can be gathered
  e.g. available train times, seating options, restrictions

National Rail greateranglia


National Rail
Plan Your Journey
Departing from
Norwich
Going to
Ipswich
Route via / avoid
Single
Return
I'm leaving
16 Mar 2024
Departing after
07
30
Adults (16+)
Children (5-15)
Passengers
1
0
Railcards
Add Railcard
Journey options
Get times and prices

Basics of web scraping

1. **Identify Websites:**

   Find websites with train ticket info.

2. **Understand Structure:**
   - Reverse engineer HTML of pages to figure out form fields to submit
   - Code makes a HTTP call (GET or POST), submitting information
   - Process HTML that is returned to find desired information
     - regex a popular way to do this
     - have to reverse engineer the HTML that is returned

3. **Choose Tool:**

   Pick a web scraping tool like BeautifulSoup, Scrapy, or Selenium in Python.

4. **Write Code and Extract Data:**

   Develop code to send requests to the site, grab HTML content, and extract the needed info based on its structure.

5. **Clean Data:**

   Check and clean the extracted data.

Things to Consider

Complexity of the site

- static pages are easiest
- dynamic pages with JavaScript or AJAX can be difficult to process
  headless browsers are a solution (e.g. Selenium or PhantomJS)

Complexity of the HTML

- using regex is sometimes difficult or not a good idea (generally)
- DOM or HTML / XML based processing a possible solution

Cookies, tracking IDs, etc.

- sites use these to track visitors or save state information
- so using them in process of fetching webpage necessary
- IE / Microsoft focused sites can be problematic to deal with

Useful Links

- Open Rail Data Google group

  https://groups.google.com/forum/#!forum/openraildata-talk

- NRE Disruptions Web Service

  need an account – may have to wait to get it (if available)

- NRE Knowledgebases (Incidents, ticket info…)

  static info, except for incidents feed

- RealTimeTrains: http://www.realtimetrains.co.uk/

  planned schedules as well as actual performance

Tutorial Source for Web Scraping:
https://realpython.com/beautiful-soup-web-scraper-python/

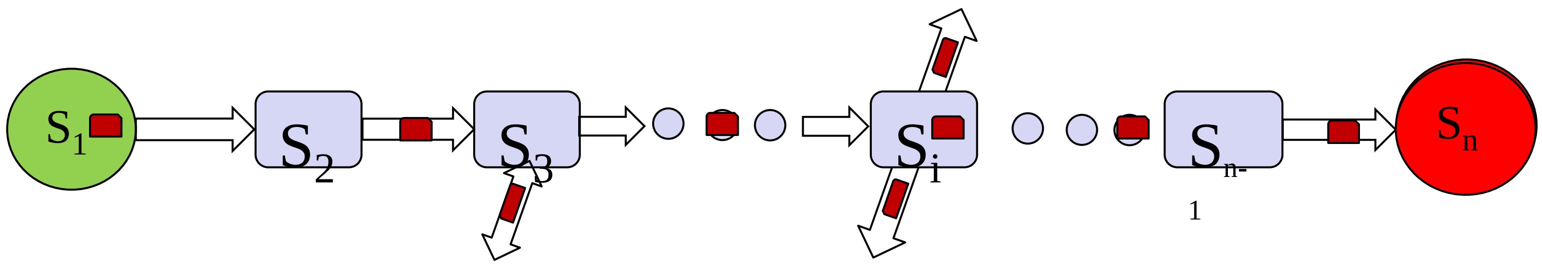
2. Train Delay Prediction

- Steps for Train Delay Prediction
  1. Devise a prediction model scheme
  2. Process the train running data
  3. Derive some input variables
  4. Transform the process data into training patterns
  5. Select learning algorithms
  6. Partition the data for training and testing
  7. Choose good models
  8. Use the chosen model for prediction.

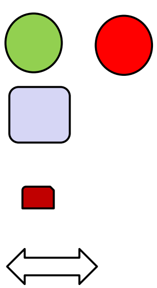
Rail network logical model

- For each journey J on a rail network, there are
  - n stations: $J = \{S_1, S_2, S_3, \ldots, S_i, \ldots, S_{n-1}, S_n\}$
  - m trains on the rail tracks between S_1 and S_n:

$$T = \{T_1, T_2, T_3, \ldots, T_m\}$$



Keys:



Departure and Destination stations

Middle stations:

Train

Rail Track

Prediction Task

- The prediction task: given the current time of a train T_x, at any station (or checkpoint) S_i, we want to predict
  (1) the arrival (and departure) time t_{ja} of this train at the next stop j and also its all the following stops:

$$T_x(t_a) = [\ t_{ja},\ t_{ka}\ , \ldots, t_{na}\ ]$$

  (2) And the arrival/departure times of all the other trains that may be affected by this train;

For train 1: $T_1(t_a) = [\ t_{1a},\ t_{2a}\ , \ldots,\ t_{na}\ ]$

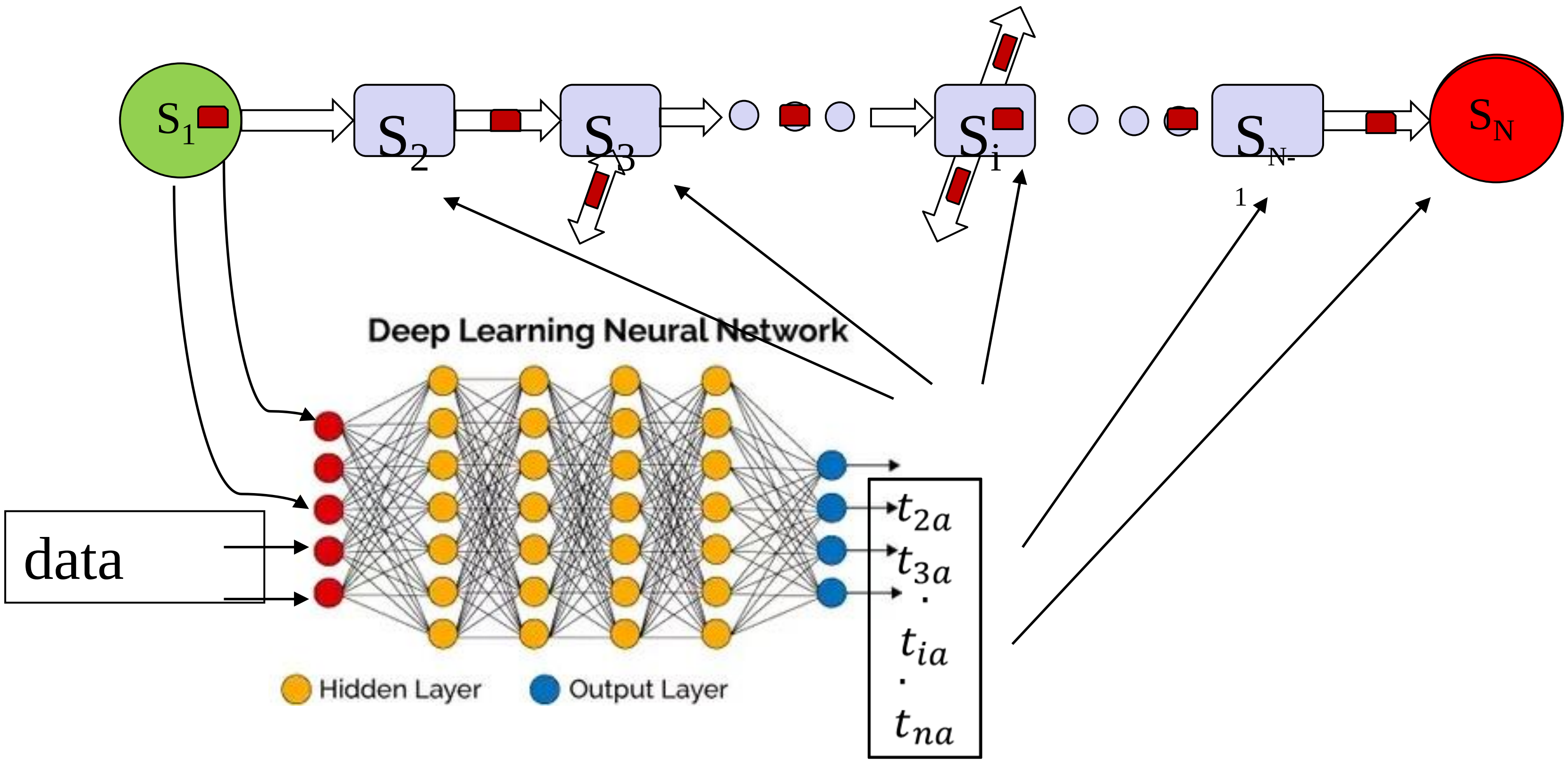
For train 2: $T_2(t_a) = [\ t_{1a},\ t_{2a}\ , \ldots,\ t_{na}\ ]$

……

For train m: $T_m(t_a) = [\ t_{1a}, t_{2a}\ , \ldots,\ t_{na}\ ]$

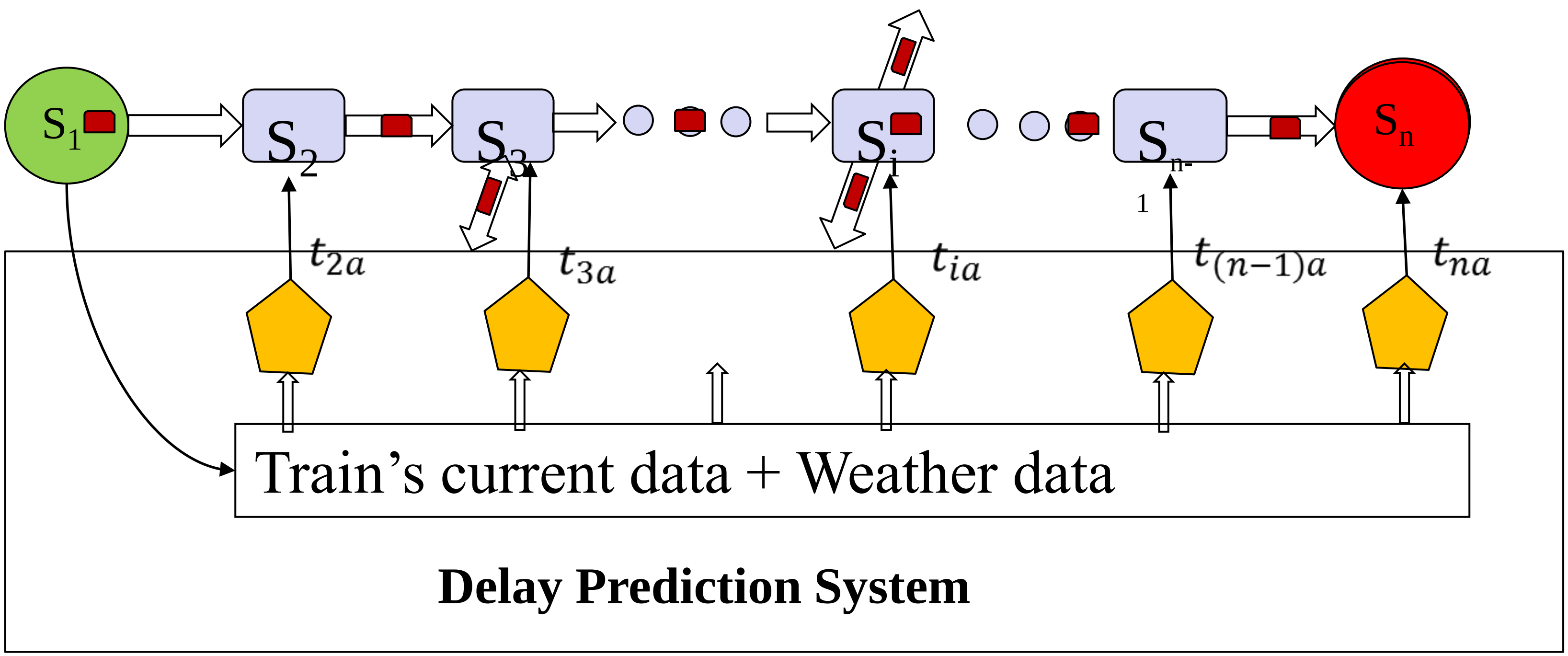
Prediction Strategy 1

- Two strategies have been devised and designed
- One big model that does all the predictions:



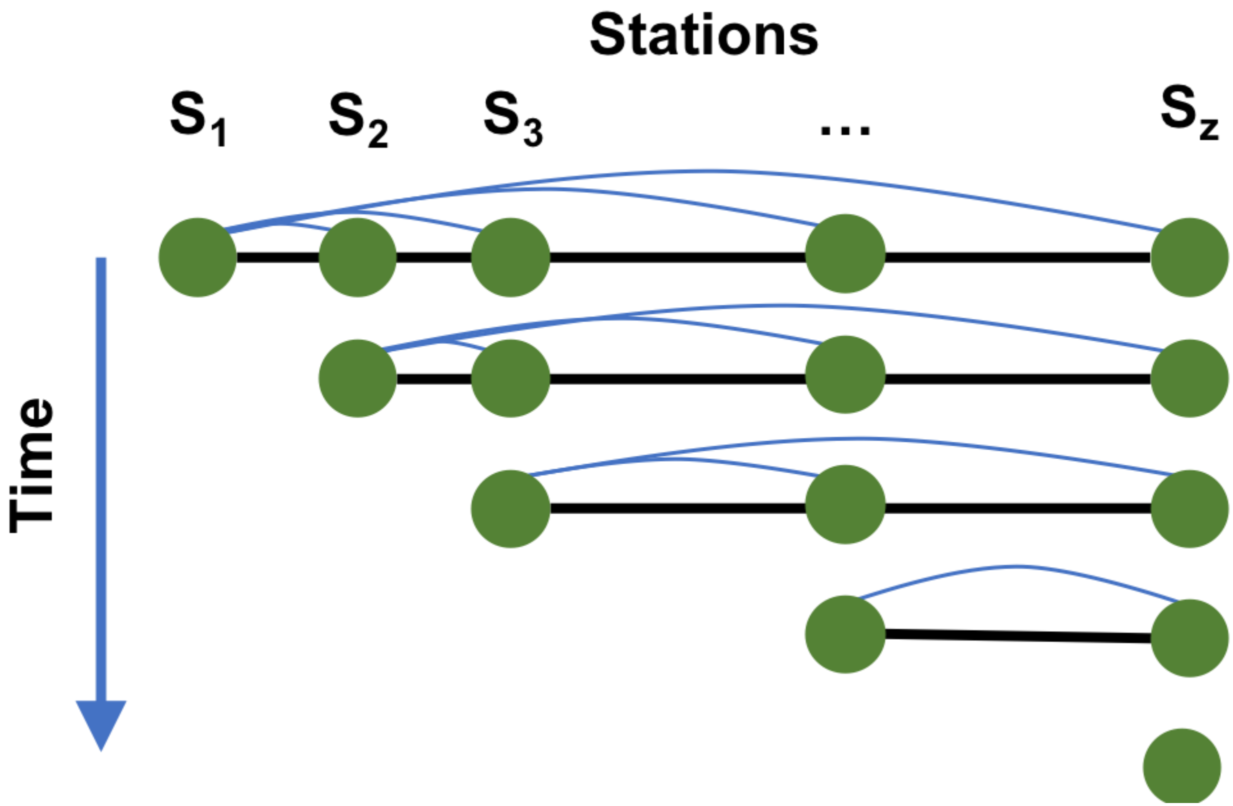
Prediction strategy 2

- Many smaller models - divide and conquer
  - One model predicts the delay time at one station
  - N models for n stations



Two strategies in building models

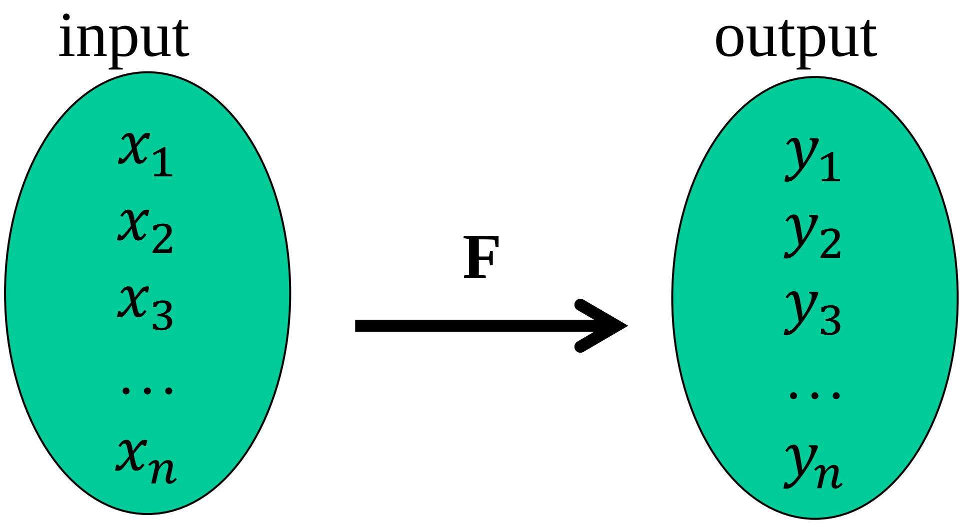
- For the entire rail journey,



3. Regression Analysis

- Regression: given a data set of n data points:

  D= $\{d_1, d_2, \ldots, d_n\}$, $d_i = (x_i, y_i)$, where, x is an input variable or a vector of m variables: X=[x1, x2, …, xm] and y is the output with continuous values.



- The task of regression is to find a function that represents the relationship between x and y. $y = \mathrm{f(x)}$

Regression example

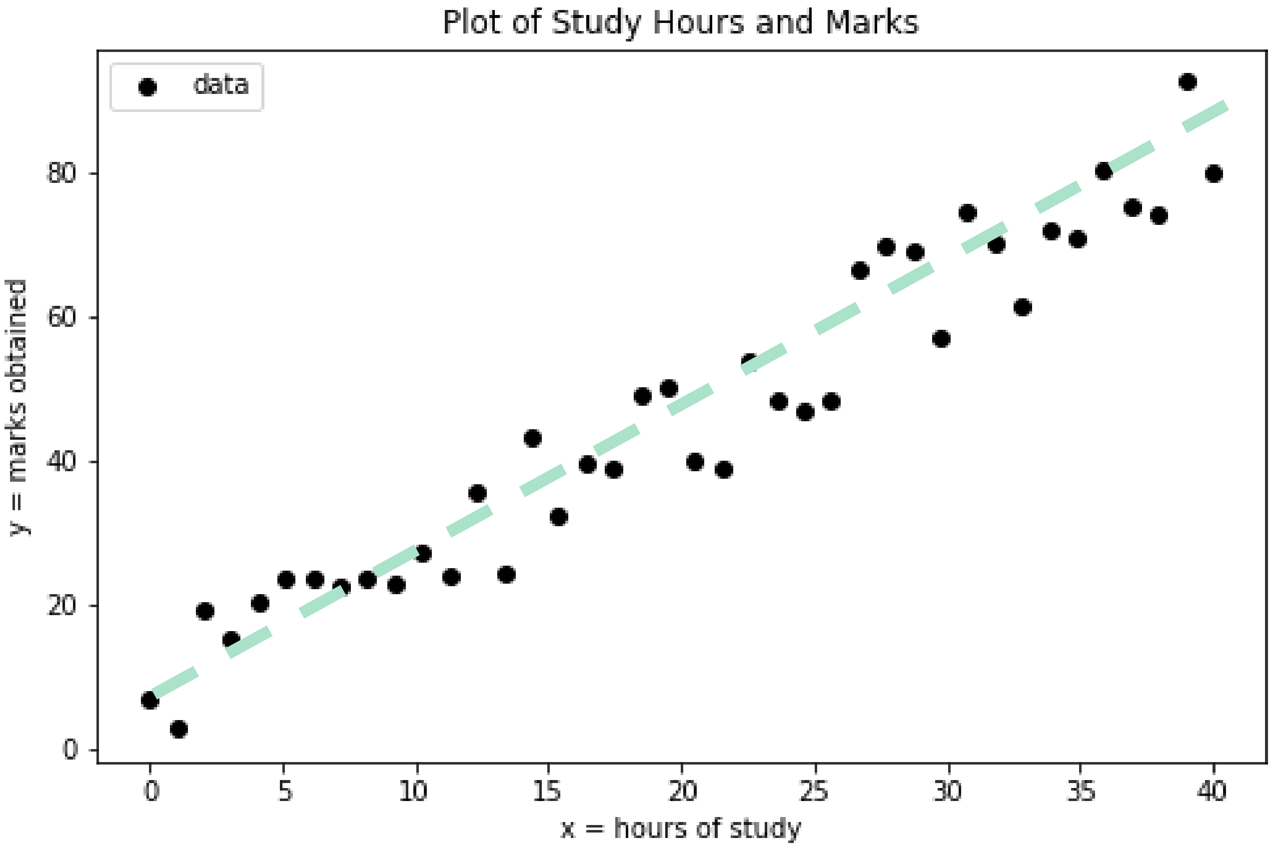
- We have collected the data (some are shown in the table) about the hours of revision (x), and the marks (y) obtained from an exam.

- Want to find a relationship between x and y.

x = hours	y= marks
…	…
15	32
20	40
23	48
25	48
26	67
28	69
29	57
30	75
38	93
40	85

Regression Example: data plot

- Plot the data using y against x.
- They appeared to have a linear relationship: $y = \mathrm{ax} + \mathrm{b}$



Linear Regression

Estimated *a* and ***b**:*

$$\hat{y} = \hat{a}x + \hat{b}$$

slope

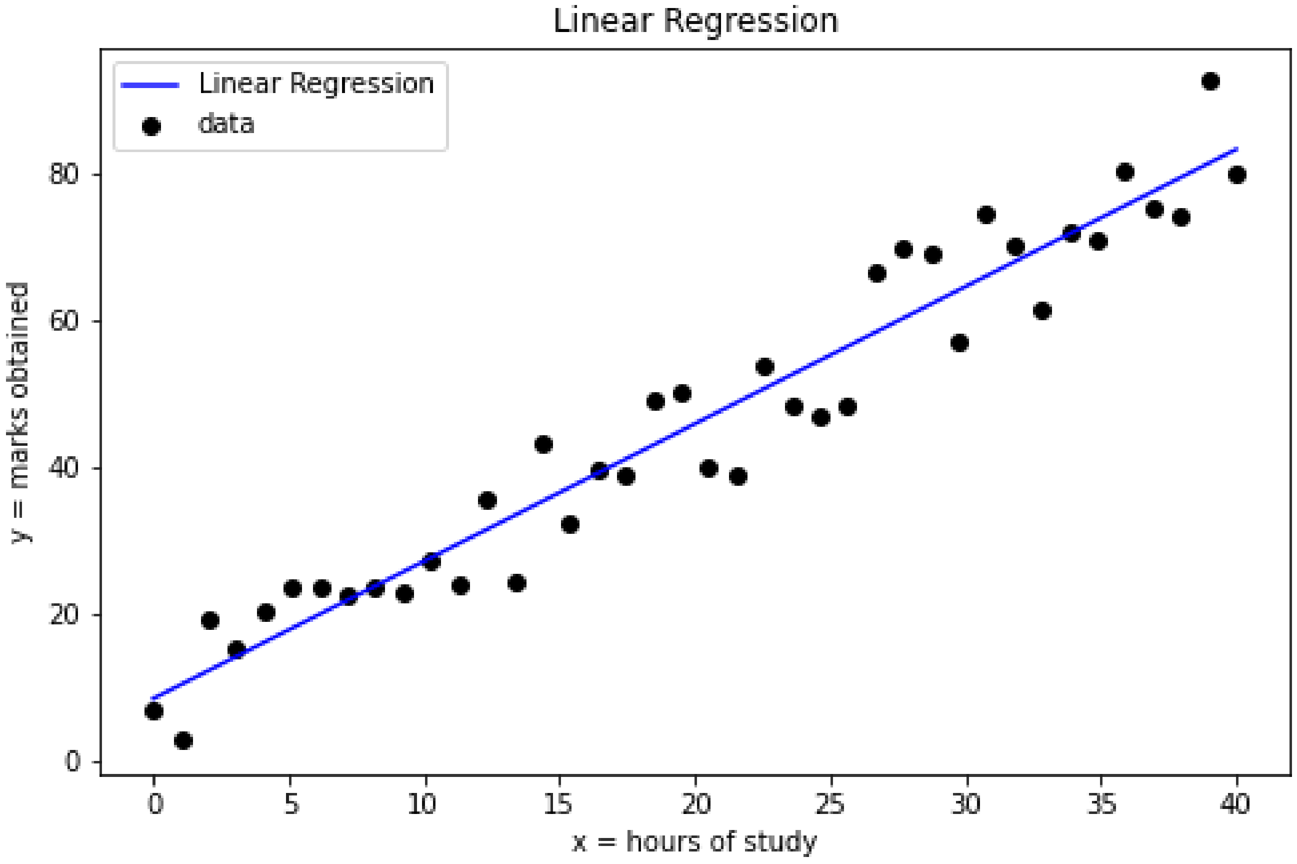
$$\hat{a} = \frac{\sum_{i=1}^{n}(y_i - \bar{y})^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

y-intercept

$$\hat{b} = \bar{y} - \hat{a}\bar{x}$$

Linear regression result

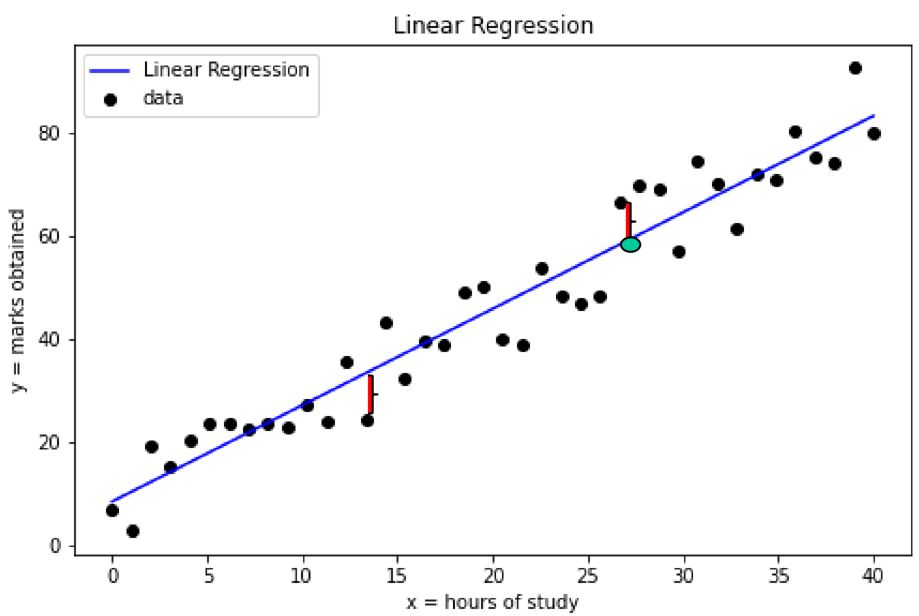
- Regression found: $\hat{a} = 1.9$, $\hat{b} = 8.4$, $\hat{y} = 1.9\text{x} + 8.4$



Linear Regression by Least Squares

- The residual errors between y and predicted ŷ

$$\varepsilon = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2$$



Linear Regression in Python

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
def f(x): #define a linear function
    return (2*x + 5).ravel()
np.random.seed(1) # set a seed for random number generator
# generate training data X
X = np.linspace(0, 80, 40)[:, np.newaxis]
y = f(X) + 20*(0.5-np.random.rand(40).ravel())
T = np.linspace(0,80,100)[:,np.newaxis] #generate test data
# Fit a linear regression model
lr = LinearRegression().fit(X, y)
y_lr = lr.predict(T)
print( "a, b = ", lr.coef_, lr.intercept_)
plt.scatter(X, y, color='black', label='training data')
plt.plot(T, y_lr, color = ‘b', label = ‘Linear Regression’)
plt.show()
```

kNN for Regression

- Given a dataset D={d_1, d_2, ,,, d_n}, d_i = (x_i, y_i)
- Set a value to k, e.g. k = 3

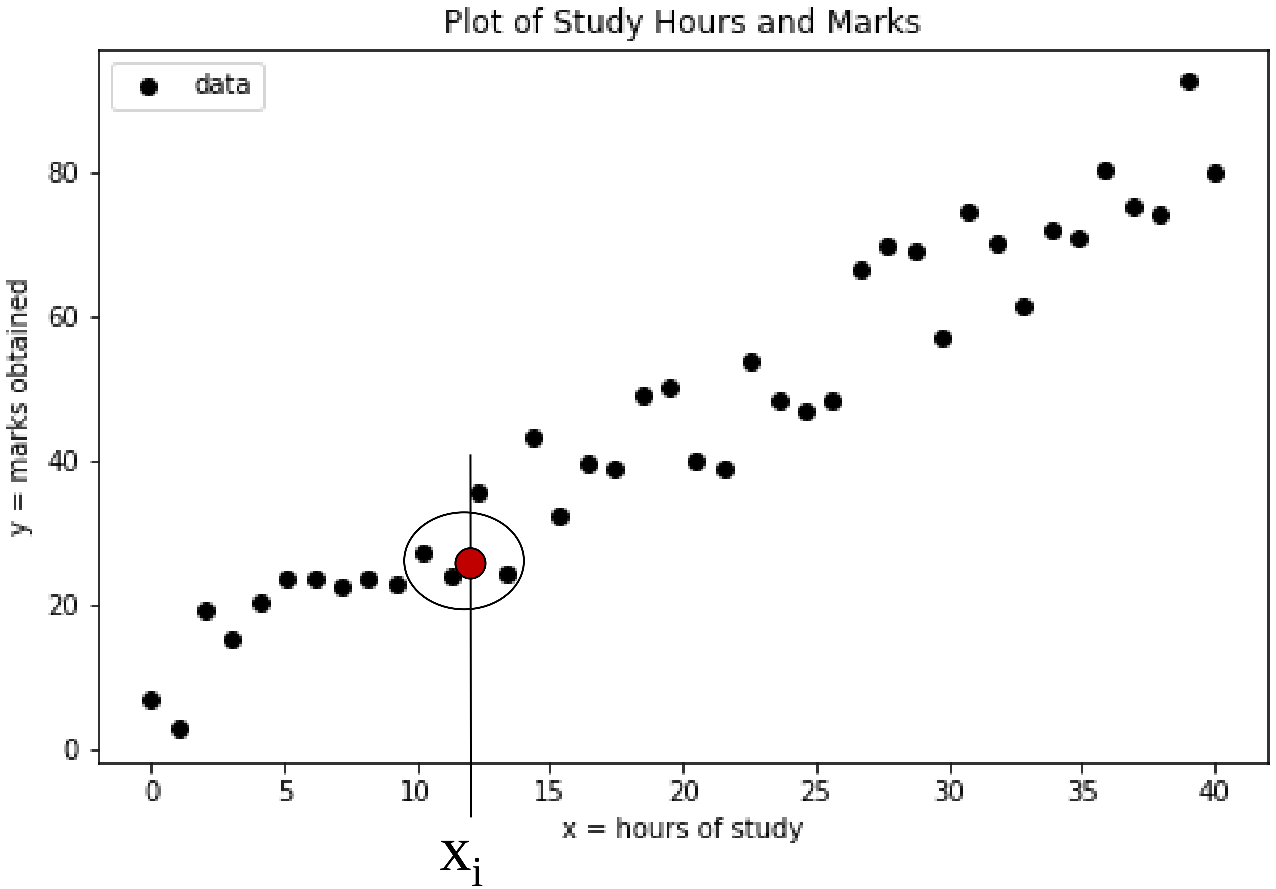
1. Take a data point $d_i = (x_i, y_i)$ from D,
2. Compute the distance: $dis(d_i, d_j), \text{ for } j \neq i$
3. *Choose k (= 3) nearest data points, e.g.* x_u , x_v , x_w
4. *Compute the mean of their outputs* y_u , y_v , y_w *as the predicted value for* x_i *, i.e.*

$$y_i\,(x_i) = (\,y_u + y_v + y_w\,)/\,k = \frac{1}{k}\sum_{q=1}^{k} y_q$$

5. *Repeat steps 1 and 4 until all data points visited.*

K-NN regression: illustration

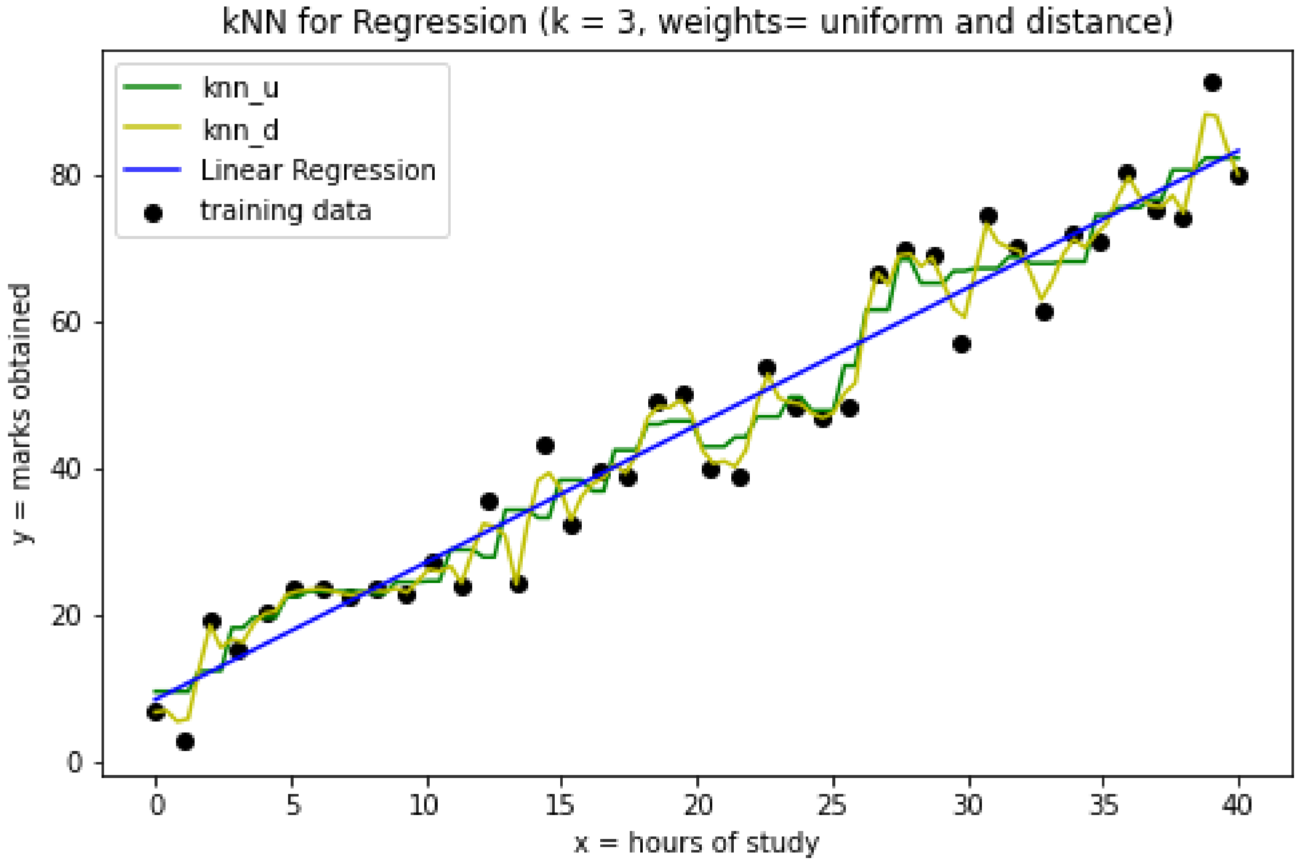
- 3 nearest data points are chosen, and their y values are averaged to be the output y_i of x_i



K-NN Regression: Results

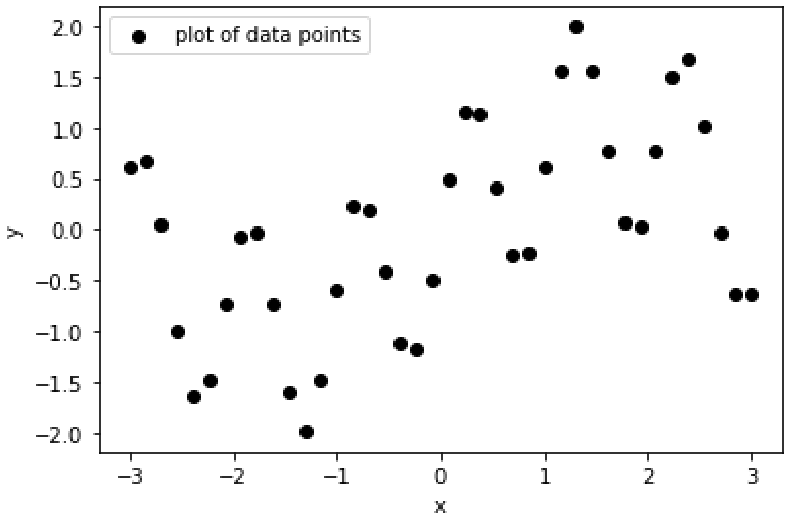
- Two kNN: normal kNN_u and weighted kNN_d
- Performance Measures: MSE and R2

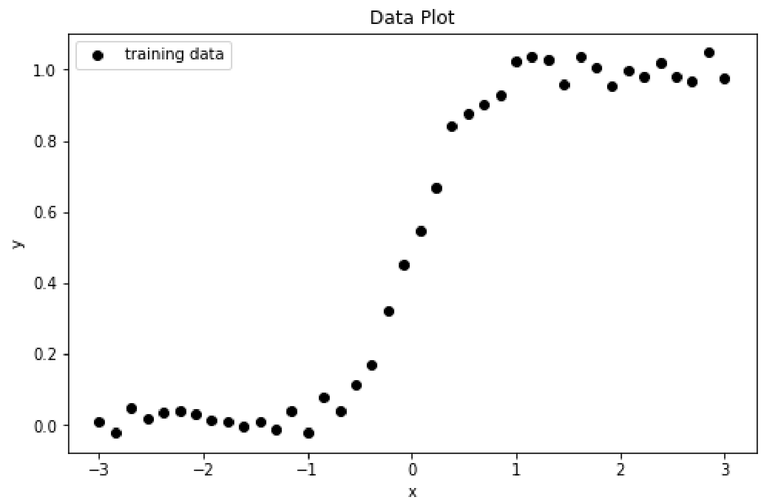
- Knn_u:
  12.96, 0.98
- knn_d:
  20.53, 0.96
- Linear:
  2.88, 0.99

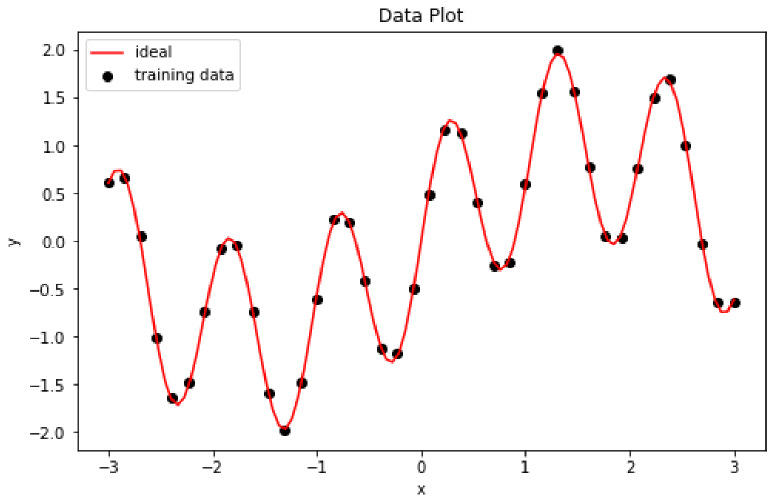


Non-linear Data and Regression

- The relationship between an output and inputs is not linear, e.g.







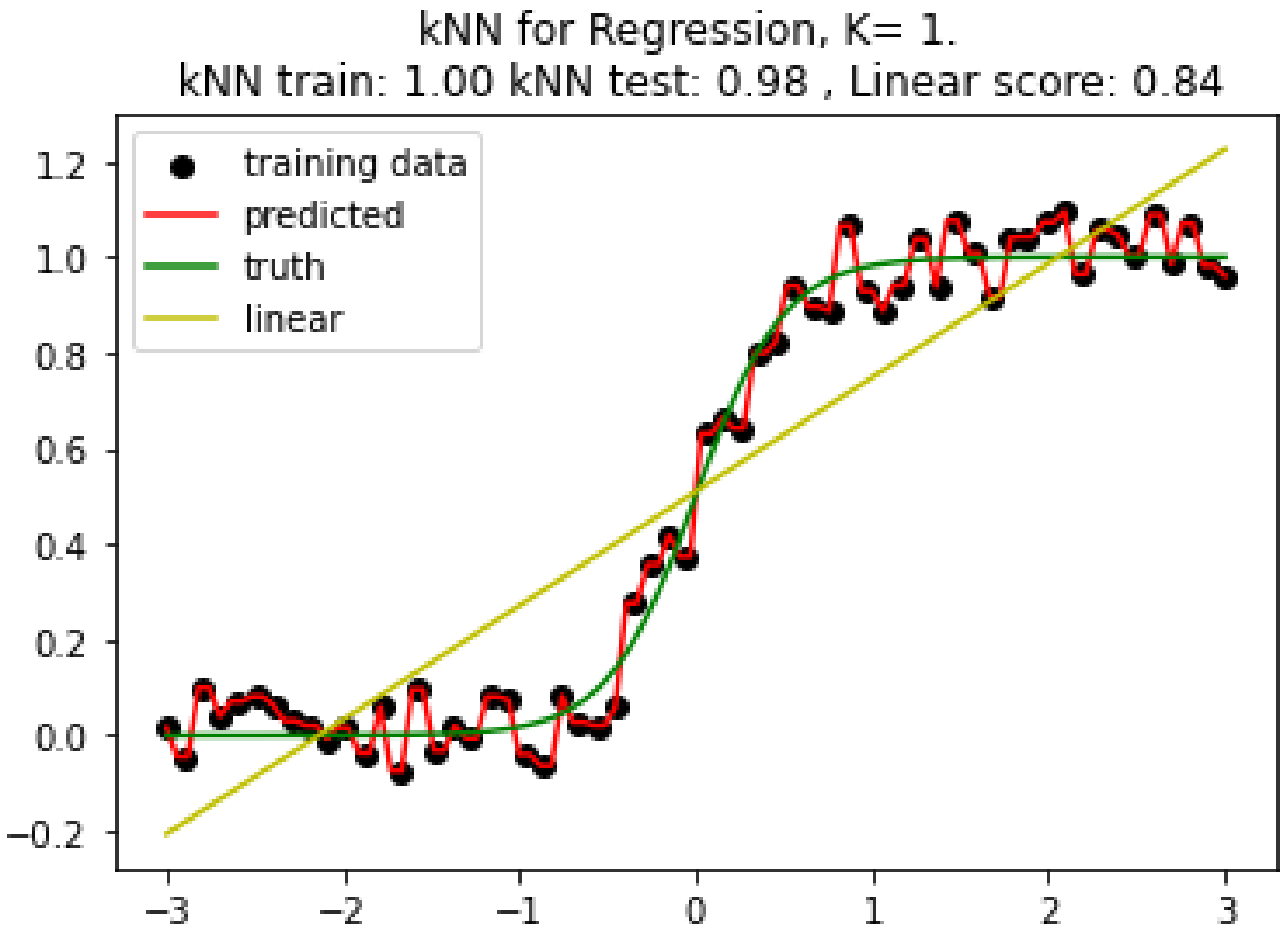
Typical Curve fitting models

- Non-linear regression
  - Such as logistic regression
- k-NN regression
  - Normal kNN, i.e. without weighting on data
  - Weighted kNN, weight data by distance
- Bayes regression
- Artificial neural networks
- Deep neural nets

Python kNN 4 Non-linear Regression

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import neighbors
def f(x): #define a non-linear function
    return 1/(1 + np.exp(-4*x)).ravel()
np.random.seed(1) # set a seed for random number generator
# generate training data X
X = np.linspace(-3, 3, 40)[:, np.newaxis]
y = f(X) + 0.2*(0.5-np.random.rand(40).ravel())
T = np.linspace(-3,3,100)[:,np.newaxis] #generate test data
# create a kNN model without weighting on data
knn_u = neighbors.KNeighborsRegressor(3, weights='uniform')
knn = knn_u.fit(X, y) # fit the model to training data
y_knn = knn.predict(T) # use the knn model on test data
plt.scatter(X, y, color='black', label='training data')
plt.plot(T, y_knn, color = 'r', label = 'predicted')
plt.show()
```

kNN for Non-linear Regression



kNN for Non-linear Regression

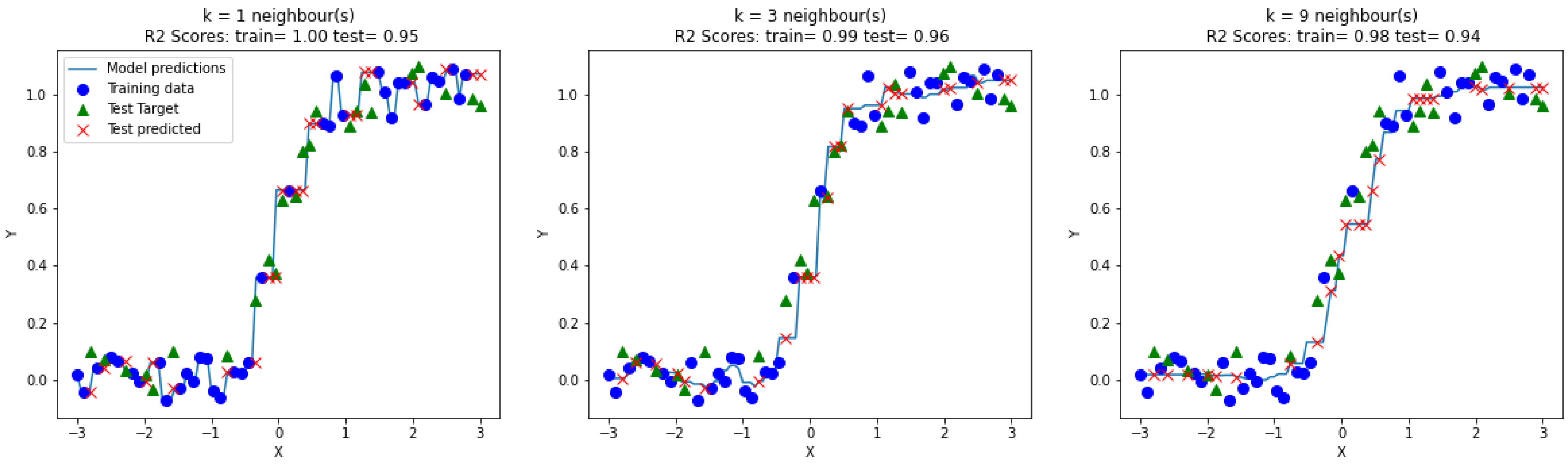
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
def f(x): #define a non-linear function
    return 1/(1 + np.exp(-4*x)).ravel()

np.random.seed(1) # set a seed for random number generator
N= 60
# generate training data X and compute the targets y
X = np.linspace(-3, 3, N)[:, np.newaxis]
y = f(X) + 0.2*(0.5-np.random.rand(N).ravel()) #added noise
T = np.linspace(-3,3,200)[:,np.newaxis] #generate test data
# split the dataset into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.40, random_state=0)
```

kNN for Non-linear Regression

```
fig, axes = plt.subplots(1, 3, figsize=(20, 5))
for n , ax in zip([1, 3, 9], axes):
    #make predictions using K = 1, 3, or 9 neighbours
    reg = KNeighborsRegressor(n_neighbors = n)
    reg.fit(X_train, y_train) # fit a knn model
    knn_test = reg.predict(X_test)
    ax.plot(T, reg.predict(T)) # test with new data points
    ax.plot(X_train, y_train, '^', c='blue', markersize=8)
    ax.plot(X_test, y_test, 'o', c='green', markersize=8)
    ax.plot(X_test, knn_test, 'x', c='red', markersize=8)
    ax.set_title("k = {} neighbour(s)\n R2 Scores: train={:.2f}
       test={:.2f}".format(n ,reg.score(X_train, y_train) ,
       reg.score(X_test, y_test)))
    ax.set_xlabel("Feature")
    ax.set_ylabel("Target")
    axes[0].legend(["Model predictions", "Training data", "Test
Target", ”Test predicted"], loc="best")
```

kNN with different k values

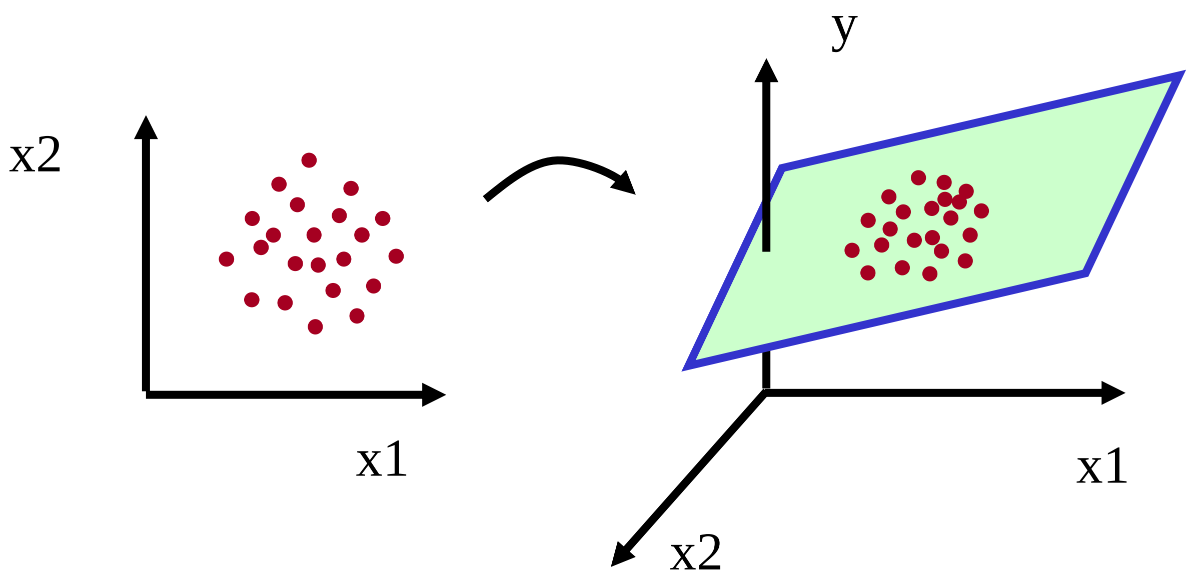
- K = 1, looks quite good, with test accuracy(R2)=0.95
- K = 3, maybe the best, test R2 = 0.96
- K = 9, overfitting, test R2 = 0.94



LR for multi-dimensions

- When there are more inputs, e.g. two inputs x1 and x2: the liner function will be a plane.

$$\hat{y} = a1*x1 + a2*x2 + b + \varepsilon$$



4. Train Running Data

- Historic Service Performance Web Service
- Network Rail's DARWIN
- UEA's DARWIN database

Historic Service Performance (HSP)

- National Rail Enquiries Web Service – need an account
- Docs: https://wiki.openraildata.com/index.php/HSP
- Example client: `hsp_api_example.py`

- Step 1: Get Train RIDs based on origin / destination / date/time of day
- Step 2: Get performance data (stops / planned times / actual times) based on a train RID
- Also the late reason or cancelled reason code for the train service

How actual journeys map to HSP data

sched_rid	dateof	stop_num	loc	pla_d	pla_a	act_d	act_a
201701067101243	2017-01-06	1	NRW	05:30:00	(NULL)	05:29:00	(NULL)
201701067101243	2017-01-06	2	DIS	05:48:00	05:47:00	05:48:00	05:46:00
201701067101243	2017-01-06	3	SMK	06:00:00	05:59:00	06:01:00	06:00:00
201701067101243	2017-01-06	4	IPS	06:14:00	06:12:00	06:14:00	06:11:00
201701067101243	2017-01-06	5	MNG	06:24:00	06:23:00	06:24:00	06:23:00
201701067101243	2017-01-06	6	COL	06:35:00	06:33:00	06:34:00	06:32:00
201701067101243	2017-01-06	7	SRA	(NULL)	07:15:00	(NULL)	07:16:00
201701067101243	2017-01-06	8	LST	(NULL)	07:27:00	(NULL)	07:25:00

Network Rail's DARWIN

Real-time data feed of rail network data

- 10 different types of XML messages
  - Planned schedules, schedule updates,
  - train performance, etc.

- Also forecasts about when trains will arrive...
  - But utilizing this data is not necessary (unless you really, really want to)
- Docs: https://wiki.openraildata.com/index.php?title=Darwin:Push_Port

UEA's DARWIN database

- ~6 years of DARWIN data
  - Area51: 2017 to 2022: https://archive.area51.dev/archive/
- Condensed performance updates for London Liverpool Street to Norwich
  - Equivalent+ to output from HSP web service
  - Passing points included and late/cancelled reason codes separated

DARWIN tables and fields

Some data attribute in DARWIN data table

rid	Train RTTI Train Identifier	**arr_et**	Estimated Arrival Time	**dep_wet**	Working Estimated Time
tpl	Location TIPLOC	**arr_wet**	Working Estimated Time	**dep_atRemoved**	true if actual replaced by estimated
pta	Planned Time of Arrival	**arr_atRemoved**	true if actual replaced by estimated	**arr_at**	Recorded Actual Time of Arrival
ptd	Planned Time of Departure	**pass_et**	Estimated Passing Time	**pass_at**	Actual Passing Time
wta	Working (staff) Time of Arrival	**pass_wet**	Working Estimated Time	**dep_at**	Actual Departure Time
wtp	Working Time of Passing	**pass_atRemoved**	true if actual replaced by estimated	**cr_code**	Cancellation Reason Code
wtd	Working Time of Departure	**dep_et**	Estimated Departure	**lr_code**	Late Running Reason

Data of one train: Norwich to LST

Departure from Norwich: 05:00, Arrived at London Liverpool Str 06:55

rid	tpl	pta	ptd	wta	wtp	wtd	pass_et	arr_at	pass_at	dep_at
201802051053467	NRCH		05:00			05:00				05:00
201802051053467	NRCHTPJ				05:01		05:01			
201802051053467	TRWSSBJ				05:01:30		05:01			
201802051053467	TROWSEJ				05:02:30				05:02	
201802051053467	DISS	05:17	05:18	05:16:30		05:18		05:17		05:19
201802051053467	HAGHLYJ				05:26:30				05:27	
201802051053467	STWMRKT	05:29	05:30	05:29		05:30:30		05:30		05:31
201802051053467	IPSWEPJ				05:39		05:38			
201802051053467	IPSWESJ				05:40				05:39	
201802051053467	IPSWICH	05:42	05:44	05:42		05:44		05:41		05:43
201802051053467	IPSWHJN				05:46				05:45	
201802051053467	MANNGTR	05:53	05:54	05:53		05:54:30		05:53		05:55
201802051053467	CLCHSTR	06:03	06:05	06:03		06:05		06:02		06:04
201802051053467	STFD	06:44		06:44		06:45		06:43		06:45
201802051053467	BOWJ				06:49				06:47	
201802051053467	BTHNLGR				06:51				06:50	
201802051053467	LIVST	06:54		06:54				06:55		

Derived Inputs

You may derive the following inputs from the train running data as inputs:

1. First station deviation from Departure time, i.e.
2. Day of the week,
3. Day of the month
4. Weekday/Weekend
5. On-Peak/Off-Peak
6. Hour of the day
7. Associated Journey
8. Associated Journey Deviation from Departure
9. Associated Journey First Stop
10. Associated Journey Second Stop

Weather Data

- If you wish, you may use Weather data as inputs to your predictive module, e.g.
  - Temperature
  - Moisture
  - Rain
  - Wind speed, direction
  - Snow
  - etc.
- Where to get weather data?
  - OpenweatherMap: https://openweathermap.org/api

https://rapidapi.com/blog/access-global-weather-data-with-these-weather-apis/

Output: Prediction target

- For a train at each station from historical data,

  Compute the difference between the actual arrival time t_a and planned arrival time, t_p, i.e.

$$t_d = t_a - t_p$$

- Then the predicted arrival time for train i at station j:

$$\widehat{t_a}(train\ i, station\ j) = t_p(i,j) + \widehat{t_d}(i,j)$$

Some References for delay prediction

- Oneto, L., Fumeo, E., Clerico, G., Canepa, R., Papa, F., Dambra, C., Mazzino, N., and Anguita, D.
  1. (2018): Train delay prediction systems: A big data analytics perspective. Big Data Research, 11:54-64
  2. (2017): Dynamic delay predictions for large-scale railway networks: Deep and shallow extreme learning machines tuned via threshold out. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 47:2754{2767. https://doi.org/10.1109/TSMC.2017.2693209.
- Yaghini, M., Khoshraftar, M., and Seyedabadi, M. (2013). Railway passenger train delay prediction via neural network model. J. Adv. Transp., 47:355{368.https://doi.org/10.1002/atr.193.
- Peters, J., Emig, B., Jung, M., and Schmidt, S. (2005). Prediction of delays in public transportation using neural networks. Int. Conf. on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06), v2.
- Al Ghamdi M., Parr G., Wang W. (2020) Weighted Ensemble Methods for Predicting Train Delays. In Lecture Notes in Computer Science, vol 12249. Springer, https://doi.org/10.1007/978-3-030-58799-4_43

5. Knowledge Base & Engines

- Some Python based: KB and REs
- PyKE: Inspired by Prolog
  - http://pyke.sourceforge.net/index.html
  - Can do forward & backward chaining
- PyKnow => **Experta**
  - https://pypi.org/project/experta/
  - Inspired by CLIPS (C-Language Integrated Production System)
- Durable Rule
  - https://pypi.org/project/durable-rules/
  - a polyglot micro-framework for real-time reasoning

AI/ML based Engines & Platforms

- Amazon: Lex
- Google: Dialogflow
- Microsoft: LUIS, QnA Maker
  - Virtual personal assistant Cortana
  - Azure Cognitive Service
- Facebook Wit.ai
- IBM: Watson

Knowledge Engine: Experta

- FACTS
  - Basic information,
  - e.g. lights=FACT(R="red", G="green", Y="amber")
- DefFACTS: default facts loaded on initialisation
- RULES
  - A rule has 2 parts: Left Hand Side and Right Hand Side:
    - LHS: conditions, RHS: actions,
    - e.g. if C1 and C2 then A
  - Can be called by KE
- Knowledge Engine

https://experta.readthedocs.io/en/latest/

Experta Example 1

```
from experta import *
class Greetings(KnowledgeEngine):
    @DefFacts()
    def _initial_action(self):
        yield Fact(action="greet")     -> 1. establish initial fact
    @Rule(Fact(action='greet'),
          NOT(Fact(name=W())))         -> 2. Fact(name) not present, so execute rule
    def ask_name(self):
        self.declare(Fact(name=input("What's your name? ")))
    @Rule(Fact(action='greet’),
          NOT(Fact(location=W())))     -> 3. Fact(location) not present
    def ask_location(self):
        self.declare(Fact(location=input("Where are you? ")))
    @Rule(Fact(action='greet’),
          Fact(name=MATCH.name),             -> 4. Now this rule executes
          Fact(location=MATCH.location))
    def greet(self, name, location):
        print("Hi %s! How is the weather in %s?" % (name, location))
engine = Greetings()
engine.reset()  # Prepare the engine for the execution.
engine.run()  # Run it!
```

Experta Example 2

```
from random import choice
from experta import *  #source: https://pypi.org/project/experta/
class Light(Fact):
    """Info about the traffic light."""
    pass
class RobotCrossStreet(KnowledgeEngine):
    @Rule(Light(color='green'))
    def green_light(self):
        print("Green light is now on: Walk now")
    @Rule(Light(color='red'))
    def red_light(self):
        print("Red light on: Stop")
    @Rule(AS.light << Light(color=L('yellow') | L('blinking-yellow')))
    def cautious(self, light):
        print("Becareful because light is", light["color"])
engine = RobotCrossStreet()
engine.reset()
engine.declare(Light(color=choice([ 'red'])))
engine.run()
```

KE: Durable_Rules -- Facts

- **Install:** pip install durable_rules
- **Facts**
  - represent the data that defines a knowledge base.
  - are asserted as JSON objects and
  - are stored until they are retracted.
  - When a fact satisfies a rule antecedent, the rule consequent is executed.

https://github.com/jruizgit/rules/blob/master/docs/py/reference.md#pattern-matching

Durable_Rules -- Rules

- **Rules:**
  - A rule is the basic building block of the framework.
  - The rule antecedent defines the conditions that need to be satisfied to execute the rule consequent (action).
  - By convention m represents the data to be evaluated by a given rule.
  - “when_all” and “when_any” annotate the antecedent definition of a rule
  - Antecedent == “if part”, consequent is the “then part”

https://github.com/jruizgit/rules/blob/master/docs/py/reference.md#pattern-matching

Durable_Rules: Rules Example

```
from durable.lang import *

with ruleset('test'):
    # antecedent
    @when_all(m.subject == 'World')
    def say_hello(c):
        # consequent
        print('Hello {0}'.format(c.m.subject))

post('test', { 'subject': 'World' })
```

https://github.com/jruizgit/rules/blob/master/docs/py/reference.md#pattern-matching

Durable_Rules: Antecedent

- A rule antecedent is an expression.
  - The left side of the expression represents an event or fact property.
  - The right side defines a pattern to be matched.
  - By convention events or facts are represented with the *m* name.
  - Context state are represented with the *s* name.

https://github.com/jruizgit/rules/blob/master/docs/py/reference.md#pattern-matching

Durable_Rules: Antecedent

- Logical operators:
  - Unary: - (does not exist), + (exists)
  - Logical operators: &, |
  - Relational operators: < , >, <=, >=, ==, !=

https://github.com/jruizgit/rules/blob/master/docs/py/reference.md#pattern-matching

Durable_Rules: Antecedent Example

```
from durable.lang import *


with ruleset('expense'):
    @when_all((m.subject=='approve')|
     (m.subject == 'ok'))
    def approved(c):
        print ('Approved subject:
     {0}'.format(c.m.subject))


post('expense', { 'subject': 'approve'})
```

https://github.com/jruizgit/rules/blob/master/docs/py/reference.md#pattern-matching

Durable_Rules: Events

- **Events**
  - An event is an ephemeral fact, a fact retracted right before executing a consequent.
  - Thus, events can only be observed once. Events are stored until they are observed.
  - Events can be posted to and evaluated by rules.
  - Events are a way to trigger other rules in a chain

https://github.com/jruizgit/rules/blob/master/docs/py/reference.md#pattern-matching

Durable_Rules: Fact/event Example

```
from durable.lang import *
with ruleset('risk'):
    @when_all(c.first << m.t == 'purchase',
          c.second << m.location != c.first.location)
    # the event pair will only be observed once
    def fraud(c):
        print('Fraud detected -> {0},
{1}'.format(c.first.location, c.second.location))

post('risk', {'t': 'purchase', 'location': 'US'})
post('risk', {'t': 'purchase', 'location': 'CA'})
```

https://github.com/jruizgit/rules/blob/master/docs/py/reference.md#pattern-matching

Inference Engines with AI/ML

- Bayes networks
  - "A Naive Bayes approach towards creating closed domain Chatbots"

  https://towardsdatascience.com/a-naive-bayes-approach-towards-creating-closed-domain-chatbots-f93e7ac33358

- Markov chain models

https://www.codingame.com/playgrounds/41655/how-to-build-a-chatbot-in-less-than-50-lines-of-code

- Use MC and Janome: https://linuxtut.com/en/14587d79dcef8722fe57/
- Decision trees (not covered in this model)
- Shallow and Deep Neural Networks, some will be covered in this module