# More Data Courses

Saturday, March 16, 2019 2:23 PM

## Edx Courses New Shortlist

~~**UCSanDiego**~~
~~**$350**~~
~~https://www.edx.org/course/python-data-science-uc-san-diegox-dse200x~~
~~**(Pandas, Numpy, Matplot - Real World data sets;**~~
~~**Also it's a university course)**~~ **(DONE - but no more access)**

**Follow up to above course: MIT Machine learning course alongside.**

~~**Then next:**~~
~~**-Machine Learning course - UCSanDiego**~~
~~https://www.edx.org/course/machine-learning-fundamentals-3~~

**OR**

**IBM**
**Machine Learning ($39)**
https://www.edx.org/course/machine-learning-with-python-for-edx
Same Topics as the Udemy course.

So either of above 2 should be good follow up or refresher.

**Andrew Ng Course**
**https://www.coursera.org/learn/machine-learning**
Recommended by a Industry professional
(Not in Python; but can do theoretical part from here)

- **Start with Logistic regression; combine with Eric course + Babson Problems**
- **Lecture Notes and Python code now available.**

https://github.com/afshinea/stanford-cs-229-machine-learning

**https://www.youtube.com/playlist?list=PLl8OlHZGYOQ7bkVbuRthEsaLr7bONzbXS**
*Cornell's entire Machine Learning class (CS 4780) is now entirely on You Tube. Taught by one of the funniest and best professors from Ucornell.*
( https://www.reddit.com/r/learnprogramming/comments/bu6645/cornells_entire_machine_learning_class_cs_4780_is/?user_id=157288578740 )

==========
==========

**Capstone Project -**
**Analytics and Machine Learning**
https://www.edx.org/course/data-science-and-machine-learning-capstone-project

**Basic but key components of Data Analysis**
https://www.edx.org/course/data-analysis-with-python

- Correlation, Normalization
- Model Creation (Simple, Multiple Regression)
- Model evaluation

**MITx Machine Learning course**
https://www.edx.org/course/machine-learning-with-python-from-linear-models-to-deep-learning

**Pluralsight**
Machine Learning Algorithms. (maybe a theoretical approach to the Algorithms)
https://app.pluralsight.com/library/courses/machine-learning-algorithms/table-of-contents

**MIT Open courseware**
***Computational Thinking and Data Science***
https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-0002-introduction-to-computational-thinking-and-data-science-fall-2016/syllabus/

- It is an introduction and basic concepts of machine learning
- Touches on topics such as Clustering, Classification, Monte Carlo etc.

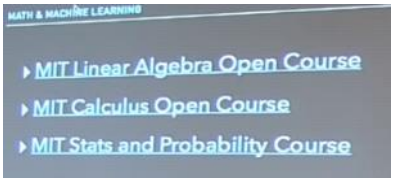
Its in **Python**.
Lecture Videos
Lecture Slides
Problem Sets everything in available
Its an Undergrad course so may be little basic but can run through quickly.
(https://www.youtube.com/watch?v=h0e2HAPTGF4 example video of this course)

**All Courses**
https://ocw.mit.edu/courses/find-by-number/

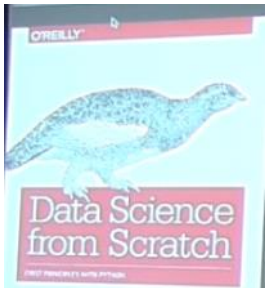
- Includes courses from **MIT Sloan (can find Babson courses and Cases to solve)**
- Also Topics like Social sciences, Anthropology, Economics
- History, Literature, Philosophy

http://cs109.org/
**Harvard Fall 2013 CS109 Data Science**

Can do even if no intention of becoming a web developer.
Use to became a better engineer or better prblem solver.
https://www.udemy.com/js-algorithms-and-data-structures-masterclass/#instructor-1




*** Search Kaggle with Tags (such as learn, tutorial, pandas etc.)
**Python and Big Query**
https://www.kaggle.com/dansbecker/getting-started-with-sql-and-bigquery



Machine Learning Algorithms. (maybe a theoretical approach to the Algorithms)
https://app.pluralsight.com/library/courses/machine-learning-algorithms/table-of-contents
Includes - Classification, Clustering, and Regression


**Udemy Pandas Course**
85 Exercises on data analysis using Pandas.
https://www.udemy.com/master-data-analysis-with-python-essential-pandas-commands/?deal_code=USERSRESTDEALL8&utm_source=email-Pro&utm_campaign=_._cn_NewCoursesMonthly_._us_AAll_._tl_T1_._tg_n_._et_5_._eg_1_._la_en_._&utm_medium=NewCoursesMonthly_AAll_T1_n&utm_content=udemy.6274145&data_h=A0MYeV9aRn4=&utm_term=NEW_TOPIC_COURSES_2

Not a Practical book.
No use of libraries.

https://www.youtube.com/playlist?list=PLu0W_9lII9ai6fAMHp-acBmJONT7Y4BSG

**github.com/thealgorithms/python**

**Google Cloud Platform**
Training -
https://google.qwiklabs.com/quests/34?utm_source=gcp-videos&utm_medium=yt&utm_campaign=qlgds

Can maybe be an option that helps teach how to push to production.

**https://www.youtube.com/watch?v=Gd7Ci2EFfcc**
- Can refer as recap for individual topics
- Statistics concepts implemented in Python starting from Two sample test, Chi sq test, then KNN, Decision tree.
Clustering, Random Forests

**GCP Training** -
https://google.qwiklabs.com/

**Starts off with Regression basics and Moves into ML** (Advanced but Could be worth a try)
https://pythonprogramming.net/machine-learning-tutorial-python-introduction/
(again can be refresher for Udemy ML tutorial)

**PANDAS**
https://www.pluralsight.com/courses/pandas-playbook-manipulating-data
https://www.pluralsight.com/courses/pandas-advanced

***ML MODEL DEPLOYMENT***
https://www.pluralsight.com/courses/deploying-machine-learning-solutions
(Would need to find more such courses)

**SQLAlchemy**
https://app.pluralsight.com/player?name=199530fa-b0d3-4b1c-934c-ef65b4462dea&mode=live&clip=0&course=python-sqlalchemy-playbook-understanding-databases&author=xavier-morera

***ML MODEL DEPLOYMENT***
https://www.pluralsight.com/paths/implementing-the-data-science-workflow-in-microsoft-azure

**ML Projects** -
http://zwmiller.com/projects.html

**Market Basket Analysis / Apriori algorithm -**
https://www.udemy.com/course/hands-on-unsupervised-learning-with-python/

https://learnxinyminutes.com/docs/files/learnpython3.py
Check and del

**Bayes ML Technique: A/B Testing**
https://www.udemy.com/bayesian-machine-learning-in-python-ab-testing/

**Hands on course**
https://www.kaggle.com/kashnitsky/mlcourse/discussion/104806
Also there are ongoing ML tutorials on Kaggle
- https://www.kaggle.com/kashnitsky/topic-9-part-1-time-series-analysis-in-python
(Time series analysis)

## BOOKS

1
https://www.amazon.ca/Introduction-Machine-Learning-Python-Scientists/dp/1449369413/ref=sr_1_7?crid=IS721OXMHJ9Z&keywords=machine+learning&qid=1566184130&s=books&sprefix=machine+%2Caps%2C172&sr=1-7

2
https://www.amazon.ca/Hands-Machine-Learning-Scikit-Learn-TensorFlow/dp/1491962291/ref=pd_bxgy_img_2/136-0128846-0060440?_encoding=UTF8&pd_rd_i=1491962291&pd_rd_r=a13221dd-bc55-41be-879e-d006d2ba0cd3&pd_rd_w=V40nh&pd_rd_wg=SJHdc&pf_rd_p=a62e2918-d998-4bbb-8337-35aac776e851&pf_rd_r=0WP1KD7SBTZ175J4V4MH&psc=1&refRID=0WP1KD7SBTZ175J4V4MH

**3**
https://www.amazon.com/Fundamentals-Machine-Learning-Predictive-Analytics/dp/0262029448/ref=zg_bs_3887_20?_encoding=UTF8&psc=1&refRID=9HK33PPS16VDZN3B1N8G#customerReviews

4
**100 Page ML book**
Quick read (only 100 pages). But good book, to make you feel more confident about ML and AI.

# ML STUDY PROGRESS

Wednesday, September 18, 2019 10:14 PM

**Andres Muller Book**:
Intro Section - **DONE**
Generalization, Over and Under Fitting - **DONE**
kNN - **DONE**
K-Means - **DONE**
Preprocessing and Scaling - **DONE**
Naïve Bayes - **DONE** (not much content)
Decision Trees (In Progress)

**NEXT STEPS** >>>>

- Do a KMeans Example (other course????) -- **DONE**

- Association Analysis & Apriori Algorithm
  - Eril Course

- Do the Linear Models section in Supervised learning
  - Has Linear and
  - Logistic Regression (Go back to **Andrews course** as doing this section)

- Naïve Bayes
  - PROBABILITY Do (Math course or Do 3Blue 1Brown)???
  - ~~Do from Andres Book NA (not properly covered)~~
  - Do from Gueron Book
  - Do Andrew Course
  - Also Eril course

- Decision Trees
  - From Book (In progress)
  - Andrew Course
  - Eril Course

- Have to do Matrices and Vectors; and Math. (Do 3Blue1Brown or Imperial College Math Course or MIT Opencourse-reddit saved post)
  - This may have math implemented in Python

  - After learning Math can then do Dimensionality Reduction and Feature extraction…

Or can Just do from **Andrews Course**.

- Can do Knn from Andrews Course as well. For the Math logic.

ALGORITHM STUDY ORDER:
Linear Regression
Logistic Regression
Knn (regression ex pending)
Decision Trees (1 example + Regression case)
Kmeans
Apriori
Ensemble learning / Random Forests (Last part)
--- Solidify all this ---

- Keep Moving stuff to GIT -

-- Follow up with --
Naïve Bayes???
Heirarchial Cluster Analysis
The Math
SVM's
Dimensionality Reduction

**Marketing Analytics Folder**:

E:\BABSON college\2nd Yr Spring Sem\Marketing Analytics\Session 3 and 4_stats review\Neoshop case
- NEOSHOP Case : Dataset and Questions. And PDFs with some solution.
- Marketing review PDF (just for some terms)

E:\BABSON college\2nd Yr Spring Sem\Marketing Analytics\Session 5 _statsAndDBmkt
Check the Bookbinder Case - A scenario of combination or Regression Log Reg and RFM.

Ignore remaining folders for now…..

**Econ Forecasting Folder**:
Session 3 - Simple Linear Regression
Session 4 - Multiple Linear Regression

For Both -
- Pre class Reading
- POST PPT's incl. Session 5 POST PPT
- Examples covered in 'InClass Exercise and Examples' & 'Review Questions' files. With answers (maybe in different files).

Should cover -
Regression, coeffients, Why R2, Diff statistical tests in Reg, some examples

Session 5 - is much of the same (Regression)
Session 6 - Dummy variables and examples

Session 7 - Mid Term

Session 8 - Reading (Evaluation of models… **Can do** this too)

# Masters

Saturday, August 10, 2019 8:16 PM

**Ryerson University**:

***MS in Data Science***
https://www.ryerson.ca/graduate/datascience/
1-year full-time or 2-year part-time
Fees: $4,878.43

Fees details
https://www.ryerson.ca/registrar/fees/detail/graduate/
(fees for Full-time is double 10k??. Need to confirm if same no of courses)

Intake September
Fall 2020 Applications not open yet. (may be around Jan)

They have an **MS in Comp Sci** but only full-time

**Ryerson - Chang School of Continued Education**

***Data Analytics, Big Data, and Predictive Analytics***
(Certificate Program)
https://continuing.ryerson.ca/public/category/courseCategoryCertificateProfile.do?method=load&certificateId=171618
6 Courses; ~$1K each; 40Hrs each.
(8 - 12 Months)
Some Machine Learning.
Some Big Data.
Some Basic Analytics

-- They have a Fast Track Option (Cant Do. its Full-time)
***CKME 995***
$4068
156Hrs.
5 Days a Week
Full - Time (9.30 am to 4.30 pm)
https://continuing.ryerson.ca/search/publicCourseSearchDetails.do?method=load&courseId=287288

~~*Hadoop, Python, and Tableau Big Data Tools*~~
~~(Webinar on Nov 29, 2019; Not Signed up yet)~~
~~https://continuing.ryerson.ca/public/category/courseCategoryCertificateProfile.do?method=load&certificateId=207747~~
~~(use Python, R, SQL, Hadoop, Hive and Pig Big Data tools.)~~
~~NOT GREAT~~

*Information Systems Management*
(Certificate Program)
(Semi Technical)

==========
==========
==========

**UofT**

**When choosing compare**:
Courses Count
Fees
No of Hours per course
No of Hours in total
Value - $/totalHrs
Actual course options

https://www.coursera.org/degrees/master-of-computer-science-illinois/academics
Example of MS in Data Science to compare Courses against.

**For Comp Sci**

***Computer Science (BSc (Hons))***
**Fees - Around 10K**

**Ryerson - Continued Education**
*Coding, Programming, and Algorithms: Computer Science Fundamentals (Certificate Program)*
https://continuing.ryerson.ca/public/category/courseCategoryCertificateProfile.do?method=load&certificateId=207550
**3 courses**
**around $1300 each**

*Computer Programming Applications*
*(Certificate Program)*
https://continuing.ryerson.ca/public/category/courseCategoryCertificateProfile.do?method=load&certificateId=173786
**6 Courses**
**around $1300 each**
**(12 Months)**
Of these the basic Comp Science courses even I can do.
(concepts around abstration, modelling design, Algorithms)

*3440 - UofT SCS Coding Boot Camp*
https://learn.utoronto.ca/programs-courses/courses/3440-uoft-scs-coding-boot-camp
"This Full Stack Flex course equips you with the fundamental skills needed to become a full stack web developer"
Part Time - 24 Weeks.
Courses - JavaScript, Node.js, HTML, CSS, jQuery, Java, and more.

==========
==========

**UofT**
https://sgs.calendar.utoronto.ca/degree/Computer-Science

***Master of Science in Applied Computing***
*(General or Data Science concentration)*
Ideal Course
FULL-TIME Only
3 Years
8 Month Internship
~12.5K

***Master of Science***
(Research oriented -
" MSc degree program is designed for students seeking to be trained as a researcher capable of creating original, internationally recognized research in computer science.")
Full-Time and PART-TIME
-- Does not say anything specific to Data Science.

~~***MS in Mathematics***~~
~~Research oriented. Study of pure and applied mathematics~~.

***MS in Statistics***
Basically it's the Math behind Data Science and ML Algorithms.
Its Not a ML course.
Part-time available.

Study of Statistical Theory and Applications or Probability.
Courses include -
Stochastic Processes
Bayesian Computation
Monte Carlo Methods
Multiple Hypothesis Testing and its Applications :-)
"*prepare students for pursuing a PhD program or directly entering the data science workforce.*"
https://sgs.calendar.utoronto.ca/degree/Statistical-Sciences

**UfT School of continuing Education**

***Data Science***
(*Certificate Course*)
https://learn.utoronto.ca/programs-courses/certificates/data-science
4 Courses
$1000 each
Includes a ML course and a Big Data Course.
https://learn.utoronto.ca/programs-courses/courses/3253-machine-learning
(ML Course of the certificate. Can maybe just do this and then eventually get the certificate)
Available at Downtown Campus (St. George)

***Artificial Intelligence***
(CERTIFICATE Program)
https://learn.utoronto.ca/programs-courses/certificates/artificial-intelligence

Part Time - 24 Weeks.
Courses - JavaScript, Node.js, HTML, CSS, jQuery, Java, and more.

**YORK University**
*BS in computer Science*
http://futurestudents.yorku.ca/program/computer-science

3 Courses
(one is ML from above)

***Individual Courses***
https://learn.utoronto.ca/programs-courses/information-technology-environment-and-engineering/information-technology-it

**Data Analysis bootcamp**
Part time - 24 Weeks
https://bootcamp.learn.utoronto.ca/data
Its fine but NOT MUCH ML.

==========
==========
==========

**YORK University**

***Computer Science (MSc)***
- ***Specialization in Artificial Intelligence***

http://eecs.gradstudies.yorku.ca/comp-sci-msc/
http://eecs.gradstudies.yorku.ca/ai/#squelch-taas-tab-content-0-1
Deadline for Fall 2020
December 15, 2019.
Fees:
FALL ENTRY ONLY
Part Time Available

**York University Continued education**
- A couple of options here (CHECK)

http://continue.yorku.ca/certificates/

# Code Sheet - Pandas

Sunday, September 22, 2019 3:08 PM

## *CREATING AND IMPORTING DATA*

```
df = pd.read_csv(''E:/Skills training/Market Basket Analysis/file_name.csv')
df.to_csv('E:/Skills training/Market Basket Analysis/NewFileName.csv')
df_name.to_excel(' ..... ')
help(fun name)
<img src="image.png">  # Adding img to jupyter notebk, if img in Notebook dir. If in sub dir then need to specify path
                       #can drag img and drop into jupytr as well but would make file heavier
```

## ***PANDAS*** -- *Series and Dataframes*

```
import pandas as pd
```

***SERIES - 1d ndArray with Axis labels***

```
pd.Series( ['milk','eggs','brush','soap'] )      #A 1D Pandas Series. And can have diff Datatypes
                                                 a 1d numpy array has no labels.
pd.Series( [[1,2,3,4,5] , [22,33,44,55]] )   # This also a 1d series (single col)
pd.Series(['milk','eggs','brush',22],index=['item1','item2','item3','item4'])     # Pandas Series with Index names
```

***DATAFRAME - 2d tabular structure with Index (rows) and Col names***

```
pd.DataFrame( [ [1,2,3,4],[22,33,'sha',55] ] )  #Creates a Dataframe. EACH LIST ADDED AS ROWS
pd.DataFrame([[1,2,3,4],[22,33,'sha',55]], index=['row1','row2'])   #Creates DF with Index names
pd.DataFrame([[1,2,3,4],[22,33,'sha',55]], index=['row1','row2'],columns=['col1','col2','col3','col4'])     #DF with
                                                                                                  Index and Column names

df2 = pd.DataFrame({'name':['lampard','greard','rooney'],'club':['chel','pool','utd'],'year':['2006','2012','2009']},
          index=['player1','player2','player3'])      #Pandas Dataframe using a dictionary
#Use this TO COMBINE 2 NPARRAYS INTO A DATAFRAME with Labels
```

**#Dataframe to Numpy Array**

```
new_arr = df.values
new_nparray = df.loc[:,'col2'].values
```

**#Numpy Array to Dataframe**

```
df_1 = pd.DataFrame(Array_name, index = ['pl1','pl2','pl3','pl4'], columns=['club1','name','titles'])
```

## *PANDAS General Commands*

```
df.head()
df.tail()
df.shape
df.info()          # can check if any nulls in any column
df.describe().transpose()          #also df.T
df[['col1','col2','col3']].describe
df.columns
df.index       #to refer to the index as a column of a dataframe

df['col_x'].replace(23,33,inplace= True)          #a ctrl+H. Find and replace.
df['col_x'].str.replace("IV","Four",inplace= True)      #A Control H on colx but for string

df['col_name'].astype('float')       #Changing data type of an entry from

data.sort_values(by='Year', ascending = False)
data.sort_values(by=['Year','Country name'])

df.set_index('colName', inplace=True)    #to set index of a dataframe to a certain column or an Array of the correct length
df.reset_index(inplace=True)    #To reset index to default

df.rename(columns = {'curr_col_name':'new_col_name' , '2nd_colname' : 'new2nd'})        # Renaming a column using
```

★ ***ALL OTHER DATAFRAME FUNCTIONS***

*Dictionary substitution*

*OPERATIONS ON DATAFRAMES*

```
demodata[['ColName1','ColName2']][26:31]    #Slicing a Dataframe
fifa_df[fifa_df['ShortPassing']>89][['Name','Overall']]          #Filtering a dataframe. Filter means essentially Filtering
                                                                 out rows
```

*#loc and iloc*
***#loc [supply label]    iloc [supply index]***
#Format supplied is *[rows,[columns]]*

```
fifa_df.loc[2:10,['Name','Age','Club']]      #using loc[]
defenders_analysis_df.loc[defenders_analysis_df['StandingTackle']>=88,'Name':'Potential']   #Filtering data

top_tacklers.iloc[0:5,0:3]       #using iloc[]
defenders_analysis_df[defenders_analysis_df['ShortPassing']>89].iloc[:,[0,2,5]]

#When multiple filter condition, result can be acomplished using ( ) as seen below
top_oceania_wines = reviews[(reviews.loc[:,'points']>=95) & \
((reviews['country']== 'Australia') | (reviews.loc[:,'country']=='Italy'))][:]
#Also here  \  used to split command into 2 lines
```

#LAMBDA FUNCTIONS (PENDING)
~~*df_name.apply(lambda x : min(dataset) + max(dataset))*~~

#Also apply()

*OTHER COMMANDS*

```
#In Pandas ROW is AXIS 0 and COLUMNS is AXIS 1

fifa_df['Nationality'].unique()        #List of unique entries

nunique()          #Count(number) of unique entries

fifa_df['Position'].value_counts()          #Unique entries in a column and its frequency
```

**# GROUP BY**

```
df.groupby('field_to_grpby)['colname_to_agg'].sum()  #Group By
df.groupby('row_name').mean()        #Group By
```

***# PIVOT TABLE*** link
```
basket = df.pivot_table(index='InvoiceNo',columns='Description',values='Quantity',aggfunc=np.sum,fill_value=0)
```

```
# Add Column with name New_Scores
pos_list = ["ST","GK","CB","LS","LB","CB", ……'CM']
df['New_Col_name'] = pos_list                #Can create a list or a nd Array and assign to a new col
OR
df['New_Scores'] = [10,20, 45, 33, 22, 11]           #Can directly assign values to a new col

dfObj['Percentage'] =  (dfObj['Marks'] / dfObj['Total'] ) * 100          #Creating new col based of existing cols

demodata.columns= ['col1','col2','col3','col4','col5']     #Renaming Columns (All col names must be specified)
```

*DATA CLEANING*

#FOR All **DROP Commands, By default original dataframe is not changed**, but **n** dataframe itself.
#Either that of assign to a new Dataframe.

```
df[col_name].isna()      #Checks NAs. Returns a Boolean Series. Can use as filter to assign values to entries with missing data
df.isna().any()    #Will list all cols. Cols with 'True' have nulls

df.dropna(inplace= True)          #Drops fields (entire Row) with missing values Nan (can change to axis 1)
df.fillna(0, inplace = True)          #Dont want to drop the rows so fill Nas with 0
df[col_name].fillna(method='bfill')     #Dont want to fill with 0, but a value
df.dropna(axis=0, subset=['Col_name'])       #Drop row only if entry in specified col has NA
df.dropna(axis=1)      #to drop the columns with NA


#To delete a column
new_df = df.drop('Col_Name',axis=1)          #Need to specify Axis. By default drop() drops rows
df.drop(['col_1','col_2','col_3'],axis = 1, inplace = True)        #To drop multiple cols in the same df

del df['col_name']        #Another approach to del a col

#To copy a dataframe
df_1 = df
func1(df_1)
#Here df_1 is a view of df and not a copy. So func1 will modify df through df_1

df2 = df.copy()      #Will create a copy
```

*JOINS / COMBINE*

```
df1.append(df2)                    #Rows of df1 and df2 are combined, duplicates remain.

df_union_all= pd.concat([df1, df2])               #Rows added to the end
df_union= pd.concat([df1, df2]).drop_duplicates()       #Rows added to the end and no duplicates

pd.concat([df1, df2],axis=1)       #add the columns in df1 to the end of df2 (rows should be identical)

df1.join(df2,on='col1',how='inner')          #This takes into account the Index column as well.
```

**.merge()** works same as join.

```
joined_df = pd.merge(left_df, right_df, how='inner', left_on= 'col_1name', right_on='col_2name)
#basically left_df inner join right_ df on left_on 'col' = right_on 'col'
```

*ANALYTICAL FUNCTIONS*

```
#Calculated for along a column/columns. Can change to rows

df['col_name'].min()
df['col_name'].max()
sum
mean/median/quantile
idxmin/idxmax         #will return the index of the row where the first minimum/maximum is found.
```

*SQL Within PYTHON*

*-- Reading from SQl*

*CORRELATIONS*

```
corr_matrix =  df.corr()                #Creates a correlation matrix of every attribute against every other attribute
```

# Code Sheet - Numpy

Tuesday, December 31, 2019 1:54 PM

## ***NUMPY*** -- *Arrays (1d arrays or nd arrays)*
## *all elements of same type*

*import numpy as np*

### *Creating Numpy Arrays*

*List1 = [11,22,54,17] #List*
*np.array(list_name) #1d np Array. Remember simpliest form of array is list. so keyword 'array' + listname*
*OR a list of lists*

*np.arange(4,13,step=2) #returns evenly spaced values between [4incl & 13excl). Step optional (default 1)*
*np.linspace(3,7,num=9) # returns 9 samples (num=9) between 3 and 7 (both included)*
*np.random.randint(2,7,size=12) #returns 12 random integers between 2 incl and 7excl*
*np.random.random(size=6) # 4 random floating point numbers*
*np.random.random(size=6).reshape(2,3) # a (2,3) array of random floating point numbers*

*Using above 3 can create 2 dim arrays --*
*np.arange(4,19).reshape (3,5) # 2 dim array with 3 rows 5 cols.*
*np.random.randint(4,9,size = 15).reshape(5,3) #2 dim array*

*arr_name.shape() #To get shape of an array*
*arr_name.ndim #To get dimension of an array 1,2 etc.*

> *A **shape** of **4,3** is a 2 dimensional array*
> *4 indicates its 4 - 1d arrays*
> *3 elements each (or 4 rows 3 cols)*
> *lll'ly*
> *5,4,3 is a 3 dimensional array.*
> *5 indicates its 5 - (4x3 arrays) 2 dimensional arrays*
> *4 incdicates 4 groups of 1d arrays in each matrice*
> *3 indicates no of elements in each array*

*Therefore*

*Arr1 = np.array([List1,List2,List3]) #Creating a 2D Numpy Array List1,2,3,4 each have 4 elements*
*Arr1 has 3 - 1d Arrays.*
*Each array has 4 elements*
*Therefore -- 2 Dimensional array; Shape 3,4*

*arr_t2 = np.array( [ [ l1,l2] , [l3,l4] ] )*
*A 3 Dimensional array.*
*has shape 2,2,4 (So 2 - (2x4) arrays)*
*each group has 2 - 1d arrays*
*each array has 4 elements.*

*#Can also have 4 dimensional array*

## *SLICING*

*#Arr1[rowno,columnno]  #Slicing a matrix*
*Arr1[ row x : row y , col p : col q]*

*test_3D_array[1,3,2] =1111  #To assign a new value to an entry in a 3 dim array*

*arr3 = np.copy(arr1)  #Will create a deep copy of an array.*

*arr2 = arr1[2:4,3]  # not a real copy. New reference (name) for same location in memory*
*#A slice does not create a copy. Changes made to elemens of a slice will effect the original*

## *OTHER FUNCTIONS*

*np.zeros((5,4,3)) / np.ones(4)  #array of 0's or 1's. Can be 1 or n dimensional.*

*array_name.size  #no of elements or length*
*arr_name.dtype  #datatype of elements in the array*
*type(arr_name)  #type*

*np.unique(test_arr[:,1])  #unique values in an Array*

*np.inner --*
*np.dot (mat1,mat2)  #matrix multiplication function of numpy. (as done in math)*

*np.logical_and  #numpy's logical AND function*

*np.random.rand(2,6)  #Random values in a given shape. from uniform distribution between [0,1)*

***APPEND***
*arrB=np.append(arrA, [5,6,7,8])  #append 5,6,7,8 to arrA*
*np.append(arrA,arrC, axis=0)  #Append along axis 0  #Remember Axis can take axis = 0,1,or 2*

*hstack (horiz stack is same as append)*

## *Boolean Masks*

Ex - Getting all entries that are divisible by 7
*my_vector = np.array([-17, -4, 0, 2, 21, 37, 105])*
*mask_arr = 0 == (my_vector%7)  #gives a array mask_arr with boolean values*
*my_vector[mask_arr]  #returns entries for True*

*my_vector[(my_vector%7)==0]  #Can be done directly*

## *BROADCASTING*

# Code sheet - Plotting

Tuesday, December 31, 2019 6:22 PM

***PLOTTING***

***~~~~SYNTAX TO REMEMBER~~~~***

***#MATLAB SYNTAX***

***plt.plot(Arr1, Arr2 ...)*** *#Simple plot of ColX or XvsY as lines or marks. No specific type of plot.*
*# NOTE -* ***df.loc[:,'col'].values*** *to convert to numpy array*

***plt.plot(df[col4X] , ...)*** *#Simple plot of ColX or XvsY as lines or marks. No specific type of plot.*
***plt.scatter(df[col1],df[col2]..)*** *#Similarily we have plt.hist*

***#OBJECT ORIENTED SYNTAX***

***fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2,figsize=(8, 4))*** *#initializing FIGURE and AXES. & initialize objects ax1,ax2 for each axes*
***ax1.scatter(…)***

***Subplots***

***plt.figure(figsize=(4,6)***
***plt.subplot(221) ; plt.hist(…)*** *#for multiple Axes/plots/vizzs in same figure. 2 rows 2 columns 1 index*

***PANDAS***

***df.plot( 'col4x' , 'col4y' , kind='…', figsize=(8,12))*** *<<-- can use this as the go to command*
*df[['col1','col2','col3']].plot(kind='bar')* *#Plot the dataframe i.e. df[[col1,col2,col3]] aginst INDEX*

***SEABORN***

***sns.boxplot(df[colA],df[colB])*** *#So works directly with dataframes*
*#OR:*
***sns.boxplot(x='colname', y = 'col2name', data=df_name)*** *#works directly with dataframes*

# numpy check and del

Sunday, June 9, 2019 8:48 PM

NEW - ADD TO CODE SHEET

#replace a value in a Column or type str (i.e. object)
ga_gsc_data['Avg. Time on Page'].str.replace('0 days ','')

---

Timedelta #Represents a "duration", the difference between two dates or times.

So "00:00:22.333333333" which is of type object, can be represented as an actual duration (type becomes timedelta)

Seems to be same as
pd.to_timedelta(df['Col name']) ???
#but check online once.

Numpy supports multidimensional Arrays and are called ndarrays (n dimensional)
And therefore is useful for working with Matrices and Vectors.

- Numpy arrays are fixed in size.
- They are much faster than Python lists.
- Elements of Numpy arrays must be of the same Type.
- Great functionality such as getting average of a vector or multiplying 2 matrices
- Pandas is actually built on Numpy. Note that while Pandas provides higher level functionality than Numpy,
you'll still be using Numpy as functionality at times.

In Numpy (seems like different from Pandas)
Rows is axis 1
Column is axis 0

Arithmetic Operations
x = np.array([[111,112],[121,122]], dtype=np.int)
y = np.array([[211.1,212.1],[221.1,222.1]], dtype=np.float64)
sum_arr = x + y

Multiplication as below is possible
x2d*y2d

- where statement
where(condition, [x, y])
OR
where(condition, arr_x, arr_y])
Return elements from arr_x where condition is TRUE and arr_y where condition is false.

# ML Applications

Monday, August 19, 2019 1:34 PM

GET YOUR PORTFOLIO RATED ON REDDIT

# ML Applications & Data Sources

Monday, September 9, 2019 10:48 PM

Your Github
https://github.com/cbragan/data-science-1

--
It makes global product-based associations and gives personalized recommendations.

**Logistic regression**
- Binary Classification to determine Spam vs Non Spam email.
-- credit card companies develop models which decide whether a customer will default on their loan EMIs or not.

--
***Marketing Applications with ML***
- For example, if your company struggles with customer churn issues, you might want to employ algorithms to figure out how to reduce the churn. In this case, ***you'd need to understand when someone is likely to churn*** so you can make an offer to reduce the risk.

-
***Clustering's*** open-ended approach still relies on the data versus human intuition in order to identify the most predominant traits in the people you're marketing to. As a result, you'll gain fresh insight into your customer universe (or total addressable market) and find new segments of customers and prospects. Marketers can then use key cohort signals to develop targeted campaigns or more personalized sales strategies, and to identify others who look similar to top-performing clusters.

-
***Classification*** These algorithms basically answer the question of whether something fits in one group or another.

**Example???**
Say a novice data scientist wants to predict whether a customer will buy a boat, given records of previous boat buyers and customers who we know are not interested in buying a boat.2 The goal is to send out promotional emails to people who are likely to actually make a purchase, but not bother those customers who won't be interested.
Can try for Big campaigns. Say the BTS campaign or the Tote bag or Gilly jacket campaign.

--
**Google Play Apps**
https://www.kaggle.com/lava18/google-play-store-apps

Key Questions -
Algorithms
*Classification -- we will use both Knn and Logistic regression*.

--
KNN for Netflix Competition???

--
Newspaper article classification
(seems to be easy language)
https://pdfs.semanticscholar.org/aa96/9114cf6e4d77c5bb3dd62a20bee3446f33ab.pdf

https://www.analyticsvidhya.com/blog/2018/12/best-data-science-machine-learning-projects-github/

Bank Marketing Dataset
https://archive.ics.uci.edu/ml/datasets/Bank+Marketing

Twitter Buzz events
https://archive.ics.uci.edu/ml/datasets/Buzz+in+social+media+

Energy Consumption (interesting dataset)
https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption

**Replicate this (EPL)**

https://www.kaggle.com/shriramganesh/awards-top-clubs-in-depth-analysis#Introduction

**VIDEO on the Titanic Dataset**
https://www.youtube.com/watch?v=j6jstahQp2A&list=PLQVvvaa0QuDfKTOs3Keq_kaG2P55YRn5v&index=36

**DATASETS**

**UCI ML datasets**
https://archive.ics.uci.edu/ml/datasets/online+retail

**Great Option for Time series forecasting**.
https://www.kaggle.com/c/demand-forecasting-kernels-only

**Data Sources Links**
https://www.kdnuggets.com/datasets/index.html

https://datasetsearch.research.google.com/
(***GOOGLE DATASETS RESOURCE*** )

**More DS Sources** -
https://www.youtube.com/watch?v=1aUt8zAG09E
data.gov
ndcc.noaa.gov/cdo-web (weather)
bls.gov/data (labor stats)
census.gov/data
quandl
ukdataservice.ac.uk
datacatalog.worldbank.org
imf.org/en/data
Amazon review ds - standford website
Yelp reviews ds made public
airbnb ds organized by city
image-net.org (for images ds)

Amazon AWS's Datasets
http://dataportals.org/
http://opendatamonitor.eu/
Datasets subreddit

**For JF** -

-->
True Women Shopper (We would classify someone as true-women if they buy 8 women items, less than 3 kids items and less than 1 mens item)
True Women-Kids Shopper
(We would classify someone as true-women if

Energy Consumption (interesting dataset)
https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption

Synthetic Datasets article
https://www.kdnuggets.com/2019/09/scikit-learn-synthetic-dataset.html

Kaggle Competition - Fraud detection
https://www.kaggle.com/c/ieee-fraud-detection/overview

Kaggle Datasets dataset **
https://www.kaggle.com/canggih/voted-kaggle-dataset

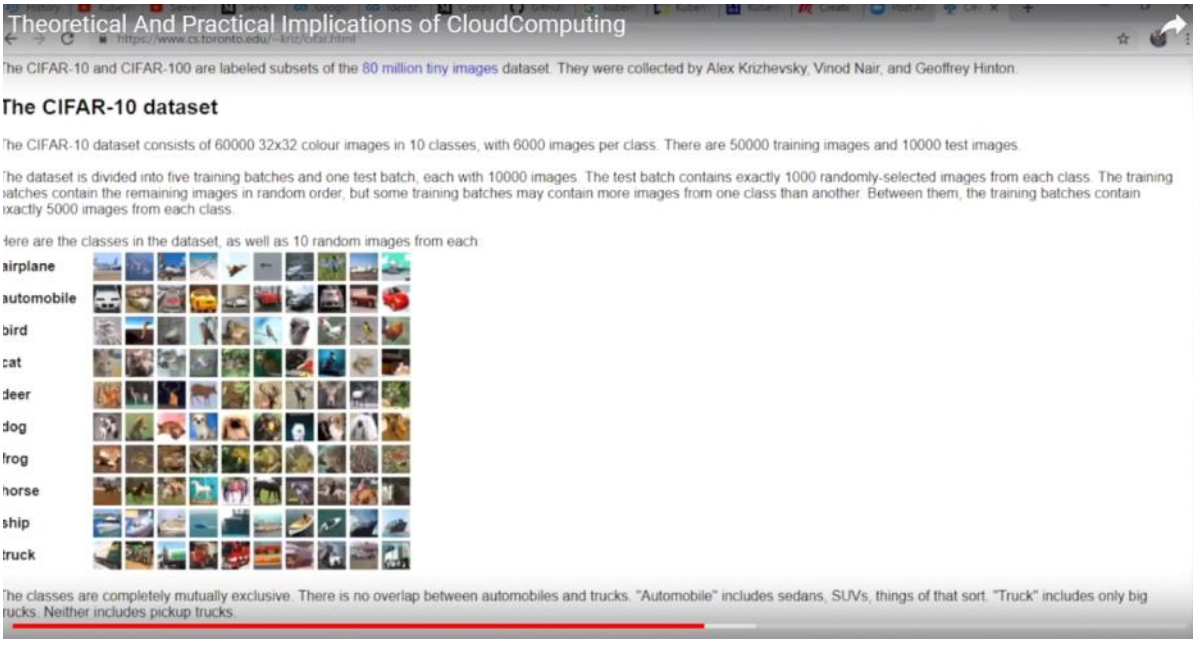
items, less than 3 kids items and less than 1 mens item)
True Women-Kids Shopper
(We would classify someone as true-women if they buy 6-8 women items, at-least 3 kids items and less than 1 mens item)
and so on…

This is a static classification - because anyone who has say
4 women items and 1 kid item is unclassified… or what would you classify her as???

Another problem with this is not enough features - its just M,W,K

*As in above* -
Can we segment customer into Platinium, Gold,Silver etc……


**ML vs Traditional Programming approach**
if spammers notice that all their emails containing "4U" are blocked, they might start writing "For U" instead. A spam filter using traditional programming techniques would need to be updated to flag "For U" emails. If spammers keep working around your spam filter, you will need to keep writing new rules forever.
In contrast, a spam filter based on Machine Learning techniques automatically notices that "For U" has become unusually frequent in spam flagged by users, and it starts flagging them without your intervention.



# GA Dataset

Sunday, December 29, 2019 8:15 PM

Google Analytics BQ dataset.
https://www.blog.google/products/marketingplatform/analytics/introducing-google-analytics-sample

(FIELDS)
https://support.google.com/analytics/answer/3437719?hl=en

Google Analytics store Dataset in Kaggle
https://www.kaggle.com/bigquery/google-analytics-sample

(DC GA stats)
https://data.world/datasets/google-analytics

# Market Basket Analysis

Monday, August 19, 2019 1:44 PM

Identifies the strength of association between pairs of products purchased together and identify patterns of co-occurrence.
Can be used in pricing strategies, product placement, and various types of cross-selling strategies.

Market Basket Analysis creates If-Then scenario rules, for example, if item A is purchased then item B is likely to be purchased.
The rules are probabilistic in nature.

**Check this**:
https://snap.stanford.edu/data/amazon0505.html

Ex - ' are you more likely to buy apples or cheese in the same transaction than somebody who did not buy milk? '
Or {onions,potatoes} --> {burger meat}

*If {A} Then {B}*

The **If** part of the rule (the {A} above) is known as the **antecedent** and the **THEN** part of the rule is known as the **consequent**.
Both antecedents and consequents can have multiple items. Ex: {Diaper, Gum} -> {Beer, Chips} is a valid rule.

**Support**

*(A+B) / Total transactions*

Fraction of transactions that contain both A and B.
*So if support of rule {Diaper, Gum} -> {Beer, Chips} is 0.7 means they occur together in 70% of all transactions.*

Support is **the relative frequency with which the rules show up**. In many instances, you may want to look for high support in order to make sure it is a useful relationship. However, there may be instances where a low support is useful if you are trying to find "hidden" relationships.

An itemset is considered as **"frequent"** if it meets a **user-specified support threshold**.
If the support threshold is set to 0.5 (50%), a frequent itemset is defined as a set of items that occur together in at least 50% of all transactions in the database.
But note - A subset of a frequent itemset must also be a
frequent itemseti.e., if {AB} is a frequent itemset, both {A} and {B} should be a frequent itemset.

**If the sales of a particular product is a certain proportion having a meaningful effect on profits, that proportion could be considered as the threshold**.

So,

*Support = freq (A,B) / N*

**NOTE on Frequent Item sets** -
Because {A} and {B} should be a frequent itemset
therefore In Apriori implementation
1- frequent itemsets is built of individual products and product sets.
Only Items with specified support threshold
are included in frequent item sets.
2- Products and product combinations from
frequent itemsets are used to form rules.
3 -Support Confidence and Lift is then looked at for each rule.

Remember for frequent itemset -
the iteration builds a set of 1-frequent itemset
then set of 2-freq itemset
then set of 3-frequent itemset and so on..

and for each k-itemset its (k-1)-itemset must also be frequent.
i.e. above the specified threshold. (refer example)

**Confidence**
*(A+B) / (A)*

Transactions of A and B bought together as a ratio of transctions with A.

Confidence is a **measure of the reliability of the rule**.

So,

*Confidence = freq (A,B) / freq(A)*
or
*Confidence = Support (A U B) / Support (A)*

Confidence takes into account Conditional probability i.e. Calculates probability of B given A has already occurred.
*A confidence of .5 in the above example says that in 50% of the cases where Diaper and Gum were purchased, the purchase also included Beer and Chips*.

Drawback with confidence - It takes into account popularity of A but not popularity of B, if B is also popular then there is a high probability that B would appear in majority of transactions. So, not necessarily associated to each other.
That's why we also have Lift.

**Conditional Probability**

2 Events are independent if an only if their joint probabilities equals the product of their probabilities.
So if
*P(A intersection B) = P(A) P(B)*
This becomes more clear when we look at definitions of their conditional probabilities.
https://en.wikipedia.org/wiki/Independence_(probability_theory)

$P(B/A_{already\ ocured}) = (P\ A\ intersection\ B) / P(A)$
If A and B are independent then P(B given A has also occurred) will be Equal to P(B)
there fore
P(B) x P(A) = P(A intersection B)

**Lift**
*((A+B)/(A) ) / (B/total transactions)*

- We now take into account popularity of B as well.

Lift tells us how much better a rule is at predicting the result than just assuming the result in the first place.

How much our confidence has increased that B will be purchased given that A was purchased.

OR

*lift (X-->Y) = supp (X U Y) / supp(X) * supp(Y)*
the ratio of the observed support to that expected if X and Y were independent.
Greater lift values indicate stronger associations.

*Lift = 1* the probability of occurrence of the antecedent and that of the consequent are independent of each other.
The basic rule of thumb is that a lift value close to 1 means the rules were completely independent

If the *lift is > 1*, that lets us know the degree to which those two occurrences are dependent on one another,
These are more 'interesting' and could be indicative of a useful pattern.

If the *lift is < 1*, that lets us know the items are substitute to each other.

--
Apriori is the best known algorithm to mine association rules. Apriori iteratively discovers pairs with the largest frequencies and then with decreasing frequencies

--
Unsupervised learning tool that looks for hidden patterns.

**Conviction**
conviction (A -->B) = (1 - supp (B)) / (1 - conf (A-->B))

**Example Run Completed**
https://pbpython.com/market-basket-analysis.html

~~**Steps**:~~
~~Decide on support threshold.~~
~~Step 1: Identify all **items** with support higher than threshold.~~
~~Step 2: Make list of all **2 - Item set** combinations.~~
~~Order does not matter. **AB = BA**~~
~~Basically take all items and pair it with other items in list.~~
~~Keep those with support above threshold.~~
~~Step 3: Identify all **3 - Item set** combinations with support above threshold. and so on.~~

~~--> In our example the output of steps is the freq_items list.~~

~~Step 4: Then we do our rules step.~~

**Applications** -

I think its 2 things -- Similar products bought together {Amazon - "Bought together"}

Similar audiences {Amazon - "so people who bought this also bought"} ???

Would need to break out the audiences and then do a A/B testing.

- Also remember it would not say customers who bought this also bought. It would say orders that have X also have Y

-

combining product incentives

- While these types of associations are normally used for looking at sales transactions; the basic analysis can be applied to other situations like click stream tracking, spare parts ordering and online recommendation engines

Meeting Next steps -
--> What metrics are we considering to make our decision on combination of Ante and conseq that we would choose.
if Lift then why? need to understand it better
Understanding each of the metrics and

How do we rank the rules or Which rule do we choose.

- List in Questions.

- try to define a use case for K-means and Knn.

==========

**Visualizing MBA**

Done using **NetworkX** Python Library

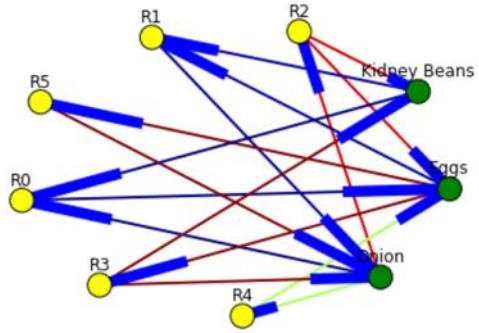
| | antecedants | consequents | support | confidence |
|---|---|---|---|---|
| 0 | (Kidney Beans, Onion) | (Eggs) | 0.6 | 1.00 |
| 1 | (Kidney Beans, Eggs) | (Onion) | 0.8 | 0.75 |
| 2 | (Onion) | (Kidney Beans, Eggs) | 0.6 | 1.00 |

==========

**Visualizing MBA**

Done using **NetworkX** Python Library
example (code): https://intelligentonlinetools.com/blog/201 rules-in-data-mining/

| | antecedants | consequents | support | confidence |
|---|---|---|---|---|
| 0 | (Kidney Beans, Onion) | (Eggs) | 0.6 | 1.00 |
| 1 | (Kidney Beans, Eggs) | (Onion) | 0.8 | 0.75 |
| 2 | (Onion) | (Kidney Beans, Eggs) | 0.6 | 1.00 |
| 3 | (Eggs) | (Kidney Beans, Onion) | 0.8 | 0.75 |
| 4 | (Onion) | (Eggs) | 0.6 | 1.00 |
| 5 | (Eggs) | (Onion) | 0.8 | 0.75 |



# Additional Apriori notes

Wednesday, November 6, 2019 11:03 PM

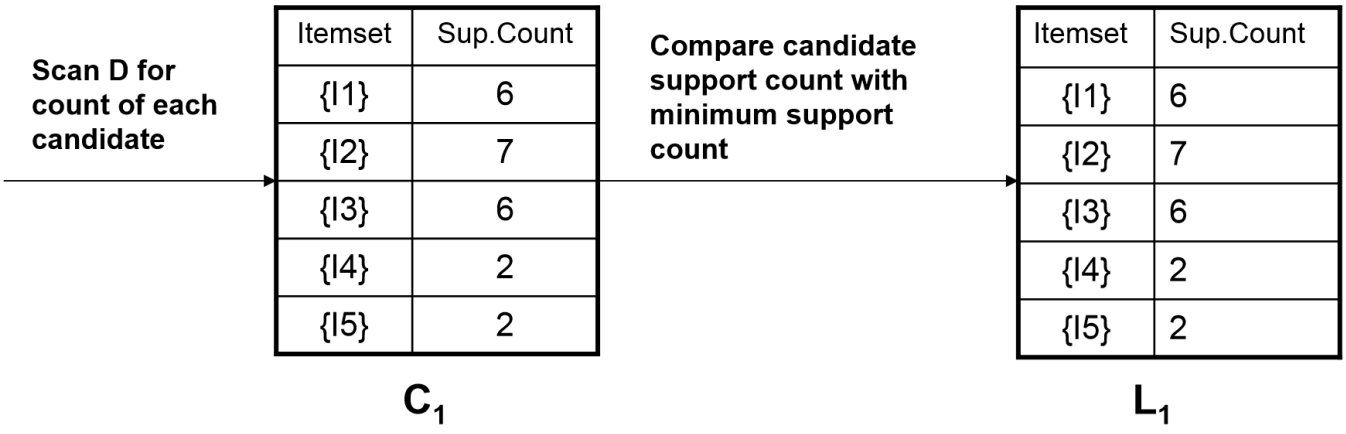
07apriori

## The Apriori Algorithm: Example

| TID | List of Items |
|---|---|
| T100 | I1, I2, I5 |
| T100 | I2, I4 |
| T100 | I2, I3 |
| T100 | I1, I2, I4 |
| T100 | I1, I3 |
| T100 | I2, I3 |
| T100 | I1, I3 |
| T100 | I1, I2 ,I3, I5 |
| T100 | I1, I2, I3 |

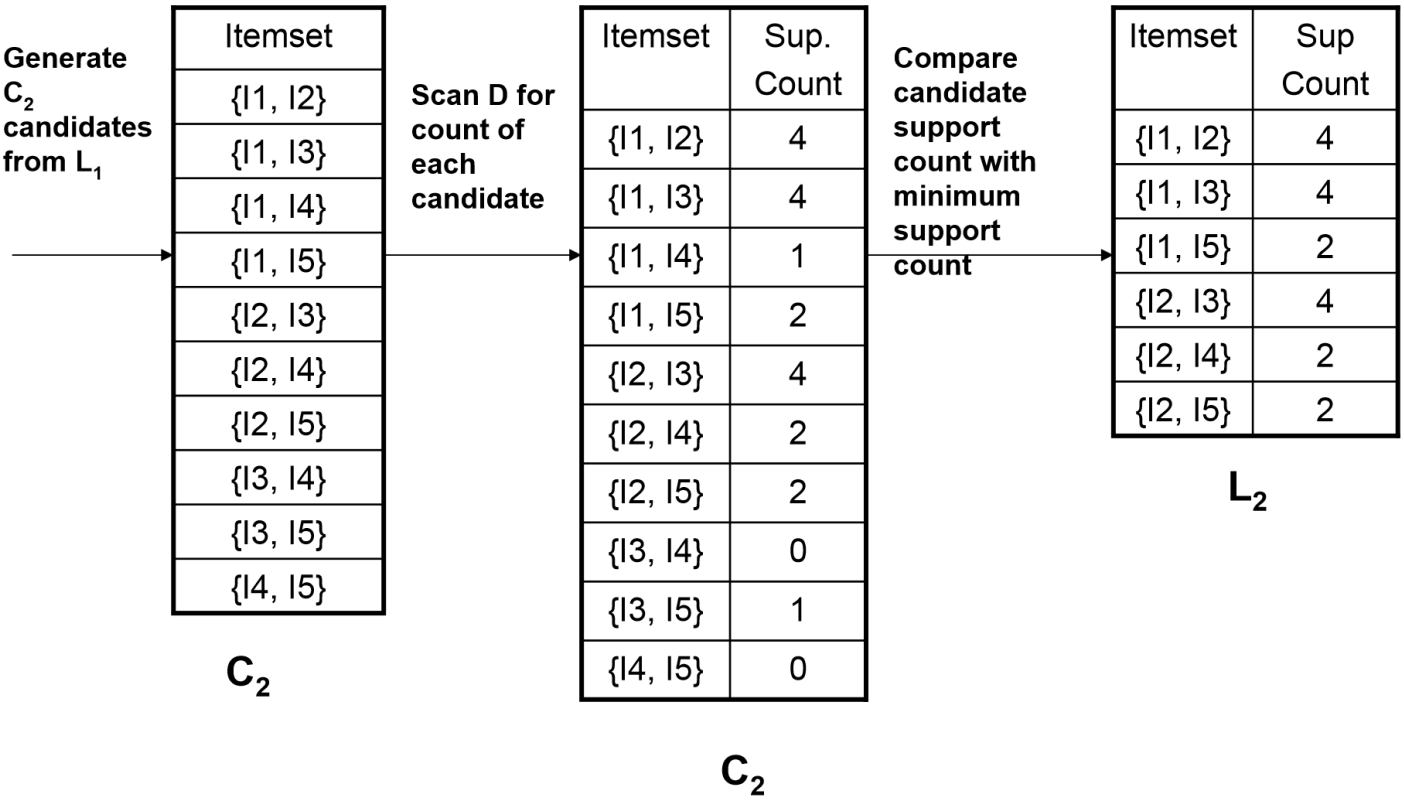
- Consider a database, D , consisting of 9 transactions.
- Suppose min. support count required is 2 (i.e. min_sup = 2/9 = 22 % )
- Let minimum confidence required is 70%.
- We have to first find out the frequent itemset using Apriori algorithm.
- Then, Association rules will be generated using min. support & min. confidence.

# **Step 1**: Generating 1-itemset Frequent Pattern




- The set of frequent 1-itemsets, $L_1$ , consists of the candidate 1-itemsets satisfying minimum support.
- In the first iteration of the algorithm, each item is a member of the set of candidate.

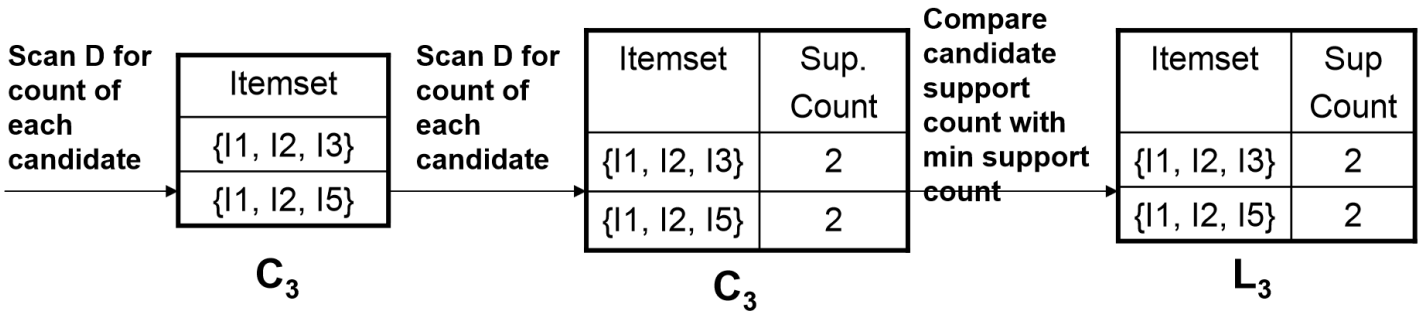
# **Step 2**: Generating 2-itemset Frequent Pattern



## **Step 2**: Generating 2-itemset Frequent Pattern

- To discover the set of frequent 2-itemsets, $L_2$ , the algorithm uses $L_1$ *Join* $L_1$ to generate a candidate set of 2-itemsets, $C_2$.
- Next, the transactions in D are scanned and the support count for each candidate itemset in $C_2$ is accumulated (as shown in the middle table).
- The set of frequent 2-itemsets, $L_2$ , is then determined, consisting of those candidate 2-itemsets in $C_2$ having minimum support.
- Note: We haven't used Apriori Property yet.

# **Step 3**: Generating 3-itemset Frequent Pattern




- The generation of the set of candidate 3-itemsets, $C_3$ , involves use of the Apriori Property.
- In order to find $C_3$, we compute $L_2$ *Join* $L_2$.
- $C_3$ = L2 *Join* L2 = {{I1, I2, I3}, {I1, I2, I5}, {I1, I3, I5}, {I2, I3, I4}, {I2, I3, I5}, {I2, I4, I5}}.
- Now, Join step is complete and Prune step will be used to reduce the size of $C_3$. Prune step helps to avoid heavy computation due to large $C_k$.

# **Step 3**: Generating 3-itemset Frequent Pattern

- Based on the Apriori property that all subsets of a frequent itemset must also be frequent, we can determine that four latter candidates cannot possibly be frequent. How ?
- For example , lets take {I1, I2, I3}. The 2-item subsets of it are {I1, I2}, {I1, I3} & {I2, I3}. Since all 2-item subsets of {I1, I2, I3} are members of $L_2$, We will keep {I1, I2, I3} in $C_3$.
- Lets take another example of {I2, I3, I5} which shows how the pruning is performed. The 2-item subsets are {I2, I3}, {I2, I5} & {I3,I5}.
- BUT, {I3, I5} is not a member of $L_2$ and hence it is not frequent violating Apriori Property. Thus We will have to remove {I2, I3, I5} from $C_3$.
- Therefore, $C_3$ = {{I1, I2, I3}, {I1, I2, I5}} after checking for all members of result of Join operation for Pruning.
- Now, the transactions in D are scanned in order to determine $L_3$, consisting of those candidates 3-itemsets in $C_3$ having minimum support.

# **Step 4**: Generating 4-itemset Frequent Pattern

- The algorithm uses $L_3$ *Join* $L_3$ to generate a candidate set of 4-itemsets, $C_4$. Although the join results in {{I1, I2, I3, I5}}, this itemset is pruned since its subset {{I2, I3, I5}} is not frequent.
- Thus, $C_4 = \varphi$ , and algorithm terminates, having found all of the frequent items. This completes our Apriori Algorithm.
- What's Next ?

  These frequent itemsets will be used to generate strong association rules ( where strong association rules satisfy both minimum support & minimum confidence).

# **Step 5:** Generating Association Rules from Frequent Itemsets

- Procedure:
  - For each frequent itemset ***"l",*** generate all nonempty subsets of ***l.***
  - For every nonempty subset ***s*** of ***l***, output the rule **"s → (l-s)"** if **support_count(l) / support_count(s) >= min_conf** where min_conf is minimum confidence threshold.

- Back To Example:

  We had L = {{I1}, {I2}, {I3}, {I4}, {I5}, {I1,I2}, {I1,I3}, {I1,I5}, {I2,I3}, {I2,I4}, {I2,I5}, {I1,I2,I3}, {I1,I2,I5}}.
  - Lets take ***l*** = {I1,I2,I5}.
  - Its all nonempty subsets are {I1,I2}, {I1,I5}, {I2,I5}, {I1}, {I2}, {I5}.

# **Step 5:** Generating Association Rules from Frequent Itemsets

- Let minimum confidence threshold is , say 70%.
- The resulting association rules are shown below, each listed with its confidence.
  - R1: I1 ^ I2 → I5
    - Confidence = sc{I1,I2,I5}/sc{I1,I2} = 2/4 = 50%
    - R1 is Rejected.
  - R2: I1 ^ I5 → I2
    - Confidence = sc{I1,I2,I5}/sc{I1,I5} = 2/2 = 100%
    - R2 is Selected.
  - R3: I2 ^ I5 → I1
    - Confidence = sc{I1,I2,I5}/sc{I2,I5} = 2/2 = 100%
    - R3 is Selected.

# **Step 5:** Generating Association Rules from Frequent Itemsets

- R4: I1 → I2 ^ I5
  - Confidence = sc{I1,I2,I5}/sc{I1} = 2/6 = 33%
  - R4 is Rejected.
- R5: I2 → I1 ^ I5
  - Confidence = sc{I1,I2,I5}/{I2} = 2/7 = 29%
  - R5 is Rejected.
- R6: I5 → I1 ^ I2
  - Confidence = sc{I1,I2,I5}/ {I5} = 2/2 = 100%
  - R6 is Selected.

    In this way, We have found three strong association rules.

# Methods to Improve Apriori's Efficiency

- Hash-based itemset counting: A *k*-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent.
- Transaction reduction: A transaction that does not contain any frequent k-itemset is useless in subsequent scans.
- Partitioning: Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB.
- Sampling: mining on a subset of given data, lower support threshold + a method to determine the completeness.
- Dynamic itemset counting: add new candidate itemsets only when all of their subsets are estimated to be frequent.

# Fifa dataset

Thursday, July 4, 2019 8:09 PM

For the project:

1 - **DATA INSPECTION / PREPERATION / CLEANING**

-- max min avg of the columns - Done

-- Create new field 'Player_type': Combine positions to do Frwd def mid

-- null/nas ????
- Check count of entries with Null's for a Variable/feature that matters
- Drop the full row entries if not too many.

-- Values in CAM/RM/RW/LB fields - figure out and fix.

2 - **EXPLORATORY DATA ANALYSIS - Best Players**

Top 10 players by Overall

Histogram of division of count of players by 'Overall'

Top 3 player-types (Image of Player (img url) + Player type + Overall)

Top 3 players for each position. (Image of Player (img url) + Position + Overall)

Highest valued players - Top 10 by 'value' (Image of Player (img url) + $Amt)

Highest wage players - Top 10 by 'wage' (Image of Player (img url) + $Amt)

Avg. 'Overall' for each team - Bar graph.

Top 200 players (by overall) divided by club
Can show on bubble chart.

Top 200 players by coutry of origin can show on map
with a Heat map on World map.

3 -
**MLR**

Dependent Variable / Target : 'VALUE'
Indep Variable / Features: Overall, Potential, Nationality, age, week foot, work rate etc.

***Key Question***
***Predict ? / why / whats the point? / Whats the Goal??***
If a new player comes on to the scene with age - , club - , potential -, crossing -, tackling- ..... what would be his value.

- Correlation between variables
- Scatter plots
- line graphs
- Then model, Rsqd etc....
- Then prediction, RMSE....

Would document all of the above in Text explanation.

4 -
**CLASSIFICATION**

Features: Crossing, finishing, heading, Fkaccuracy,Preferred foot etc.
Target: Position

Do kmeans / follow with decision trees / then Random forests.

PYTHON NOTEBOOKS - **FIFA_NOTEBOOK** (EXP DATA ANALYSIS)
- MOVE EXPLORATORY ANALYSIS FROM THIS NB TO BELOW NB
- JUST 5 METRICS FOR NOW.

- VER2 WOULD BE TO ADD IN MORE EXP ANALYSIS (BOX PLOTS / HIST ETC. TO THE NB)

- VER3 WOULD BE GETTING SOME INPUTS FROM THE KAGGLE FIFA KERNAL

**FIFA_DS_V1** (CLUSTERING)

NEED TO ADD A DESCRIPTION TO THE NOTEBOOK. MAYBE AN IMAGE.

==> UPLOADING TO GITHUB AND THEN MAKING FINAL CHANGES IN MARKDOWN CELLS WOULD BE BEST.

check units and how can fix.
Also would scaling be an option. At end of exercise do scaling and check if getting same results for the same conditions.

Finding Best Players:

- Top 10 defenders in europe based on 'Overall'

- Top 10 forwards in South America based on 'Overall'

- Top 10 midfielders in Europe who have highest sum of tackling, short passes, heading

- Best players with the lowest release clause.

- List out top 10 GK's by 1 GK attribute

Finding Best Teams:
- Get top 3 players of a Team by 'Overall'
  List out this for all teams

- List all teams and sum of their overalls.
  Plot as bar-chart or bubble chart.

segmenting players based on attributes
looking at combination of attributes to determine good players

- Build PDF for LinkedIn

- Build moving visual for Reddit

==========

Some Takeaways:-

LEVERAGE TABLEAU + PYTHON

(but may not be possible with Tableau Public.
Can try Google Data Studio)

(under classification complete AUC ROC -- Also Read article sent on Gmail Id on Classif metrics.)

5 -
**CLUSTERING**

Run it on the dataset and see what comes --

Some assumptions / Hypothesis by which it may segment /
sense of no of initial k:

say k = 7 or 10 based on number of different Positions.
or

- Inspect clusters
- Visualize clusters.

1st -
Features

# Fifa ds 1

Sunday, November 3, 2019 3:53 PM

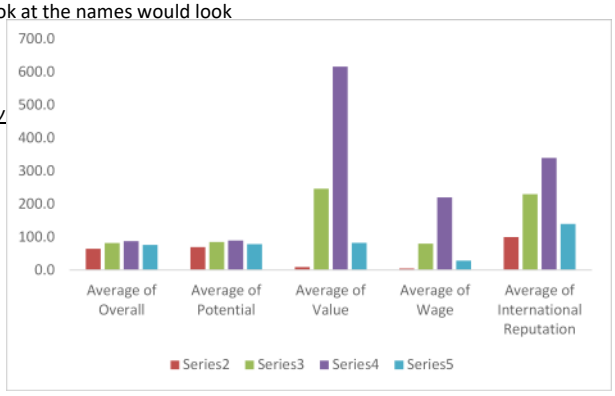
**CLUSTERING**

1 - Run based on k decided by player_type i.e Mid forward etc.
but include all features
k = 4
Your Hypothesis: The dataset would be segmented based on player-type, essentially we would see players of certain player type grouped together.
**Your observations / Examining the clusters**:

| | Cluster 0 | Cluster 1 | Cluster 2 | Cluster 3 |
|---|---|---|---|---|
| Count of players | 15431 | 346 | 65 | 2076 |

- 86% of players are grouped under cluster 0.

- a **Step 1** could be to *maybe start examining with the smallest cluster* to see what actully got them together.
  Cluster 2 (smallest cluster with just 65 entries):
  This was our best players - ~~Messi, Ronaldo, Debryne, Hazard etc.~~. Ideally would not look at the names would look at features of the Cluster.




- Or another 1st step could be to *consider the important features and see how they vary* (something as the UCSD example
  Compare -
  Avg. Overall, Avg. Wage, Avg. Value, Potential, International Reputation.
  - Bar chart comparing the values.
  - Even a box plot showing min max etc for each cluster against each other.

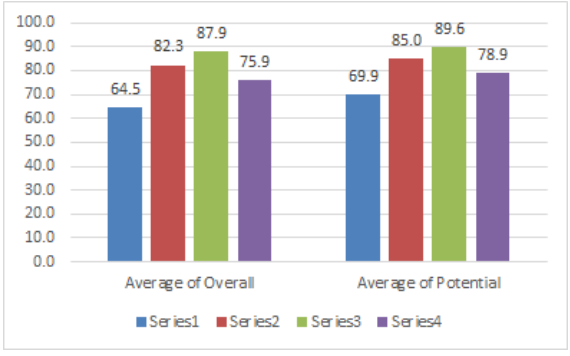
  Cluster No 2 has Players with the highest Value,Wage, International rep.
  Next is Cluster no 1 with highest Wage,Value and international rep.

  Avg. Overall and Avg. potential is quite close to cluster 1

  But Cluster No 1 is certainly the next cluster with high value players.

***Can actually use these labels to classify new incoming data.***

**==>**
This is actually plotting the means of the features. We should be able to actually use the centers file.

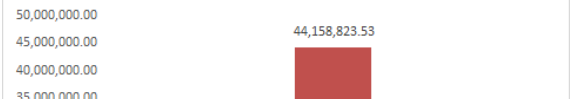


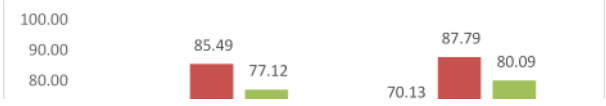

But what about Categorical variables.

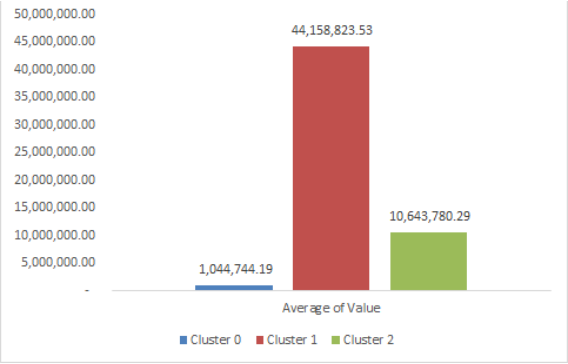
- Trying to Analyze if any trend in PLAYER-Types across clusters

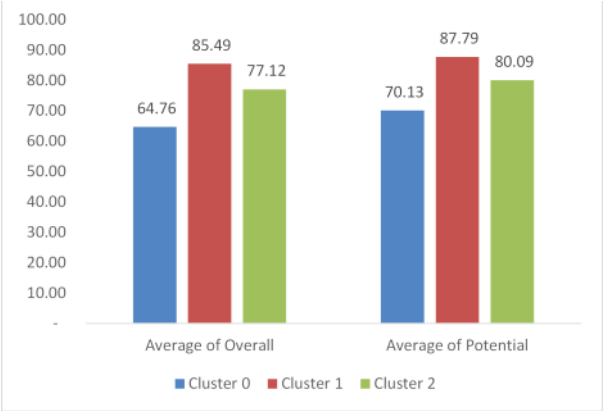
| | Cluster 0 | Cluster 1 | Cluster 2 | Cluster 3 |
|---|---|---|---|---|
| Forward | 14% | 16% | 25% | 17% |
| Midfielder | 41% | 53% | 57% | 49% |
| Defender | 33% | 23% | 12% | 28% |
| Goalkeeper | 12% | 9% | 6% | 7% |

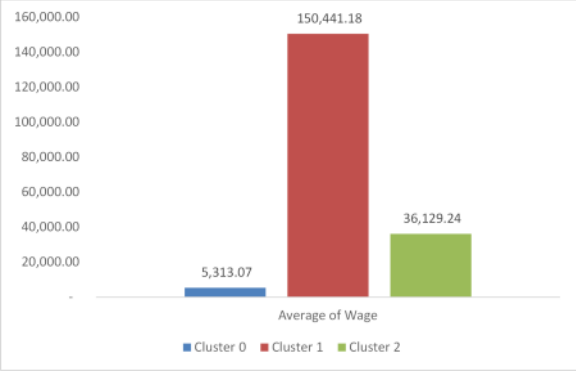
2 - Run multiple iterations and get a WCSS plot
then
run run for k as recommended by WCSS
Based off WCSS we got K=3






| | Cluster 0 | Cluster 1 | Cluster 2 |
|---|---|---|---|
| Count of Player_Type | 15891 | 170 | 1857 |

| PLAYER_TYPE | 0 | 1 | 2 |
|---|---|---|---|
| Defender | 33% | 18% | 27% |
| Forward | 14% | 19% | 17% |
| Goalkeeper | 12% | 8% | 6% |
| Midfielder | 41% | 55% | 50% |

OBSERVATIONS FOR K=3

- Smallest Cluster is Cluster 1 (only areound 1.2% of Total Players)
- Avg Value of Cluster 1 is 4x Avg value of next highest group
- Avg. Wage of Cluster 1 is 3x Avg Wage of next highest grp.
- Cluster 1 also has the highest overall and the highest Potential.
- Distribution of Player type consistent with other clusters.

Clearly Cluster 1 is the Highest Value or the 'Best' players.

- But perhaps 3 cluster is too less.

❖

Next would be to reduce the no of features, especially features that contribute to the same behavior in the algorithm.

3 - Run with dataframe containing only features of Player_type or a few selective features.

- Run for a pre determined k
  Check WCSS
  Check for a assumed value (based on no of players)

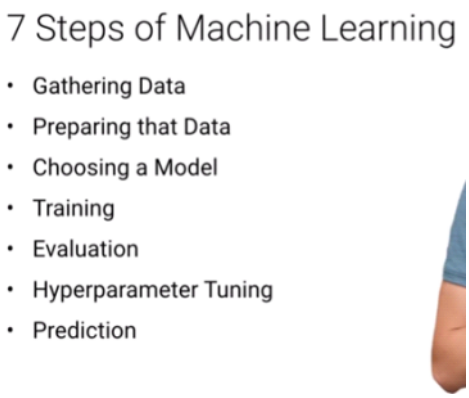
- Run for multiple iterations and check what's optimal value of K
  Run for recommended value

# Random (not from course)

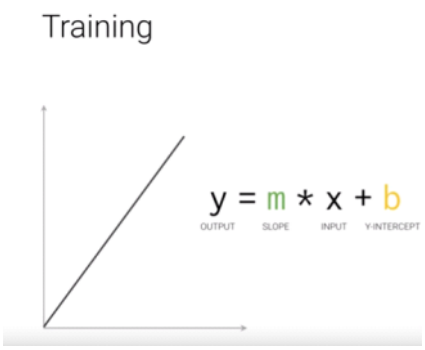
Tuesday, August 6, 2019 11:15 PM

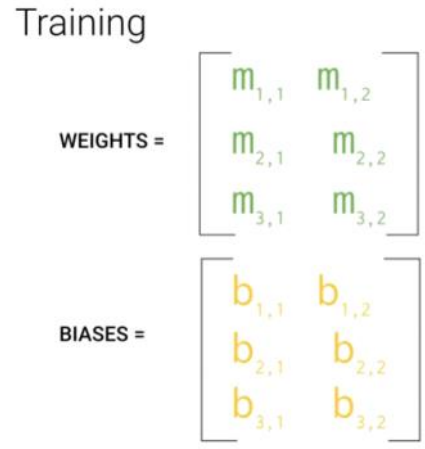
Why is it machine learning

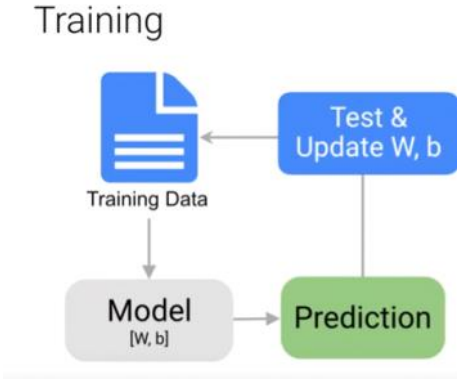
use this to remember steps of ML




So we train the model on the training set and then we test it (evaluate) on a test set so d
Then we use it to make further predictions.
But when training, after we train it once, we can make changes to the the model based c



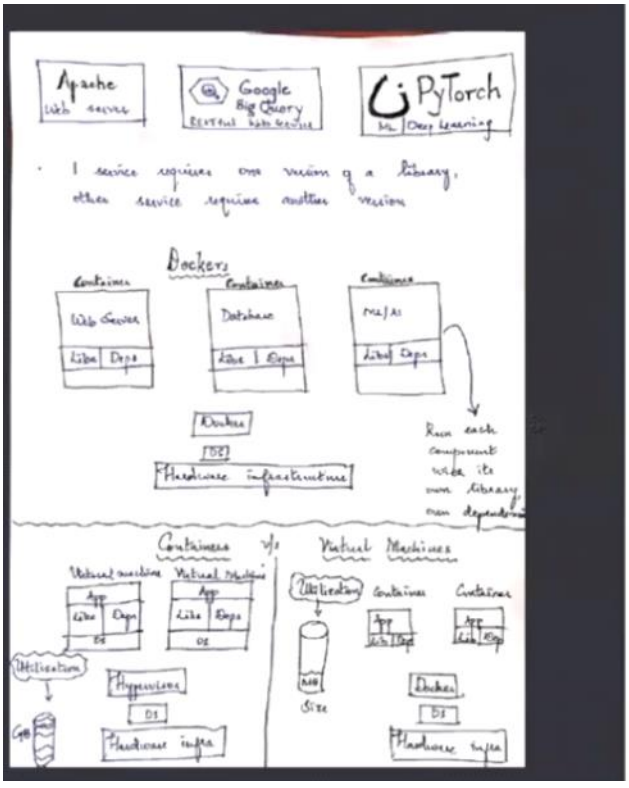



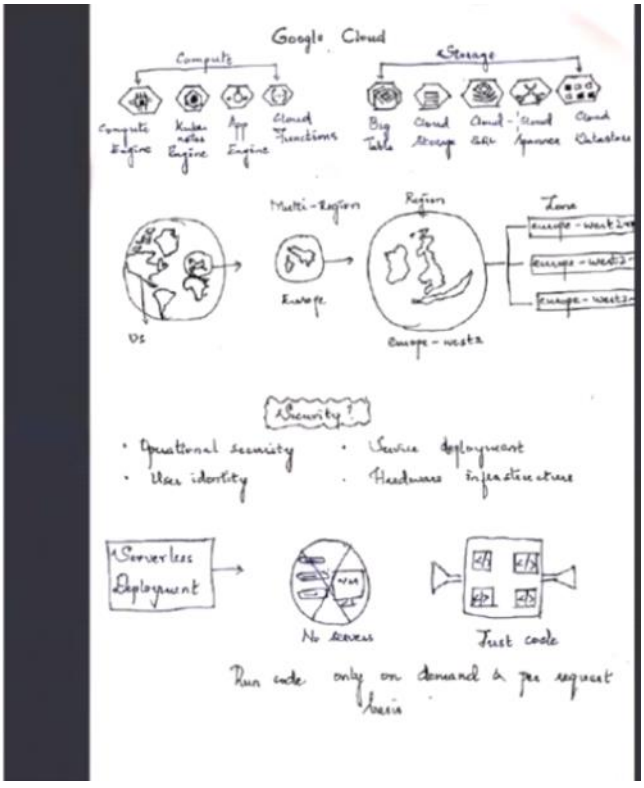

---

---

---
---

**Cloud Computing very basic intro**



# Udemy ML - Eril

Wednesday, May 1, 2019 10:40 PM

# Sec 0: Intro

Sunday, April 28, 2019 9:16 PM

**Q and A**

Machine-Le
arning-A-...

# Sec 1: Part 1 Data Processing

Saturday, March 16, 2019 3:01 PM

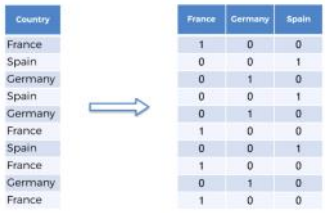
-- **Dependent and Independent Variables**
A good step 1 is determine the Dependent and the Dependent and the Independent variable.
Remember in R indexes start at 1 (not 0 as in Python).

Distinction between R and Python when working towards a ML dataset: (R vs Python)
In Python you need to distinguish between the Matrix of Features and the Dependent variable.
Matrix of features is basically a matrix of the 3 independent variables.

-- **Lecture 15** Dealing With Missing Data
Method followed in tutorial is to take mean of the column.
Used sklearn package and SimpleImputer class

-- **Lecture 16 Categorical Data**
1 - Because ML models are based off mathematical equations it is natural that we would want to represent Categorical variables as numbers.
1st step we do is assign Numerical values to the categorical variables, because the ML model is an equation and for it to work we need numbers.
2 - After assigning numerical values to the categorical, We want to prevent the ML equations from thinking that 1 Categorical is higher value than the other that is why we use dummy variables (called dummy encoding).

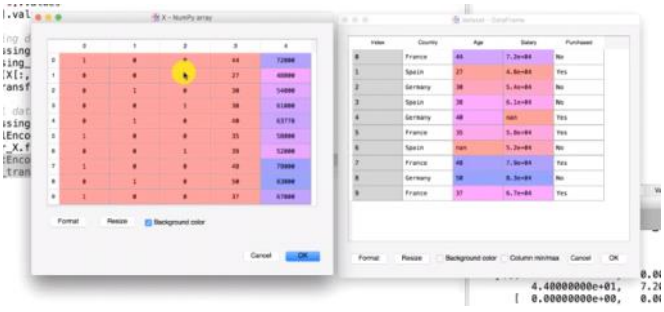
Used Sklearn package > LabelEncoder class and the OneHotEncoder class.
(Review these classes in more detail)



| Country |
|---|
| France |
| Spain |
| Germany |
| Spain |
| Germany |
| France |
| Spain |
| France |
| Germany |
| France |

| France | Germany | Spain |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 0 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |

Below is the change that happens to the actual dataset after encoding.
(on the right is the original dataset)



3 - For the dependent variable we Only encode the categorical variables to Values but no need to create dummy encoding because ML Equation knows it's a dependent variable and therefore there will be no order between the entries.

--> It's a more direct approach in R (you have done it in the CLV model, using Factor) (R vs Python)
*Dataset$Country = factor(dataset$Country, level = c('France', 'Spain', 'Germany')*

-- **Lecture 18 Split Dataset into Training and Test set**
We establish the correlations between independent variables and dependent variables from the training set and once the ML model understands the correlations we will test if the ML model can apply correlations on the test set i.e. if it can make the correct predictions.

One take away is the ML model should learn the Correlations in the training dataset, but should be able to adapt when correlation on the test dataset are slightly different from the training dataset.
You create test, train dataset of the dependent variable and of the independent variable.
Give the test size (in %). A good choice is 20-25%

So better the learning better the prediction but if it learns too much from the training set it will not predict correctly in the test set. That's over fitting.

shift+tab - For help from inside a function (in Jupyter)

**R Data Structures**
PluralSight R course

**Python Tutorials**

Same Steps in Python and R
https://www.dataquest.io/blog/python-vs-r

Python Plotting Tutorial
https://www.dataquest.io/blog/jupyter-notebook-tutorial/

Variable Inspector in Jupyter
https://stackoverflow.com/questions/37718907/variable-explorer-in-jupyter-notebook
For Kaggle:
Do Add custom Package > 'nbextensions'

https://www.kaggle.com/docs/kernels#modifying-the-default-environment

**General Python Commands**

*#Seperate the Independent and Dependent Variables*
*X = carl-tab.iloc[:,:-1] # Matrix of features*
*y = carl-tab.iloc[:,3]*

*dataset['Age'] #To choose 1 column*

*dataset['Age'] #To choose a row of a column*

*dataset['Salary'][2:4] #To choose multiple rows of a column*

*dataset[['Age','Salary']] #To choose 2 columns, basically both columns are passed as a list*

*dataset[['Age','Salary']][2:5] #Choose specific rows of 2 columns.*

*from sklearn.preprocessing import LabelEncoder*
(package) . (library) (class)

*label_encoderX = LabelEncoder()*
(Creating an object of a class)

*df.label_encoder.fit(y)*
(using the object with a method of the class)

In sklearn, data is usually denoted with a capital X, while labels are denoted by a lowercase y. This is inspired by the standard formulation f(x)=y in mathematics,

-- **Lecture 19: <u>Feature Scaling</u>**

Machine learning models use the Euclidian distance {i.e. sqrt of (x2-x1)^2 + (y2 - y1)^2 }

So when 1 of the variables (say salary) has a much wider range than the other variable (say Age) the Euclidian distance will be dominated by Salary. That's why variables need to be transformed so they are in the same scale.

| Standardisation | Normalisation |
| --- | --- |
| $x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation }(x)}$ | $x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$ |

---
SCALING is to basically scaling and shifting the data.
You need to
- **FIT** the scaler object to the data (i.e basically figure out what the range/extremes/mean etc fo the features areas)
- **TRANSFORM** the features (i.e. shift the data of the features)

We FIT based on the Training data.
And TRANSFORM both Training and Test data.

- Feature Scaling would be applied to Dependent variable as well if it includes a wide range of values.

- Even if ML Algorithms are not based on Euclidian distances we would still do Feature scaling, this reduces the time for the Model to run (for ex in Decision tree).

- Some libraries may need you to manually do feature scaling, some may not need it.

-->
All machine learning models in scikit-learn are implemented in their own classes, which are called ***Estimator*** classes

# Sec 3: Regression

Sunday, April 7, 2019 4:04 PM

# Sec 4: Simple Linear Regression

Sunday, April 7, 2019 4:05 PM

--> Don't do the Linear Regression intuition videos.
You already know that. And you have done another tutorial on it.

Refer link for Regression Notes -->
Pluralsight-Regression

**Video 26** : Linear Regression in Python

**Library** - From package import the library **Sklearn linear_model** library

**Class** - the select the class; **LinearRegressionClass()**

**Object** - Create an object of that class. This object would be the regressor that runs on the data.

**Method** - Will use a method (called '**fit**') to fit to the training set. The method is called with the object. This method will take the Xtrain and ytrain data as parameters.

-- When the **regressor is fit** to the dataset using the fit() method, basically the simple linear regression machine is learning the correlations in the training set to predict the test set observations.

All of the predicted values will be stored in a vector. So the predictions of all of the observations in the test set.
Will use the regressor object to make the predictions and will use the **predict()** method. The parameter used in this case would be the Test matrix (X_test, matrix of features).

**Sec 5 - Lecture 37**
Assumptions of linear regression

**Assumptions of a Linear Regression:**

1. Linearity
2. Homoscedasticity
3. Multivariate normality
4. Independence of errors
5. Lack of multicollinearity

# Sec 5: Multivar Linear Reg

Friday, April 19, 2019 1:24 PM

y = bo +b1x1 + b2x2 + ……. +bnXn

**Video 38** - Dummy variables
Done for categorical variables

The regression equation now changes.
y = bo +b1x1 + b2x2+ b4D4 + b5D5…

But you should not include all dummy variables in your model. Always omit 1 dummy variable column.
You wont loose any information, because if the model knows code for 5 of the 6 variables, it can conclude whats the 6th.

**Video 39** - Dummy variable trap

The phenomenon where one or several independent variables in a linear regression predict another is called multicollinearity as a result of this effect the model.
If this exists the model cannot distinguish the effects of Dep var1 from Dep var 2 and so wont work properly.
This is the Dummy variable trap.

So to sum up whenever you're building a model always admit to one dummy variable and this applies irrespective of the number of dummy variables they are in that specific dummy set. If you have nine then you should only include eight if you have 100. Then you should only include 99 of them.
Also note that if you have two sets of dummy variables then you need to apply the same rule to each set. i.e. 2 categorical variables that are to be represented as dummy variables for the model.

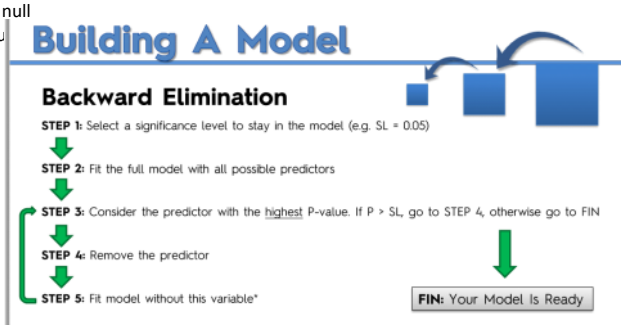
**Lecture 40** -
**P Value** - refer Eril other course / Your notes

Alternate Explanations -
The p-value is NOT the probability the claim is true. The p-value is NOT the probability the null hypothesis is true.
The p-value is actually the probability of getting a sample like ours, or more extreme than ours IF the null hypothesis is true.
A small p-value indicates that by pure luck alone, it would be unlikely to get a sample like the one we have if the null hypothesis is true. If this is small enough we start thinking that maybe we aren't super lucky and instead our assu about the null being true is wrong. Thats why we reject with a small p-value.

**5 methods of building models:**

1. All-in
2. Backward Elimination
3. Forward Selection
4. Bidirectional Elimination
5. Score Comparison

Stepwise Regression




**Video 41** -
MLR Intuition; How to Build the Multiple Linear Regression Model
Backward Elimination is the one covered.
Retain Variables with highest P-value;

**Video 42, 43, 44** -
**Setting up and Running the Regression**

Important note here on Dummy Variable Trap.
The Linear Regression Library normally takes care of this but in some cases you may need to do it manually.

**Video 45, 46, 47** -
**Getting to the optimal model**

Evaluating the model, Backward elimination, Identifying independent variables that have high Statistic Significance i.e. great impact on the dependent variable.
So goal is to find the optimal list of independent variables that have the highest impact on the Dependent variable. This effect can be +ive or -ive

We use the **statsmodel** library to check for this.

Note - the **lower the P value the more significant the variable**. So eliminate highest P value variable is eliminated in each round.

# Sec 20 21:Clustering- Code (kmeans)

Sunday, July 14, 2019 8:23 PM

**REFER UCSD FOR CLUSTERING THEORY**

## *K Means Clustering Code*

```
#Can determine no of initial clusters based on knowledge of dataset
#OR
# Using the elbow method to find the optimal number of clusters

from sklearn.cluster import KMeans
wcss_values = []
for i in range(1, 11):
    kmeans_obj = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans_obj.fit(X)
    wcss_values.append(kmeans_obj.inertia_)

# 'init' parameter for choosing initial no of clusters can be random or as we choose
# why init = k-means++
#other available parameters - max_iter for max # of iterations allowed and n_init.
# random_state to ensure same output each time
# 'inertia' another name for wcss inertia attribute in scikitlearn computes the wcss


y_kmeans = kmeans.fit_predict(X)

#fit_predict that returns for each observation which cluster it belongs to.
#For no of clusters = 5, each entry of y_kmeans would be an int no (0 1 2 3 4) .


#==========#

#Plot the WCSS
plt.plot(range(1, 11), wcss_values)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()


#Plot the clusters


Note that labels can be added back to the original file and they get added appropriately
based on the INDEX
example - say the dataframe X used in clustering algorithm does not have a field like 'Player
Name', but X is actually a Subset of the original dataframe. So output of model with X or
predicted label can be added back to the original dataframe wherein each label matches
correctly to the player appropriately.
(remove nulls from original df before pulling out X, & don't shuffle the output series)


#For a df with 7 features and initial no of clusters = 12
# each of the 12 clusters will have list of seven floating point numbers,
# which denote where the cluster center stands in the seven dimensions of our feature space.


# 2Dimensional data can be viewed as Scatter plot (See alongside)
#Also we have  pd_centers and parallel_plot


centers = model.cluster_centers_

#Provide Features and centers to pd_centers and get returned in a dataframe.
def pd_centers(featuresUsed, centers):
        colNames = list(featuresUsed)
        colNames.append('prediction')

        # Zip with a column called 'prediction' (index)
        Z = [np.append(A, index) for index, A in enumerate(centers)]

        # Convert to pandas data frame for plotting
```

```
# K-Means Clustering

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('E:/Skills training/Machine Learning A-Z/Part 4 -
Clustering/Section 24 - K-Means Clustering/Mall_Customers.csv')
dataset.head()

'''
Spending Score is a field calculated based on Shops per year,
Amount Spent per year, Income, Visits to mall etc. higher the score more the spend

--> Need to segment based on Anual Income and Spending score

'''

X = dataset[['Annual Income (k$)','Spending Score (1-100)']]
X.head()


# Splitting the dataset into the Training set and Test set
"""from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state
= 0)"""

# Feature Scaling
"""from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train)"""

# Using the elbow method to find the optimal number of clusters
from sklearn.cluster import KMeans              #Import the KMeans class
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

#Above we:
# Created an object of KMeans Class as kmeans
#iterated through i between 1 and 11, where i is no of clusters.
#for each no of clusters fit the data X to kmeans object algorithm
#lastly Calculated WCSS and Stored different values of WCSS in a list

#Also 'init' parameter for choosing initial no of clusters can be random or as we
chose
# why init = k-means++
#We have other parameters such as max_iter for max no of iterations allowed
# and n_init. Also 'random_state' is added, check output if not added or for diff
values


#Plot the WCSS

plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()


#Fitting K-Means to the dataset with desired no of clusters
#step same as done above
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
#only here we use fit_predict that returns for each observation which cluster it
belongs to.
#These will be stored in a vector call y kmeans; Each entry is a int Number(0-4) .


# Visualising the clusters

#NOTE - YOU WILL VISUALIZE ONLY FOR 2 DIMENSIONS, if MORE THAN 2
```

```
    # Zip with a column called 'prediction' (index)
    Z = [np.append(A, index) for index, A in enumerate(centers)]

    # Convert to pandas data frame for plotting
    P = pd.DataFrame(Z, columns=colNames)
    P['prediction'] = P['prediction'].astype(int)
    return P
```

It takes in the cluster centers generated by the model, the ones that we just displayed in centers, and creates a Pandas data frame P. Here header is the name of the each feature.
It also adds a new column that denotes the number of the cluster itself. This new column here is called **prediction**. (just the no of the cluster)

```
def parallel_plot(data):
    my_colors = list(islice(cycle(['b', 'r', 'g', 'y', 'k']), None, len(data)))
    plt.figure(figsize=(15,8)).gca().axes.set_ylim([-3,+3])
    parallel_coordinates(data, 'prediction', color = my_colors, marker='o')
```

It takes in the data for plotting, which is the data frame that we just generated out of this pd_centers and generates a colorful graph with different colors to each cluster.
A **parallel coordinates plot** is a quick way to visualize cluster centers along all the seven dimensions, of our features space.

***SUMMARY OF STEPS***

***1 - Data Preparation:***
*Build Dataframe of required features.*
*Remove Nulls and Nas.*

***2 - Scaling***

***3 - Build X***
*- Dataframe with required features (as previous)*
*- Convert Categorical to Numerical*

***4 - KMEANS MODEL***

***4a -*** *Build model with K based on any Hypothesis you may have.*
*& Predict the target labels.*

***4b -*** *Check for optimal value of K based on Elbow method.*
*Build model with optimal value of K.*
*& Predict the target labels.*

***5 - Map the labels back to the dataframe of features.***
*Note - not the dataframe with all features also not the dataframe with dummies.*
*You want to see the real characteristics of features each of the Labels are based on*

***6 - Examine the clusters***
*- Any Hyp that you may have made is that seen.*
*- What else can you say from the clusters obtained.*

*- Plotting the centers.*

```
# Visualising the clusters

#NOTE - YOU WILL VISUALIZE ONLY FOR 2 DIMENSIONS, if MORE THAN 2 DIMENSIONs
#THEN CANNOT VISUALIZE

#Scatterplot of all observations by clusters and Centroids
#Scatterplot is quite simple just supply all x values and all y values

plt.scatter(X['Annual Income (k$)'][y_kmeans==0], X['Spending Score (1-100)'][y_kmeans==0],
        s = 100, c = 'red', label = 'Cluster 1')

plt.scatter(X['Annual Income (k$)'][y_kmeans==1], X['Spending Score (1-100)'][y_kmeans==1],
        s = 100, c = 'blue', label = 'Cluster 2')

plt.scatter(X['Annual Income (k$)'][y_kmeans==2], X['Spending Score (1-100)'][y_kmeans==2],
        s = 100, c = 'green', label = 'Cluster 3')

plt.scatter(X['Annual Income (k$)'][y_kmeans==3], X['Spending Score (1-100)'][y_kmeans==3],
        s = 100, c = 'cyan', label = 'Cluster 4')

plt.scatter(X['Annual Income (k$)'][y_kmeans==4], X['Spending Score (1-100)'][y_kmeans==4],
        s = 100, c = 'magenta', label = 'Cluster 5')


plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
        s = 300, c = 'yellow', label = 'Centroids')
#cluster_center attribute used to get the cluster centroids.

plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

*For Clustering need not take all features into the clustering model.*
*if we want to segment say just by:*
*income, spending_score, age, then use just those features to get an insight.*
*See what kind of insights or segments you may find.*
*For ex - you may see people of certain age, income between 50-70K spend less than 40$ a month.*

# Generic ML Code

Saturday, April 20, 2019 12:36 PM

**STEPS:**

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
df = pd.read_csv('Data.csv')
df.head()
df.shape
df.columns
df.describe.transpose()


#separate independent (FEATURES) and dependent (LABEL) variables
X = df.iloc[:, 2:10]
y = df.iloc[:, 3]

#OR

X = df[['Col1','Col2']]

#Remember, we do Feature Scaling also label encoding for Dependent variables is different from independent variables.
#So, seperating into X and y comes before data scaling and encoding categorical data.


# Taking care of missing data

df.isnull()                                  #Check missing values
df = df.dropna(axis=0, subset=['Col_name'])      #Drops entire row if NA in specified column. default is drop row if NA in any col
df = df.dropna(axis=1)                           #to drop the columns.
NOTE - must create new dataframe when using dropna()
```

**#OR Can use Imputer (now deprecated - need to use SimpleImputer)**

```
from sklearn.preprocessing import Imputer                    #Import the class
imputer = Imputer(missing_values = 'NaN', strategy = 'mean', axis = 0)          #Object instantiated
imputer = imputer.fit(X[:, 1:3])            #Fit it to the column or train the imputer Note - can only do for numerical columns
X[:, 1:3] = imputer.transform(X[:, 1:3])     #Transfor the col using the transform() method
```

```
# Categorical data
```

2 things are done here -

- Convert from Categorical to Numerical {Label Encoder}
- then Encode as 100 010 etc. to remove heirarchy of the numbers {OneHotEncoding} - Create dummy variables

```
# Independent Variable - Convert to Numerical and creating dummy variables
new_df = pd.get_dummies(df_name,columns=['categorical-ColName'],drop_first=True)
#drop_first=True drops one column from the resulted dummy features. The purpose is to avoid multicollinearity
#if Colname not specified it detects all categorical col names and converts to dummy.
#Single step command  (does both Label encoding i.e. categorical to numerical and creating dummy variable)
#Other option is column transformer(alongside) .

# Dependent Variable - Convert to Numerical
labelencoder_y = LabelEncoder()                #Convert to numerical
y = labelencoder_y.fit_transform(y)
```

- For the dependent variable we Only encode the categorical variables to Values but no need to create dummy encoding because ML Equation knows it's a dependent variable and therefore there will be no order between the entries.

Use *Get dummy* for independent variables.

For dependent variables use *Label encoder*.

```
#Remember to include in model: (total no of dummy variables columns - 1; always omit 1)
# if 2 sets of dummy variables for 2 different fefatures , omit 1 from each.
#Your not losing any information. if the model knows the rest, it can conclude the last.


# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X = LabelEncoder()

X[:, 0] = labelencoder_X.fit_transform(X[:, 0])
#Change this to be index of column that is to be encoded
onehotencoder = OneHotEncoder(categorical_features = [0])
#Change here as well
X = onehotencoder.fit_transform(X).toarray()
#Note output is a Numpy array not a Dataframe

#Categorical features is Deprecated.
#Above implementation is for 1D Series but can apply to a dataframe.

Standard steps happening here is--
 - Import the Encoder Class
 - Instantiate an object of the Encoder Class
 - Fit to the Column you want to encode using the fit method
 - then transform the column of the dataframe using transform method
```

**#Using column transformer**

```
from sklearn.compose import ColumnTransformer, make_column_transformer

pre_1 = make_column_transformer((OneHotEncoder(),['categ_col_name']),remainder='passthrough')

df_name_new = pre_1.fit_transform(dfname)
#returns an Array. But can convert to a dataframe.
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)     #Test size change as required
```

#the train_test_split function shuffles the dataset and then makes the split. It does this using a pseudorandom number generator. To make sure we get same outcome each time we do the split we provide the pseudorandom number generator with a fixed seed using the random_state parameter.
Also, default split test_size is 0.25

```
# Feature Scaling

from sklearn.preprocessing import StandardScaler      #import class
scalerObj_X = StandardScaler()                        #Create object of the class
scalerObj_X.fit(X_train)          # FIT based on the training set using the fit method
X_train_scaled = scalerObj_X.transform(X_train)          # TRANSFORM the training set
X_test_scaled = scalerObj_X.transform(X_test)          #TRANSFORM the Test Set

sc_y = StandardScaler()
y_train_scaled = sc_y.fit_transform(y_train)        # Scale dependent variable only if needed???
```

Again steps are pretty standard -
- import the class
- instantiate the object of the class
- Apply (FIT it to the df using the FIT() method and transform the dataframe using the TRANSFORM() method)
- Can do *fit_transform* which is equivalent to calling *fit* and then *transform*
    - Besides Standard Scaler there is also ***MinMaxScaler***. But does not work as well if there are heavy outliers.

```
#Create/Train the Model (i.e. fit to Training data)

#Evaluate the Model (i.e. Test it with Test data)

#Apply model to into production or application.
```

# Sklearn Design

Saturday, October 12, 2019 11:10 AM

**SCIKIT-LEARN DESIGN**

Scikit-Learn's API is remarkably well designed. The main design principles are:[16]

- **Consistency**. All objects share a consistent and simple interface:
- ***Estimators***. Any object that can estimate some parameters based on a dataset is called an *estimator* (e.g., an imputer is an estimator). The estimation itself is performed by the fit() method, and it takes only a dataset as a parameter (or two for supervised learning algorithms; the second dataset contains the labels). Any other parameter needed to guide the estimation process is considered a hyperparameter (such as an imputer's strategy), and it must be set as an instance variable (generally via a constructor parameter).
- ***Transformers***. Some estimators (such as an imputer) can also transform a dataset; these are called *transformers*. Once again, the API is quite simple: the transformation is performed by the transform() method with the dataset to transform as a parameter. It returns the transformed dataset. This transformation generally relies on the learned parameters, as is the case for an imputer. All transformers also have a convenience method called fit_transform() that is equivalent to calling fit() and then transform() (but sometimes fit_transform() is optimized and runs much faster).
- ***Predictors***. Finally, some estimators are capable of making predictions given a dataset; they are called *predictors*. For example, the LinearRegression model in the previous chapter was a predictor: it predicted life satisfaction given a country's GDP per capita. A predictor has a predict() method that takes a dataset of new instances and returns a dataset of corresponding predictions. It also has a score() method that measures the quality of the predictions given a test set (and the corresponding labels in the case of supervised learning algorithms).[17]
- **Inspection**. All the estimator's hyperparameters are accessible directly via public instance variables (e.g., imputer.strategy), and all the estimator's learned parameters are also accessible via public instance variables with an underscore suffix (e.g., imputer.statistics_).
- **Nonproliferation of classes**. Datasets are represented as NumPy arrays or SciPy sparse matrices, instead of homemade classes. Hyperparameters are just regular Python strings or numbers.
- **Composition**. Existing building blocks are reused as much as possible. For example, it is easy to create a Pipeline estimator from an arbitrary sequence of transformers followed by a final estimator, as we will see.
- **Sensible defaults**. Scikit-Learn provides reasonable default values for most parameters, making it easy to create a baseline working system quickly.

**Datasets loaded by Scikit-Learn** generally have a similar dictionary structure including:

- A DESCR key describing the dataset
- A data key containing an array with one row per instance and one column per feature
- A target key containing an array with the labels

1 - the default datasets in sklearn are stored as a Bunch object. Its format is key-values.
the data is kind of nested within this object

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

the iris_dataset now already has 2 dataframes 'data' and 'target' that are already seperated out. We directly split these into train and test

--
The **fit, predict**, and **score** methods are the common interface to supervised models in scikit-learn.

# Python - Analytics Basics Eril

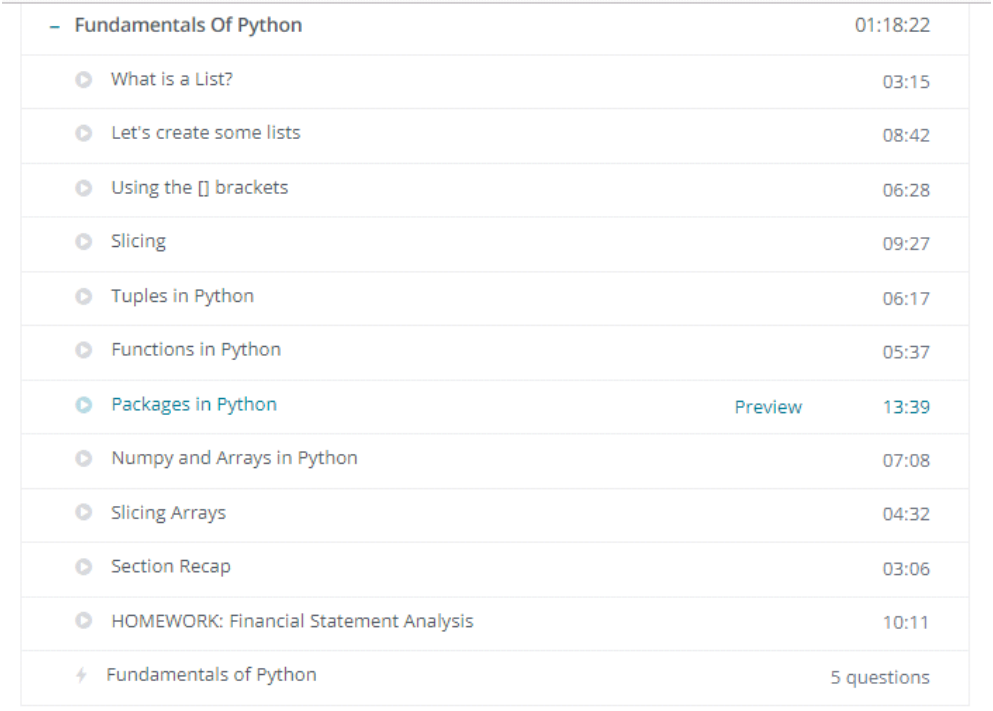
Tuesday, June 26, 2018 10:22 PM

# Contents

Tuesday, June 26, 2018 10:22 PM







| | | |
|---|---|---|
| – Matrices | | 01:58:43 |
| Project Brief: Basketball Trends | | 08:16 |
| Matrices | Preview | 03:31 |
| Building Your First Matrix | | 16:50 |
| Dictionaries in Python | | 14:20 |
| Matrix Operations | | 08:34 |
| Your first visualization | | 11:04 |
| Expanded Visualization | | 09:37 |
| Creating Your First Function | | 11:09 |
| Advanced Function Design | | 11:15 |
| Basketball Insights | Preview | 11:17 |
| Section Recap | | 04:07 |
| HOMEWORK: Basketball free throws | | 08:43 |
| Matrices | | 5 questions |
| – Data Frames | | 01:59:26 |

| | | |
|---|---|---|
| – Data Frames | | 01:59:26 |
| Importing data into Python | Preview | 08:25 |
| Exploring your dataset | | 10:51 |
| Renaming Columns of a Dataframe | | 02:56 |
| Subsetting dataframes in Pandas | | 16:31 |
| Basic operations with a Data Frame | | 09:49 |
| Filtering a Data Frame | | 18:52 |
| Using .at() and .iat() (advanced tutorial) | | 09:01 |
| Introduction to Seaborn | Preview | 10:47 |
| Visualizing With Seaborn: Part 1 | | 10:05 |
| Keyword Arguments in Python (advanced tutorial) | | 10:42 |
| Section Recap | | 04:30 |
| HOMEWORK: World Trends | | 06:57 |
| Data Frames | | 5 questions |

| | | |
|---|---|---|
| – Advanced Visualization | | 02:36:28 |
| What is a Category data type? | | 10:29 |
| Working with JointPlots | | 07:38 |
| Histograms | | 07:52 |
| Stacked histograms in Python | | 18:29 |
| Creating a KDE Plot | | 07:59 |
| Working with Subplots() | | 14:05 |
| Violinplots vs Boxplots | Preview | 08:55 |
| Creating a Facet Grid | | 12:28 |
| Coordinates and Diagonals | | 07:54 |
| BONUS: Building Dashboards in Python | | 16:31 |
| BONUS: Styling Tips | Preview | 15:46 |
| BONUS: Finishing Touches | | 14:48 |
| Section Recap | | 05:37 |
| HOMEWORK: Movie Domestic % Gross | | 07:57 |
| Advanced Visualization | | 5 questions |
| – Homework Solutions | | 01:45:54 |

| | |
|---|---|
| – Homework Solutions | 01:45:54 |
| Homework Solution Section 2: Law Of Large Numbers | 08:57 |
| Homework Solution Section 3: Financial Statement Analysis (Part 1) | 10:30 |
| Homework Solution Section 3: Financial Statement Analysis (Part 2) | 13:39 |
| Homework Solution Section 4: Basketball Free Throws | 17:23 |
| Homework Solution Section 5: World Trends (Part 1) | 15:45 |
| Homework Solution Section 5: World Trends (Part 2) | 14:35 |
| Homework Solution Section 6: Movie Domestic % Gross (Part 1) | 16:46 |
| Homework Solution Section 6: Movie Domestic % Gross (Part 2) | 08:19 |
| – Bonus Lectures | 02:04 |
| ***YOUR SPECIAL BONUS*** | 02:04 |

# Basics and Importing data

Saturday, June 30, 2018 12:22 PM

R is a vectorized programming language. Python is not a vectorized programming language it is an object oriented programming language, therefor you will be using more of loops.

**Note on Installing packages** (not in video):
*C:/users/Carlyle: python --version*
*C:/users/Carlyle: pip --version*
Both must run from Your command line. If running then you can do:
*C:/users/Carlyle: pip install numpy*
(from your command line itself to install a package)

Then in Python shell you do *import numpy* and proceed to use the package. Note you can import either the entire package or selected functions of the package.

Also my Python location is --
C:\Users\Carlyle\AppData\Local\Programs\Python

Basket analysis (Python)
https://www.kaggle.com/asindico/customer-segments-with-pca

Customer Segmentation (Python)
https://github.com/Hari365/customer-segmentation-python

Path for dataset input in Kaggle
../input/DemographicData.csv

A **module** is simply a file containing Python definitions, functions, and statements. Putting code into modules is useful because of the ability to import the module functionality into your script or IPython session

A **package** is just a way of collecting related modules together within a single tree-like hierarchy. Very complex packages like NumPy or SciPy have hundreds of individual modules so putting them into a directory-like structure keeps things organized and avoids name collisions.

(Try the Kaggle Kernals, so no need to run Python notebooks on local machine)

Kaggle **Kernels**

- Public & private datasets
- Hosted notebooks (Kernels)
- 4 CPUs, 16GB RAM
- Docker container with pre-installed Python and R packages

**Jupyter Notebooks**

- To start Jupyter notebook. In cmd type

C:\users\carlyle> jupyter notebook

- After closing Jupyter Notebook in browser, **running kernal must also be shutdown**.
- Then lastly do QUIT (top right corner) to shutdown the server.

To surpress errors in Python or specfically in Jupyter notebooks:
*import warnings*
*warnings.filterwarnings('ignore')*

### shift+tab - For help from inside a function (in Jupyter)

**Regular Expressions**
When working with Text
https://www.youtube.com/watch?v=abrcJ9MpF60

**Arrays** -
There are 2 type of arrays in Python - Inbuilt arrays and Numpy Arrays (arrays that come with Numpy package)

You pass a list to an array function to get an array.
*import numpy as np*
*List1 = [11,22,54,17]*
*Arr1 = np.array(List1)*

Differences between Arrays and Lists --
- You **Cannot have different data types inside an Array**.
- Given Python is OO, a List as an object has functions already associated with it,
  Ex - *listname.count()*
  The thing is Arrays have way more Methods (functions) available.
  Ex - *arrayname.mean()*

**Slicing of Arrays**
Slicing of arrays is done similar to slicing of lists.
*>>>Arrayname[2:5]*
But the difference is when slicing Arrays you do not create a copy. When a list is sliced you create a new list (the sliced list). With arrays when you slice you are just pulling out a view of the original array. Any change made to the slice, will become a change in the original array. Done to save memory.
Need to be careful of this.

==========

**Reading and Writing to File**

*pd.read_csv(''E:/Skills training/Market Basket Analysis/file_name.csv')*
*df-name.to_csv('E:/Skills training/Market Basket Analysis/NewFileName.csv')*

*Can set 1 of the columns to be the index.*
using *index_col()*
this will take int or logical input
*Also we have.*
*df_name.to_excel(' ..... ')*

**Hiding warnings**

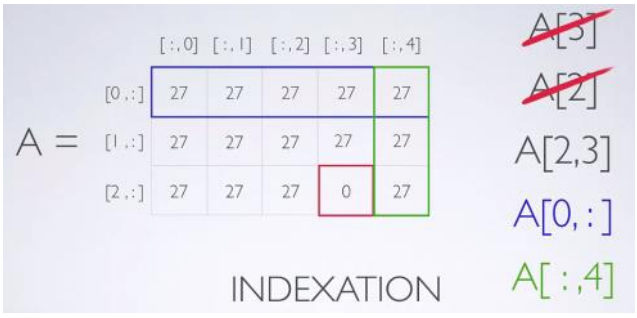
*import warnings #for managing error message*
*warnings.filterwarnings('ignore')*

# ~~Matrices -CHEK and DEL~~

Saturday, July 07, 2018 5:12 PM

**Matrices Indexes**

For matrices as well Row and Column indexing starts at 0.
As seen in figure alongside.




To get a value from a matrix you use:
***MatName[row,col]***

Just remember -
**Indexing for everything in Python starts at 0**.

**Building a Matrix**

1d array:
**np.reshape()** and **np.array()**

One way is --
***np.reshape(*,*,order = 'C')***
* is you data & * is your dimensions, 'C' represents the language C from which this comes.
Here the Matrix is built row wise. So 1st row, then 2nd row and so on. Alterternative is 'F"

```
>>>np.arange(2,20)            # basically 'a range'
array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
>>>raw_data=np.arange(5,20)        #create a 1d numpy array.

>>>np.reshape(raw_data,(5,3))         #reshape it to 5 rows and 3 cols
array([[ 5,  6,  7],
       [ 8,  9, 10],
       [11, 12, 13],
       [14, 15, 16],
       [17, 18, 19]])
```

Given that *raw_data* is an object of np. So we can directly do
*raw_data.reshape((5,3))*
This will also work and give the same output. Again an **object contains data as well and functions that work on the data**.

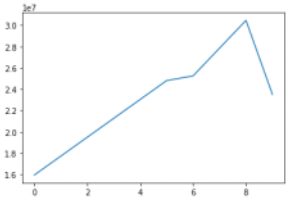
Another way is using --
***np.array()***
This Puts Rows into your matrix.
***Mat_1 = np.array([List1,List2,List3,List4])***
In this case each list is added as a row of the matrix
JUST REMEMBER - it must be a list of lists so [ lis1, lis2, lis3 ] are important

```
lis_r1 = ['hi','how','u']
lis_r2 = ['mmm','ppp','qqq']
lis_r3= [2,5,8]
mat_test = np.array([lis_r1,lis_r2,lis_r3])
print(mat_test)

[['hi' 'how' 'u']
 ['mmm' 'ppp' 'qqq']
 ['2' '5' '8']]
```

# Note that in above output the numbers are converted to string, because as mentio

***np.unique(arr_name)*** *#list unique entries in a numpy array*

```
In [18]: Salary[0]
Out[18]: array([15946875, 17718750, 19490625, 21262500, 23034375, 24806250,
                25244493, 27849149, 30453805, 23500000])
In [8]: plt.plot(Salary[0])
Out[8]: [<matplotlib.lines.Line2D at 0xc0139d0>]
```




**Slicing of Matrices**
i.e. to get an entry from a matrice

```
mat1                        #Prints the whole array
mat1[rowno,columnno]        #Print a single entry of specified row and column no
```
or
```
mat1[Row No]                # Print entire row. See diagram alongside. How to print
```
or
```
mat1[ row x : row y, col p :col q]
```

**Dictionaries** can be used to get specific values from a matrix.
Ex - if we have:
Pdict1 -- Dictionary where Key is PlayerName and Value is a Number for each player
Sdict2 -- Dictionary where Key is Year and Value is a Number for each year
Mat1 -- Matrix with different Player scores for different years.

Then to get a particular players score from the matrix we can do:
*mat1[Pdict['playername'],Sdict['year']]*

**Matrices Operations**

Multiplication or Division in Matrices is NOT like multiplication in Matrices like Maths

**Ex**: Div of FieldGoals matrice by Games matrice to get average goals per game.
*FieldGoalsPerGame = np.matrix.round(FieldGoals / Games )*

FieldGoalsPerGame is now a new matrice with resultant values.
In this way we can do sum, mul, div etc. But note in this case there were equal no of elements in both matrices.

**Visualizations**

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

%matplotlib inline                    # to show visuals in same window. Not mandatory.
plt.rcParams['figure.figsize'] = 8,4  #setting the area size where the plot is shown. Not mandatory. One time only

# Plotting 10 year Salary of 1 player (1st player in the Martix).
plt.plot(Salary[0])
```

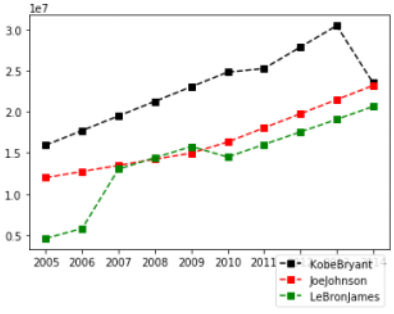
**#Plotting Multiple Player Salaries**
**# Customizing the chart,axis, legend and etc.**

```
plt.plot(Salary[0],c='Black',ls='--',marker='s',label=Players[0])
plt.plot(Salary[1],c='Black',ls='--',marker='s',label=Players[1])
plt.plot(Salary[2],c='Black',ls='--',marker='s',label=Players[2])
plt.xticks(list(range(0,10)),Seasons)
plt.show()
```

**Using Functions and loops** --

```
def games_plot(player_dimension,playerlist):
    colordict={'KobeBryant':'Green','LeBronJames':'Blue','DerrickRose':'Black'}
    for name in playerlist:
        plt.plot(player_dimension[Pdict[name]],c=colordict[name],ls='--',marker='o',label=name)
    plt.legend(loc='Upper left',bbox_to_anchor=(1,1))
    plt.xticks(list(range(0,10)),Seasons)
    plt.show()

playerlist=['KobeBryant','LeBronJames','DerrickRose']
games_plot(Games,playerlist)
games_plot(FieldGoals/Games,playerlist)
```

```
plt.plot(Salary[0],c='Black',ls='--',marker='s',label=Players[0])
plt.plot(Salary[1],c='Red',ls='--',marker='s',label=Players[1])
plt.plot(Salary[2],c='Green',ls='--',marker='s',label=Players[2])
plt.legend(loc='Upper left',bbox_to_anchor=(1,0))
plt.xticks(list(range(0,10)),Seasons)
plt.show()
```




--
Can also assign ***playerlist = Players***, here Players is the original player matrice, so a default parameter assigne
In this case all Players are plotted and when calling the function we don't specify any Player arguments. But if we do specify player arguments then it overrides the default parameter.

# Pandas Notes - 1

Sunday, July 29, 2018 12:12 PM

**Pandas** is one of the best packages to work with Data Frames.
Pandas is quite similar to working with data in R.

Pandas has two main data structures it uses, namely, *Series* and *DataFrames*
--> Pandas **Series one-dimensional labeled array**.
--> Pandas **DataFrame is a 2-dimensional labeled data structure**.

## SERIES

Series is a simple 1-D array like a Python list.
But remember Data type would be Series.

```
pd.Series(np.random.randn(5))    #Creates a Series of 5 random numbers
0   -1.307166
1    0.474930
2    0.599106
3   -0.862343
4    0.323514
dtype: float64
```

A Series is a 1D data frame.
*ingredients = pd.Series({'Flour':'4 cups', 'Milk':'1 cup','Eggs':'2 large','Spam':'1 can'})*

## DATAFRAME

2 Dimensional Tabular structures. Made up of Index(rows) and Columns.
So a structure similar to what you would see in Excel, SQL etc.
Its basically multiple Series combined together. Can have many different datatypes.

```
pd.DataFrame([[1,2,3,4],[22,33,'sha',55]]  #Creates a Dataframe
pd.DataFrame([[1,2,3,4],[22,33,'sha',55]], index=['row1','row2']    #Creates DF with Index names
pd.DataFrame([[1,2,3,4],[22,33,'sha',55]], index=['row1','row2'],columns=['col1','col2','col3','col4
```

Create an example dataframe

```
# Create an example dataframe
data = {'name': ['Jason', 'Molly', 'Tina', 'Jake', 'Amy'],
        'year': [2012, 2012, 2013, 2014, 2014],
        'reports': [4, 24, 31, 2, 3]}
df = pd.DataFrame(data, index = ['Cochice', 'Pima', 'Santa Cruz', 'Maricopa', 'Yuma'])
df
```

| | name | reports | year |
|---|---|---|---|
| Cochice | Jason | 4 | 2012 |
| Pima | Molly | 24 | 2012 |
| Santa Cruz | Tina | 31 | 2013 |
| Maricopa | Jake | 2 | 2014 |
| Yuma | Amy | 3 | 2014 |

For 1D dataframe
*fruits = pd.DataFrame({'Apples':30,'Bananas':21}, index = [0])*
*fruits*
(For this scenario 1D data, index is mandatory)

For 2-D Data frame
*fruit_sales = pd.DataFrame({'Apples':[35,41],'Bananas':[21,34]},*
*index = ['2017 Sales','2018 Sales'])*

Or can create dictionary seperately and then create the data frame. (Example alongside)

*filter_1 = fifa_df['Overall']==90*
#Remember this as countd(email) as in tableau.

**Some Examples of Functions\Methods in Pandas**:
*import pandas as pd*
*demodata1 = pd.read_csv('D:\\Tor_Tutorials\\[FreeTutorials.Us] ….\\Downloaded data\\DemographicData.csv')*

***demodata* is now an Object. Its type is Data Frame**
So **you will have methods (like .head(), .describe() etc.) that you will call with that object**.
Don't look at *demodata* as a variable. It's an object.
That's why there is a way to get rows of the object and columns of the object

```
# No of Rows
len(demodata)
# No of columns
len(demodata.columns)
# Top 6 rows
demodata.head(6)
# Bottom rows
demodata.tail(7)
#transpose
demodate1.describe().transpose()          #preferable to use describe with transpose if many columns in dataframe

# Describe only certain columns i.e. Max min average of certain columns - Age, Overall, Potential, Finishing, Vision
```

Shift+Tab
For Help in Jupyter Notebooks

B to add a line below

*fifa_df[['Age','Overall','Potential','Finishing','Vision']].describe()*

*# All Columns names*
*demodata.columns*
*# More info on columns*
*demodata.info()*
*#Renaming Column names (All column names must be specified)*
*demodata.columns= ['col1','col2','col3','col4','col5']*

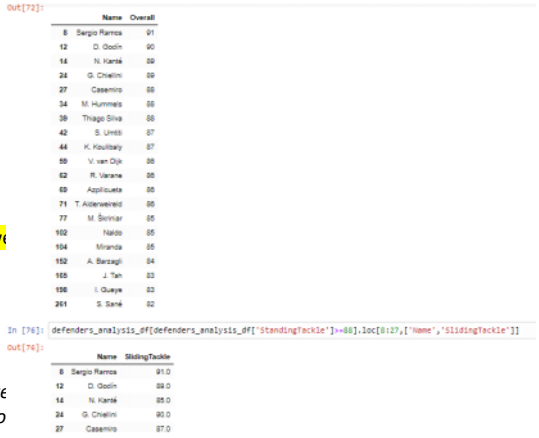
***# Statistics on the data. Will work on the Numeric data columns***
*demodata.describe()*

*t1 = CountryFile['CountryName']*
*type(t1)*
**pandas.core.series.Series**

*t1 = CountryFile[['CountryName']]*
*type(t1)*
**pandas.core.frame.DataFrame**

**## Subsetting Data Frames in Pandas** -
- Subset by Rows
- Subset by Columns
- Subset a combination of both

# Similar to Slicing of Lists



**1 -**
*#Extracting or getting a subset of Rows*
(default extract of a dataset is Rows, WHY? its just how we normally look at part of a dataset; we
*demodata[26:31]* *#prints the mentioned set of rows*
*demodata[2:30:2]* *#using the step (2) in the end to print alternate entries*

**2 -**
*#Extracting or subsetting columns*
*demodata['ColName']* *# Will use ' ' because the Column name is a not a variable cre*
*demodata[['ColName','ColName2']]* *#To get 2 columns as a dataframe we Pass the Co*

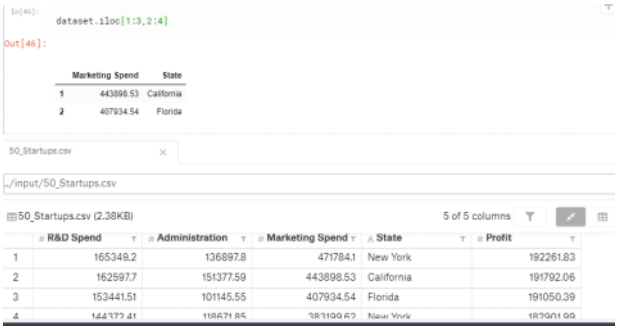
*demodata.ColName* *#Just another option to view a single column at a time (requires col name be a single word)*

**3 -**
*#Subsetting as a section of rows and columns*
*demodata['ColName1'][0:11]*
***demodata[['ColName1','ColName2']][26:31]***

(#Order Rows or Columns does not matter)
*demodata[26:31][['ColName1','ColName2']]* *# same as above*

What is happening here is first you get a DF object that's a subset of columns and then get a subset or extract of rows of that object. Or vice versa.
ex - *fifa_df[['Name','Age']][2:7]*
is same as *fifa_df[2:7][['Name','Age']]*

Try making smaller subset of the data and then trying operations on it when working
i.e. creating a new dataframe object that's a subset of the bigger dataframe
and then working with it.
maybe that would be more intuitive when working.



**loc and iloc**
Another way of getting rows and columns.
Format supplied is ***rows,columns***
loc - gives data based on label supplied
iloc - gives data based on index

**loc**
if a integer is supplied then it is still considered and a Label of the Index.
Because it is labels that are supplied both start and end labels are included in output.
**LOC is Very straight forward,** should not be a bother

Another example to help remember
*top_fifa_tacklers.loc[ : , ['Name','Overall']]*

Supply rows throgh filter + Column labels
*defenders_analysis_df.loc[defenders_analysis_df['StandingTackle']>=88,'Name':'Potential']*
supply the rows , Supply the cols
And choosing selective entries and selective columns (without : )
*defenders_analysis_df.loc[defenders_analysis_df['StandingTackle']>=88,['Name','Potential']]*
Same for rows
*fifa_ds.loc[[8,27,77],['Name','Age','Club']]*

**iloc**

for iloc inputs are integers.
Again indexing is same as Python (i.e. starting 0) but here 2nd boundary is not included in the output

Inputs can also be
- A list or array of integers, e.g. ``[4, 3, 0]``.
- A slice object with ints, e.g. 1:7.
- A boolean array.

For iloc think of 2 steps:
#Step 1 - check what the Column positions (i.e. Indexes) are
*top_tacklers.head(10)*
#
#Step 2 - Choose the Index No based on column you want
*top_tacklers.iloc[0:5,0:3]*
or
*top_tacklers.iloc[0:5,[0,2,3]]*

*defenders_analysis_df[defenders_analysis_df['ShortPassing']>89].iloc[:,[0,2,5]]*
Having row Filter inside iloc does not seem to be working, so Indirectly used a row filter and then selected cols with iloc
though entering a list of boolean values to filter out rows will work

**Examples of loc and iloc**:

Print all rows of few columns
dataset.loc[:,'Administration':'Profit']
dataset.iloc[:,2:4]

Print all columns but selected rows
dataset.loc[3:5,:] # here no row label so integer counted as label index
dataset.iloc[3:5,:]

As an example remember in above loc will output 3 rows because boundary included
but iloc will output 2 rows.

**Group By in Pandas**

*df.groupby('field_to_grpby)['colname_to_agg'].sum() #Group By*

**Ex -**
*ga_page_data.groupby([page_name])[['Bounce Rate','% Exit']].mean()*
*#aggregating 2 fields*
*#mean of Bounce rate , mean of %Exit*
*#Grouped by page_name*

```
[1]: #Filter in single step
     #Finding players with overall > 90 and whats their potential
     fifa_df[fifa_df['Overall']>90][['Name','Age','Nationality','Club','Overall','Potential']]
```
Get the data frame; Filter out the rows as per condition; Choose the required columns

[1]:

| | Name | Age | Nationality | Club | Overall | Potential |
|---|---|---|---|---|---|---|
| 0 | L. Messi | 31 | Argentina | FC Barcelona | 94 | 94 |
| 1 | Cristiano Ronaldo | 33 | Portugal | Juventus | 94 | 94 |
| 2 | Neymar Jr | 26 | Brazil | Paris Saint-Germain | 92 | 93 |
| 3 | De Gea | 27 | Spain | Manchester United | 91 | 93 |

**Operations With Data Frames**

*testproduct = demodata['Birth rate'] * demodata['Internet users'] # Multiplying 2 Columns*
This is quite a powerful thing because the other option would be to create a loop and have each entry multiplied by the other.

*testproduct_1 = (demodata['Birth rate'][2:4])*(demodata['Internet users'][2:4])*

**#Adding a new Column**
*demodata['Calc_Column'] = demodata['Birth rate']*demodata['Internet users']*

**#Removing a Column**
(done using the drop() method)
*demodata = demodata.drop ('Cal_col1',1)*

By default drop() will drop rows specified, we need to specify which axis we want to drop from.
**In Pandas Row Numbers is Axis 0 and Column Names is Axis 1.**
Also, we have to overwrite the existing Data Frame to update the Data Frame.

Finding **Unique entries**
*demodata['Income Group'].unique()*
*OR*
*demodata.Calc_Column.unique()*
Depending on how you usually access columns of the data frame
Ex -
*len(fdata_endurance['ClubName'].unique()) #Count of unique entries in a column called 'ClubName'*

**Anonymous functions**
**short, throw-away functions for one-time use only.**
~~*df_name.apply(lambda x : min(dataset) + max(dataset))*~~
- apply() function to apply a function along an axis of the DataFrame.
OR

**Filtering a Data Frame**

**Filter means essentially Filtering out rows**

Sort values based on a given column
*fdata_endurance.sort_values('Overall' , ascending=False)*

*demodata['Internet users']<2 #Lists all 'Internet users' whose valus is less than 2. This is now filtered rows.*

We can now view all columns of these filtered entries.
*filter_1 = demodata['Internet users']<2*
*demodata[filter_1]*

But ideally you can apply the Filter directly to the data frame, No need to create the filter first.
*demodata[demodata['Internet users']<2]*

```
life_condition = data['Life Ladder']  > 4
year_condition = data['Year']  > 2014
gen_condition = data['Generosity'] > .2

data.loc[life_condition & year_condition & gen_condition]
```

-- Combining 2 filter conditions
If we write something like:
*demodata[filter_1 AND filter_2]*
It would throw an error. Because Python is not a vectorized programming language. It is expecting 1 single value for filter_1 and filter_2 but it gets a series of numbers and therfore throws an ambigious entries error (this above statement would work in R).

So to combine 2 filters we use a different And, a **bitwise** And; **'&'**
*demodata[(demodata['Internet users']<2) & (demodata['Birth rate']>40.00)]*

-->When multiple filter condition, result can be acomplished using **( )** as seen below
*top_oceania_wines = reviews[(reviews.loc[:,'points']>=95) & ((reviews['country']== 'Australia') | (reviews.loc[:,'country']=='Italy'))][:]*

- Sorting a column based on its values (nos or text)
*p_data.sort_values(by='Birth rate',ascending=False)*

*fdata_endurance[['Name','Club','Endurance']][fdata_endurance['Endurance']<50]*
***#Just as normal subsetting, just that row numbers are supplied based on a condition***

Similarily Can do with iloc as well:
*bool_arr = list(fdata_endurance['Endurance']<50) #iloc can take an 'Array' of boolean values as input. Therefor this step is reqd.*
*fdata_endurance.iloc[bool_arr,0:2]*

Will fill up None, NaN, Null values etc.
*df_name['Col1_name] = df_name['Col1_name'].fillna(method='ffill')*

Another Example -
*is_animation = movies['genres'].str.contains('Animation')* #Checks if column genres contains string 'Animation' and creates a Series that contains Boolean values
*movies[is_animation][5:15]* ##This is applied as a filter to the ratings table. And Printing few filter entries

**.at and .iat**
To access individual entry in a dataframe
.iat Looks for Integer location.
.at Looks for Labels, even integers are treated as labels.

*demodata.iat[2,4]*
*demodata.at[3,'Birth rate']*

There is a difference between the 1st character in .iat and .at. In .iat the 1st integer counts the number of Rows and checks corresponding column and gives the cell value, in .at the 1st character points to the Label in the table (not count of row number) and then the column and returns the corresponding cell value.

**Regex to modify string field**
https://stackoverflow.com/questions/39684548/convert-the-string-2-90k-to-2900-or-5-2m-to-5200000-in-pandas-dataframe

#Convert 120M to 120,000,000
*df['CloName'].replace({'K': '*1e3', 'M': '*1e6'}, regex=True).map(pd.eval).astype(int)*

# Pandas Notes - 2 (incl joins and Sqlite)

Monday, August 19, 2019 1:51 PM

## Data Cleaning

Check for missing values in the data by running ***df.isnull()***
***df.dropna()*** to drop the rows or **df.dropna(axis=1)** to drop the columns.

A different approach would be to fill the missing values with other values by using **df.fillna(x)** which fills the missing values with x (you can put there whatever you want) or **s.fillna(s.mean())**

read_sql
read_sql_query
read_sql_table

*df.isnull()* #Show which entries in a DataFrame are NA; returns the full DataFrame with either True or False depending if null or NA

***replace()*** # Globally change values in a DF
*df['col_x'].replace(23,33,inplace= True)* *#a ctrl+H. Find and replace.*
*df['ColX'] = df['ColX'].str.replace("hi","bye")* *#A Control H on colx but for string*

***dropna()*** *#Drops fields (entire row) with missing values Nan (can change to axis 1)*
*df.dropna(axis=0, subset=['Col_name'])*
# By default it will drop row if any entry in the row is Na, but can specify to check if specific column has na and only then drop the row. Can specify multiple columns.

***df_temp.dropna(inplace=True)*** **#FOR All DROP Commands, By default original dataframe is not changed, but inplace = true drops from the original dataframe itself.**

*ffill() and backfill*() #Fills the entry with the previous cell value or the next cell value.

*df.interpolate()* *#it tries to fill the entry using a linear or poly interpolation* (equation of a line)

*new_df = df.drop('Col_Name',1)* #Need to specify Axis. By default *drop()* drops rows

*df['col_name'].astype('float')* *#Changing data type of an entry.*

## Join/Combine

The last set of basic Pandas commands are for joining or combining data frames or rows/columns
**df1.append(df2)**— appends the rows in df1 to df2 and then sorts it(columns should be identical)

**-->**
**pd.concat([df1,df2])** -- adds rows of df2 to the end of df1 OR add

Union All and Union in Pandas

"""Union All """
*df_union_all= pd.concat([df1, df2])*
*print(df_union_all)*

""" Union in pandas"""
*df_union= pd.concat([df1, df2]).drop_duplicates()*
*print(df_union)*

**pd.concat([df1, df2],axis=1)**— add the columns in df1 to the end of df2 (rows should be identical)

**df1.join(df2,on='col1',how='inner')**—**SQL-style join** the columns in df1 with the columns on df2 where the rows for col have identical values. **how** can be equal to one of: **'left', 'right', 'outer', 'inner'**
#note - .join() looks at the 'on' field and the Inde**x???**

So a solution could be to set Index to be the field that joining on.

*cust_regpromo = RegPromoDF.join(jfcust.set_index('Order_No'),on='Order_No',how='inner')*
**OR**

**.merge()** should work same as join.
*joined_df = pd.merge(left_df, right_df, how='inner',*
*left_on= 'col_1name', right_on='col_2name)*
*#basically left_df inner join right_ df on left_on'col' = right_on 'col'*

-- **Reading from SQl**
*import sqlite3*
*conn = sqlite3.connect("../input/pitchfork-data/database.sqlite")*
*music_reviews = pd.read_sql_query("SELECT * FROM artists", conn)*

*To check / change the current working directory.*
*import os*
*print(os.getcwd())*
*os.chdir(c:\\users\desktop)*
*If file in same directory as the python/jupyter working directory then no need to specify the full path. Can do*
*df2 = pd.read_csv(filename.csv)*

*%reset -f Resets Python, clears all variables and the ctrl in console clears console view.*

**ANALYTICAL FUNCTIONS**

*max/min*
*sum*
*mean/median/quantile*
*idxmin/idxmax*

By Default they will be applied along the Column (i.e. along axis =0), but can change to along the columns (i.e setting axis =1, so calculated from left to right)

**-- HEIRARCHIAL INDEXING**
Its kinda like level of aggregation.

**Typing on multiple lines**
Need to be careful if not could get indentation error
*clear_only_filter = (cust_regpromo['PRICETYPE'] == 'CLEARANCE') &*
*(cust_regpromo['ENTRYLEVELVOUCHERUNITDISCOUNT']==0) *
*& (cust_regpromo['Promotions fired on Order Entry']== '')*

# Pandas - 3

Sunday, May 12, 2019 8:12 PM

**Video: Data Merging** -

Like Join in databases

One advantage of Python over say Tableau is in Python you do a lot of data cleaning and data manipulation in Python itself rather in Tableau you would probably clean your data in the DB and then import it.
Things like string manipulation, extracting subsets of a Table etc. are all quite easy to do.

**Concat (vertical)**
The concat function in pandas can be used to stack DataFrames and create a new DataFrame out of them.
The index for the resulting DataFrame will have row indexes from the original tables preserved.
If the two DataFrames given to the concat function have columns that are separate, the resulting DataFrame will have the columns from both frames represented. In that case, some of the cells for the columns that didn't exist in the original DataFrames will end up having NaN or missing values as they will be missing in the first DataFrame.

**Concat (horizontal)**
Instead of having extra rows with missing numbers, we could also try an inner join.
Inner join is a useful operation for merging data, as it combines the column values of two DataFrames, into a new DataFrame, just like we see here. Notice the way to do this is to specify to concat that the join type is inner.
In the horizontal stacking unfortunately, this **isn't the perfect merge for our data** either.

**Append()**
Behavior same as Concat()

**Merge()**
The operation which will give us a true combination of these two frames is called **merge()**.
(Seems like a proper inner join)

*t = df_1.merge(df_2, on='movieId', how='inner')*

## Video: String Operations

Great for Data cleaning and analysis:
split()
replace()
contains() # returns Boolean True or False is a string in part of entries in a column
extract() # to extract part of the data (the string) using a Regular expression

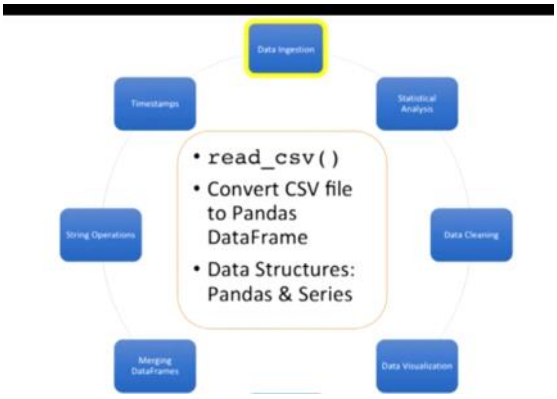
## Video: Parsing Timestamps

Unix Time
- Int format

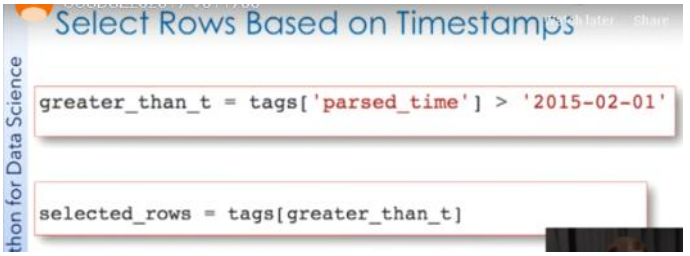
**Datetime64ns** -- General Datatype for Datetime

*pd.todatetime()*
- The unit argument here tells the function what the unit of the input is.

**Regular Expressions**
https://www.regular-expressions.info/


| | | | | | |
|---|---|---|---|---|---|
| 1 | 65 | 208 | dark hero | 1368150078 | 2013-05-10 01:41:18 |



# UC San Diego - Python for DS

Wednesday, May 1, 2019 8:56 PM

# Intro Section

Wednesday, May 1, 2019 8:59 PM

Leo and Altintas (Instructors)
Interesting examples such as using ML to predict how well a software application will run on a certain hardware device and using ML to predict student outcomes (pass or fail) at the end of a school year.
Also, using data to predict how wildfires will spread.
Final Goal - Get a dataset, Develop research question that you would want to answer through the dataset, analyze and visualize the data to answer the question

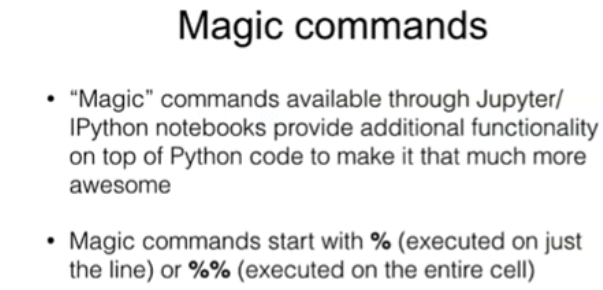
**Jupyter Notebooks**
The notebook file is a file format with the **ipynb** extension that saves code, images, and text in a single document, easy enough to be shared.

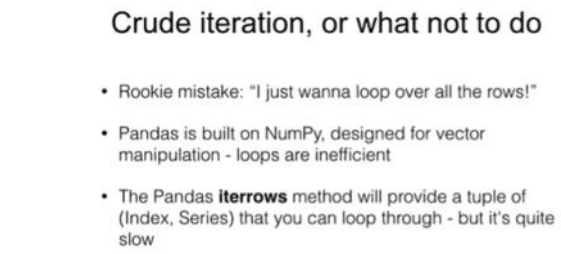
Like I can say, print("Hello World!"),
which is our traditional greeting in any computer science programming course :)

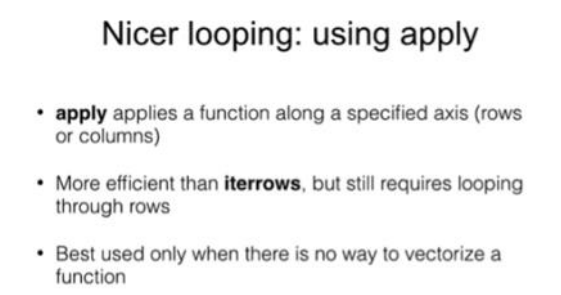
*_ / 1e6*
In Jupyter " _ " refers to output of the last cell executed.

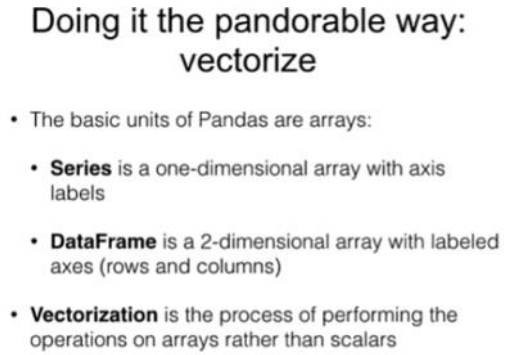
The purpose of a cell in the notebook can be changed from a cell to run code to a '**Markdown cell**' to document something as text.
Markdown cells have support for HTML (from within the cell) and can do some customization. Also '**Latex**' (for scientific equations)
Ex: # This is a heading
(when done in the Markdown cell, will add a header in the markdown cell)
OR

**Markdown guide**
*<h2> Header that we want </h2>*
* Tag for bolding text*
*<br> This is tag for next line*
*<u>For underlining </u>*



# Optimizing with NumPy arrays

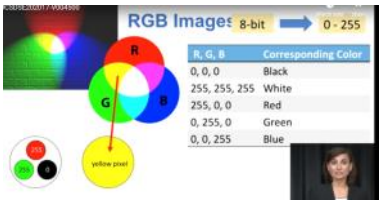
- We've gotten our runtime down from 184 ms to 370 µs
- That's more than 500-fold improvement!

| Methodology | Avg. single run time | Marginal performance improvement |
|---|---|---|
| Looping with iterrows | 184.00 | |
| Looping with apply | 78.10 | 2.4x |
| Vectorization with Pandas series | 1.79 | 43.6x |
| **Vectorization with NumPy arrays** | **0.37** | **4.8x** |

# WIFIRE Case

Sunday, May 5, 2019 5:15 PM

Multiple pixels come together to form an image. (think of it as being like images from mosaic tiles)
The smaller the individual pixel, meaning more pixels used -- the better the image.
Each pixel is made up of a combination of 3 colors (RBG).
Total shades 256x256x256 = 16M total shades



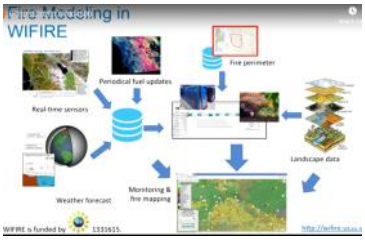

In Python RGB images are NDArrays of shape [Height,Weight X 3] for each RGB layer.

*photo_data[200:800, : ,1]  #selecting entries in 3d array*

*photo_data[200:800, : , :]  #All columns, All layers and rows 200 - 800*

*If just Rows and Column are mentioned, then all entries of the 3rd dimension are included (need not have : )*

**==>** An Application is use a photo of yourself to create your own filters.



# ML - Clustering

Saturday, May 25, 2019 6:47 PM

Clustering is similar to classification, but you don't know what your looking for.

- articulate the goal of cluster analysis.
- And list some ways that cluster results can be applied.

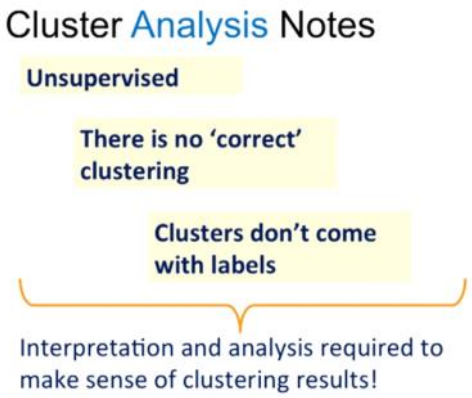

CAN DO CLUSTERING FOR FIFA DS.
SEE Below

**In Clustering** *you don't know what you are looking for*, and you are trying to identify some segments or clusters in your data.
***When you use clustering algorithms on your dataset, unexpected things can suddenly pop up like structures, clusters and groupings you would have never thought of otherwise.***

## ***1*** - *GOAL - organize similar items*

In **cluster analysis**, as you would remember, **the goal is** to organize similar items in your data set into groups, or clusters. By segmenting your data into clusters, you can analyze each cluster more carefully. (**marker** - students grouping together at Babson based on country, language etc.)

Which group a sample is placed in is based on some **measure of similarity**.
The goal of cluster analysis - is to segment data so that the differences between samples in the same cluster are minimized.
Visually, you can think of this as getting samples in each cluster to be as close as possible and the samples from different clusters to be as far apart as possible.



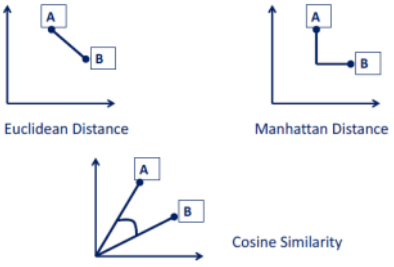

## ***2*** - *Measures of Similarity*

Cluster analysis requires some sort of **metric to measure similarity** between two samples.
Some common similarity measures are:

**Euclidean** distance, which is the distance along a straight line between two points, A and B, as shown on this plot;

**Manhattan** distance, which is calculated on a strictly horizontal and vertical path as shown in the right plot, to go from point A to point B you can only step along either the X axis or the Y axis in the two-dimensional case.
So the path to calculate the Manhattan distance crosses two segments along the axis, instead of along a diagonal path as with the Euclidean distance.

**Cosine** similarily measures the Cosine of the angle between points A and B as shown in the bottom pl




## ***2a*** -

Since distance measures such as Euclidean distance are often used to measure similarity between samples in clustering algorithms, note that it may be necessary to **normalize the input variables** so that no one value dominates the similarity calculation.

Essentially scaling with input variables puts the variables on the same scale so that all variables have equal weighting in the calculation to determine similarity between samples.

## ***3*** - *Clustering Results*

Some things to note about cluster analysis are:
-- Cluster analysis is an **unsupervised task**.
This means that there is no target label for any sample in the data set. So you don't know what each cluster represents.
must *Analyze samples in the cluster to come up with what they are*.
-- So there is no correct clustering results. It all depends on the clusters.
-- Cluster outliers are anomalies and need to be investigated further.
**Question???** In Clustering you don't submit labels but do you Hypothesize what the clusters may be based on what you know about the data??

## ***4*** - *Application*

- Data segmentation.

- Classify new data samples i.e. determine what category/cluster a new data sample falls under.

- Once cluster labels have been determined, samples in each cluster can be used as label data for another classification task.
This process can be used to provide much needed label data for classification.

- Basis for detecting an anomaly
If a sample is very far away or very different from any of the cluster centers, then that sample is a cluster outlier and can be flagged as an anomaly.
Example:
- a sample with value of 150 for age.
- credit card fraud detection or network intrusion detection application.

Example Can try --
**- Fifa Dataset**:
players who can play multiple positions, you don't know what the combinations of positions are but there are groups of players who would generally have abilities to play 2 different positions.
Ex - a RB and RW
Classification:
could be give attributes as inputs, supply label names as RB,LB, GK,DEF etc. and then see how players get classified based on attribute values.

- Apriori Dataset / Original Tableau dataset.

# K-Means clustering

Saturday, May 25, 2019 7:12 PM

*K-means algorithm steps*

- The first step is to select **k initial centroids**. A centroid is simply the center of a cluster. So, K is no of clusters.
- Next, assign each sample in a data set to the closest centroid. This means you calculate the distance between the sample and each cluster center and assign the sample to the cluster with the closest centroid.
- Then you calculate the mean or average of each cluster to determine a new centroid.
- A sample may then move to a different cluster whose mean (centroid) value is closest to the value of the sample.
- The mean is then re-calculated.
- Those two steps (i.e. assign sample to cluster and centroid recalculation) are done repeatedly until some stopping criteria is reached.

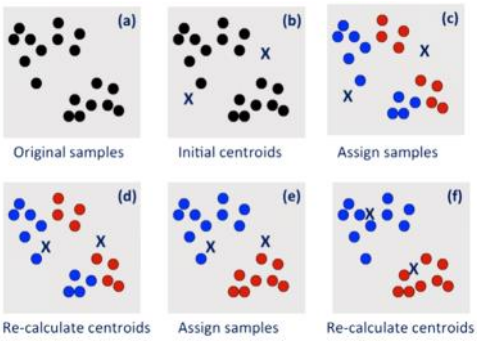


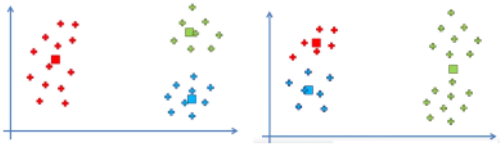

*Illustration of the steps*

A) original data set with some samples.
B) initial centroids are randomly selected (note - 2 clusters).
C) 1st iteration. Here samples are assigned to the closest centroid.
D) centroids are re-calculated.
E) 2nd iterations. Samples are assigned to the closest centroid. Some samples changed cluster assignments during this step.
F) centroids are re-calculated again.
Cluster assignments and centroid re-calculation are **repeated until some stopping criterion** is reached and you get your final clusters as shown in f (not ends with 2 clusters as well).

*Components of K-means steps*

==> ***Selecting Initial Centroids***

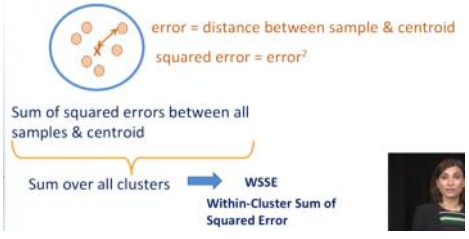
The issue is that the final clusters are sensitive to initial centroids. This means that if cluster results with one set of initial centroids can be different from results with another set of initial centroids.
(see diagram alongside) So know that **there can be 'wrong' or 'bad' clusters**.



The easiest and most widely used approach is to apply k-means several times with different initial centroids randomly chosen to cluster your data set and then select the centroids that give you the best clustering results.
Also, can read on k-means++ algorithm.
The initial centroids need not be values from your dataset.

==> ***WSSE*** (evaluate Cluster results)

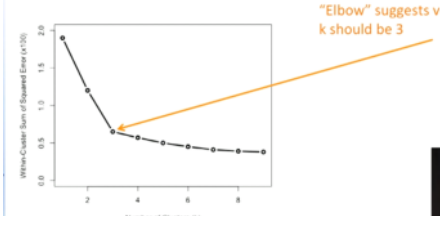
An error measure known as the Within-Cluster Sum of Squared Error.
The error associated with a sample within a cluster.
It is the **distance between the sample and the cluster centroid**. The squared error for the sample then is the square of that distance.
We sum up all the squared errors for all samples for a cluster to get the squared error for that cluster.
We then do the same thing for all clusters to get the final calculation for the Within-Cluster Sum of Squared Error for all clusters in the results of a cluster analysis run.
Given two clustering results, the one with the smaller **WSSE** provides the better solution numerically.
However, as we've discussed before, there's **no ground truth to mathematically determine which set of clusters is more correct than the other**.
In addition, note that increasing the number of clusters (i.e value for k) always reduces the WSSE.
So WSSE should be used with caution. **It only makes sense to use WSSE to compare two sets of clusters with the same value for k** and generated from the same data set.
Also the **set of clusters with the smallest WSSE may not always be the best solution** for the application at hand.
Again, interpretation and domain knowledge about what the cluster should represent and how they will be used are crucial in determining which clustering results are best.
Note that there are several other metrics that are used to evaluate cluster results.



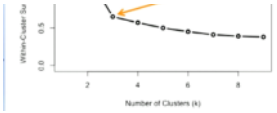

==> ***Choosing the right k***

$$WCSS = \sum_{P_i \text{ in Cluster 1}} \text{distance}(P_i, C_1)^2 + \sum_{P_i \text{ in Cluster 2}} \text{distance}(P_i, C_2)^2 + \sum_{P_i \text{ in Cluster 3}} \text{distance}(P_i, C_3)^2$$

Several methods:
-Visualization techniques can be used to examine the data set to see if there are natural groupings of the samples. Scatterplots and the use of dimensionality reduction are useful here to visualize the data.
-A good value for k is application dependent. So domain knowledge of the application can drive the selection for the value of k.
Example:
- if you want to cluster types of products customers are purchasing, a natural choice for kmight be the number of broad product categories offered.
- k might be selected to represent the geographical locations of respondents to a survey in which case a good value for k would be the number of regions you're interested in analyzing.
- Data-Driven methods that calculate some metric for different values of k to determine the best.
Elbow method:
As we saw in the previous slide, WSSE or **Within-Cluster Sum of Squared Error** measures **how much data samples deviate from their respective centroids in a set of clustering results**. If you plot WSSE for different values for k, we can see how this error measure changes as the value of k changes as seen in the plot. The bend in this error curve indicates a drop in gain by adding more clusters. This Elbow in the curve provides a suggestion for a good value of k.
Note that the elbow cannot always be unambiguously determined especially for complex data.



**deviate from their respective centroids in a set of clustering results**. If you plot WSSE for different values for k, we can see how this error measure changes as the value of k changes as seen in the plot. The bend in this error curve indicates a drop in gain by adding more clusters. This Elbow in the curve provides a suggestion for a good value of k.
Note that the elbow cannot always be unambiguously determined especially for complex data.
In many cases, the error curve will not have a clear suggestion for one value but multiple values.
This can be used as a guideline for the range of values to try for k.




==> ***When to stop***
1. When No changes to the centroids.
i.e **stop when no samples change clusters**, and re-calculating the centroids will not result in any changes.
2. A second stopping criteria could be to stop when the number of samples change in clusters is below a certain threshold.
i.e. **stop when very few samples are changing clusters** each iteration.

## *Interpreting the Results*

The centroid is the mean of the samples assigned to that cluster. Can consider the centroid as a representative sample for that cluster.
- One step to interpret would be to examine the cluster centroids.
Comparing the values of the variables between the centroids will reveal how different or alike clusters are and provide insight into what each cluster represents.
Example: if the value for say variable h is different for a different customer clusters, this indicates that the clusters are including different customers segments by h among other variables.

# K-means code session

Tuesday, May 28, 2019 10:49 PM

IMP CODE POINTS ALREADY NOTED IN KMEANS CODE SHEET UNDER ERIL

- **Examine the dataset**.

  Remove rows with NA's that can affect the result { df.dropna() }
  -- ***dropna()*** drops complete row that has any na values.

- Select Features of Interest for Clustering.
  Create a LIST of features (column names)
  and Use that LIST as filter to get data only of those columns.

- We noticed that some readings are in hundreds, while others are in decimals. It's a large difference, especially if you make it a measure of clustering when you're creating clustering metrics.

- **SCALING**
  To keep values of different columns comparable we scale the values of the features.

  So we will use **Standard Scaler** for that. It's an object, and if we give a data frame to it,
  i.e. to a function of this object, we'll get a nicely scaled input data for our clustering operation.

  *X = StandardScaler().fit_transform(df_name)*

  The **fit_transform()**
  function here combines fit and transform operations, which means it first calculates how much the data set should be transformed to be scaled.
  So it looks at different values and finds how to scale that. Then it's going to apply that transformation to the data frame we give it, in this case select_df, and it's gonna apply the transformation it came up with to that data frame.

  The transformed output obtained will be input for the K-means modelling.

- **Do K-means clustering**
  Input is dataframe X
  If we chose 7-dimensions (i.e. 7 features) and No of clusters = 12 then

  *kmeans = KMeans(n_clusters=12)*
  *model = kmeans.fit(X)*
  *print("model\n", model)*

  The type of the model object is a K-means object.

  # so each of the 12 clusters will have list of seven floating point numbers,
  # which denote where the cluster center stands in the seven dimensions of our feature

  So we are get the model and assign the cluster centers of that model into 'centers' which will be an array.
  We use an attribute of the Kmeans model to build this array.

  *centers = model.cluster_centers_*

  we have now the cluster centers for each 12 clusters.

```
Out[30]: array([[-0.83790332, -1.20123662,  0.37555238,  0.36931133,  0.47401457,
                  0.35726703,  1.36262794],
                [ 0.71641154,  0.44913707,  0.28554319, -0.53643487,  0.47281305,
                 -0.54268212, -0.76657478],
                [ 0.06113923, -0.78891905, -1.19730343, -0.5707786 , -1.0433185 ,
                 -0.58536331,  0.87915471],
                [ 0.13262266,  0.84100449,  1.41242073, -0.6382135 ,  1.67658107,
                 -0.58900661, -0.71382497],
                [-1.18250526, -0.870469  ,  0.44688125,  1.98314596,  0.53837217,
                  1.94404955,  0.90883808],
                [ 1.19007801, -0.25468312, -1.15493166,  2.12261318, -1.0534084 ,
                  2.23936256, -1.13456932],
                [-0.21055439,  0.6330789 ,  0.40856917,  0.73359542,  0.51674121,
                  0.67175466, -0.15171463],
                [-0.16370459,  0.8634041 , -1.31126777, -0.58955336, -1.16710698,
                 -0.60484116, -0.64125855],
                [-0.7065958 ,  0.53374882,  0.17516434, -0.58436054,  0.34479277,
                 -0.59780203, -0.10742667],
                [ 0.23405125,  0.32061871,  1.8879544 , -0.65181511, -1.55175807,
                 -0.57669197, -0.28385048],
                [ 1.36760818, -0.08151489, -1.20649644, -0.0510817 , -1.07519624,
                 -0.03084637, -0.97766689],
                [ 0.24463799, -0.99618746,  0.65805408, -0.54707845,  0.0478233 ,
                 -0.52984474,  1.16129529]])
```

Centers Array: 12 Clusters, 7 numbers in each cluster that represent the centers of each cluster.

- **Plot the clusters**
  It'll be great to **plot these cluster centers** so we **can actually examine which clusters are close to each other and what separates different clusters** from each other.

  To visualize we create 2 functions:

  This could be a step you can maybe do in Tableau. To visualize it.

```
def pd_centers(featuresUsed, centers):
    colNames = list(featuresUsed)
    colNames.append('prediction')

    # Zip with a column called 'prediction' (index)
    Z = [np.append(A, index) for index, A in enumerate(centers)]

    # Convert to pandas data frame for plotting
    P = pd.DataFrame(Z, columns=colNames)
    P['prediction'] = P['prediction'].astype(int)
    return P
```

  Function one here is called **pd_centers**.
  It takes in the cluster centers generated by the model, the ones that we just displayed in centers, and creates a Pandas data frame P. Where the header is the name of the each feature.

It also adds a new column that denotes the number of the cluster itself. This new column here is called **prediction**. (just the no of the cluster)

```
def parallel_plot(data):
    my_colors = list(islice(cycle(['b', 'r', 'g', 'y', 'k']), None, len(data)))
    plt.figure(figsize=(15,8)).gca().axes.set_ylim([-3,+3])
    parallel_coordinates(data, 'prediction', color = my_colors, marker='o')
```

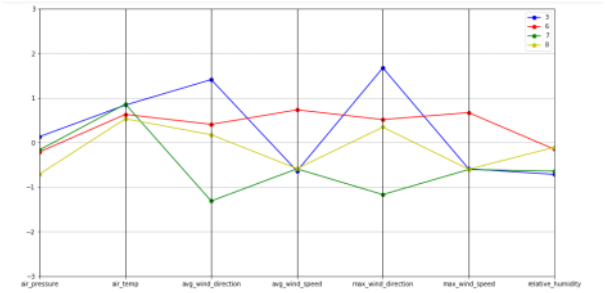
The second utility function is for plotting, and it's called **parallel_plot**.
It takes in the data for plotting, which is the data frame that we just generated out of this pd_centers, and generates a colorful graph with different colors to each cluster.
A **parallel coordinates plot** is a quick way to visualize cluster centers along all the seven dimensions, of our features space.

To summarize the plotting step -->
We'll give the features and the centers to the pd_centers, and we'll get that nice data frame that we generated. Then we plot it

When we plot with conditions such as
Dry days (Humidity <0.5)
Warm Days (Air_temp >0.5)
Cool Days (Rel humidity >0.5 & Air_temp<0.5)

We will basically filter out few clusters and only have few clusters left.
The left clusters are the **clusters that have the behavior**, so to say,
of dry days, warm days, and cool days.

**Warm Days**




For example, let's **look at red and blue clusters** in each graph.

You can notice that they're close to each other for a few features, and far apart on others.
I**f they differ on a particular feature**, **this is one of the points which is likely helpful to distinguish this cluster from others**.

For example, red and blue are pretty far here,
far here, far here, closer here.
So they are pretty different two clusters, right?

They are **different clusters**, **<u>but how different and which features are different for, this plot will help us to identify</u>** that.

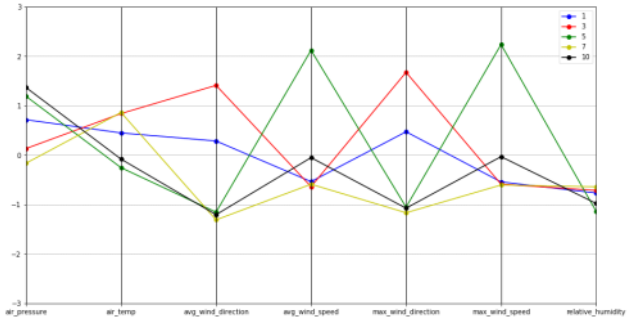
For the dry days graph, the differences are in,
as we looked at, air pressure, air temperature,
features related to wind speed.
Let's look at the warm days, if they are closer
than for dry days.
If you follow the two lines and look at the labels,
looks like they actually differ for all features
related to wind.

And in the cool days graph, they vary for most features,
if you look at that, related to wind speed, right.
These are pretty similar up to the point,
the moment we look at average wind and maximum wind speed,
we see that they are different.

So **we try to gain an intuition about how these clusters differ**.

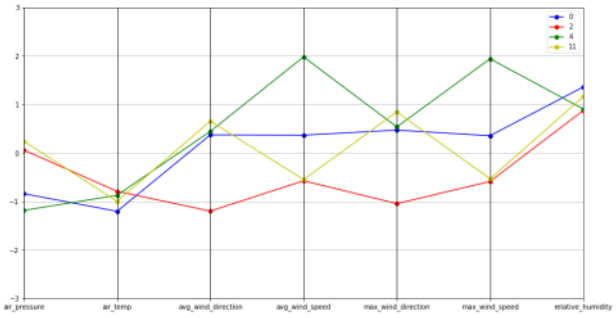
Now think how you could further use this information **???**?

**Dry Days**



## Cool Days

```
17]: parallel_plot(P[(P["relative_humidity"] > 0.5) & (P["air_temp"] < 0.5)])
```



# Classification

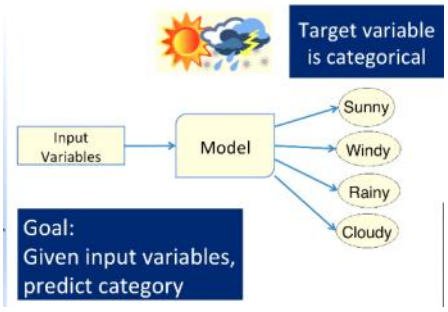
Tuesday, June 11, 2019 10:45 PM

- In this lesson, we will focus on a category
of machine learning problem called classification.

Discuss
- whether classification is supervised or unsupervised ???

and
- binomial classification differs from multinomial classification.

In a classification problem, the input data is presented to the machine learning model,
and the task is to **predict the target** (category) corresponding to the input data.
The **target is a categorical variable**.




So the classification task is to predict the category or label of the target given the input data.

Example, To predict the type of weather.

Target:Weather
Possible values for weather in this case is sunny, windy, rainy, or cloudy.

Input data: Measurements like temperature, relative humidity, atmospheric pressure,
wind speed, wind direction, et cetera.

So given specific values for temperature, relative humidity, and all those other measurements, the task for the model is to predict if the weather will be sunny, windy, rainy, or cloudy for the day.

Dataset:
Each row is a sample with **input variables** Temperature, humidity, and pressure,
**and target variable**, weather.


Each row has specific values for the input variables and a corresponding value for the target variable.
The classification task is to predict the value of the target variable from the values of the input variables.

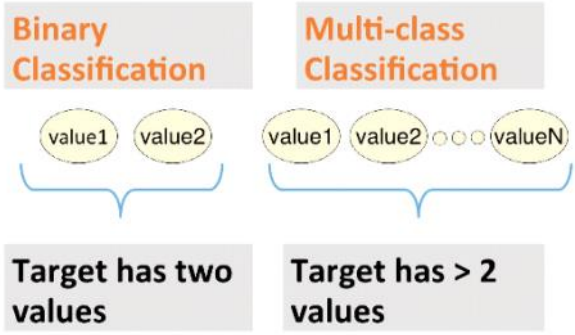
Since a target is provided, we have label data, and so classification is a supervised task.
Recall that in a supervised task, the target, or design output for each sample is given.

The Target variable goes by many names such as target, label, output, class variable, category, and class.

A classification problem can be binary, or multi-class.
With the **binary classification**, the target variable has two possible values, for example yes and no.
With **multi-class classification**, the target variable has more than two possible values, for example
the target can be short, medium, and tall.

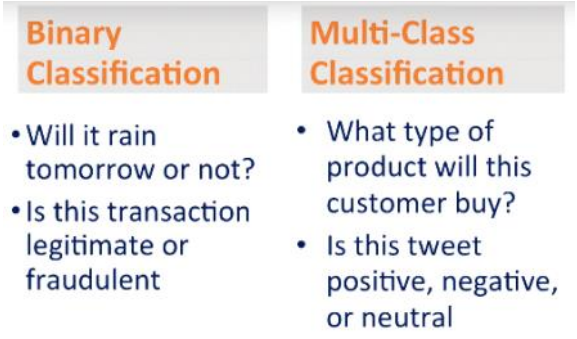
Remember, though, the **target is always a categorical variable in classification**.

See examples alongside.




Unlike regression where you predict a continuous number, you use
classification to predict a category. There is a wide variety of classification
applications from medicine to marketing.
Classification models include:
linear models like - Logistic Regression, SVM
nonlinear ones like - K-NN, Kernel SVM and Random Forests.

applications from medicine to marketing.
Classification models include:
linear models like - Logistic Regression, SVM
nonlinear ones like - K-NN, Kernel SVM and Random Forests.



# Classification Models

Tuesday, June 11, 2019 11:15 PM

- how building a model
differs from applying a model.

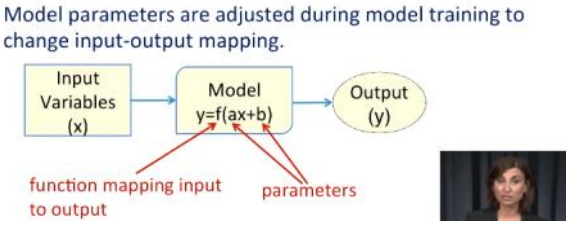
- What building a classification model means,
- explain the difference between building and applying a model,
- summarize why the parameters of a model need to be adjusted,
- describe the goal of a classification algorithm and name some common algorithms for classification.

In a model
Input parameters are adjusted during model training to change input-output mappings
y = mx+c
values of c may be changed to get different values of y. done to train the model.

In the general sense, we can say that the model has parameters and uses equations to determine the relationship between its inputs and outputs.

To kind of summarizwe, building a classification model, as well as other Machine Learning models, involves two phases. The first is the training phase, in which the model is constructed, and its parameters adjusted using training data. A learning algorithm is used to train the model.

The second is the testing phase. This is where the learned model is applied to new data that is data not used in the training model.



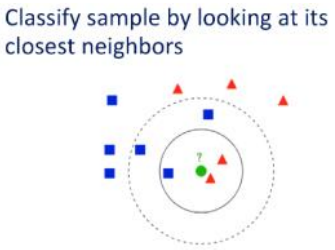

In building a model, we want to adjust the parameters in order to
reduce the model's error.
In the case of supervised tasks, such as classification, this means getting the model's outputs to match the targets, or desired outputs, as much as possible.
Since the classification task is to predict the category or class given the input variables,
you can think of the classification problem visually as carving up the input space into regions corresponding to different class labels.
Remember, training data is previously seen by the model and the test data is not previously seen by the model.

In this diagram, for example, the classification model
needs to form the boundaries to define the regions
separating red triangles from blue diamonds
from green circles, from yellow squares.
In this example, if a sample falls between the region
in the upper right corner, it will be classified
as a blue diamond.
Classification decisions are based on these regions,
and the regions are defined by the boundaries, as indicated
by the dash lines in this diagram.
Thus, these boundaries are referred to
as decision boundaries.

Buil**ding a classification model then means** using the data to adjust a model's parameters in order to form decision boundaries to separate the target classes.
Note that the term classifier is often used to mean classification model.

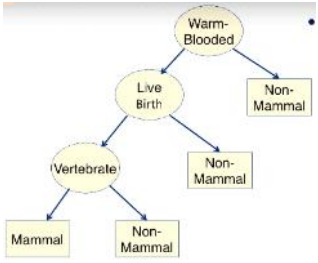
The goal in building a classifier model is to have the model perform well on training, as well as test data.

Some commonly used algorithms for building a classification model:
The classification model processes the input data it receives and provides an output.
Since classification is a supervised task, a target or desired output is provided for each sample.
The goal is to get the model outputs to match the targetsas much as possible.
A classification model adjusts its parameters to get it outputs to match the targets.
To adjust a model's parameters, a learning algorithm is applied. This occurs in the training phase when the model is constructed.
-->
- kNN or k-nearest neighbors,
- Decision Trees
- Naïve Bayes.

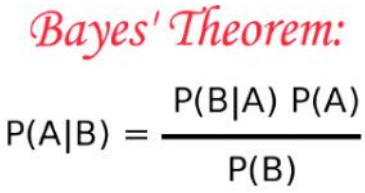



**kNN stands for K-nearest Neighbors**.
This technique relies on the notion that samples with similar characteristics, that is samples with similar values for input, likely belong to the same class.
So classification of a sample is dependent on the target values of the neighboring points.

**Decision Tree**
A Decision Tree is a classification model that uses a tree-like structure to represent multiple decision paths.
Traversing **each path** leads to a different way to classify an input sample.




**Naïve Bayes Model**
Uses a probabilistic approach to classification. Bayes' Theorem is used to capture the relationships within the input data and the output class.
Simply put, the Bayes' Theorem compares the probability of an event in the presence of another event.
Here we see the probability of A if B is present.
An example for this is probability of having a fire if the weather is hot.
You can imagine event B depending on more than one variable, e.g. weather is hot and windy.



# Decision Trees

Wednesday, June 19, 2019 10:07 PM

By the end of this video, you should be able to explain
- how a decision tree is used for classification,
- describe the process of constructing a decision tree for classification,
- and interpret how a decision tree comes up with a classification decision.

The idea behind decision trees for classification
is to split the data into subset,
where each subset belongs to only one class.
This is accomplished by dividing
the input space into pure regions.
That is, regions with samples from only one class.
With real data, completely pure subsets may not be possible,
so the goal is to divide the data into subsets
that are as pure as possible.
That is, each subset contains as many samples as possible
of a single class.
Graphically, this is equivalent to dividing the input space
into regions that are as pure as possible.
Boundaries separating these regions
are called decision boundaries,
and the decision tree model makes classification decisions
based on these decision boundaries.
A decision tree is a hierarchical structure
with nodes and directed edges.
The node at the top is called a root node.
The nodes at the bottom are called leaf nodes.
Nodes that are neither the root node or the leaf nodes
are called internal nodes.
The root and internal nodes have test conditions.
Each leaf node has a class label associated with it.
A classification decision is made by traversing
the decision tree, starting with the root node.
At each node, the answer to the test condition
determines which branch to traverse to.
When a leaf node is reached, the category at the leaf node
determines the classification decision.
The depth of a node is the number of edges
from the root node to that node.
The depth of the root node is zero.
The depth of a decision tree is the number of edges
in the longest path from the root node to the leaf node.
The size of a decision tree
is the number of nodes in the tree.
This is an example of a decision tree,
and it can be used to classify an animal
as a mammal or not a mammal.
According to this decision tree,
if an animal is warm-blooded, gives live birth,

and has a vertebrae, then it is a mammal.
If an animal does not have any of these
three characteristics, then it's not a mammal.
A decision tree is built by starting with all samples
at a single node, the root node, and adding additional nodes
when the data is split into subsets.
At the high level, constructing a decision tree
consists of the following steps.
Start with all samples at a node.
Partition the samples into subsets
based on the input variables.
Here, the goal is to create
subsets of records that are purest.
That is, each subset contains as many samples as possible
belonging to just one class.
Another way to say this is that the subsets
should be homogenous, or as pure as possible.
Repeatedly partition data into successively pure subsets
until stopping criteria are satisfied.
And an algorithm for constructing a decision tree model
is referred to as induction algorithm,
so you may hear the term tree induction
used to describe the process of building a decision tree.
Note that at each split,
the induction algorithm only considers the best way
to split the particular portion of the data.
This is referred to as a greedy approach.
Greedy algorithms solve a subset of the problem
at the time, and is a necessary approach
when solving the entire problem is not feasible.
And by feasible, I mean computable
in a reasonable amount of time or space.
Using a greedy algorithm is necessary for decision trees.
It is not feasible to determine the best tree
given a dataset, so the tree has to be built
in a piecemeal fashion by determining the best way
to split the current node at each step,
and combining these decisions together
to form a final decision tree.
In constructing a decision tree,
how is the data partitioned?
How does a decision tree determine the best way
to split the set of samples at a node?
Again, the goal is to partition the data at the node
into subsets that are as pure as possible.
In this example, the partition shown on the right
results in more homogenous subsets, since these subsets
contain more samples belonging to a single class
than the resulting subsets shown on the left.
So the partition on the right results in purer subsets,
and is the preferred partition.
Therefore, we need a way to measure the purity of a split
in order to compare different ways

to partition a set of data.
It turns out that it works better mathematically
if you measure the impurity,
rather than the purity, of a split.
So the impurity measure of a node
specifies how mixed the resulting subsets are.
Since we want the resulting subsets to have
homogenous class labels, not mixed class labels,
we want the split that minimizes the impurity measure.
A common impurity measure used for determining
the best split is called the Gini index.
The lower the Gini index, the higher the purity
of the split, so the decision tree will select the split
that minimizes the Gini index.
Besides the Gini index, other impurity measures include
entropy, or information gain, and misclassification rate.
The other factor in determining the best way
to partition a node is which variable to split on.
The decision tree will test all variables
to determine the best way to split the nodes,
using a purity measure such as the Gini index
to compare the various possibilities.
Recall that the tree induction algorithm repeatedly
splits nodes to get more and more homogenous datasets.
So when does this process stop building subsets?
When does the algorithm stop growing the tree?
There are several criteria that can be used to determine
when a node should no longer be split into subsets.
The induction algorithm can stop expanding a node
when all samples in the node have the same class label.
This means that this set of data is as pure as possible,
and further splitting will not result
in any better partition of the data.
And since getting completely pure subsets
is difficult to achieve with real data,
this stopping criterion can be modified
to when a certain percentage of the samples in the node,
say 90%, for example, have the same class labels.
The algorithm can stop expanding a node when the number
of samples in the node falls below a certain minimum value.
At this point, the number of samples is too small
to make much difference in the classification results
with further splitting.
The induction algorithm can also stop expanding a node
when the improvement in impurity measure is too small
to make much of a difference in classification results.
Additionally, the tree, or the algorithm, can stop expanding
a node when the maximum tree depth is reached.
This is to control the complexity of the resulting tree.
There can be other criteria that can be used
to determine when tree induction should stop,
but we'll stop here.
Let's take a look at an example

to illustrate the induction process.
Let's say we want to classify loan applicants as being
likely to repay a loan, or not likely to repay a loan,
based on their income and amount of debt they have.
Building a decision tree for this classification problem
could proceed as follows.
Consider the input space of this problem,
as shown in the left figure.
One way to split this dataset into a more homogenous subset
is to consider the decision boundary where income is t1.
To the right of this decision boundary are mostly red
samples, and to the left are mostly blue samples.
The subsets are not completely homogenous,
but that is the best way to split the original dataset
based on the variable income.
The decision tree boundary is represented
in the decision tree by the condition
income is greater than t1 at the root node.
This is the condition used to split the original dataset.
Samples with income greater than the threshold value of t1
are placed in the right subset, and samples with income
less than or equal to t1 are placed in the left subset,
just as shown in the left diagram.
Because the right subset almost perfectly predicts
that lenders will repay properly,
the right subset is now labeled as red,
meaning that the loan applicant is likely to repay the loan.
The second step, then, is to determine
how to split the region outlined in red.
As shown in the left diagram and input space,
the best way to split this data is specified
by the second decision boundary, with debt equals t2.
This is represented in the decision tree on the right
with the addition of the node
with condition debt greater than t2.
Samples with a value for debt greater than t2
are shown in the region around the decision boundary.
This region contains all blue samples,
and so the corresponding node is labeled blue,
meaning that the loan applicant
is not likely to repay the loan.
The third and final split looks at how to split
the region outlined in red in the left diagram.
The best split is specified
by the boundary with income equals t3.
This splits the red region into two pure subsets.
The split is represented in the decision tree
by adding a node with condition income is greater than t3,
and the left resulting node is labeled blue,
and the right resulting node is labeled red,
corresponding to the resulting subsets
with the red border in the left diagram.
We end with the final decision tree on the right,

which implements the decision boundaries
shown as dashed lines in the left diagram.
These decision boundaries partition the dataset as shown.
The label for each region is determined
by the label of the majority of the samples.
These labels are reflected in the leaf nodes
of the decision tree shown on the right.
You may have noticed that the decision boundaries
of a decision tree are parallel
to the axes formed by the variables.
This is referred to as being rectilinear.
The boundaries are rectilinear because each split
considers only a single variable.
There are variants of the tree induction algorithm
that consider more than one attribute
when splitting a value.
However, each split has to consider all combinations
of combined variables, and so such induction algorithms
are more computationally intensive,
or we can also call them more computationally expensive.
There are a few important things to note
about the decision tree classifier.
The resulting tree is often simple to understand
and interpret, and this is one of the biggest advantages
of decision trees for classification.
It is often possible to look at the resulting tree
to see which variables are important
to the classification problem,
and understand how the classification is performed.
For this reason,
many people will start out with a decision tree classifier
to get a feel for the classification problem, even if
they end up using a more sophisticated model later on.
The tree induction algorithm, as described in this lesson,
is relatively computationally inexpensive,
so training a decision tree for classification
can be relatively fast.
The greedy approach used by the tree induction algorithm
determines the best way to split the proportion
of the data at a node, but does not guarantee
the best solution overall for the entire dataset.
Decision boundaries are rectilinear,
and this can limit the expressiveness
of the resulting model, which means it may not be able
to solve complicated classification problems
that require more complex decisions,
or decision boundaries, to be formed.
In summary, the decision tree classifier
uses a tree-like structure to specify a series of conditions
that are tested to determine the class label for a sample.
The decision tree is constructed by repeatedly
splitting the data, and partitioning the data
into successively more homogenous subsets.

The resulting tree can often be easy to interpret.
So now, let's look at our notebook
to do a decision tree classification
on our weather dataset in Python.

# Pluralsight - Linear Regession

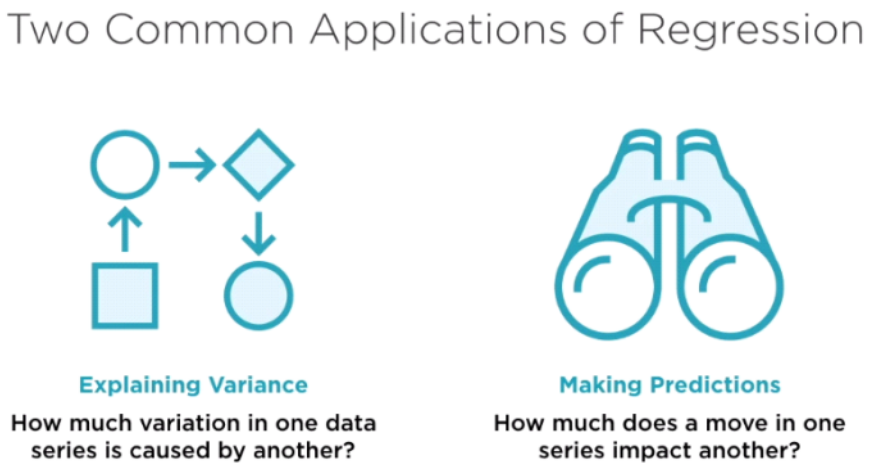
Monday, October 08, 2018 12:03 PM

# Modelling Relationships between variables

Monday, May 20, 2019 1:30 PM

**Modelling Relationships between variables using Regression.**

**Video 1**




- **Explaining Variance:**
  Quantifying the relationship between 2 variables.

- **Prediction:**
  Fit a Regression line for Y a given set of points.
  Predict a new set of values for Y.
  Can put a confidence interval for the forecast.

**Video 2: Rising Stock**

**Beta**
How much a particular stock moves if the market as a whole moves by 1%

**Alpha**
Rise in stock price that cannot be explained by a rising market.
Company has done something right.

These names also come from Regression
y = alpha + beta x

**Video 6: Probability distribution ad Bell curve**

# Simple regression

Monday, October 08, 2018 1:01 PM

**Video 1 - Setting up the regression problem.**

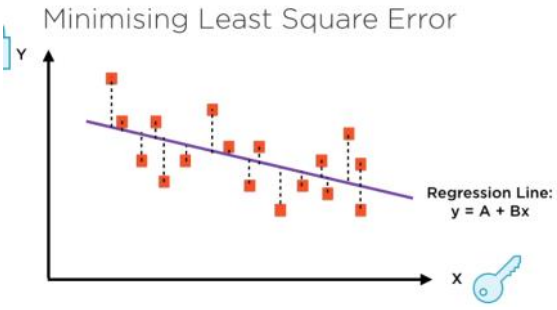
**Cause and Effect**
We should use regression only when we clearly know
whats the **Cause**
(Independent Variable x; also referred to as explanatory variable)
And
Whats the **Effect**
(Dependent variable y)

Dependent variable (y) i.e the effect --> on the Y axis
Independent variable (x) i.e. the cause --> on the X axis

Then find the best fit line:
y = a + bx

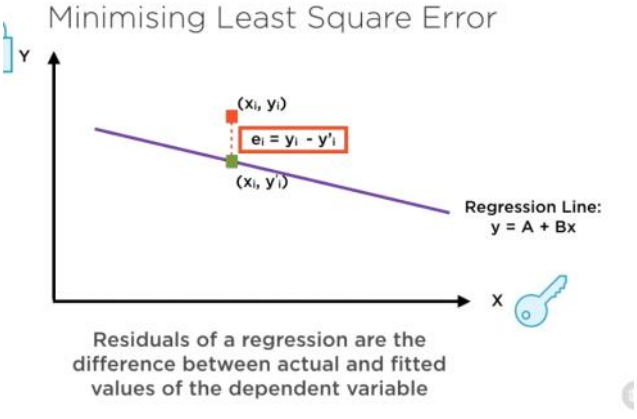
**Video 2 - Best Fit line**

Minimizing the Least square error




The greater the sum of lengths of these **dotted lines** the worse the fit of the regression line is.
Note that its sum of the squares of the dotted lines.

**Residuals**
These dotted lines are the residuals, they are critical in analyzing the regression problem.
The **residuals are the difference between the actual and fitted values of the dependent variable**.

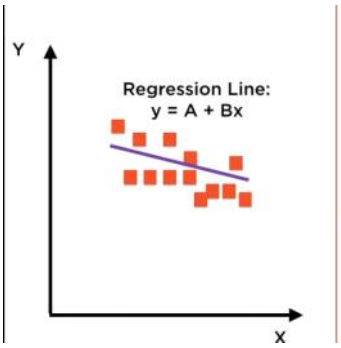



**The Best Fit Line is one that Minimizes the Sum of the Square of the Lengths of the Errors (the residuals)**

Finding this best fit line is the **objective of the regression problem.**
If the line was a perfect fit the Residuals (errors) would all be equal to 0.

**Have we chosen correctly by fitting a straight line**? This is examined by looking at properties of the residual.
If we find that the below conditions are not followed or do not hold true then we know that we have not set up the regression problem correctly.

***Important to remember this***




Ideally, residuals should
- have zero mean
- common variance
- be independent of each other
- be independent of x
- be normally distributed

**Note** - Often times in practical cases, some of these assumptions may not be satisfied.
Also, the method of least squares does not guarentee that Residuals are normally distributed.
These are idealised properties of residuals, but in reality some may not be completely satisfied everytime.

**Video 3** - Demo of setting up a simple **Regression in Excel simple example**
easy if want to show someone.

**Video 4** & **Video 5** - **Solving the regression problem**

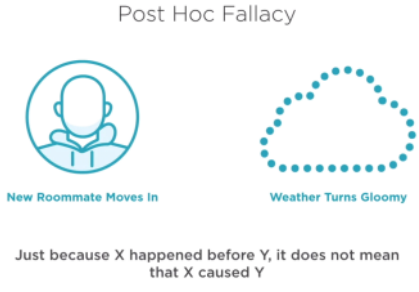
The **objective of the regression problem** is to find the **Best Fit Line**, One that minimizes the Least Sqaure Errors (or residuals).
Finding Best fit line means obtaining the **Regression Coefficients**

There are 3 different methods


We discussed so far is the Method of Least Squares.

**Limitation** of Regression is that it **cannot be used to Find a Cause and Effect relationship**. But if cause and effect is known then regression can be used.

**Video - Explaining Variance**
Quantifying relationships. How much variation in one data series is caused by variation in another data series.
- Post Hoc Fallacy
- Correlation Not Same as Causation




**NEVER EVER USE <u>REGRESSION</u> TO ESTABLISH A CAUSE AND EFFECT RELATIONSHIP**

Sometimes regression is just used when -
X happens before Y (Post Hoc Fallacy)
X and Y just happened together (Correlation is Causation Fallacy)
These are not good cases to use regression to explaing variance. Must be used only if X causes Y. So use regression only when we are sure that genuine causation actually exists.

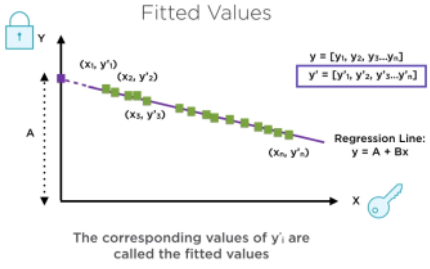
**<u>Video</u> - $R^2$**
*Getting to Rsqd.

Independent variable;
A **vector x** (i.e. a data series)
x = [x1, x2, x3….xn]

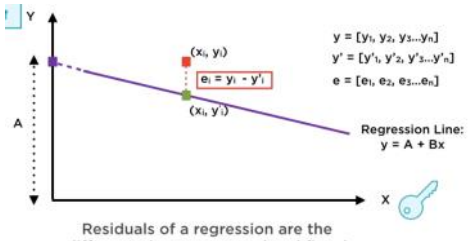
Dependent Variable
A vector y
y = [y1,y2,y3…yn]

Regression line
y=A + Bx
x and y is passed through a regression to get the '**best-fit**' regression line.

Fitted values
From the regression line we have the fitted values of y
A vector y`.
y` = [y`1,y`2,y`3 ….. y`n]



The corresponding values of $y'_i$ are called the fitted values



Residuals of a regression are the difference between actual and fitted values of the dependent variable

Residuals
Difference between the actual and the fitted values of the dependent variable.
so e1 = y1 - y`1
A vector e of all residuals is
e = y - y`

**Variance in y**
y = y` + e
variance(y) = variance (y`) + variance(e) + cov (y`,e; which always=0)
variance(y) = variance (y`) + variance(e)

Total Variance (TSS; var(y))
A measure of how volatile the dependent variable is.

Explained Variance (ESS; var(y))

Residual Variance (RSS)
Variance in the dependent variable that cannot be explained by the regression.

TSS = ESS + RSS

$R^2$ = TSS/ESS
% of variance explained by the regression.
**Higher the $R^2$ Better the regression**.
It says how much of the dependent variable can actually be measured by the independent variable.
Example stock has increase 10% this year and market has increased 8% in same period, we regress Returns of the stock on market returns and if the R2 is very high, we can say it's a high Beta stock, since its rise mostly explained by the rise in the market.

**<u>Video</u> - Prediction using Simple Regression**

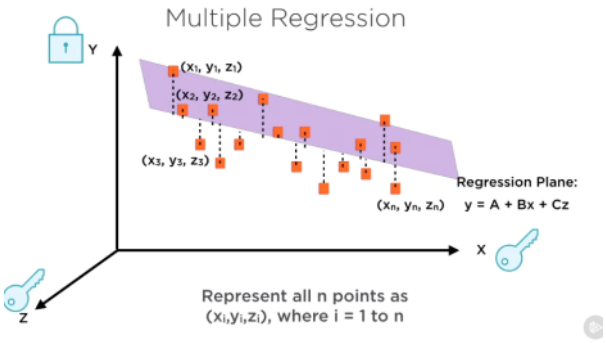
Along with prediction of values we can also have the 95% prediction interval.
This range has a minimum error when you forecast a value of Y at the average value of X.
The smaller the variance of the residual the tighter the prediction interval.

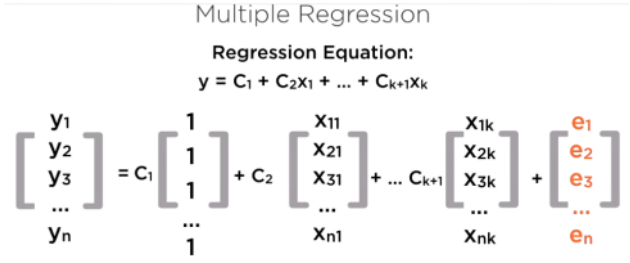
# Multiple Regression

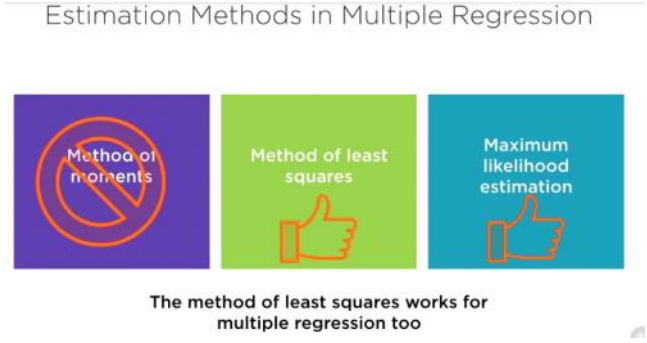
Monday, October 08, 2018 1:00 PM

Multiple Causes (Independent variables) Single Effect (Dependent Variable)

The Regression Problem -
To find the best fit plane.



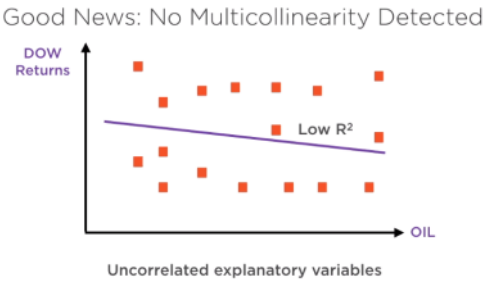



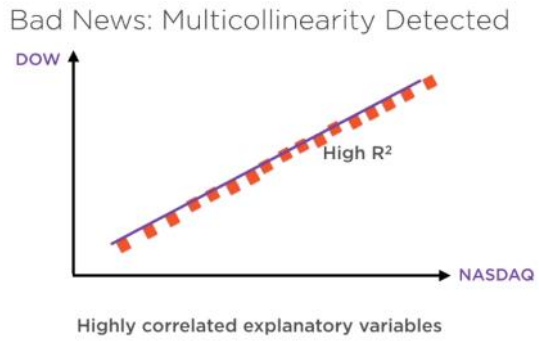




**Video 2 - Risks with Multiple Regression**

Most **Risks of Simple regression** exist here as well i.e.
No Cause - Effect {low Rsqd and can be found by looking at a scatter plot of the 2 variables}
Mis-specified Relationship {High Rsqd but residuals are not independent of each other} - Solution is transform to logs.
Incomplete Relationships. {Low Rsqd and residuals are not independent of each other} - Solution add more variables

But an additional risk here is **Multicolinearity**.
It's basically choosing multiple X variables that closely resemble each other. Meaning choosing 2 Independent variables that are highly correlated (i.e. if we regress one on the other we will get a very high $R^2$)

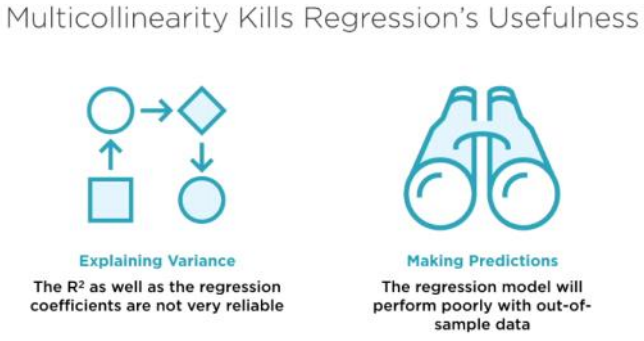



So,
Stock Prices = A + B(Dow) + C(Oil) +D(Nasdaq) **NO**
Stock Prices = A + B(Dow) + C(Oil) **FINE**

**<u>Video 3</u> - Why is Multi collinearity bad**

- Does not explain variance correctly between variables.
- Also leads to inaccurate predictions.
  It is a form of overfitting, the model works extremely well with the data that you have i.e. the sample data, but this model performs really poorly with Out of sample data.




**Solutions** -
**Common sense**:
Think deeply about each X variables.
Eliminate closely related ones.
Need to dig down and think about the underlying causes for each variable. {ex - For effects on a stock we need to consider GDP, oil prices, seasonality and when analyzing we will see that we only need to include 2 of the 4 maybe GDP and interest rates because changes in the others are just related to these 2}

It **can be very tempting to throw every variable into the regression**, because it may feel that every added variable is increasing the Rsqd, but that is not the right thing to do.

**<u>Video 4</u>: Multicollinearity more solutions**

**Factor analysis**
Finding underlying factors that drive the correlated variables.
Dimensionality Reduction technique to identify few underlying causes in data.

The output of the Factor analysis is a small number of factors that represent all of the underlying causes.

- PCA (Principal Component Analysis) is a way of accomplishing this for regression.


**Video 5: Benefits of Multiple Regression**

**Powerful:**

- Achieves the 2 applications of Regression: Explaining Variance and Making Predictions.
- In addition **MLR allows for controlled experimentation** i.e. it lets us determine how much variation is caused in the output when 1 of many predictor variables are changed, keeping all of the other variables constant.

**Versatile:**

- Can easily be applied to Non Linear relationships

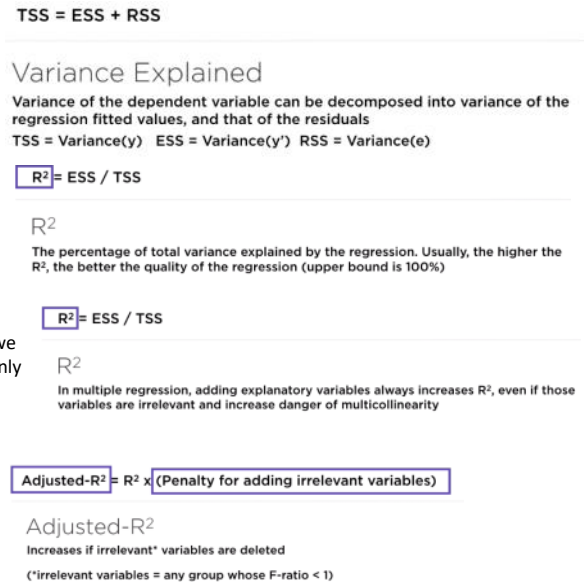
**Deep:**

- Crosses over into Machine learning

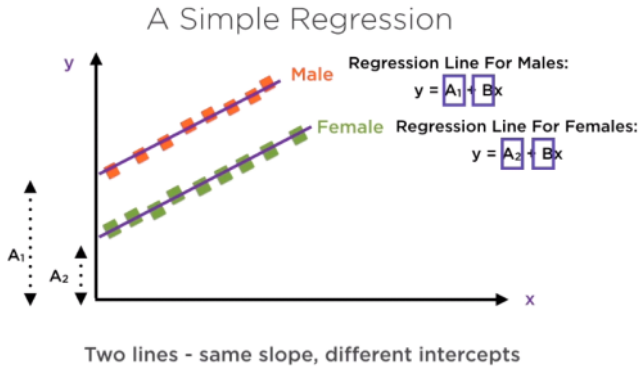

**Video 6: Interpreting Results of a MLR**

In case of a simple Regression -
$R^2$ (Measures quality of fit; the higher the better)
And Residuals (check for regression assumptions)
Combination of these 2 are enough to determine quality of the model.
{note - Standard errors are usually of little significance}

For MLR it is much more complicated:
**R2** (measures quality of fit, higher the better; but it is not a reliable statistic for MLR, because it only goes up as we add more and more variables and if we are including irrelevant variables / Multicollinear variables then we are only hurting the model) So cannot rely on R2 for MLR
**Adjusted R2** (Penalizes the model for irrelevant variables included) (***CAN RELY** on this)
**Residuals**
**F-Statistic**
**Std. Errors of Coefficients** (normally ignored for Simple Reg)




**Video 7: MLR that has Categorical variables**



Regression Line For Males:

$y = A_1 + Bx$

Regression Line For Females:

$y = A_2 + Bx$

**Combined Regression Line:**

$$y = A_1 + (A_2 - A_1)D + Bx$$

D = 0 for males

$$y = A_1 + (A_2 - A_1)D + Bx$$

$$= A_1 + Bx$$

Given data with k groups, set up k-1 dummy variables, else multicollinearity occurs

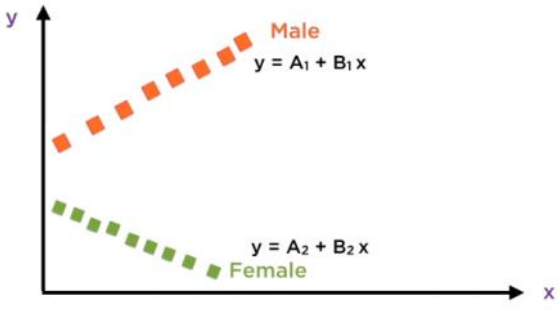
**Dummy Variables**

Binary - 0 or 1

**Categorical Variables**

Finite set of values - e.g. days of week, months of year...

To include non-binary categorical variables, simply add more dummies




Dummy variables can also be extended for use where groups have different slopes

Regression Line For Males:

$y = A_1 + B_1 x$

Regression Line For Females:

$y = A_2 + B_2 x$

**Combined Regression Line:**

$$y = A_1 + (A_2 - A_1)D_1 + B_1x + (B_2 - B_1)D_2$$

$D_1 = 0$ for males

$= 1$ for females

$D_2 = 0$ for males

$= x$ for females

# Implementing Regression

Sunday, June 9, 2019 1:59 PM

Simple and Multiple Regression in Python --- Pending
(16 min + 15 min)

# Andrew NG -ML

Wednesday, July 24, 2019 9:30 PM

***COURSE NOTES***

https://www.reddit.com/r/learnmachinelearning/comments/dvb2eo/official_cs229_lecture_notes_by_stanford/?utm_source=share&utm_medium=web2x

COURSE ASSIGNMENTS in PYTHON
https://www.reddit.com/r/learnmachinelearning/comments/do1zcv/andrew_ngs_ml_courser_assignments_in_python/?utm_source=share&utm_medium=web2x

# Classification - Logistic Regression

Wednesday, July 24, 2019 9:31 PM

## Notes in

Lecture6

COURSE NOTES - https://www.reddit.com/r/learnmachinelearning/comments/dvb2eo/official_cs229_lecture_notes_by_stanford/?utm_source=share&utm_medium=web2x

**An application** -
spam detection in email service providers can be identified as a classification problem. This is s binary classification since there are only 2 classes as spam and not spam. A classifier utilizes some training data to understand how given input variables relate to the class. In this case, known spam and non-spam emails have to be used as the training data. When the classifier is trained accurately, it can be used to detect an unknown email.

**Example**:

$$\theta = \begin{bmatrix} 5 \\ -1 \\ 0 \end{bmatrix}$$

$y = 1\, if\, 5 + (-1)x_1 + 0x_2 \geq 0$

$5 - x_1 \geq 0$

$-x_1 \geq -5$

$x_1 \leq 5$

In this case, our decision boundary is a straight vertical line placed on the graph where $x_1 = 5$, and everything to the left of that denotes y = 1, while everything to the right denotes y = 0.

Again, the input to the sigmoid function g(z) (e.g. $\theta^T X$) doesn't need to be linear, and could be a function that describes a circle (e.g. $z = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2^2$) or any shape to fit our data.

-- Imaging we have 100 dimensions, which is essentially 100 features.
Dimensionality reduction is reducing the number of features used.

# Intro to ML - Andres Muller

Saturday, August 31, 2019 10:45 AM

https://www.youtube.com/watch?v=Qd68h4UGlNY&list=PL_pVmAaAnxIQGzQS2oI3OWEPT-dpmwTfA

Book Videos???

# Supervised Learning

Tuesday, September 17, 2019 10:22 PM

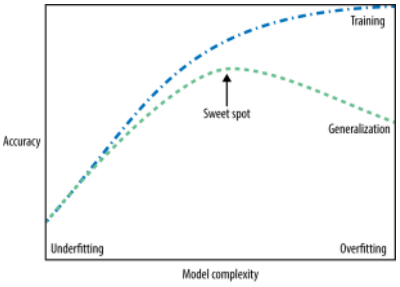
# Generalizn, Overfitting and Underfitting

Saturday, August 31, 2019 10:45 AM

Supervised learning is used whenever we want to predict a certain outcome from a given input, and we have examples of input/output pairs. We build a machine learning model from these input/output pairs, which comprise our training set. Our goal is to make accurate predictions for new, never-before-seen data.

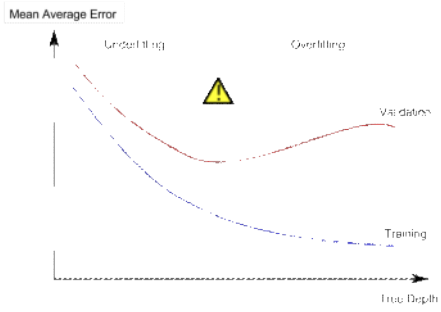
There are two major types of **supervised machine learning** problems, called **Classification** and **Regression**.
For regression tasks, the goal is to predict a continuous number.
A way to distinguish between classification and regression tasks is to ask whether there is some kind of continuity in the output. If there is continuity between possible outcomes, then the problem is a regression problem.

**Generalization, Overfitting, and Underfitting**




Supervised learning

Build a model using available data (Training the model)
|
Evaluate the model by making predictions on new data to check if accurate (Test the model)
If a model is able to make accurate predictions on unseen data, we say it is able to **Generalize** from the training set to the test set.
Intuitively we expect simple models to generalize better to new data. Therefore, **we always want to find the simplest model**.
Building a model that is too complex for the amount of information we have, is called overfitting.
**Overfitting** occurs when you fit a model too closely to the particularities of the training set and obtain a model that works well on the training set but is not able to generalize to new data (model too specific to whats seen in the training set)
But, if you (i.e. the model) are too generic, you might not be able to capture all the aspects of and variability in the data, and your model will do badly even on the training set. Choosing too simple a model is called **Underfitting**.
A larger dataset allows building more complex models without overfitting. However, simply duplicating the same data points or collecting very similar data will not help. But generally, more data points (more records) will yeild more variety.



# kNN

Tuesday, September 10, 2019 8:50 PM

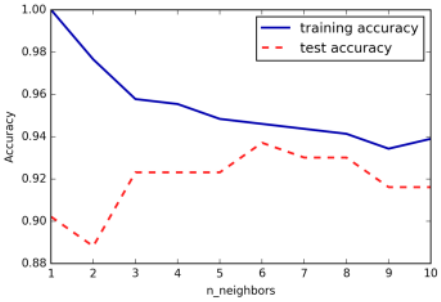
Instead of considering only the closest neighbor, we can also consider an arbitrary number, k, of neighbors.

When predicting the label for a data point considering it has more than one neighbor, we use *voting* to assign a label. Count how many neighbors belong to class 0 and how many neighbors belong to class 1. We then assign the class that is more frequent.
So the majority class among the k-nearest neighbors.

**Considering more and more neighbors leads to a smoother decision boundary**.
A smoother boundary corresponds to a simpler model, so using many neighbors corresponds to low model complexity.

(SEE ALONGSIDE)

**From the Wisconsin example**




Considering a **single nearest neighbor** leads to a model that's **too complex**, the prediction on the training set is perfect but test-set accuracy is low.

When **multiple neighbours** are considered, the **model becomes simpler**. The training accuracy drops but test accuracy may improve.
However when considering too many neighbours, the model is over generalized and performance may be quite worse.
The best performance is somewhere in the middle.

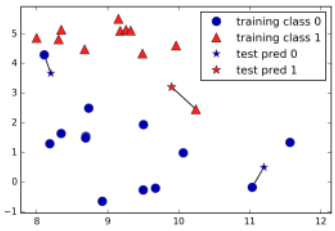
-
**When using the k-NN algorithm, it's important to preprocess your data**

-
**Note:**
This approach often does not perform well on datasets with many features (**hundreds or more**), and it does particularly badly with datasets where most features are 0 most of the time (so-called sparse datasets).

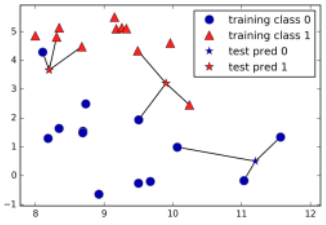
--> There's also a **regression variant of the k-nearest neighbors algorithm**.

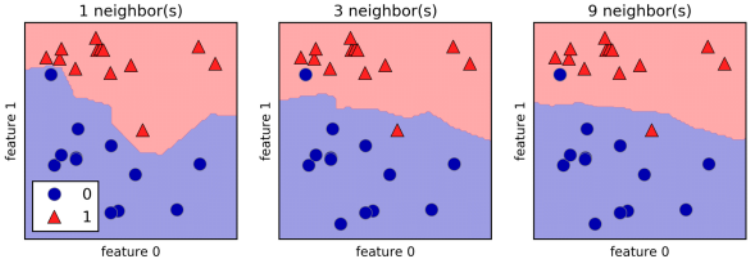
--> Lastly, there are two important parameters to the KNeighbors classifier:
the **number of neighbors** and how you measure **distance between data points**.
(Distance between datapoints to be reviewed beyond the book)

neighbours = 1




neighbours = 3
You can see that the prediction for the new data point at the top left is not the same as the prediction when we used only one neighbor.



# kNN_1

Sunday, September 1, 2019 10:41 AM

**How it works**

--> Build the model
Building this model only consists of fitting to the training set.

--> Make Predictions on test set
T**o make a prediction for a new data point**: the algorithm
- finds the point in the training set that is closest to the new point.
- assigns the label of this training point to the new data point.

--> Evaluate the model

The k in k-nearest neighbors signifies that instead of using only the closest neighbor to the new data point, we can consider any fixed number k of neighbors in the training (for example, the closest three or five neighbors to the new data point). Then, make a prediction using the majority class among these neighbors.

---

## Building the Model

- Import the Class
***KNeighborsClassifier*** *class* in the ***neighbors*** module.
- Instantiate the class into an ***Object*** (basically create an object of the class)
- Set ***number of neighbours***.
- Using the Object of knn class, build the model. Call the ***fit()*** method of the knn Object. ***Parameters*** of the fit method will be *Xtrain* and *ytrain.*

The **fit method returns the knn object itself (and modifies it in place)**, so we get a string representation of our classifier. this modified classifier **Object** is then used to make predictions.

## Predict and Evaluating the Model

(making predictions on the test set)
- We make predictions on the Test set and then compare the predicted values to actual values.
- Predict values of the test set using the ***predict()*** method.
- We can use the ***score*** method of knn object to compare predicted and actual values. It gives us an accuracy score out of 1. Parameters for score are X_test and y_test. (***Score*** is % predictions that are correct)

The model_score is basically = mean(pred==ytest)

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Based on the accuracy, you apply the model into production to new incoming data.

==> **NOTE**
'accuracy' is generally not the preferred performance measure for classifiers, especially when you are dealing with skewed datasets (i.e., when some classes are much more frequent than others).

So we use the ***confusion matrix***.

**#Getting the confusion matrix**
*from sklearn.metrics import confusion_matrix*
*confusion_matrix(y_train, y_train_pred)*

*o/p*
*array([[53272, 1307],*
*[ 1077, 4344]])*

***Each row in a confusion matrix represents an actual class, while each column represents a predicted class***. The first row of this matrix considers non-5 images (the *negative class*): 53,272 of them were correctly classified as non-5s (they are called *true negatives*), while the remaining 1,307 were wrongly classified as 5s (*false positives*). The second row considers the images of 5s (the *positive class*): 1,077 were wrongly classified as non-5s (*false negatives*), while the remaining 4,344 were correctly classified as 5s (*true positives*). A perfect classifier would have only true positives and true negatives, so its confusion matrix would have nonzero values only on its main diagonal (top left to bottom right)

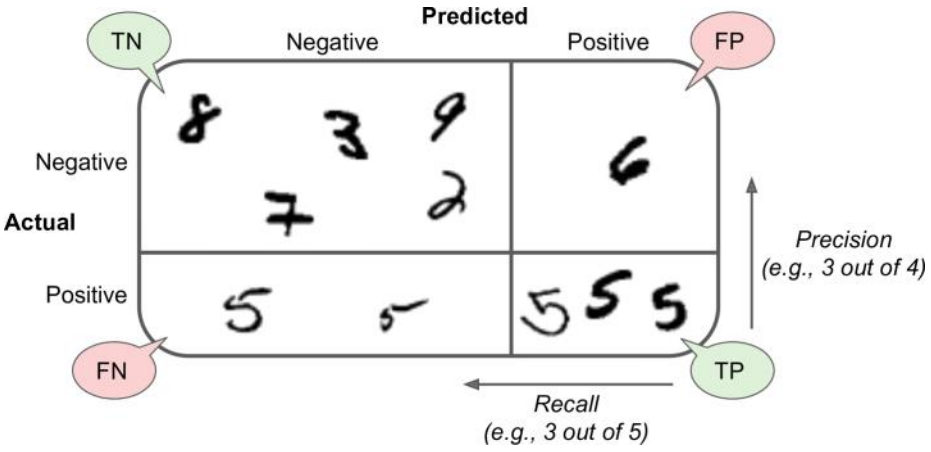
**Additional Metrics - PRECISION & RECALL**

***PRECISION***
*how precise, of all predicted as truE how many right*

$$\text{precision} = \frac{TP}{TP + FP}$$

***RECALL***
*Of all Trues how many could it find. Also called sensitivity or true positive rate (TPR)*

$$\text{recall} = \frac{TP}{TP + FN}$$




**Mnemonics**
**Precision <==> Positive (TP & FP)**
**reCALL <==> FONE (TP & FN)**

From Above:
Precision -- 75% (So of all the numbers it predicted to be 5, only 75% were right)

Recall -- 60%  (It detects only 60% of all the actual 5's)

DEPENDING ON THE SCENARIO YOU MAY WANT TO MAXIMIZE ONE OVER THE OTHER ==>  A TRADEOFF

**F1 SCORE**

Combination of Precision and Recall.
A simple way to compare two classifiers.
The F1 score is the harmonic mean of precision and recall .
Whereas the regular mean treats all values equally,
the harmonic mean gives much more weight to low values.

The **F1** score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0.
The relative contribution of precision and recall to the F1 score areequal. The formula for the F1 score is::

F1 = 2 * (precision * recall) / (precision + recall)

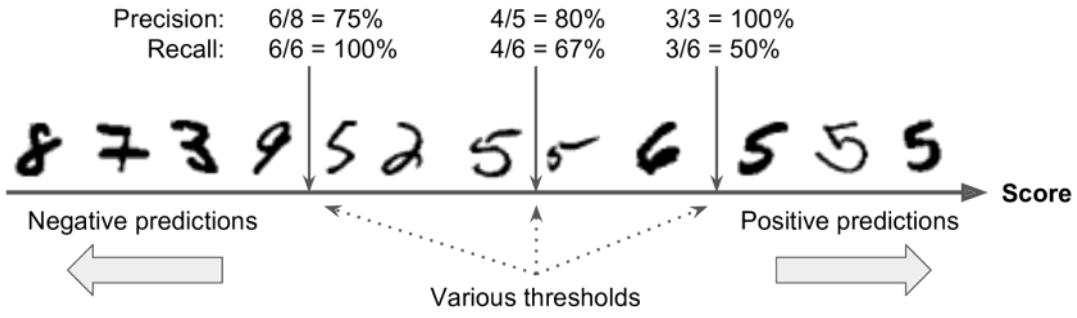
In the **multi-class and multi-label case**, this is the average of the F1 score of each class with weighting depending on the ``**average**`` parameter.
"**average**" parameter is required for multiclass/multilabel targets. If ``None``, the scores for each class are returned. Otherwise, this determines the type of averaging performed on the data.

The classifier will only get a high F1 score if ***both recall and precision are high*** --> BUT THIS IS ***NOT ALWAYS DESIRED***

For example, if you trained a classifier to detect videos that are safe for kids, you would probably prefer a classifier that rejects many good videos (low recall) but keeps only safe ones (high precision), rather than a classifier that has a much higher recall but lets a few really bad videos show up in your product (in such cases, you may even want to add a human pipeline to check the classifier's video selection). On the other hand, suppose you train a classifier to detect shoplifters on surveillance images: it is probably fine if your classifier has only 30% precision as long as it has 99% recall (sure, the security guards will get a few false alerts, but almost all shoplifters will get caught).
Unfortunately, you can't have it both ways: increasing precision reduces recall, and vice versa. This is called the ***precision/recall tradeoff***.

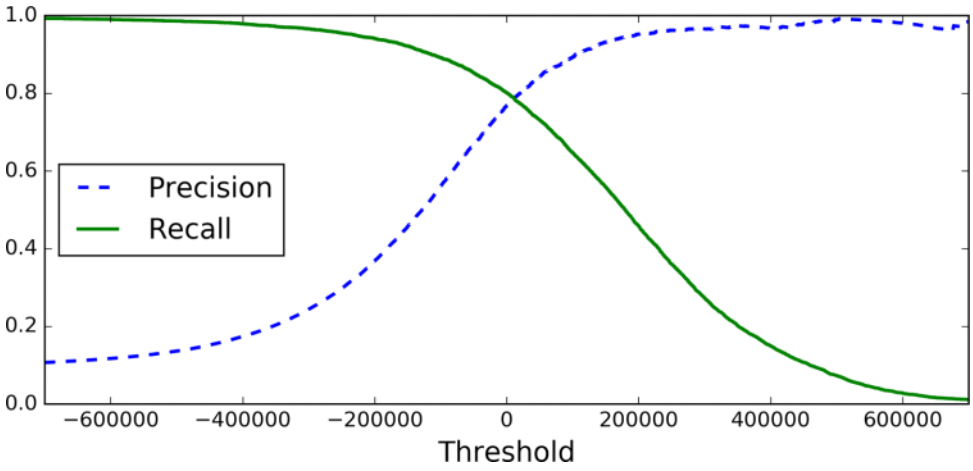
***Model DECISION THRESHOLD***

Scikit-Learn does not let you set the **threshold** directly, but it does give you access to the decision scores that it uses to make predictions. Instead of calling the classifier's predict() method, you can call its decision_function() method, which returns a score for each instance, and then make predictions based on those scores using any threshold you want.
As you change the threshold - Precision and Recall change (both opposite to the other)



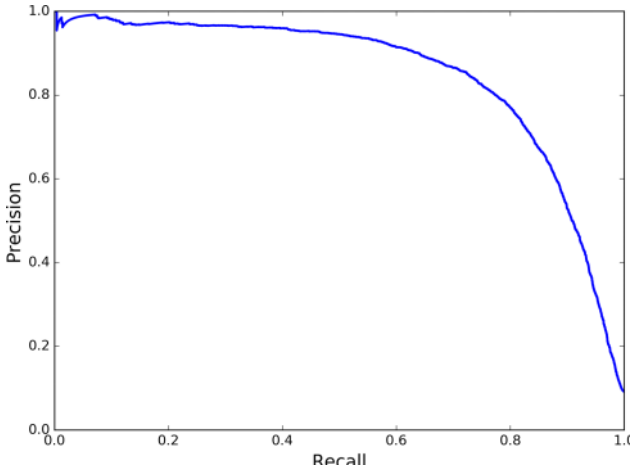

*As threshold increases --> more strict --> fewer wrong positives i.e False Positives --> Precision increases*

Can compute precision and recall for all possible thresholds using the ***precision_recall_curve()*** function.
Need to define a function that supplies - precision, recall and threshold for different values of threshold




**PRECISION RECALL CURVE**

Can also plot precision vs recall to see the point at which it falls sharply.



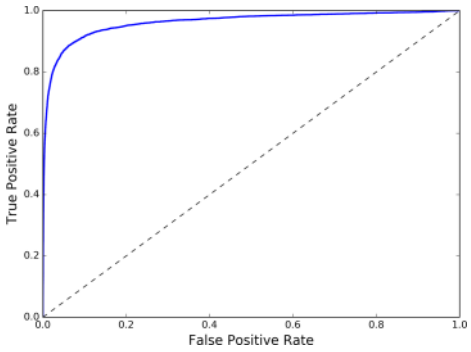

TIP

If someone says "let's reach 99% precision," you should ask, "at what recall?"

**<u>The ROC Curve</u>**

*Reciever Operating Curve*

- The ROC curve plots the **true positive rate** (**recall;sensitivity**) against the **false positive rate**.
- The FPR is the ratio of negative instances that are incorrectly classified as positive. It is equal to one minus the true negative rate, which is the ratio of negative instances that are correctly classified as negative. The TNR is also called specificity.
- ROC curve plots sensitivity (recall) versus 1 – specificity.




Once again there is a tradeoff: the higher the recall (TPR), the more false positives (FPR) the classifier produces. The dotted line represents the ROC curve of a purely random classifier; a good classifier stays as far away from that line as possible (**Good Classifier ==> toward the top-left corner**).

One way to compare classifiers is to measure the **area under the curve (AUC)**. A perfect classifier will have a ROC AUC equal to 1, whereas a purely random classifier will have a ROC AUC equal to 0.5.

***PR CURVE or ROC CURVE***

As a rule of thumb, you should prefer the PR curve whenever the positive class is rare or when you care more about the false positives than the false negatives, and the ROC curve otherwise.
For example, looking at the previous ROC curve (and the ROC AUC score), you may think that the classifier is really good. But this is mostly because there are few positives (5s) compared to the negatives (non-5s). In contrast, the PR curve makes it clear that the classifier has room for improvement (the curve could be closer to the top-right corner).

***Side Notes*** -
(add in from Jupyter notebook)

# KNN Code

Tuesday, November 19, 2019 10:11 PM

```
#kNN Model

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)

from sklearn.neighbors import KNeighborsClassifier
knn_obj = KNeighborsClassifier(n_neighbors=1)
knn_obj.fit(X_train, y_train)

y_pred = knn_obj.predict(X_test)        #Making Predictions


#Model Evaluation
knn_obj.score(X_test, y_test)  #How much of X test was predicted correctly
                               #Can run on Training as well to evaluate model


predictions_compare = pd.df({'Actual': y_test.flatten(), 'Predicted': y_pred.flatten()})

#Evaluating using Cross validation
from sklearn.model_selection import cross_val_score
cross_val_score(knn_obj, X_train, y_train, cv=3, scoring="accuracy")     #Check once for knn. Also check why its used


knn_obj.classes_              #check once for knn
#When a classifier is trained, it stores the list of target classes in its classes_ attribute.
```

***EXTENDED EVALUATION OF CLASSIFICASTION MODELS - CONFUSION MATRIX, PRECISION CURVE, ROC (AUC)***

***#1*** *- Getting the CONFUSION MATRIX*

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_train, y_train_pred)       #NOTE - Calculated by comparing  Actual and Predicted values of training

o/p
array([[53272,  1307],
       [ 1077,  4344]])
```

*Note - We know Each row in the confusion matrix represents an actual class. And each col in the matrix represents a predicted class. So for Multiclass Classif with say 10 classes, we will see a 10X10 Matrix.*

***#2*** *- Additional Evaluation Metrics - Precision and Recall*

```
from sklearn.metrics import precision_score, recall_score
precision_score(y_train, y_train_pred, average =None)
recall_score(y_train, y_train_pred, average = None)

from sklearn.metrics import f1_score
f1_score(y_train, y_train_pred)          # f1 Score- Again Comparing actual and predicted values of the model (train)
```

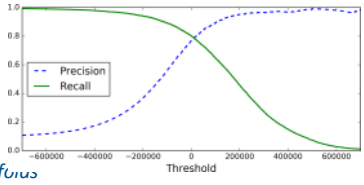
*In MC we will have Precision and Recall val for each class. And F1 Score for each class. OR we can have Average F1 Score depending on the "average" parameter*

*"average" parameter is required for multiclass/multilabel targets. If ``None``, the scores for each class are returned. Otherwise, this determines the type of averaging performed on the data.*

***#3*** *- Controlling the DECISION THRESHOLD of the Model*
*(sgd_clf is Stoch grad descent classif object as an example)*

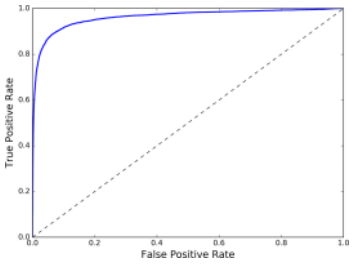
*Instead of* calling the classifier's *predict()* method, you *can directly call its* ***decision_function()*** *method*, which returns a score for each instance, and then make predictions based on those scores using any threshold you want. The confidence score for a sample is the ***signed distance of that sample to the hyperplane***. It's an array of shape (no of Samples X Features) i.e. a Sample-Class combination.

```
y_scores = sgd_clf.decision_function(X_train)        #Predicts confidence score.
threshold = 0
y_correct_predictions = (y_scores > threshold)        #will give predictions that were right based of existing threshold

threshold = 200000                          # Threshold is the baclance between Precision and Recall (the tradeoff)
y_correct_predictions = (y_scores > threshold)        #will give predictions that were right based off new threshold
```




```
#Precision Recall Curve  (using the decision_function to get scores for different folds)
from sklearn.model_selection import cross_val_predict
y_scores = cross_val_predict(sgd_clf, X_train, y_train, cv=3,
                             method="decision_function")          #returns an array of scores based on predictions made for diff folds

#Compute precision-recall pairs for different probability thresholds
#Note: this implementation is RESTRICTED TO BINARY CLASSIFICATION task.
```

```
from sklearn.metrics import precision_recall_curve
precisions, recalls, thresholds = precision_recall_curve(y_train, y_scores)

#Can now plot the precision recall as functions of the threshold value
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall")
    plt.xlabel("Threshold")
    plt.legend(loc="center left")
    plt.ylim([0, 1])

plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.show()
```




```
#ROC
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)

#Plot using Matplotlib
def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')

plot_roc_curve(fpr, tpr)
plt.show()


#AUC to compare classifiers
from sklearn.metrics import roc_auc_score
roc_auc_score(y_train, y_scores)
```

***MULTICLASS CLASSIFICATION***

- ERROR ANALYSIS in Model

```
plt.matshow(confusion_mx, cmap=plt.cm.gray)        #Visualization of the Confusion matrix
plt.show()

row_sums = confusion_mx.sum(axis=1, keepdims=True)
norm_conf_mx = confusion_mx / row_sums              #Normalizing each row entry (actuals for a class) to get error rates

#Now let's fill the diagonal with zeros to keep only the errors, and let's plot the result:
np.fill_diagonal(norm_conf_mx, 0)
plt.matshow(norm_conf_mx, cmap=plt.cm.gray)
plt.show()
#Can then clearly see errors made by the classifier; the misclassifications; can decide where classifier is to be improved.
--> Some Solution to errors
Gather more training data
Engineer a new feature
Pre Process the images (using Scikit img, Pillow or OvenCV)
```

| | | |
|---|---|---|
| | CLASSIFICATION MODEL EVALUATION | |
| | SGD | (MNIST Example) |
| (using Cross val) | SCORE | 0.87 |
| | CONFUSION MATRIX | |
| Always a balacnce between both these - THRESHOLD | PRECISION | |
| | RECALL (TPR;SENSITIEVITY) | |
| | F1 SCORE | |
| | Get thereshold value and then 2 plots: | |
| | 1 - PRECISION_RECALL CURVE (precision recall vs Threshold) | |
| | 2 - PRECISION vs RECALL Curve | |
| | ROC CURVE (TPR or recall/sensitivity VS FPR) | |
| | CONFUSION MATRIX - IMAGE MAP (Multiclass Classification) | |

## *AUC For MULTICLASS ???*

# Decision Trees

Sunday, September 22, 2019 12:30 AM

Decision Trees, a model that **learns a hierarchy of if/else questions**, **leading to a decision**. So, Learning a decision tree means learning the sequence of if/else questions that gets us to the true answer most quickly.
Each node in the tree either represents a question or a terminal node (also called a *leaf*) that contains the answer.
The top node, also called the *root*.
A leaf of the tree that contains data points that all share the same target value is called *pure*.
A prediction on a new data point is made by checking which region of the partition of the feature space the point lies in, and then predicting the majority target (or the single target in the case of pure leaves) in that region.

It is also possible to use trees for **regression tasks**, using exactly the same technique. To make a prediction, we traverse the tree based on the tests in each node and find the leaf the new data point falls into. The output for this data point is the mean target of the training points in this leaf.

**Complexity of Trees**
Building a tree until all leaves are pure leads to models that are very complex and highly overfit to the training data. The presence of pure leaves mean that a tree is 100% accurate on the training set; each data point in the training set is in a leaf that has the correct majority class.

2 common strategies to **prevent overfitting**:
- Stopping the creation of the tree early (also called ***pre-pruning***),
Possible criteria for pre-pruning include **limiting the maximum depth of the tree**, limiting the maximum number of leaves, or requiring a minimum number of points in a node to keep splitting it.

- building the tree but then removing or collapsing nodes that contain little information (also called post-pruning or just pruning).

Scikit-learn only implements pre-pruning, not post-pruning.

--> The tree that is visualized in the diagram is built based on the model that was trained using the training set (so the 426 samples)
And we can see in the tree -- The class of the node (M or B) and the the actual no of samples in each class.
The score accuracy should be 100% if no pruning done, check tree in that case.)

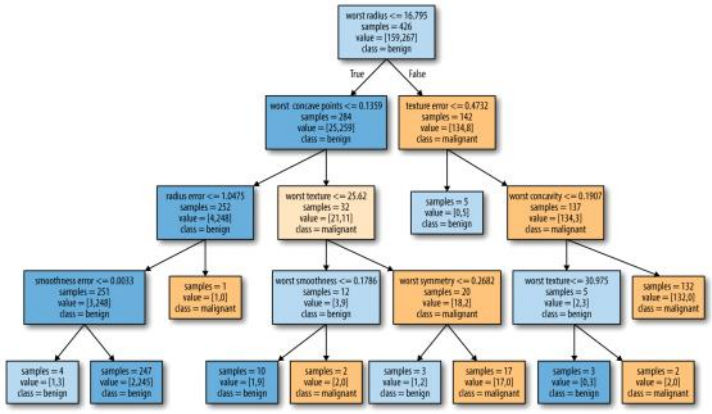
**Feature Importance**
Properties summarize the workings of the tree.
Feature Importance - rates how important each feature is for the decision a tree makes. It is a number between 0 and 1 for each feature, where 0 means “not used at all” and 1 means “perfectly predicts the target.”
Feature Importance always sums up to 1.

if a feature has a low value in feature_importance_, it doesn’t mean that this feature is uninformative. It only means that the feature was not picked by the tree, likely because another feature encodes the same information.

CONTINUE FROM DECISION TREE REGRESSOR

Also try changing max depth and no of leafs to see difference in score.

===> Gueron book has topic on 'Estimating Class Probabilities' in Decision Trees section.




*Code for Decision Trees*

```
from sklearn.tree import DecisionTreeClassifier

tree_obj = DecisionTreeClassifier(max_depth=4,random_state=0)
tree_obj.fit(X_train,y_train)

y_pred = tree_obj.predict(X_test)

#Evaluating the model
print('Accuracy of training set:',tree_obj.score(X_train,y_train))
print('Accuracy of test set:',tree_obj.score(X_test,y_test))

#Unpruned trees are prone to overfitting and not generalizing well to new data

#We apply pruning to the tree, which will stop developing the tree before we perfectly fit to the training data.
#Done using the max depth of the tree

#Can also improve accuracy by limiting Max no of Leaf Nodes
tree_obj1 = DecisionTreeClassifier(max_leaf_nodes=100, random_state=0)
tree_obj1.fit(X_train, y_train)
y_pred1 = tree_obj1.predict(X_test)
score_1 = tree_obj1.score(y_pred1, y_test)
```

```
#Can run model on X_test to see if overfitted or underfitted.
#Max leafs and Max Depths can be great options if overfitted.


#Code for Decision Tree Regressor exactly same

from sklearn.tree import DecisionTreeRegressor
tree_obj_reg = DecisionTreeRegressor(max_leaf_nodes=100, random_state=0)
tree_obj_reg.fit(X_train, y_train)
.....
......
...........
#Check model MAE,RMSE  (Regressor Only. Wont make sense for classification)
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error

mae_value = mean_absolute_error(test_y, predicted_y)

mse_value = mean_squared_error(y_test,y_pred)
rmse_value = np.sqrt(mse_value)      #Use this to get sense of error in predicted values


#Check model MAE,RMSE using K fold Cross validation
from sklearn.model_selection import cross_val_score
scores = cross_val_score(tree_reg_obj, X_train, y_train,
                         scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-scores)
#each time a different subset of data is taken to evaluate the rmse.
#increases our confidence in the model, if rmse is low on avg across all
```

# Naïve Bayes - NA

Saturday, September 21, 2019 1:51 PM

# Unsupervised Learning

Tuesday, September 17, 2019 10:24 PM

# Intro, Preprocessng, Scaling

Thursday, September 12, 2019 9:19 PM

We will look into two kinds of unsupervised learning in this chapter:
**transformations of the dataset** and **clustering**.

**Unsupervised transformations** of a dataset are algorithms that create a new representation of the data which might be easier for humans or other machine learning algorithms to understand compared to the original representation of the data.
A common application of unsupervised transformations is **dimensionality reduction**, which takes a high-dimensional representation of the data, consisting of many features, and finds a new way to represent this data that summarizes the essential characteristics with fewer features.
A common application for dimensionality reduction is reduction to two dimensions for visualization purposes.

Another application for unsupervised transformations is finding the parts or components that "make up" the data.

An example - topic extraction on collections of text documents.
Identifyinh what topics appear in each document. This can be useful for tracking the discussion of themes like elections, gun control, or pop stars on social media.

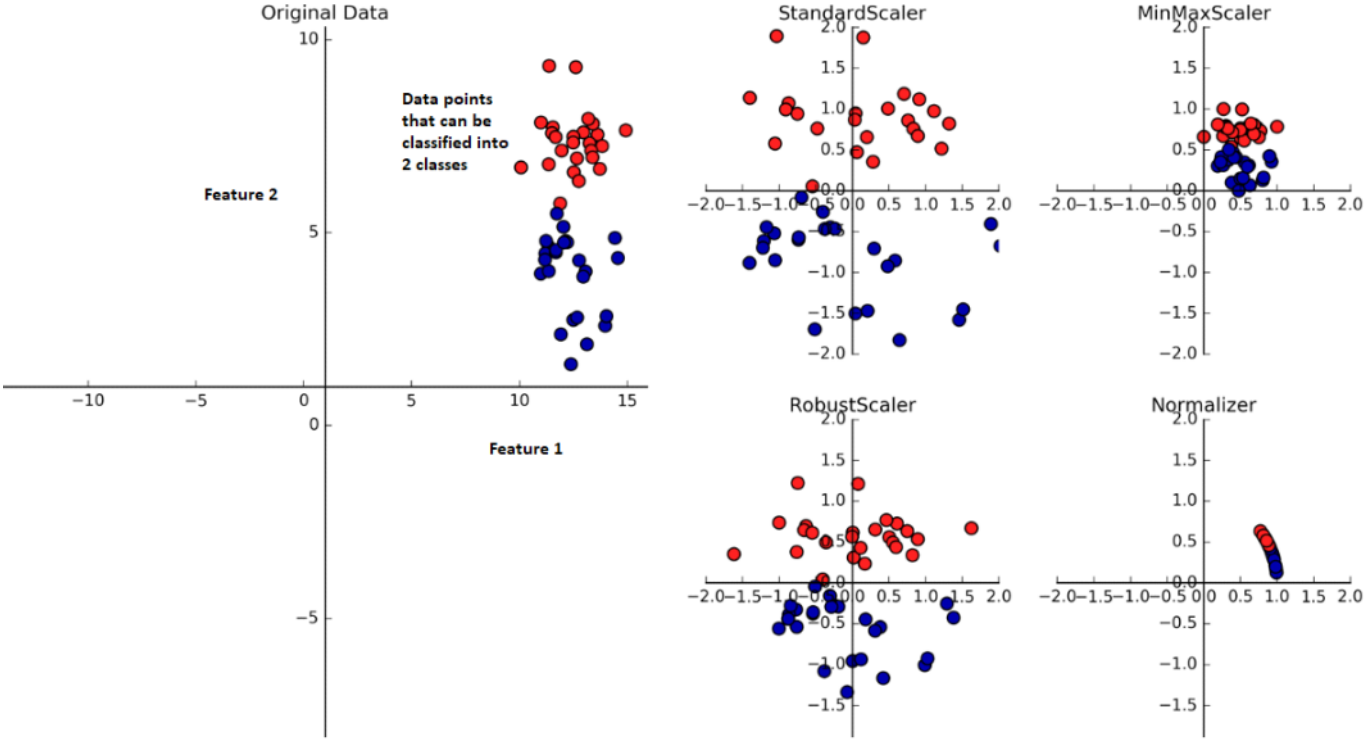
**Clustering** algorithms, on the other hand, partition data into distinct groups of similar items.

**Challenges with Unsupervised learning**

Unsupervised learning algorithms are usually applied to data that does not contain any label information, so we don't know what the right output should be. There is no way for us to "tell" the algorithm what we are looking for, and often the only way to evaluate the result of an unsupervised algorithm is to inspect it manually.

Unsupervised algorithms are used often in an exploratory setting, when a data scientist wants to understand the data better, rather than as part of a larger automatic system.

**Preprocessing and Scaling**



***To keep values of different columns comparable we scale the values of the features.***

The **StandardScaler** in scikit-learn ensures that for each feature the mean is 0 and the variance is 1, bringing all features to the same magnitude. However, this scaling does not ensure any particular minimum and maximum values for the features.

The **RobustScaler** works similarly to the StandardScaler in that it ensures statistical properties for each feature that guarantee that they are on the same scale. However, the RobustScaler uses the median and quartiles,1 instead of mean and variance. This makes the RobustScaler ignore outliers that can lead to trouble for other scaling techniques.

The **MinMaxScaler** shifts the data such that all features are exactly between 0 and 1.

The **Normalizer** does a very different kind of rescaling. It scales each data point such that the feature vector has a Euclidean length of 1. In other words, it projects a data point on the circle (or sphere, in the case of higher dimensions) with a radius of 1. This means every data point is scaled by a different number (by the inverse of its length). This normalization is often used when only the direction (or angle) of the data matters, not the length of the feature vector.

Remember -
We call FIT() on the Training set to determine min max range mean etc.
We call TRANSFORM() on the Training and Test set.

# preprocessing using 0-1 scaling
*scaler_obj = MinMaxScaler()*
*scaler_obj.fit(X_train)*
*X_train_scaled = scaler_obj.transform(X_train)*

The effect of scaling the data can be quite significant.

# Clustering

Tuesday, September 17, 2019 10:25 PM

# Kmeans

Tuesday, September 17, 2019 11:04 PM

-
Clustering is the task of partitioning the dataset into groups, called clusters.
The goal is to split up the data in such a way that points within a single cluster are very similar and points in different clusters are different.

-
K-means has 2 steps:
assigning each data point to the closest cluster center
and
Recalculating the mean and setting it as the cluster center

-
**the algorithm**
Algorithm initialized by declaring three data points randomly as cluster centers.
Then -
1 - each **data point is assigned to the cluster center** that is **closest to it**.
2 - Next, the cluster centers are **updated to be the mean** of the assigned points.
Now some data points will have new cluster centers close to it, so they will move to the different cluster, again mean changes (i.e. cluster center changes)
Above 2 steps are repeated.
Repeated until clusters remain unchanged. Then Algorithm Stops.

Given **new data points**, k-means will assign each to the closest cluster center.

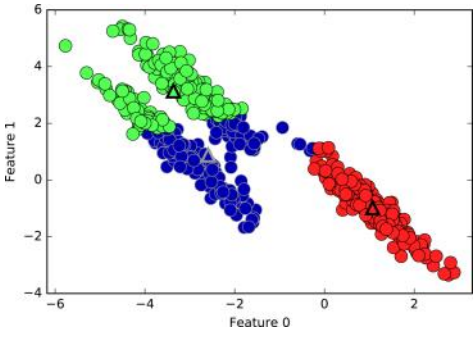
--
SAME STEPS as Other Algorithms (From Cluster Module, Import the Kmeans class, Instantiate the class with an object instance, apply (use fit method) the object to the dataset, use predict method for new data points)

*from sklearn.cluster import Kmeans*
*kmeans_obj = KMeans(n_clusters=3)*
*kmeans.fit(X)*

During the algorithm, each training data point in X is assigned a cluster label. You can find these labels in the attribute:
*kmeans_obj.labels_*
The cluster centers are stored in the attribute:
*cluster_centers_*

You can also assign cluster labels to new points, using the *predict* method.

-
**Failure cases of k-means**
- Even if you know the "right" number of clusters for a given dataset, k-means might not always be able to recover them.
- Each cluster is defined solely by its center,this means that k-means can only capture relatively simple shapes. k-means also assumes that all clusters have the same "diameter" in some sense.
- it always draws the boundary between clusters to be exactly in the middle between the cluster centers. Can sometimes lead to surprising results.
- k-means also assumes that all directions are equally important for each cluster.
See alongside, the 3 clusters should have been the data along the diagnols, but thats not the case.




- Additionally it relies on a **random initialization** (initial cluster centers) which means the outcome of the algorithm depends on a random seed. By default, scikit-learn runs the algorithm 10 times with 10 different random initializations, and returns the best result.
- Also the requirement to specify the number of clusters you are looking for (which might not be known in a real-world application).

**K-means as Decomposition**
k-means tries to represent each data point using a cluster center.
You can think of that as each datapoint being represented using only a single component (cluster center).
This view of k-means as a decomposition method, where each point is represented using a single component, is called **vector quantization**.

(DECOMPOSITION EXAMPLE PENDING - Do after PCA and NMF)

-
It is also possible to get an even more expressive representation of the data by using the distances to each of the cluster centers as features. This can be accomplished using the ***transform*** method of kmeans

-
Kmeans works well with large datasets but Scikit-learn even includes a more scalable variant in the **MiniBatchKMeans**

class, which can handle very large datasets.

-
NO EXAMPLE IN BOOK

- But did FIFA Example

# Gueron Book

Thursday, October 3, 2019 9:56 PM

# Ch1 - Intro Section

Thursday, October 3, 2019 9:56 PM

Applying ML techniques to dig into large amounts of data can help discover patterns that were not immediately apparent. This is called *data mining*.

**Supervised - Unsupervised - SemiSupervised - Reinforcement Learning**
A typical ***Supervised*** learning task is *classification* (ex - a spam filter), the target is called the *label*.
Another typical task is to *predict a target numeric value*, such as the price of a car, given a *set of features or predictors* (mileage, age, brand, etc.). This sort of task is called *regression*.

Example of an ***Unsupervised*** task is *anomaly detection*—for example, detecting unusual credit card transactions to prevent fraud, catching manufacturing defects, or automatically removing outliers from a dataset before feeding it to another learning algorithm. The system is trained with normal instances, and when it sees a new instance it can tell whether it looks like a normal one or whether it is likely an anomaly.

*Hierarchical clustering* algorithm may also subdivide each group into smaller groups. So you can target sub groups.

*Dimensionality Reduction* is an unspervised learning algorithm. in which the goal is to simplify the data without losing too much information. One way to do this is to merge several correlated features into one. Called *Feature Extraction*.

*Association rule learning* is also an unspervised task.

*anomaly detection,* ex- detecting unusual credit card transactions to prevent fraud, catching manufacturing defects, or automatically removing outliers from a dataset before feeding it to another learning algorithm. Also can be unsuprevised task.

It is often a good idea to try to reduce the dimension of your training data using a dimensionality reduction algorithm before you feed it to another Machine Learning algorithm (such as a supervised learning algorithm). It will run much faster, the data will take up less disk and memory space, and in some cases it may also perform better.

***Semi-Supervised*** Some photo-hosting services, such as Google Photos, are good examples of this. Once you upload all your family photos to the service, it automatically recognizes that the same person A shows up in photos 1, 5, and 11, while another person B shows up in photos 2, 5, and 7. This is the unsupervised part of the algorithm (clustering). Now all the system needs is for you to tell it who these people are. Just one label per person,4 and it is able to name everyone in every photo.

In ***Reinforcement Learning*** The learning system, called an agent can observe the environment, select and perform actions, get a reward or penalty based on the action, thereby learn whats the best strategy (called a Policy) to get the most reward over time.

**Batch (offline learning) and Online Learning**

*whether or not the system can learn incrementally from a stream of incoming data?*

In **batch learning**, the system is incapable of learning incrementally: it must be trained using all the available data. This will generally take a lot of time and computing resources, so it is typically done offline.
-the system is trained,
-then it is launched into production and runs without learning anymore; just applies what it has learned.
-If you want the system to learn about new data, you need to train a new version of the system from scratch on the full dataset (not just the new data, but also the old data), then stop the old system and replace it with the new one.
-Resource heavy.

In **online learning**, you train the system incrementally by feeding it data instances sequentially, either individually or by small groups called mini-batches.
Good option if you have limited computing resources: once an online learning system has learned about new data instances, it does not need them anymore, so you can discard them (unless you want to be able to roll back). This can save space.
Online learning algorithms can also be used to train systems on huge datasets that cannot fit in one machine's main memory (this is called ***out-of-core learning***). The algorithm loads part of the data, runs a training step on that data, and repeats the process until it has run on all of the data.**

Online Learning does not mean on the Live System. Would be offline learning but Incremental.
So learning in parts and then launch (see img).
One important parameter of online learning systems is how fast they should adapt to changing data: this is called the *learning rate*.
You would have this based on if system will rapidly adapt to new data and tend to quickly forget the old data.
A big challenge with online learning is that if bad data is fed to the system, the system's performance will gradually decline. To reduce this risk, you need to monitor your system closely and promptly switch learning off (and possibly revert to a previously working state) if you detect a drop in performance.

**Instance-Based Versus Model-Based Learning**

*Categorizing Machine Learning systems by how they generalize*.

Instance-based learning: the system learns the examples by heart, then generalizes to new cases using a similarity measure.
If you were to create a spam filter this way, it would just flag all emails that are identical to emails that have already been flagged by users (learning by heart)—not the worst solution, but certainly not the best.
Instead of just flagging emails that are identical to known spam emails, your spam filter could be programmed to also flag emails that are very similar to known spam emails.
This requires a measure of similarity between two emails. A (very basic) similarity measure between two emails could be to count the number of words they have in common.
The system would flag an email as spam if it has many words in common with a known spam email.

Model Based Learning - is the the usual Linear Regression based Model learning that we do.

CODE EXAMPLE HERE
Can Try.
Good Data set.

For example, suppose you want to know if money makes people happy, so you download the *Better Life Index* data from the OECD's website as well as stats about GDP per capita from the IMF's website. Then you join the tables and sort by GDP per capita. Table 1-1 shows an excerpt of what you get.

*Table 1-1. Does money make people happier?*

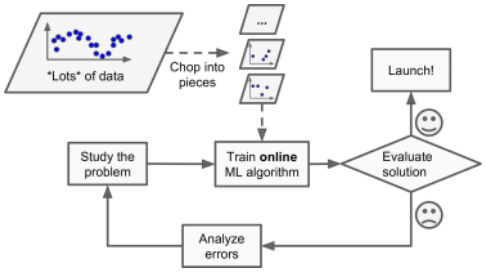
| Country | GDP per capita (USD) | Life satisfaction |
|---|---|---|
| Hungary | 12,240 | 4.9 |
| Korea | 27,195 | 5.8 |

**Supervised**

- k-Nearest Neighbors
- Linear Regression
- Logistic Regression
- Support Vector Machines (SVMs)
- Decision Trees and Random Forests
- Neural networks[2]

**UNSUPERVISED**

- Clustering
  - k-Means
  - Hierarchical Cluster Analysis (HCA)
  - Expectation Maximization
- Visualization and dimensionality reduction
  - Principal Component Analysis (PCA)
  - Kernel PCA
  - Locally-Linear Embedding (LLE)
  - t-distributed Stochastic Neighbor Embedding (t-SNE)
- Association rule learning
  - Apriori
  - Eclat


"Lots" of data
Chop into pieces
...
Launch!
Study the problem
Train online ML algorithm
Evaluate solution
Analyze errors

# Ch1a - Challenges of ML

Wednesday, October 23, 2019 9:30 PM

# Ch2 - ML Project

Thursday, October 3, 2019 9:59 PM

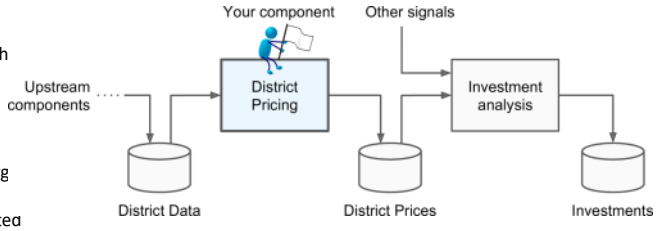
When starting off a ML project:

Frame the Problem or **the Question** or ***the Objective***
- What we hope to get out of it; as done for JF)
- the so what…? -- ***Possible Insight***
- ***Benefit or Application*** of doing this.

**Machine Learning Pipeline**
(alongside)

**Housing Dataset example**
- the median house value seem to be capped. This may be a serious problem since it is your target attribute (your labels). Your Machine Learning algorithms may learn that prices never go beyond that limit.

- You will often see people set the random seed to 42. This number has no special property, other th to be The Answer to the Ultimate Question of Life, the Universe, and Everything ????




==> **Test Set Generation**
We generally use random sampling methods to generate the Test set. This is fine if the dataset is larg but if not you run the risk of introducing sampling bias.
The US population is composed of 51.3% female and 48.7% male, so for a 1000 person well-conducted survey in the US one would try to maintain this ratio in the sample: 513 female and 487 male. This is called ***stratified sampling***. If they used purely random sampling, there would be about 12% chance of sampling a skewed test set with either less than 49% female or more than 54% female.

If in the housing sample you are told that the median income is a very important attribute, You may want to ensure that the test set is representative of the various categories of incomes in the whole dataset.
You would categorize the income by groups:

```
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])
```

Then do stratified sampling based on the income category:

```
from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

Now remove the 'income_cat' attribute so the data is back to its original state

```
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)
```

NOTE -- In Book Gueron seperated out Test set before seperating into Dep and Indep variables or doing any scaling etc.

## #Correlations in the data

```
corr_matrix = df.corr()
corr_matrix['house_value'].sort_values(ascending = False)     #checks correl of house_val with other var
```

--> Standard correlation coefficient (also called ***Pearson's r***) between every pair of attributes
Range of corr coeff is -1 to 1.
Closer to 1 is strong +ive correlation (as 1 attribute increases the other also increases).
Closer to -1 is negative correlation.
0 means no linear correlation.

The correlation coefficient only measures linear correlations ("if x goes up, then y generally goes up/down"). It may completely miss out on nonlinear relationships (e.g., "if x is close to zero then y generally goes up").

--> Another way using Pandas scatter matrix.

```
from pandas.plotting import scatter_matrix
attributes = ["median_house_value", "median_income", "total_rooms",
              "housing_median_age"]
scatter_matrix(df[attributes], figsize=(12, 8))
```

**Sparse Martix**

the output of OneHotEncoder is a SciPy sparse matrix, instead of a NumPy array.
A Sparse matrix is very useful when you have categorical attributes with thousands of categories. After one-hot encoding we get a matrix with thousands of columns, and the matrix is full of zeros except for a single 1 per row. Using up tons of memory mostly to store zeros would be very wasteful, so instead a sparse matrix only stores the location of the nonzero elements. You can use it mostly like a normal 2D array.
But if youwant to convert it to a (dense) NumPy array, just call the toarray() method.
Get_dummies also has option for Sparse.

**Custom Transformers**
Creating your own classes for data preperation, so with your own fit() and transform() methods.

**Linear Regression Model -** *Calif Housing Dataset*:
For the Calif Data set we see 67K error with training set itself, this is not high score.
An example of a model underfitting the training data. Basically could be that the features do not provide enough information to make good predictions.
One of the fixes to underfitting is:
- Feed the model with better features (more complex model, less generalized).
- Or could also Select a more powerful algorithm.

**Model Evaluation using Cross Validation**. **
Basically a *sklearn* feature called ***K-fold cross validation*** that randomly splits the training set into 10 distinct subsets called folds, then it trains and evaluates the Decision Tree model 10 times, picking a different fold for evaluation every time and training on the other 9 folds. The result is an array containing the 10 evaluation scores.

*from sklearn.model_selection import cross_val_score*
*scores = cross_val_score(tree_reg, housing_prepared, housing_labels,*
*scoring="neg_mean_squared_error", cv=10)*
*tree_rmse_scores = np.sqrt(-scores)*

Can then look at the different scores, mean and std. deviation.

BUT - For this you should have done all the Pre-processing including Scaling (remember normally Scaling after Splitting) on the entire Dataset of features.

Cross Validation seems powerful -
If we can randomly take any set of observations and make accurate predictions on them, with a low rmse on Avg. then it can go a long way in our confidence in the model.

TIP

You should save every model you experiment with, so you can come back easily to any model you want. Make sure you save both the hyperparameters and the trained parameters, as well as the cross-validation scores and perhaps the actual predictions as well. This will allow you to easily compare scores across model types, and compare the types of errors they make. You can easily save Scikit-Learn models by using Python's `pickle` module, or using `sklearn.externals.joblib`, which is more efficient at serializing large NumPy arrays:

```
from sklearn.externals import joblib

joblib.dump(my_model, "my_model.pkl")
# and later...
my_model_loaded = joblib.load("my_model.pkl")
```

**Fine Tuning the model:**
**Grid search -** Trying different combinations of Hyperparameters??, no of estimators??, maximum features?? etc.
Not practiced - to explore from diff source in more detail.

**Randomized search** -
RandomizedSearchCV

**Ensemble Methods** -
Another way to fine-tune your system is to try to combine the models that perform best. The **group (or "ensemble"**) will often perform better than the best individual model, especially if the individual models make very different types of errors.
Ex. - Random Forests work by training many Decision Trees on random subsets of the features, then averaging out their predictions. Building a model on top of many other models is called Ensemble Learning,.

**PreLaunch and Launch** **
**Pre Launch**: Highlighting what you have learned, what worked and what did not, what assumptions were made, and what your system's limitations are etc.

**Launch**: plugging the production input data sources into your system and writing tests.
- Write monitoring code to check your system's live performance at regular intervals and trigger alerts when it drops. Thus catch performance degradation which is quite common because models tend to "rot" as data evolves over time, unless the models are regularly trained on fresh data.
So, generally want to train your models on a regular basis using fresh data. You should automate this process as much as possible.
- Evaluating your system's performance will require sampling the system's predictions and evaluating them.
- also make sure you evaluate the system's input data quality. (e.g., a malfunctioning sensor sending random values, or another team's output becoming stale)

# Linear Reg Notes

Sunday, October 20, 2019 4:24 PM

A linear regression model in two dimensions is a straight line; in three dimensions it is a plane, and in more than three dimensions, a hyperplane.

--> REGRESSION can be described as a technique for studying the relationship between two or more variables so as to DESCRIBE, PREDICT, and CONTROL the variable of interest.

**Packages** for Linear Regression:
SciKit Learn
or
**STATSMODELS** another package for liner Regression.
Seems easy and comfortable to interpret outputs.

*MAIN COMPONENTS TO REGRESSION:*

Check relation of LABEL vs INDIV FEATURES - Does is seem to be LINEAR? Which ones? Which variables to include?
Checking CORRELATION between the variables
MODEL
COEFFICIENT of each variable/feature & INTERCEPT - The regression equation.
CONFIDENCE INTERVAL
P VALUES (is there a relationship)
Rsqd & Adj Rsqd
MAE MSE RMSE - Evaluating the Model

**LINEAR REGRESSION BASIC THEORY**
**Notebook -** Pluralsight - Linear Regession

**Deciding on the Model would be a combination of**:
Variables chosen - based on P values; Correlation between the variables - don't want substitutes.
Rsqd / Adj Rsqd (remember this is rsqd of the model)
Prediction on the test set - so RMSE of the test set * (biggest determinant)
Business knowledge

We May say P Values / Correlation / Rsqd etc. would also be more about understanding/describing the dependent variable.
RMSE more so when making predictions.

How can you know which values of coefficient and intercept will make your model perform best? To do this, you need to specify a performance measure. You can either define a utility function (or ***fitness function***) that measures how good your model is, or you can define a ***cost function*** that measures how bad it is.
For linear regression problems, people typically use a cost function that measures the distance between the linear model's predictions and the training examples; the objective is to minimize this distance.

=====

**Correlations in the data**

```
corr_matrix = df.corr()
corr_matrix['house_value'].sort_values(ascending = False)    #checks correl of house_val with other var

# visualize the relationship between the features and the response using scatterplots (seasborn)
sns.pairplot(data, x_vars=['TV','Radio','Newspaper'], y_vars='Sales', size=7, aspect=0.7)
sns.pairplot(data, x_vars=['TV','Radio','Newspaper'], y_vars='Sales', size=7, aspect=0.7, kind='reg')
```

**Coefficients**
Basically the regression equation. Obtained from the model, basically after the model is fit to the data we can see what exactly are the coefficients of each feature.

Interpreting Coefficients -
Holding all other variables constant, a 1 unit change in x1 will change y by _____ .

```
#Coefficients of each feature/variable
print(lmodelobj2.intercept_)
print(lmodelobj2.coef_)
# OR print the coefficients
list(zip(feature_colNames, model_obj.coef_))
```

**Confidence Interval**
A 95% confidence intervals for our model coefficients, which are interpreted as follows:
If the population from which this sample was drawn was sampled 100 times approximately 95 of those confidence intervals would contain the "true" coefficient. CODE?????????

**P Values**
*P is Low then Reject the null*
if we reject the null for any feature we can say, there is a relationship between the dependent and Independent variable

or basically…… babson class interpretation -- i.e. there is enough evidence to suggest that for a given change in x1…….

I think --- basically its running your t-test (that you know) for each of the independent variables with the dependent variable to check - ex. mean of sales (dependent variable) for 100 houses is Val P , no_of_indians (independent variable) changed by 10%, everything else held constant, new mean of sales for 100 houses is Val Q , did the value of houses increase?
So null and Alt hyp (there was an increase in value of houses). if you can reject the null, then can say there is a relationship between no_of_indians (indep. variable) and sales (dep. variable).

P Values in sklearn CODE????

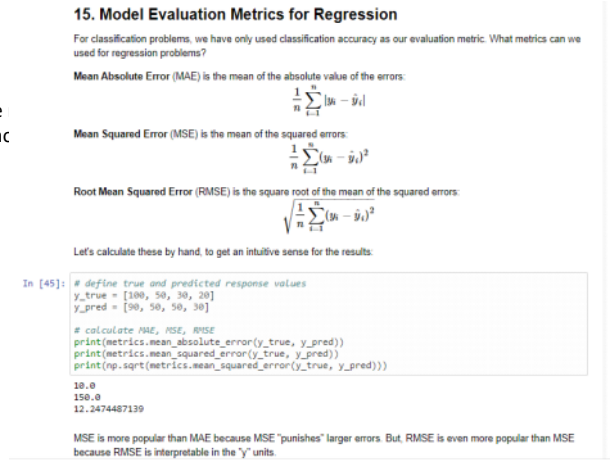
**Rsqd**
Rsqd is how much of the variance in the observed data can be explained by the model.
***lm2.score(X, y)***

Issues with Rsqd……
R-squared will always increase as you add more features to the model, even if they are unrelated to the
Selecting the model with the highest R-squared is not a reliable approach for choosing the best linear mc

Therefore Adj Rsqd.

---

**MAE / MSE / RMSE*****
MSE is more popular than MAE because MSE "punishes" larger errors. But, **RMSE is even more popular than MSE because RMSE is interpretable in the "y" units**.
ONLY FOR REGRESSION PROBLEMS.
WONT Make ssense for Classification Problems



### 15. Model Evaluation Metrics for Regression

For classification problems, we have only used classification accuracy as our evaluation metric. What metrics can we used for regression problems?

**Mean Absolute Error** (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

**Mean Squared Error** (MSE) is the mean of the squared errors:

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

**Root Mean Squared Error** (RMSE) is the square root of the mean of the squared errors:

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

Let's calculate these by hand, to get an intuitive sense for the results:

In [45]:
```
# define true and predicted response values
y_true = [100, 50, 30, 20]
y_pred = [90, 50, 50, 30]

# calculate MAE, MSE, RMSE
print(metrics.mean_absolute_error(y_true, y_pred))
print(metrics.mean_squared_error(y_true, y_pred))
print(np.sqrt(metrics.mean_squared_error(y_true, y_pred)))
```

```
10.0
150.0
12.2474487139
```

MSE is more popular than MAE because MSE "punishes" larger errors. But, RMSE is even more popular than MSE because RMSE is interpretable in the "y" units.

Why don't you consider **Gradient Boosting Decision Trees (GBDT) for Regression** which you will find many Python implementation for (XGboost, LightGBM and CatBoost).
The good things about GBDTs (more relevant to your problem) are:

- They have an intrinsic way to calculate feature importance (due to the way trees splits work .e.g Gini score and so on).
- They can deal with categorical variables that you have (sex, smoke, region)
- Also account for any possible correlations among your variables. Simple linear models fail to capture any correlations which could lead to overfitting.
- There are many ways to regularize GBDTs, which may come very handy!

With GBDTs you only have to be careful with continuous variable, in your case the bmi variable, not to be artificially overruling your trees (trees have hard time dealing very continuous data). You can easily overcome this challenge by rounding up/down or binning your continuous variable or other methods.

If you have strong reasons to stick to linear regressions, maybe you could use **LASSO which is a regularized linear regression that harshly penalizes (=0) the less important variables**. People actually use LASSO for feature selection as well.

# Lin Regression Code

Wednesday, October 23, 2019 4:08 PM

*LINEAR REGRESSION CODE*

#BUILDING THE MODEL

```
from sklearn.linear_model import LinearRegression
lin_reg_obj = LinearRegression()
lin_reg_obj.fit(X_train,y_train)
y_pred = lin_reg.predict(X_test)

from sklearn.metrics import mean_squared_error
mse_value = mean_squared_error(y_test,y_pred)
rmse_value = np.sqrt(mse_value)                    #This is main metric would use to evaluate.
                                                   #Note besides predicting on Test set we would also
                                predict for

                                                   training set and check how well the model was fit
                                                   #Was it overfit or underfit
```

# FINE TUNING THE MODEL

```
#Coefficients of each feature/variable
print(lmodelobj2.intercept_)
print(lmodelobj2.coef_)
# OR print the coefficients
list(zip(feature_colNames, model_obj.coef_))          #also have residuals_ to check residuals

 #Checking for Correlated attributes.
corr_matrix =  df.corr()          #Creates a correlation matrix of every feature against every other
attribute

# visualize the relationship between the features and the response using scatterplots (seasborn)
sns.pairplot(df, x_vars=['TV','Radio','Newspaper'], y_vars='Sales', size=7, aspect=0.7)
sns.pairplot(df, x_vars=['TV','Radio','Newspaper'], y_vars='Sales', size=7, aspect=0.7, kind='reg')

# Using Cross vlaidation to get range of rmse for diff combinations of Test-Control
#Would run agter all pre-processing is done
from sklearn.model_selection import cross_val_score
scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-scores)

-------------------------------------------------------------------------

#Linear Regression using SGD
from sklearn.linear_model import SGDRegressor
sgd_reg = SGDRegressor(max_iter=50, penalty=None, eta0=0.1)       # runs 50 epochs, starting with a
learning rate of 0.1
```

```
sgd_reg.fit(X, y.ravel())

#Solution
>>> sgd_reg.intercept_, sgd_reg.coef_
```

# Ch4 - Training Models

Wednesday, October 23, 2019 7:49 PM

**Linear Regression model**:
two very different ways to train it -
- Using a direct "closed-form" equation that directly computes the model parameters that best fit the model to the training set (i.e., the model parameters that minimize the cost function over the training set).
- Using an iterative optimization approach, called Gradient Descent (GD), that gradually tweaks the model parameters to minimize the cost function over the training set, eventually converging to the same set of parameters as the first method.

We will look at a few variants of Gradient Descent that we will use again and again when we study neural networks in Part II: Batch GD, Mini-batch GD, and Stochastic GD.

Polynomial Regression - a more complex model that can fit nonlinear datasets. Since this model has more parameters than Linear Regression, it is more prone to overfitting the training data,

go through the linear algebra and calculus introductory tutorials available as Jupyter notebooks in the online supplemental material.

Linear Regression Equation

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

Linear Regression model prediction (**vectorized form**)

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

- θ is the model's parameter vector, containing the **bias term** θ0 and the **feature weights** θ1 to θn.
- x is the instance's feature vector, containing x0 to xn, with x0 always equal to 1.
- θ · x is the dot product of the vectors θ and x, which is of course equal to .

$$\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

- hθ is the hypothesis function, using the model parameters θ.

NOTE -
In Machine Learning, vectors are often represented as *column vectors*, which are 2D arrays with a single column.
If θ and x are column vectors, then the prediction is:

$$\hat{y} = \boldsymbol{\theta}^{\mathbf{T}} \mathbf{x}$$

where is the transpose of θ (a row vector instead of a column vector) and it is the matrix multiplication of theeta transpose and x. It is of course the **same prediction**, except it is now represented as a single cell matrix (vector) rather than a scalar value.

**Cost function** for Lin Reg is RMSE (or MSE). So this is what we want to minimize.

$$\text{MSE}(\mathbf{X}, h_{\boldsymbol{\theta}}) = \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2$$

h(theeta) is just to indicate that it is parameterized by Theeta.

**The Normal Equation**
To find **the value of θ that minimizes the cost function**, there is a *closed-form solution*—in other words, a mathematical equation that gives the result directly. This is called the ***Normal Equation***.

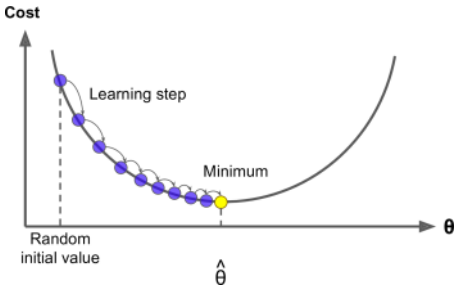
$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^T \mathbf{X})^{-1} \ \mathbf{X}^T \ \mathbf{y}$$

==========
==========

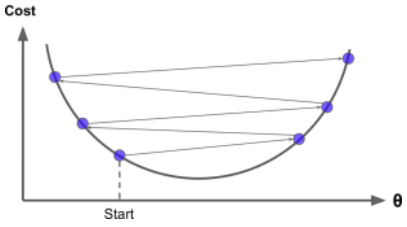
## GRADIENT DESCENT

- The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function.
- It measures the local gradient of the error function with regards to the parameter vector θ, and it goes in the direction of descending gradient. Once the gradient is zero, you have reached a minimum!
- You start by filling θ with random values (this is called random initialization), and then you improve it gradually, taking one baby step at a time, each step attempting to decrease the cost function (e.g., the MSE), until the algorithm converges to a minimum.

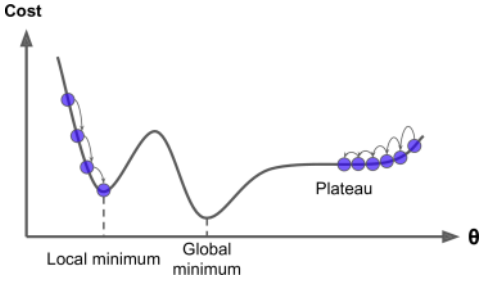
NOTE - GD is NOT Specific to Linear Regression




- An important parameter in Gradient Descent is the **size of the steps**, determined by the **learning rate** hyperparameter. If the learning rate is too small, then the algorithm will have to go through many iterations to converge, which will take a long time
- On the other hand, if the learning rate is too high, you might jump across the valley and end up on the other side, possibly even higher up than you were before. This might make the algorithm diverge, with larger and larger values, failing to find a good solution (see Figure below)




- Finally, not all cost functions look like nice regular bowls.
if the random initialization starts the algorithm on the left, then it will converge to a local minimum, which is not as good as the global minimum. If it starts on the right, then it will take a very long time to cross the plateau, and if you stop too early you will never reach the global minimum.




- The MSE cost function for a Linear Regression model happens to be a Convex function.
This implies that there are no local minima, just one global minimum. It is also a continuous function with a slope that never changes abruptly.4 These two facts have a great consequence: Gradient Descent is guaranteed to approach arbitrarily close the global minimum.

- However, it can be an elongated bowl if the features have very different scales.
Therefore when using Gradient Descent, you should ensure that all **features** have a similar scale (e.g., using Scikit-Learn's StandardScaler class), or else it will take much longer to converge. (so no scaling for y seems like.....)

**Training a model means** searching for a **combination of model parameters that minimizes a cost function** (over the training set). *It is a search* in the model's parameter space: the more parameters a model has, the more dimensions this space has, and the harder the search is.

**BATCH GRADIENT DESCENT**
To implement Gradient Descent, you need to compute the gradient of the cost function with regards to each model parameter θj. In other words, you need to calculate how much the cost function will change if you change θj just a little bit. This is called a partial derivative.

After Partial derivative of Theetha J:

$$\frac{\partial}{\partial \theta_j} \text{MSE}(\boldsymbol{\theta}) = \frac{2}{m} \sum_{i=1}^{m} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) \, x_j^{(i)}$$

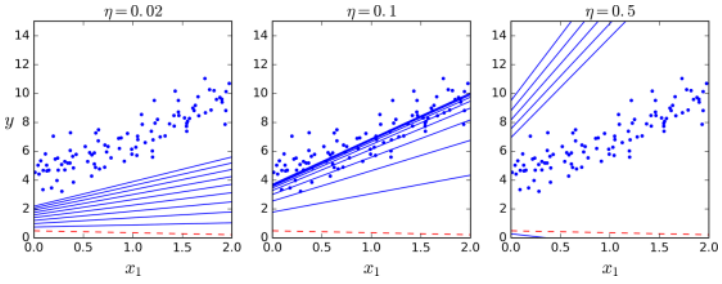
Instead of computing these partial derivatives individually, you can compute them all in one go. The **gradient vector**, noted ∇θMSE(θ), contains all the partial derivatives of the cost function (one for each model parameter)

$$\nabla_{\boldsymbol{\theta}} \operatorname{MSE}(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \operatorname{MSE}(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1} \operatorname{MSE}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \operatorname{MSE}(\boldsymbol{\theta}) \end{pmatrix} = \frac{2}{m} \mathbf{X}^T (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

This formula involves calculations over the full training set X, at each Gradient Descent step! This is why the algorithm is called Batch Gradient Descent: it uses the whole batch of training data at every step. As a result it is terribly slow on very large training sets. However, Gradient Descent scales well with the number of features.

Once you have the gradient vector, which points uphill, just go in the opposite direction to go downhill. This means subtracting ∇θMSE(θ) from θ. This is where the **learning rate η** comes into play: multiply the gradient vector by η to determine the size of the **downhill step**:

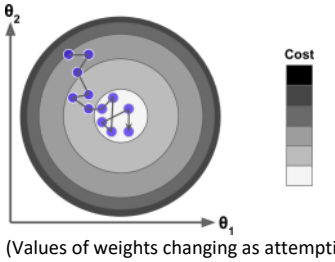
$$\boldsymbol{\theta}^{(\text{next step})} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \operatorname{MSE}(\boldsymbol{\theta})$$




On the left, the learning rate is too low: the algorithm will eventually reach the solution, but it will take a long time. In the middle, the learning rate looks pretty good: in just a few iterations, it has already converged to the solution. On the right, the learning rate is too high: the algorithm diverges, jumping all over the place and actually getting further and further away from the solution at every step.
To find a good learning rate, you can use grid search (see Chapter 2). However, you may want to limit the number of iterations so that grid search can eliminate models that take too long to converge.
To determine the no of iterations is to set a very large number of iterations but to interrupt the algorithm when the gradient vector becomes tiny—that is, when its norm becomes smaller than a tiny number ϵ (called the tolerance)—because this happens when Gradient Descent has (almost) reached the minimum.

**Convergence rate** - When the cost function is convex and its slope does not change abruptly (as is the case for the MSE cost function), Batch Gradient Descent with a fixed learning rate will eventually converge to the optimal solution, but you may have to wait a while.
it can take O(1/ϵ) iterations to reach the optimum within a range of ϵ depending on the shape of the cost function. If you divide the tolerance by 10 to have a more precise solution, then the algorithm may have to run about 10 times longer.
Something to keep in mind.

**STOCHASTIC GRADIENT DESCENT**
Batch Gradient descent is slow. Because it considers all available data at once.
Stochastic Gradient Descent considers only part of the data at a time. So It uses an instance of the entrie dataset to calculate the min cost. It just picks a random instance in the training set at every step and computes the gradients
SGD is Faster. Can work on huge training sets.
Because its working with instances of the whole data does not get stuck in a local minima. The model moves toward a total minimum cost for the dataset. But even after reaching a minimum the value tends to move around a bit.



(Values of weights changing as attempting towards lower cost)

Stochastic ((random)) GD is random in its nature. This randomness is good to escape from local optima, but bad because it means that the algorithm can never settle at the minimum. One solution to this dilemma is to gradually reduce the learning rate. The steps start out large (which helps make quick progress and escape local minima), then get smaller and smaller, allowing the algorithm to settle at the global minimum.

The function that determines the learning rate at each iteration is called the learning schedule. If the learning rate is reduced too quickly, you may get stuck in a local minimum, or even end up frozen halfway to the minimum. If the learning rate is reduced too slowly, you may jump around the minimum for a long time and end up with a suboptimal solution if you halt training too early.

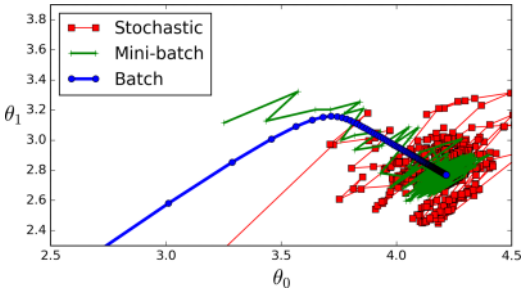
SGD with Skearn uses the SGDRegressor class, which defaults to optimizing the squared error cost function.
*from sklearn.linear_model import SGDRegressor*
*sgd_reg = SGDRegressor(max_iter=50, penalty=None, eta0=0.1)  # runs 50 epochs, starting with a learning rate of 0.1*
*sgd_reg.fit(X, y.ravel())*

*#Solution*
*>>> sgd_reg.intercept_, sgd_reg.coef_*

Each iteration is called a *epoch*.

**MINI-BATCH GRADIENT DESCENT**
- Similar to Batch but smaller batches + Multiple instances.

- The main advantage of Mini-batch GD over Stochastic GD is that you can get a performance boost from hardware optimization of matrix operations, especially when using GPUs.
  - Mini Batch is less erratic than Stochastic
    So reaches quite close to the minimum. But on the flip side there are chances of getting stuck in a local minima.




We see all 3 reach towards the global minimum. With a good learning rate SGD and Minibatch can perform as well as Batch.

**Summary For Gradient Descent Models -**
Normal eqn (Scikit learn n/a) -- Closed form
SVD (LinearRegressor() )
Batch Gradient descent ((SGDRegressor() - but Code for implementation??)
Stochastic Gradient Descent (SGDRegressor())
Mini Batch (SGDRegressor())

There is almost no difference after training: all these algorithms end up with very similar models and make predictions in exactly the same way.

*Table 4-1. Comparison of algorithms for Linear Regression*

| Algorithm | Large *m* | Out-of-core support | Large *n* | Hyperparams | Scaling required | Scikit-Learn |
|---|---|---|---|---|---|---|
| Normal Equation | Fast | No | Slow | 0 | No | n/a |
| SVD | Fast | No | Slow | 0 | No | LinearRegression |
| Batch GD | Slow | No | Fast | 2 | Yes | SGDRegressor |
| Stochastic GD | Fast | Yes | Fast | ≥2 | Yes | SGDRegressor |
| Mini-batch GD | Fast | Yes | Fast | ≥2 | Yes | SGDRegressor |

# Ch3 - Classification

Wednesday, October 23, 2019 7:30 PM

**CLASSIFICATION** ---

- **Refer Muller one note section for initial Classification notes.**
- **Continue Multiclass classification from here**

- **Training Models Chapter** --

Continue with Gradient Descent types - **DONE**
Poly Reg
Learning curves
Regularized Lin Models (Omit for now)
Log Regression

- Do Maths course and then continue with
  _____

- Followup Classification with DECISION TREES chapter.

# MULTICLASS CLASSIFICATION

Sunday, December 8, 2019 12:31 PM

**MULTICLASS CLASSIFICATION**:

Whereas binary classifiers distinguish between two classes, multiclass classifiers (also called multinomial classifiers) can distinguish between more than two classes.

Some algorithms (such as Random Forest classifiers or naive Bayes classifiers) are capable of handling multiple classes directly. Others (such as **Support Vector Machine** classifiers or **Linear classifiers**) are **strictly binary classifiers**. However, there are various strategies that you can use to perform multiclass classification using multiple binary classifiers.

For example, one way to create a system that can classify the digit images into 10 classes (from 0 to 9) is to train 10 binary classifiers, one for each digit (a 0-detector, a 1-detector, a 2-detector, and so on). Then when you want to classify an image, you get the decision score from each classifier for that image and you select the class whose classifier outputs the highest score. This is called the ***one-versus-all*** (*OvA*) strategy (also called one-versus-the-rest).

Another strategy is to train a binary classifier for every pair of digits: one to distinguish 0s and 1s, another to distinguish 0s and 2s, another for 1s and 2s, and so on. This is called the ***one-versus-one*** (*OvO*) strategy. If there are N classes, you need to train N × (N – 1) / 2 classifiers. For the MNIST problem, this means training 45 binary classifiers! When you want to classify an image, you have to run the image through all 45 classifiers and see which class wins the most duels. The main advantage of OvO is that each classifier only needs to be trained on the part of the training set for the two classes that it must distinguish.

Some algorithms (such as Support Vector Machine classifiers) scale poorly with the size of the training set, so for these algorithms OvO is preferred since it is faster to train many classifiers on small training sets than training few classifiers on large training sets. For most binary classification algorithms, however, OvA is preferred.

Scikit-Learn detects when you try to use a binary classification algorithm for a multiclass classification task, and it **automatically runs OvA** (except for SVM classifiers for which it uses OvO).

**MNIST Image example**:

*sgd_clf.fit(X_train, y_train)*
*sgd_clf.predict([some_digit])*
*array([5], dtype=uint8)*

Under the hood, Scikit-Learn actually trained 10 binary classifiers, 1 for each number i.e. a 0-detector, a 1- detector, 2-detector and so on. <br> it got their decision scores for the image, and selected the class with the highest score.

We can call the ***decision_function()*** method. Instead of returning just one score per instance, it now returns 10 scores, one per class.

*some_digit_scores = sgd_clf.decision_function([some_digit])*

*sgd_clf.classes_* #When a classifier is trained, it stores the list of target classes in its classes_ attribute.

If you want to force ScikitLearn to use one-versus-one or one-versus-all, you can use the **OneVsOneClassifier** or **OneVsRestClassifier *classes***.

**MULTICLASS - Using Random Forest Classifier**
For a Random Forest ClassifierSklearn does not have to run OvA or OvO because ***Random Forest is a Multiclass Classifier***.
*from sklearn.ensemble import RandomForestClassifier*

```
rf_clf = RandomForestClassifier(random_state=42)
rf_clf.fit(X_train,y_train)

rf_clf.predict([Fst_digit])     #Or can predict for full training set to check results.
rf_clf.predict_proba([Fst_digit])   #to get list of probabilities the classifier assigned to each instance for each class
                                    #Or can predict for full training set

rf_clf.score(X_train,y_train)     #To check score i.e. Evaluate the model.
```

**Further EVALUATION OF MODEL (MC CLASSIFICATION)**

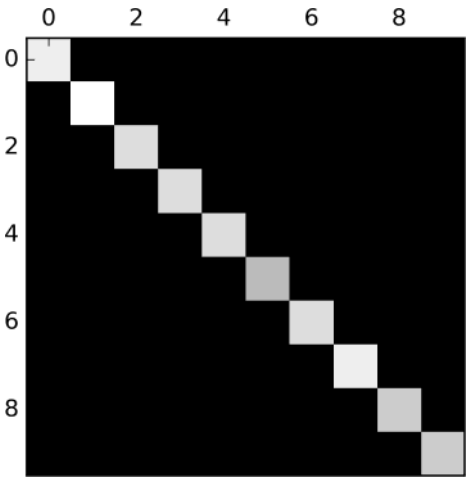
***CONFUSION MATRIX***

```
#Confusion Matrix is based on Actuals and Predicted values. So get predicted values:
y_train_predict = sgd_clf.predict(X_train)
confusion_matrix(y_train,y_train_predict)

array([[5709,   5,  91,  10,   9,   6,  29,   4,   7,  53],
       [   1, 6648,  41,   2,   2,  10,   0,   4,  19,  15],
       [  19, 143, 5536,  37,  31,   7,  25,  56,  57,  47],
       [  22, 114, 510, 4865,  14, 171,  24,  43,  59, 309],
       [   7,  92,  70,   1, 4543,   9,  44,   8,  28, 1040],
       [  89,  78, 220, 189,  62, 4030,  81,  10, 244, 418],
       [  51,  97, 203,  14,  17,  64, 5413,   6,  22,  31],
       [  24,  76, 108,   7,  47,   3,   3, 5254,   4, 739],
       [  41, 836, 518,  75,  29, 315,  20,  24, 2896, 1097],
       [  16,  52,  30,  31,  49,  10,   4,  46,   7, 5704]],
      dtype=int64)


#Each row in the confusion matrix represents an actual class
#And each col in the matrix represents a predicted class

Basically we have TP TN FP FNs for each class
  ^ So Class 0 will have some PRECISION and some RECALL <br>
   Simillarily for Class 1 will have some PRECISION and some RECALL and so on... <br>
  ^ F1 SCORE is HM of PRECISION and some RECALL.
   So we can have F1 SCORE of Each class
   OR
  ^ Using the "average" Parameter we can get the AVERAGE F1 SCORE
  ^ We can also have AVERAGE of PRECISION & RECALL for all the classes
```

**ERROR ANALYSIS in Model**

Confusion matrix here has a lot of numbers so we can look at it as images.

```
plt.matshow(confusion_mx, cmap=plt.cm.gray)
plt.show()
```



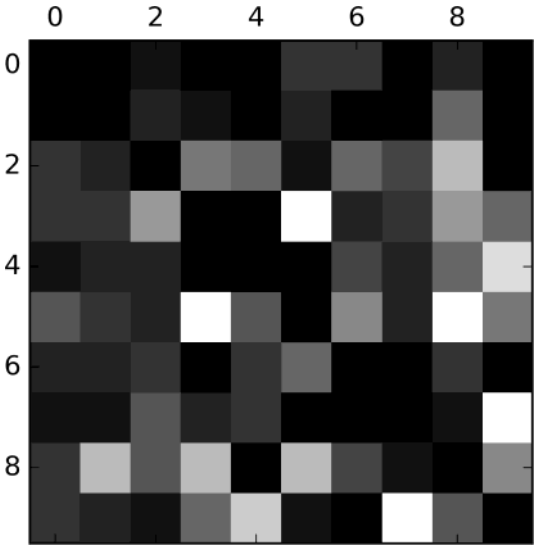

This confusion matrix looks fairly good, since most images are on the main diagonal, which means that they were classified correctly. The 5s look slightly darker than the other digits, which could mean that there are fewer images of 5s in the dataset or that the classifier does not perform as well on 5s as on other digits. In fact, you can verify that both are the case.

Let's focus the plot on the errors. First, you need to divide each value in the confusion matrix by the number of images in the corresponding class, so you can compare error rates instead of absolute number of errors (which would make abundant classes look unfairly bad):

```
row_sums = confusion_mx.sum(axis=1, keepdims=True)
norm_conf_mx = confusion_mx / row_sums
```

Now let's fill the diagonal with zeros to keep only the errors, and let's plot the result:

```
np.fill_diagonal(norm_conf_mx, 0)
plt.matshow(norm_conf_mx, cmap=plt.cm.gray)
plt.show()
```

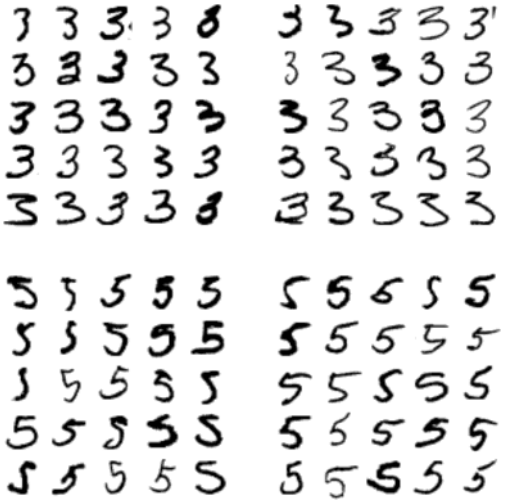



Now you can clearly see the kinds of errors the classifier makes. Remember that rows represent actual classes, while columns represent predicted classes. The columns for classes 8 and 9 are quite bright, which tells you that many images get misclassified as 8s or 9s. Similarly, the rows for classes 8 and 9 are also quite bright, telling you that 8s and 9s are often confused with other digits. Conversely, some rows are pretty dark, such as row 1: this means that most 1s are classified correctly (a few are confused with 8s, but that's about it). Notice that the errors are not perfectly symmetrical; for example, there are more 5s misclassified as 8s than the reverse.

Analyzing the confusion matrix can often give you insights on ways to improve your classifier. Looking at this plot, it seems that your efforts should be spent on improving classification of 8s and 9s, as well as fixing the specific 3/5 confusion. For example, you could try to gather more training data for these digits. Or you could engineer new features that would help the classifier—for example, writing an algorithm to count the number of closed loops (e.g., 8 has two, 6 has one, 5 has none). Or you could preprocess the images (e.g., using Scikit-Image, Pillow, or OpenCV) to make some patterns stand out more, such as closed loops.

Analyzing individual errors can also be a good way to gain insights on what your classifier is doing and why it is failing, but it is more difficult and time-consuming. For example, let's plot examples of 3s and 5s (the plot_digits() function just uses Matplotlib's imshow() function; see this chapter's Jupyter notebook for details):

```
cl_a, cl_b = 3, 5
X_aa = X_train[(y_train == cl_a) & (y_train_pred == cl_a)]
X_ab = X_train[(y_train == cl_a) & (y_train_pred == cl_b)]
X_ba = X_train[(y_train == cl_b) & (y_train_pred == cl_a)]
X_bb = X_train[(y_train == cl_b) & (y_train_pred == cl_b)]

plt.figure(figsize=(8,8))
plt.subplot(221); plot_digits(X_aa[:25], images_per_row=5)
plt.subplot(222); plot_digits(X_ab[:25], images_per_row=5)
plt.subplot(223); plot_digits(X_ba[:25], images_per_row=5)
plt.subplot(224); plot_digits(X_bb[:25], images_per_row=5)
plt.show()
```

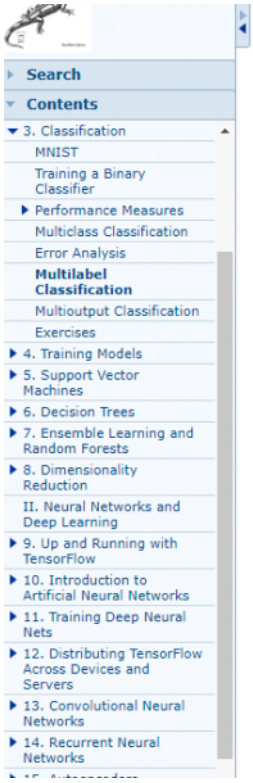


The two 5×5 blocks on the left show digits classified as 3s, and the two 5×5 blocks on the right show images classified as 5s. Some of the digits that the classifier gets wrong (i.e., in the bottom-left and top-right blocks) are so badly written that even a human would have trouble classifying them (e.g., the 5 on the 8th row and 1st column truly looks like a 3). However, most misclassified images seem like obvious errors to us, and it's hard to understand why the classifier made the mistakes it did.3 The reason is that we used a simple SGDClassifier, which is a linear model. All it does is assign a weight per class to each pixel, and when it sees a new image it just sums up the weighted pixel intensities to get a score for each class. So since 3s and 5s differ only by a few pixels, this model will easily confuse them.

The main difference between 3s and 5s is the position of the small line that joins the top line to the bottom arc. If you draw a 3 with the junction slightly shifted to the left, the classifier might classify it as a 5, and vice versa. In other words, this classifier is quite sensitive to image shifting and rotation. So one way to reduce the 3/5 confusion would be to preprocess the images to ensure that they are well centered and not too rotated. This will probably help reduce other errors as well.

# MULTILABEL CLASSIFICATION

Sunday, December 22, 2019 7:34 PM




## Multilabel Classification

Until now each instance has always been assigned to just one class. In some cases you may want your classifier to output multiple classes for each instance. For example, consider a face-recognition classifier: what should it do if it recognizes several people on the same picture? Of course it should attach one label per person it recognizes. Say the classifier has been trained to recognize three faces, Alice, Bob, and Charlie; then when it is shown a picture of Alice and Charlie, it should output [1, 0, 1] (meaning "Alice yes, Bob no, Charlie yes"). Such a classification system that outputs multiple binary labels is called a *multilabel classification* system.

We won't go into face recognition just yet, but let's look at a simpler example, just for illustration purposes:

```
from sklearn.neighbors import KNeighborsClassifier

y_train_large = (y_train >= 7)
y_train_odd = (y_train % 2 == 1)
y_multilabel = np.c_[y_train_large, y_train_odd]

knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_multilabel)
```

This code creates a `y_multilabel` array containing two target labels for each digit image: the first indicates whether or not the digit is large (7, 8, or 9) and the second indicates whether or not it is odd. The next lines create a `KNeighborsClassifier` instance (which supports multilabel classification, but not all classifiers do) and we train it using the multiple targets array. Now you can make a prediction, and notice that it outputs two labels:

```
>>> knn_clf.predict([some_digit])
array([[False,  True]])
```

And it gets it right! The digit 5 is indeed not large (`False`) and odd (`True`).

There are many ways to evaluate a multilabel classifier, and selecting the right metric really depends on your project. For example, one approach is to measure the $F_1$ score for each individual label (or any other binary classifier metric discussed earlier), then simply compute the average score. This code computes the average $F_1$ score across all labels:

```
>>> y_train_knn_pred = cross_val_predict(knn_clf, X_train, y_multilabel, cv=3)
>>> f1_score(y_multilabel, y_train_knn_pred, average="macro")
0.9770907847752502
```

This assumes that all labels are equally important, which may not be the case. In particular, if you have many more pictures of Alice than of Bob or Charlie, you may want to give more weight to the classifier's score on pictures of Alice. One simple option is to give each label a weight equal to its *support* (i.e., the number of instances with that target label). To do this, simply set `average="weighted"` in the preceding code.[4]

# MULTIOUTPUT CLASSIFICATION?

Sunday, December 22, 2019 7:40 PM

## Multioutput Classification

The last type of classification task we are going to discuss here is called *multioutput-multiclass classification* (or simply *multioutput classification*). It is simply a generalization of multilabel classification where each label can be multiclass (i.e., it can have more than two possible values).

To illustrate this, let's build a system that removes noise from images. It will take as input a noisy digit image, and it will (hopefully) output a clean digit image, represented as an array of pixel intensities, just like the MNIST images. Notice that the classifier's output is multilabel (one label per pixel) and each label can have multiple values (pixel intensity ranges from 0 to 255). It is thus an example of a multioutput classification system.

> **NOTE**
>
> The line between classification and regression is sometimes blurry, such as in this example. Arguably, predicting pixel intensity is more akin to regression than to classification. Moreover, multioutput systems are not limited to classification tasks; you could even have a system that outputs multiple labels per instance, including both class labels and value labels.

Let's start by creating the training and test sets by taking the MNIST images and adding noise to their pixel intensities using NumPy's `randint()` function. The target images will be the original images:

```
noise = np.random.randint(0, 100, (len(X_train), 784))
X_train_mod = X_train + noise
noise = np.random.randint(0, 100, (len(X_test), 784))
X_test_mod = X_test + noise
y_train_mod = X_train
y_test_mod = X_test
```

Let's take a peek at an image from the test set (yes, we're snooping on the test data, so you should be frowning right now):

On the left is the noisy input image, and on the right is the clean target image. Now let's train the classifier and make it clean this image:

```
knn_clf.fit(X_train_mod, y_train_mod)
clean_digit = knn_clf.predict([X_test_mod[some_index]])
plot_digit(clean_digit)
```

Looks close enough to the target! This concludes our tour of classification. Hopefully you should now know how to select good metrics for classification tasks, pick the

Looks close enough to the target! This concludes our tour of classification. Hopefully you should now know how to select good metrics for classification tasks, pick the appropriate precision/recall tradeoff, compare classifiers, and more generally build good classification systems for a variety of tasks.

# EXERCISES ????

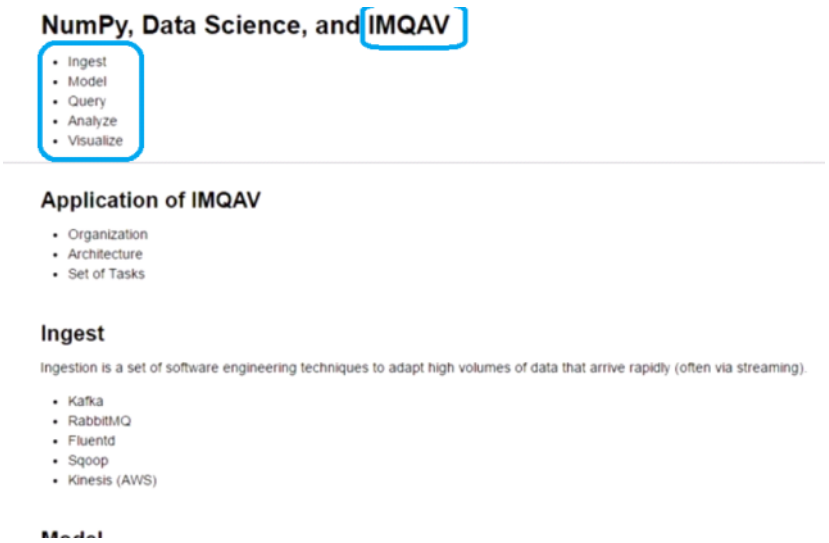
Sunday, December 22, 2019 8:02 PM

# NUMPY - LYNDA

Tuesday, December 31, 2019 5:28 PM

# NUMPY 1

Sunday, December 29, 2019 12:45 PM




- ***All elements of a numpy array must be of the same type***.
A note - An int cannot accomadate a complex no, so when creating an array from a list with int and complex numbers, the int is ***promoted*** to a complex number.

- The operator and its impact will vary, depending upon the type of data structure that you're using.
*my_list = [-17, 0, 4, 5, 9]*
*my_list * 4*
*Out: [-17, 0, 4, 5, 9, -17, 0, 4, 5, 9, -17, 0, 4, 5, 9, -17, 0, 4, 5, 9]* *# The list is just replicated 4 times.*

*array_name * 4* *#Will multiply each element by 4. remember elementwise multiplication*

-- The SIMPLEST way to **create a numpy ARRAY** is
create a SIMPLE LIST
use the ARRAY function.

***np.array(list_name)*** *#create a numpy array. how? using array function*

- Numpy also has INTRINSIC Functions that can be used to create numpy arrays.

***np.arange(4,13)*** *#Creates an array with a range of integers. 4 to 12. Can have a **step** parameter as well*
***np.arange(4,19).reshape (3,5)*** *# 2 dim array with 3 rows 5 cols*

*my_array= np.arange(35)*
*my_array.shape = (7,5)* *#7 rows 5 columns*

*np.random.randint(2,7,size=12)* *#returns 12 random integers between 2 incl and 7excl*
*np.random.random(size=6)* *# 6 random floating point numbers*
*np.random.random(size=6).reshape(2,3)* *# a (2,3) array of random floating point numbers*

```
In [5]: np.zeros((4,3))
Out[5]: array([[0., 0., 0.],
               [0., 0., 0.],
               [0., 0., 0.],
               [0., 0., 0.]])

In [4]: np.zeros((5,4,3))
Out[4]: array([[[0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.]],

               [[0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.]],

               [[0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
```

- **Numpy functions**

*linspace()* - It creates an array whose elements are evenly spaced over a specified interval.
*np.linspace(5, 15, num=9)* *#array from 5 to 15 with 9 elements. Step stize is autocalculated*

*linspace()* - It creates an array whose elements are evenly spaced over a specified interval.
*np.linspace(5, 15, num=9)* *#array from 5 to 15 with 9 elements. Step stize is autocalculated*

*np.zeros((5,4,3))* / *np.ones(4)* *#array of 0's or 1's. Can be 1 or n dimensional.*

--
*np.zeros((4,3))*
((4,3)) is a 2 dimensional array
4 indicates its 4 - 1d arrays
each 1d array has 3 elements each (so 4 rows 3 cols)
lll'ly
5,4,3 is a 3 dimensional array.
5 indicates its 5 - 2 dimensional arrays
4 incdicates 4 groups of arrays in each matrice
3 indicates no of elements in each array

```
[0., 0., 0.],
[0., 0., 0.]],

[[0., 0., 0.],
 [0., 0., 0.],
 [0., 0., 0.],
 [0., 0., 0.]],

[[0., 0., 0.],
 [0., 0., 0.],
 [0., 0., 0.],
 [0., 0., 0.]],

[[0., 0., 0.],
 [0., 0., 0.],
 [0., 0., 0.],
 [0., 0., 0.]]])
```

*Arr1 = np.array([List1,List2,List3])* *#Creating a 2D Numpy Array List1,2,3 each have 4 elements*
*Arr1 has 3 - 1d Arrays.*
*Each array has 4 elements*
*Therefore -- 2 Dimensional array; Shape 3,4*

*arr_t2 = np.array( [ [ l1,l2] , [l3,l4] ] )*
*A 3 Dimensional array.*
*has shape 2,2,4 (So 2 - (2x4) arrays)*
*each group has 2 - 1d arrays*
*each array has 4 elements.*

*#Can also have 4 dimensional array ==>*

```
]: arr_t4 = np.array([[[11,13],[11,12],[13,14]],[[11,12],[13,14],[12,14]] ])

]: arr_t4

]: array([[[[ 67,  78,  89,  98],
            [ 77,  88, 999, 945]],

           [[ 67,  78,  89,  98],
            [ 44,  33,  55,  65]],

           [[ 77,  88, 999, 945],
            [  2,   3,   4,   5]]],


          [[[ 67,  78,  89,  98],
            [ 44,  33,  55,  65]],

           [[ 77,  88, 999, 945],
            [  2,   3,   4,   5]],

           [[ 44,  33,  55,  65],
            [  2,   3,   4,   5]]]])

]: arr_t4.ndim

]: 4

]: arr_t4.shape

]: (2, 3, 2, 4)
```

**SLICING**

***array_name[row,col]*** *# note - indexing starting from 0*

*test_3D_array[1,3,2] =1111* *#To assign a new value to an entry in a 3 dim array*

**Boolean Masks**
Ex - Getting all entries that are divisible by 7
*my_vector = np.array([-17, -4, 0, 2, 21, 37, 105])*
*mask_arr = 0 == (my_vector%7)* *#gives a array mask_arr with boolean values*
*my_vector[mask_arr]* *#returns entries for True*

*my_vector[(my_vector%7)==0]* *#Can be done directly*
*an_array[(an_array>20)&(an_array<30)]* *#example*

*np.logical_and* *#numpy's logical AND function*

*array_name.shape*
*array_name.ndim* *#gives number of dimensions*
*array_name.size* *#no of elements or length*
*arr_name.dtype* *#datatype of elements in the array*
*type(arr_name)*

*np.unique(test_arr[:,1])* *#unique values in an Array*

*np.inner --*
*np.dot (mat1,mat2)* *#matrix multiplication function of numpy*

**Broadcasting** Broadcasting

*np.dot (mat1,mat2)  #matrix multiplication function of numpy*

**Broadcasting**

Broadcasting describes how numpy performs operations between arrays of different sizes.
The smaller array is broadcast across the larger array.

*arr_name.sum()  #Will give sum of all elements*
*arr_name.sum(axis =0)  #Sum calculated along axis 0. Broadcast along 0th axis*
*arr_name.sum(axis=1)  #bx along 1th axis*

Below is automatic Broadcasting across 2 arrays.

```
In [31]: my_vector2
Out[31]: array([14, 21, 28])

In [34]: my_vector1
Out[34]: array([[ 4,  6,  8],
                [10, 12, 14]])

In [35]: my_vector2*my_vector1
Out[35]: array([[ 56, 126, 224],
                [140, 252, 392]])
```

This will work for Multi Dim arrays as well.

```
Broadcasting
start array:
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]

add_rows array:
 [1 0 2]

result of start + add_rows:
 [[1. 0. 2.]
 [1. 0. 2.]
 [1. 0. 2.]
 [1. 0. 2.]]          #basically add_rows array is broadcasted
```

**STRUCTURED ARRAYS**

Basically Complex Data Structures that may contain numeric, 'string' elements, or even other arrays.

```
In [1]: person_data_def = [('name','S6'),('height','f8'),('weight','f8'), ('age', 'i8')]
        person_data_def
Out[1]: [('name', 'S6'), ('height', 'f8'), ('weight', 'f8'), ('age', 'i8')]
```

**A Structured array**

```
In [4]: people_array = np.zeros((4,), dtype=person_data_def)    # note the dtype here
        people_array
Out[4]: array([(b'', 0., 0., 0), (b'', 0., 0., 0), (b'', 0., 0., 0),
               (b'', 0., 0., 0)],
              dtype=[('name', 'S6'), ('height', '<f8'), ('weight', '<f8'), ('age', '<i8')])

In [5]: people_array[3] = ('Delta', 73, 205, 34)
        people_array[0] = ('Alpha', 65, 112, 23)
        people_array
Out[5]: array([(b'Alpha', 65., 112., 23), (b'',  0.,   0.,  0),
               (b'',  0.,   0.,  0), (b'Delta', 73., 205., 34)],
              dtype=[('name', 'S6'), ('height', '<f8'), ('weight', '<f8'), ('age', '<i8')])

In [6]: people_array[0:]
Out[6]: array([('Alpha', 65.0, 112.0, 23), ('', 0.0, 0.0, 0), ('', 0.0, 0.0, 0),
               ('Delta', 73.0, 205.0, 34)],
              dtype=[('name', 'S6'), ('height', '<f8'), ('weight', '<f8'), ('age', '<i8')])

In [7]: ages = people_array['age']
        ages
```

**Record Arrays**

Record arrays are similar to Structured arrays, they are structured arrays wrapped inside an ndarray subclass called numpy.record.array
This wrapping allows field access by attribute on an array object.
Record arrays are also using a special data type, a numpy.record, which allows field access by attribute on the individual elements with an array.
The documentation tells us that numpy.record is a thin wrapper around structured arrays. This means that most of the functionality

that is available within structured arrays is available without the wrapper. The benefits of the wrapper though accrue, while you are creating code using both Python and NumPy concurrently.
For now use Structured arrays and keep record arrays for a later time.

Structured and record arrays are designed for heterogeneous data, while maintaining NumPy's requirement that every element in an array use the same amount of memory space.

-- A Note:
# Using both integer indexing & slicing generates an array of lower rank
*row_rank1 = an_array[1, :]  # Rank 1 view*
*Output:*
*[21 22 23 24] (4,)*

# Slicing alone: generates an array of the same rank as the an_array
*row_rank2 = an_array[1:2, :]  # Rank 2 view. # notice the [[ ]]  in the output*
*Output:*
*[[21 22 23 24]] (1, 4)*

# NUMPY 2

Sunday, December 29, 2019 11:47 PM

**Copies and Views** -
A copy is a copy. So data is replicated at a different location in memory
Views provide 2 or more differently named references to the same location in memory.

*arr1 = np.array([1,2,45,668,89])*
*arr2 = arr1 #Just creates a different view*

*arr3 = np.**copy**(arr1) #Will create a deep copy of an array*

*# SAME OR DIFFERENT*
*arr1 is arr2*
*out: True #if out is True then Same*

*#OR can use*
*id(arr1) #will give memory location. compare with id(arr2)*

*# just give one of the elements a new value and check if the other changed as well*

***Changes made to the slice or the new reference of the same memory location, will effect the original***

**Adding and Removing Elements from Array**

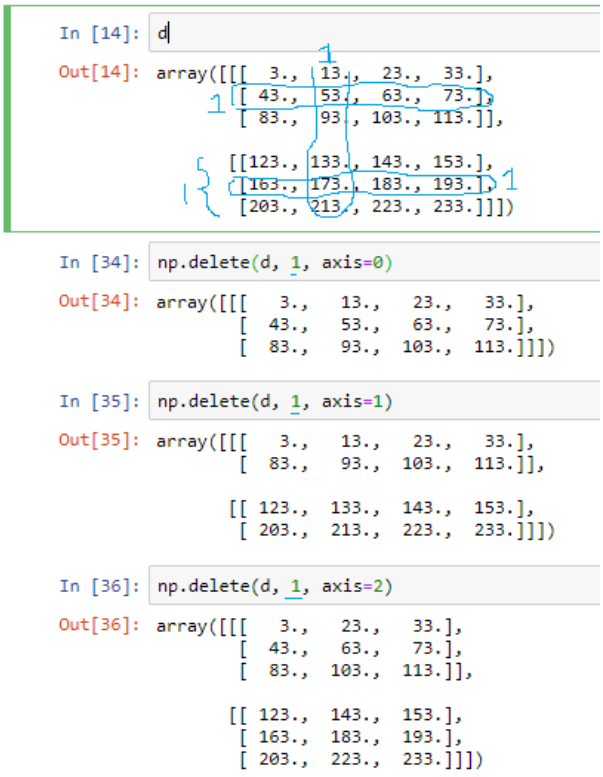
**APPEND**
*arrB=np.append(arrA, [5,6,7,8]) #append 5,6,7,8 to arrA*
*np.append(arrA,arrC, axis=0) #Append along axis 0*
*#Remember Axis can take axis = 0,1,or 2*

*hstack (horiz stack is same as append)*
*np.vstack((K-array ,M-array)*
*np.concatenate([K, M], axis = 0)*

**INSERT**

```
In [29]: c
Out[29]: array([[[  3,  13,  23,  33],
                 [ 43,  53,  63,  73],
                 [ 83,  93, 103, 113]],

                [[123, 133, 143, 153],
                 [163, 173, 183, 193],
                 [203, 213, 223, 233]]])

In [12]: after_insert_array = np.insert (c, 1, 444, axis=0)
         after_insert_array
Out[12]: array([[[  3,  13,  23,  33],
                 [ 43,  53,  63,  73],
                 [ 83,  93, 103, 113]],

                [[444, 444, 444, 444],
                 [444, 444, 444, 444],
                 [444, 444, 444, 444]],

                [[123, 133, 143, 153],
                 [163, 173, 183, 193],
                 [203, 213, 223, 233]]])
```

**DELETE**




- 2 SECTIONS PENDING
- COMPLETE & ADD CERTIFICATE ON LINKEDIN
- MOVE CODE / NOTES TO CHEAT SHEET AS NEEDED
- (Pending is broadcast onwards)
- CHECK any other Note IN ONE NOTE -- AGGREGATE (ex - UCSD NOTEBOOKS)
- CHECK OFFICE NOTES points IF ANYTHING
- THAT ENDS NUMPY

**Joining & Spliting Arrays -**

*joined_arr = np.**concatenate**((a,b),axis=0)  #note you pass a tuple of the 2 arrays*

*concatenate creates a copy not a view.*

*np.stack()  #*

*np.**split**(ary,indices_or_sections,axis)  #seperates an arrray into 2 or more sub arrays*
*If `indices_or_**sections**` is an integer, N, the array will be divided*
*into N equal arrays along `axis`.*
*if an array then, ``[2, 3]`` would, for ``axis=0``, result in*
*- ary[:2]  so 0 -2*
*- ary[2:3]  2- 3*
*- ary[3:]  3 onwards*

*#Need to see a proper example.*

**Array Shape manipulation -**

*reshape()*
*#Seen earlier*
*#Note reshape() creates a view not a copy.*

*ravel()*
*#Creates a continuous flat array*
*#documentation tells us that if we apply ravel to a matrix, the return value is a one-dimensional N-D array.*
*Otherwise, the return value is an array of the same subtype as the array to which ravel is applied.*
*Basically converts a nd array into a 1 dimensional array.*

flat()

*Flat is used as an iterator*

**Rearranging Array elements**

*np.fliplr(arr_name)  #flip left to right*
*NOTE - The flipping take place along the last index*
*so for a (3,4,5) array it would be the '5' elements that are flipped*

*np.flipud(arr_name)  #flip up down*

*np.roll(arrname,4)  #roll elements by specified no of places.*
*#note that the roll is not fixed to one axis. an element could move into a different axis after rolled.*

*np.rot90(arrname)*

**Transpose like operations**
*np.transpose(arr_name)*
*np.transpose(arr_name, axes = (0,3,1)  #To indicate on which element we want to transpose.*

*np.swap(arrname,axis 1,axis 2)  #Interchange 2 axes of an array*
*1st parameter is the 1st axis where swapping will take place.*
*2nd parameter is the next axis where swapping will take place.*

*http://stackoverflow.com/questions/15483156/generalizing-matrix-transpose-in-numpy*

**Tiling Arrays**
*Creating an array built on rectangular repetitions of another array.*

*np.tile(arrname, repition_count)*

# Pandas Time Series

Monday, December 30, 2019 9:23 PM

https://www.youtube.com/watch?v=r0s4slGHwzE&list=PLeo1K3hjS3uvMADnFjV1yg6E5nVU4kOob

*df = pd.read_csv("aapl.csv",parse_dates=["Date_col"], index_col="Date_col") #Parse-says which column we want to treat as type DATE*

*#There are certain benefits of making the Date column as index; for Slicing, getting data by Month, data for range of dates etc. Ex-*

*df['2017-01']['ColName'] #Will give all entries of Jan 2017. Note because Only partial date entered. month &yr*
*df['2017-01-08':'2017-01-03']*

***#RESAMPLING***
*- looking at data by a particular FREQUENCY (Month, Minute, Week, Semi-annual, Quarterly etc.)*
*- Kind of like Changing granularity from Day to Month. AND then aggregating.*

*df['colName].resample(rule='M').sum() #Column entries will be aggregated (i.e. sum) at Month level*
*df['colName'].resample(rule='3M').sum() #Col sum for every 3 Months*
*df['colName'].resample(rule='3M').sum().plot() #Plots column against the Index (i.e. Date)*

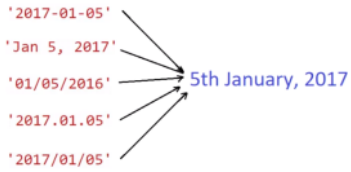
*#resample also has option to choose **'on'** which col and '**level**'.*

*plt.plot(ga_kw_traffic_data['Users'].resample(rule='M').sum()) #A Time series plot. Df with index time series*

***date_range()*** *#to create a range of dates*
*rng = pd.date_range(start = '2/1/2017', end = '6/1/2018', freq = 'B') # here B is business days.*
*#This array can then be added to a dataframe as a col or an index.*
*rng = pd.date_range(start = '2/1/2017', periods=365, freq = '**D**') # If end date is not known but duration is known*

***asfreq()***
*df.asfreq('D', method= 'pad') #To change time-freq of your dataframe. 'Pad' says entries that do not have a value take value of previous entry.*

***to_datetime()***
*- To represent a date in the desired format*




*dates = ['2017-01-05', '**Jan 5, 2017**', '01/05/2017', '2017.01.05', '2017/01/05','20170105']*
*pd.to_datetime(dates)*
OUT: *DatetimeIndex(['2017-01-05', '**2017-01-05**', '2017-01-05', '2017-01-05',*
*'2017-01-05', '2017-01-05'],*
*dtype='datetime64[ns]', freq=None)*

*#Note - As seen above DATES DISPLAYED as **YMD***

*d1 = '06-02-2018' # Pandas interprets this as Month Day Year*
*to_datetime() will produce timestamp of 2nd June 2018*
*if in data Day is first then use*
*d2 = pd.to_datetime(d1,format='%d-%m-%Y') #will produce timestamp of 6th Feb 2018*

*EPOCH time - no of seconds that have passed since Jan 1, 1970 00:00:00 UTC*

***pd.Period***
*#Extracts out Time period from a Date-String or a Timestamp*

```
d2 = pd.to_datetime('14-03-2018')       #d2 is a timestamp

f = pd.Period(d2, freq='Y')      #Extracts out the full YEAR as a TIMEPERIOD
OUT: Period('2018', 'A-DEC')

f.start_time                     #the object f then has different methods. (check with dir(f))
OUT: Timestamp('2018-01-01 00:00:00')

lll'ly
Timpd1 = pd.Period('16-03-2018', freq = 'M')
Timpd1         #Extracts the Month as a PERIOD from the string.   OUTPUT:  Period('2018-03', 'M')
Timpd1.month         #Gives OUTPUT:3
Timpd1.start_time     #Gives OUTPUT: Timestamp('2018-03-01 00:00:00')
Also can do
Timpd1+1      #Gives OUTPUT Period('2018-04', 'M')

period_range()
#Similar to date_range but creates range of PERIOD (not dates or timestamps)
pd.period_range('2011','2019',freq='Y')
```

MOST VIDEOS ON TIMESERIS FROM THIS PERSON COVERED

```
Timd1.to_timestamp()         #Converts PERIOD to TIMESTAMP
```

# DATA VIZ PYTHON

Monday, December 30, 2019 9:21 PM

# Plotting Code

Tuesday, December 24, 2019 9:43 PM

***PLOTTING***

***%matplotlib inline***

**MATPLOTLIB works with lists, generally numpy arrays. In fact, all sequences are converted to numpy arrays internally.**

**Can have WRAPPERS around MATPLOTLIB Code -**
**PANDAS WRAPPER**
**SEABORN WRAPPER**

**MATPLOTLIB Has 2 SYNTAX - MATLAB Syntax & OBJECT ORIENTED Syntax**
(You can have charts that would use a combination of both)

***#MATLAB SYNTAX***
***plt.plot(Arr1, Arr2 ...)*** *#Simple plot of ColX or XvsY as lines or marks. No specific type of plot.*
*# NOTE -* ***df.loc[:,'col'].values*** *to convert to numpy array*

***plt.plot(df[col4X] , ...)*** *#Simple plot of ColX or XvsY as lines or marks. No specific type of plot.*
***plt.scatter(df[col1],df[col2]..)*** *#Similarily we have plt.hist*

***#OBJECT ORIENTED SYNTAX***
***fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2,figsize=(8, 4))*** *#initializing FIGURE and AXES. & initialize objects ax1,ax2 for each axes*
***ax1.scatter(…)***

***Subplots***
***plt.figure(figsize=(4,6)***
***plt.subplot(221) ; plt.hist(…)*** *#for multiple Axes/plots/vizzs in same figure. 2 rows 2 columns 1 index*

***PANDAS***
***df.plot( 'col4x' , 'col4y' , kind='…')***
***df[['col1','col2']].plot('col1','col2',kind='bar')*** *#Plots specified columns of the dataframe[[col1 , col2]]*
***df[['col1','col2','col3']].plot(kind='bar')*** *#Plot the dataframe i.e. df[[col1,col2,col3]] aginst index*

***SEABORN***
***sns.boxplot(df[colA],df[colB])*** *#So works directly with dataframes*
*#OR:*
***sns.boxplot(x='colname', y = 'col2name', data=df_name)*** *#works directly with dataframes*

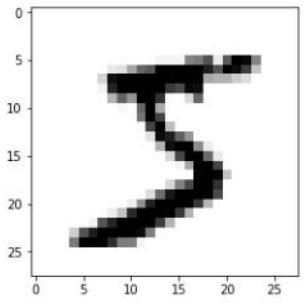
# Plot examples

Tuesday, December 24, 2019 9:53 PM

***df.plot(kind='scatter',x='median_income',y='median_house_value')*** #Simple Scatter plot that plots all points for x & y

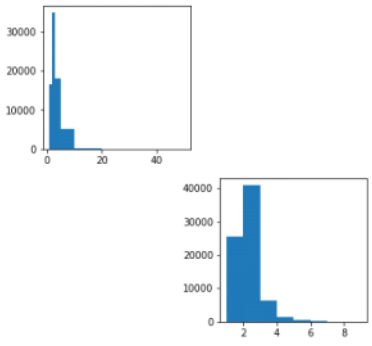
```
df.plot(kind="scatter", x="longitude", y="latitude",
      alpha=0.4,
   s=df["population"]/100,
      label="population", figsize=(10,7),
   c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,
)
plt.legend()                                    #Scatter Plot with more details

#To Compare Test and Predicted
plt.scatter(X_test, y_test,  color='gray')
plt.plot(X_test, y_pred, color='red', linewidth=2)
plt.show()

plt.style.available    #To see the available aesthetic style
plt.style.use('fivethirtyeight')    #to use a different style

plt.imshow(array_name,cmap=mpl.cm.binary)  #cmap is color map and mpl is matplotlib
#Alongside is how a 28x28 Pixel image looks, its feature set is 784 entries each representing
#pixel intensity white(0)  -  Black(255)

plt.hist(data['TOTAL_PERSONS'],label='Total_injured_persons',bins=[1,2,3,5,10,20,50],edgecolor='black')
#start and end number of each bin can be supplied to the histogram function. [1, 2, 3, 4]
then the first bin is ``[1, 2)`` (including 1, but excluding 2) and the second ``[2, 3)``.  The last bin,
however, is ``[3, 4]``, which *includes* 4.

#Labels, Title, Limits setting, TIcks - MATLAB syntax
plt.xlim(left=1,right=100)
plt.ylim(bottom=0, top=500)
plt.xlabel('nameX') ; plt.ylabel('adf', fontsize=5); plt.title('tit3')
plt.grid()   #to add gridlines. Can have different color, transparency, only for 1 axis if needed etc.
plt.plot(month_number,interest_paid,marker='*',markersize=10,c='r')    #Marks

#Labels, Title, Limit setting,Ticks - OO Syntax
ax1.set_xlim(left = 1, right = 100)
ax.set_xlabel('labelname')
axes.grid()       #Can maybe use a style with a certain type of grid lines
axes.legend(loc=''center right)  or plt.legend(loc=(1.2,0.5))    #think of this as %of axis. so 50% up Y ax

#Save plot to files
plt.tight_layout() #automatically arranges subplot sizes so subplot fits into the figure area
plt.savefig('loc\filename',dpi)
OR
fig.tight_layout()
fig.savefig('loc\filename',dpi)
```

```
In [31]: plt.imshow(Fst_digit_img,cmap=mpl.cm.binary)
Out[31]: <matplotlib.image.AxesImage at 0x16f00f95cc0>
```



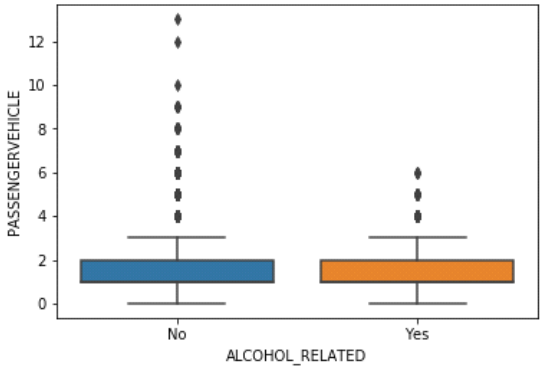
## *USE OF MULTIPLE SUBPLOTS (AXES) IN THE SAME FIGURE*

```
In [28]: plt.figure(figsize=(6,6))
         plt.subplot(221); plt.hist(data['TOTAL_PERSONS'],bins=[1,2,3,5,10,20,50])
         plt.subplot(224);plt.hist(data['NO_OF_VEHICLES'],bins=[1,2,3,4,5,6,7,8,9])
Out[28]: (array([2.5606e+04, 4.1041e+04, 6.2440e+03, 1.4770e+03, 3.9300e+02,
                 9.6000e+01, 2.7000e+01, 1.7000e+01]),
          array([1, 2, 3, 4, 5, 6, 7, 8, 9]),
          <a list of 8 Patch objects>)
```



## -- *BOX PLOTS*

```
[11]: sns.boxplot(data['ALCOHOL_RELATED'],data['PASSENGERVEHICLE'])
      plt.show()
```

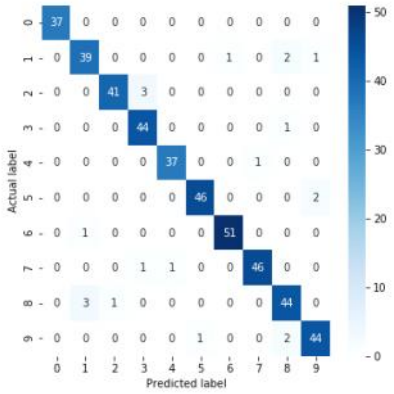



The Box plot would not be that appeasing using basic matplot lib.

-- *HEAT MAPS*

```
In [39]: plt.figure(figsize=(6,6))
         sns.heatmap(confusion,
                     annot = True,
                      cmap='Blues')        #Heatmap -- data,annotations,colormap - can be sequential or qualitative ('Pastell')
         plt.ylabel('Actual label')
         plt.xlabel('Predicted label')
```

```
Out[39]: Text(0.5, 33.0, 'Predicted label')
```




This is actually harder and more code to do in matplotlib, without the seaborn wrapper.

-- Interactive Data vizualization libraries.
FOLIUM
BOKEH

--- Another example
*ga_page_data[['Page','Pageviews']].sort_values(by='Pageviews',ascending=False).head(5).plot('Page','Pageviews',kind='bar')*

# Offic viz(matplt) notes Add

Monday, December 30, 2019 9:23 PM

# MATPLOT LIB - Lynda

Tuesday, December 24, 2019 7:49 PM

- MATPLOTLIB Library has 2 different styles of syntax
Matlab syntax
Object Oriented syntax
You can have charts that would use a combination of both.

matlab syntax:
typically *plt.plot()* command

OO Syntax
fig,axes = plt.subplots()

- SMALL PART MAYBE PENDING
- COMPLETE & ADD CERTIFICATE ON LINKEDIN
- MOVE CODE / NOTES TO CHEAT SHEET AS NEEDED (CHECK ALL SHEETS IN ONE NOTE) -- **AGGREGATE**
- INCLUDE OFFICE ONE NOTE NOTES
- THAT ENDS MATPLOTLIB FOR NOW

MATPLOTLIB WRAPPERS -
Pandas
Seaborn

Why -
better integration with Pandas data structures (remember MATPLOTLIB is with numpy arrays)
better design asthethics (color, styles etc.)
Support for categorical variables to show aggregate statistics.

Plot using Pandas

```
df.boxplot(column= 'area_mean', by='diagonsis)     #Pandas wrapper around matplotlib
plt.title('plot name')     #can use pure matplotlib along with the Pandas wrapper
```

Seaborn -

# EDA few notes

Wednesday, December 25, 2019 5:29 PM

## *EXPLORATORY DATA ANALYSIS*

- - EDA is about understanding the breadth and depth of the data

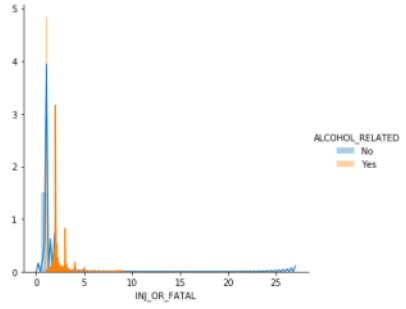
- - A few "Things to find" for any new dataset you get:
  - Structure of the data (maybe schema / table structure)
  - Patterns / Trends
  - Key variables
    And How key variables plot / visualize
  - Any anomalies
  - Check any Hypothesis or assumptions you may have about the data
    ….
    …….. Can be more but this is a start.

-- Understanding the data
How many features (basically fields); How many observations;
What are the features.
Numerial or categorical

- **Univariate analysis**.
Only one variable is involved in this analysis. In fact, the main purpose of the analysis is to describe, understand the population distribution, detect outliers, summarize and find patterns for a single feature.
so using say -
*df['col_name'].describe()*
And looking at things like like mean, median, std, and percentiles.

**- Probability Density Function**
  - ***It just the distribution of the data***.

*sns.FacetGrid(data,hue="ALCOHOL_RELATED",height = 5).map(sns.distplot,"INJ_OR_FATAL"). add_legend()*




- **Cumulative Distribution Function**
PENDING (find explanation)

-- **Categorical Variables**
The characteristics of interest for a categorical variable are simply the range of values and the frequency of occurrence of each value

"The representation and presentation of data to facilitate understanding"
- Andy Kirk
(Book on data Visualization; SAGE)

Data Visualization types -
**Conceptual and Data Driven**

Conceptual -- Classic Demand Supply Curve; this can then be backed up with data using an example such as Uber search pricing.

Visualizations often encourage us, and enable us to look deeper into the data.

Visualizations can be **Declarative** (where you are trying to convey results obtained) or **Explorative** (where you are using a visualization to explore different relationships and interpret more of the data)

A Good Data visualization is:
1. Trustworthy
2. Accessible
3. Elegant (you should focus on what is relevant and remove anything that isn't adding to the figure)

-- Andy Kirk

Nate Silver:
*538*

# Check and del

Tuesday, December 31, 2019 1:52 PM

**Introduction to Seaborn**

Matplotlib plots a Series, you can plot multiple series.
*plt.plot(df_name['ColumnName'] #Basically each row entry of the Column is plotted*

Seaborn is an additional python package that works on top of Matplotlib. It gives additional visualization options.

```
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
#Distribution:
Viz1 = sns.distplot(demodata['Internet Users'])
Or
Viz1 = sns.distplot(demodata['Internet Users'], bins = 10)
#This will give a distribution (i.e. no of countries) of No of internet users. A histogram.
#Box Plot:
vis2 = sns.boxplot(x='Income Group',y='Birth rate',data=demodata)
#Gives different Box Plots of Birth rate as it varies for different income groups.

#Linear Model Plot
viz3 = sns.lmplot(x = 'Internet Users', y = 'Birth rate', data=demodata)

# if required we can switch the plot to not fit the regression.
viz3 = sns.lmplot(x = 'Internet Users', y = 'Birth rate', data=demodata, fit_reg=False, hue='Income Group')
```

Can also just check **Seaborn Gallery**

# MATH

Saturday, October 19, 2019 12:33 PM

1a -
Reddit Recommendations (Saved Reddit Post)
http://joshua.smcvt.edu/linearalgebra/ (Text book)

Quick Notes / Cheat sheets / Problems
http://tutorial.math.lamar.edu/

b.
THE MIT COURSES COME MOST RECOMMENDED

2 Great books on Math+Statistics+ Machine Learning Models **(But in R)**

**ESLR**: https://web.stanford.edu/~hastie/Papers/ESLII.pdf

**ISLR**: http://faculty.marshall.usc.edu/gareth-james/ISL/ISLR%20Seventh%20Printing.pdf
(ISLR - easier read)

https://www.youtube.com/watch?v=5N9V07EIfIg&list=PLOg0ngHtcqbPTlZzRHA2ocQZqB1D_qZ5V

2 -
Youtube Channel -- **3BlueOneBrown**

3 -
**Coursera course**. ***
**Imperial college**
https://www.coursera.org/specializations/mathematics-machine-learning

Videos but same as above course:
https://www.youtube.com/watch?v=T3TpdPmTLso
https://www.youtube.com/watch?v=m998PdOCFcY

4 -
**Microsoft**
~~https://www.edx.org/course/essential-math-for-machine-learning-python-edition-2~~

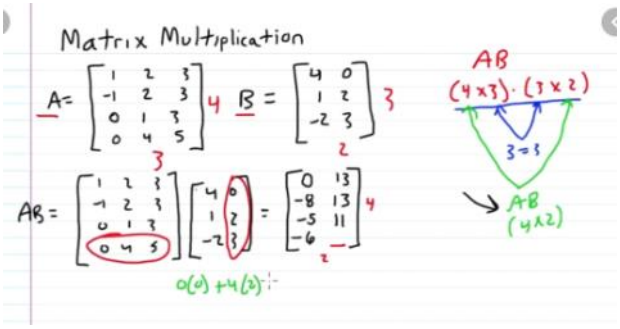
5 -
Probability and Statistics using python (**UCSD**)
https://www.edx.org/course/probability-and-statistics-in-data-science-using-python-2

There are bunch of courses on the Math behind modelling / Probability concepts etc on Edx.

# Matrices and Vectors

Saturday, August 10, 2019 11:27 AM

Vectors in 3d
-- If v w u are vectors
just by multiplying scalers with these can give you all possible vectors in a 3d space.
--3d Vectors represented in planes.




-- Higher order vector combinations are represented by Hyperplanes.

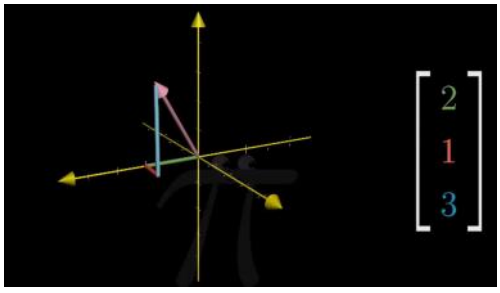
https://www.youtube.com/watch?v=kjBOesZCoqc
***3Brown1Blue*** - can consider doing all their videos and then donate.

What is a Matrix -
Geometric meaning of putting nos in a matrix.
Visualizing a vector space.

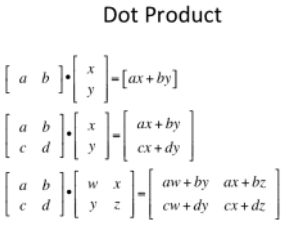
-- Linear transformation is converting input to an output
Like rotating a vector by 60 degrees or 90 degrees etc. basically transforming it to a new coordinate system.
ex:Man walking on a street. Highlighting just the man is an example of linear transformation.

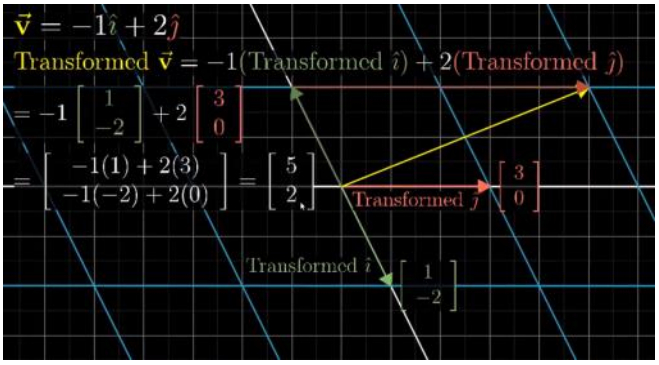
$$\underset{4\times 2 \text{ matrix}}{\begin{bmatrix} a_{11} & a_{12} \\ \cdot & \cdot \\ a_{31} & a_{32} \\ \cdot & \cdot \end{bmatrix}} \underset{2\times 3 \text{ matrix}}{\begin{bmatrix} \cdot & b_{12} & b_{13} \\ \cdot & b_{22} & b_{23} \end{bmatrix}} = \underset{4\times 3 \text{ matrix}}{\begin{bmatrix} \cdot & x_{12} & x_{13} \\ \cdot & \cdot & \cdot \\ \cdot & x_{32} & x_{33} \\ \cdot & \cdot & \cdot \end{bmatrix}}$$

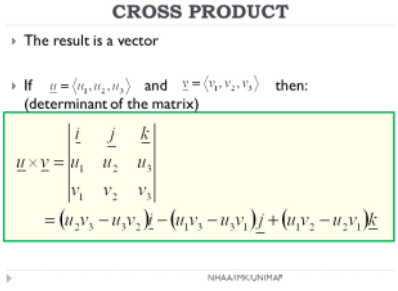
To understand how this vector transformation works in a geometric space, when say a vector is multiplied think of it as each column (*I's, j's,k's*) being moved along that axis by some distance.

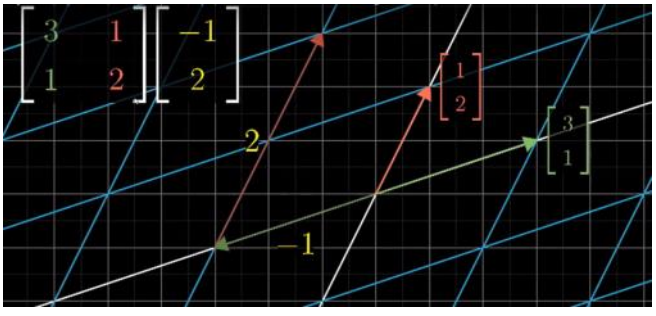



-- Determinant
The determinant for a 2D matrix is the Area or the factor by which th area of a unit matrix is
multiplied.
For a 3D matrix it's the volume of the cube / parallelopiped.




A vector has both Mag and Direction.
The Cross Product a × b of two vectors is another vector that is at right angles to both (**Direction**). Some times called Vector Product.

The **Magnitude** (length) of the cross product equals the area of a parallelogram with vectors a and b for sides.
The Dot Product which gives a scalar (ordinary number) answer, and is sometimes called the scalar product.

**Quick Summary of Matrix operations**
https://www.ritchieng.com/linear-algebra-machine-learning/

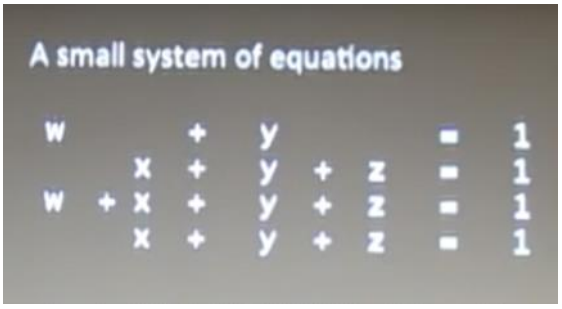
https://www.youtube.com/watch?v=HHUqhVzctQE&list=PLQVvvaa0QuDfKTOs3Keq_kaG2P55YRn5v&index=21

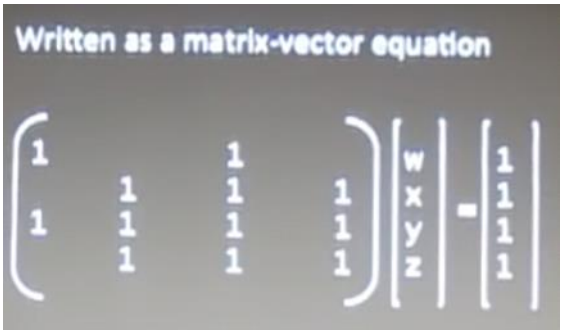
# Visualizing Matrices

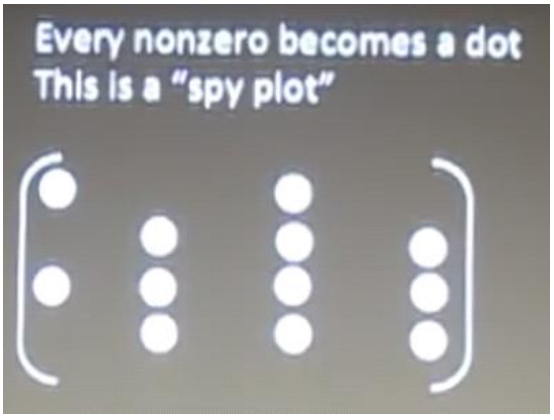
Sunday, September 22, 2019 11:48 AM

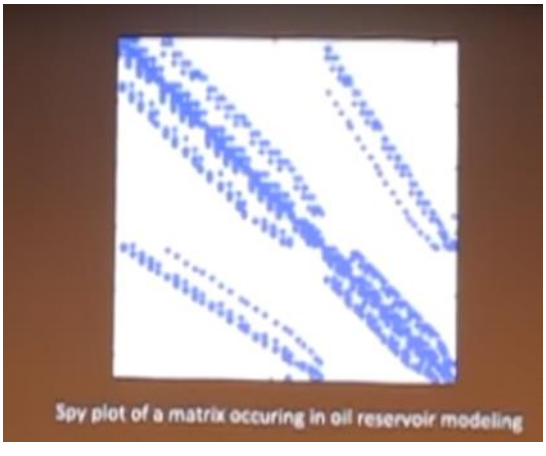
If I want to solve 4 unknowns, I need 4 equations that tell me how they relate to each other.
In the system of equations the pattern of Non-Zeros for each coefficient gives information about the system.

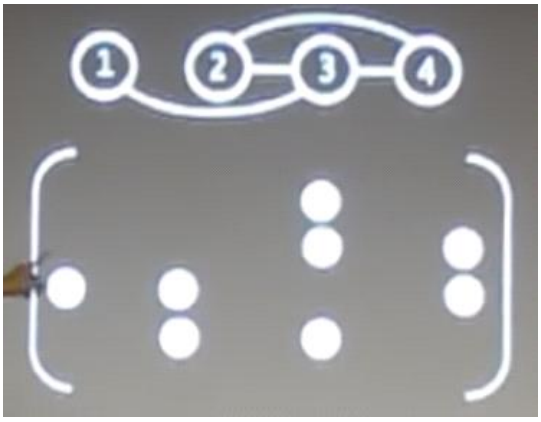
By Margot Gerritson.

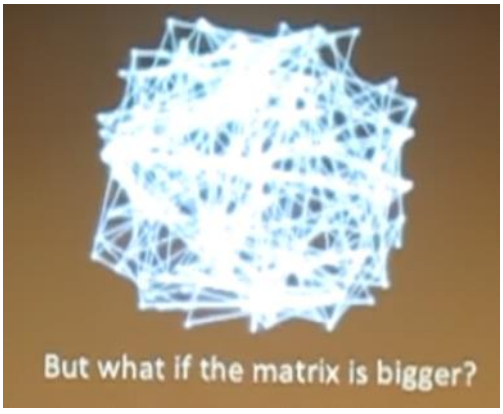


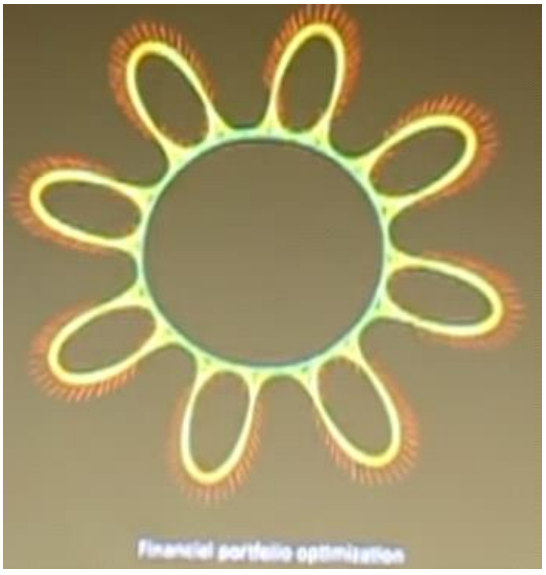






Zoomed In Social
Media Matrix

# Probability - Random variables

Monday, September 03, 2018 1:20 PM

**Khan Academy Tutorial**
https://www.khanacademy.org/math/statistics-probability/random-variables-stats-library/binomial-random-variables/v/10-percent-rule-independence

**Random variable**
A variable that can take on different values. Binomial variable is an example of a random variable.
Ex: X = # of Heads when a coin is tossed 5 times.
Here X can take a number of different values.
But a random variable will be a binomial variable if it follows certain conditions as below.

**Binomial variables**
Conditions for a Binomial variable

The conditions for a binomial random variable are:

- ✓ The outcome of each trial can be classified as either success or failure.
- ✓ Each trial is independent of the others.
- ✓ There is a fixed number of trials.
- ✓ The probability $p$ of success on each trial remains constant.

--> Choosing 2 cards from 6 can be done in 6C2 ways.

Examples -

Not binomial

In a game involving a standard deck of $52$ playing cards, an individual randomly draws $7$ cards without replacement. Let $Y$ = the number of aces drawn. Not independent trials

Binomial

$60\%$ of a certain species of tomato live after transplanting from pot to garden. Eli transplants $16$ of these tomato plants. Assume that the plants live independently of each other. Let $T$ = the number of tomato plants that live.

Not fixed # of trials

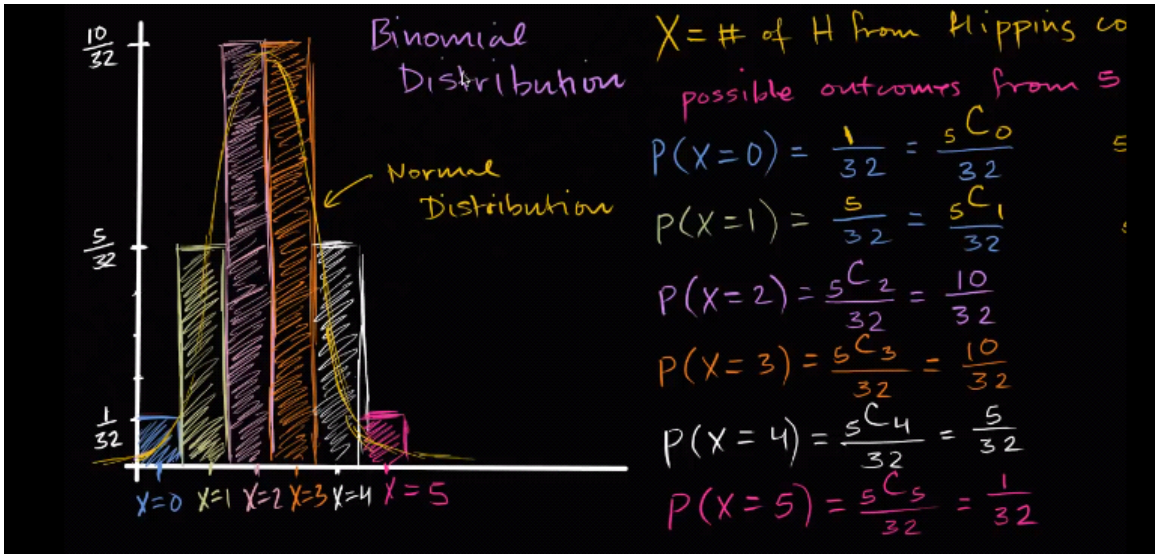
In a game of luck, a turn consists of a player continuing to roll a pair of six-sided dice until they roll a double (two of the same face values). Let $X$ = the number of rolls in one turn. Not Binomial

**10% Rule of assuming 'Independence'**
If your sample size is <= 10% of the population
then it is ok to treat your Random variable as Binomial

**Binomial Distribution**

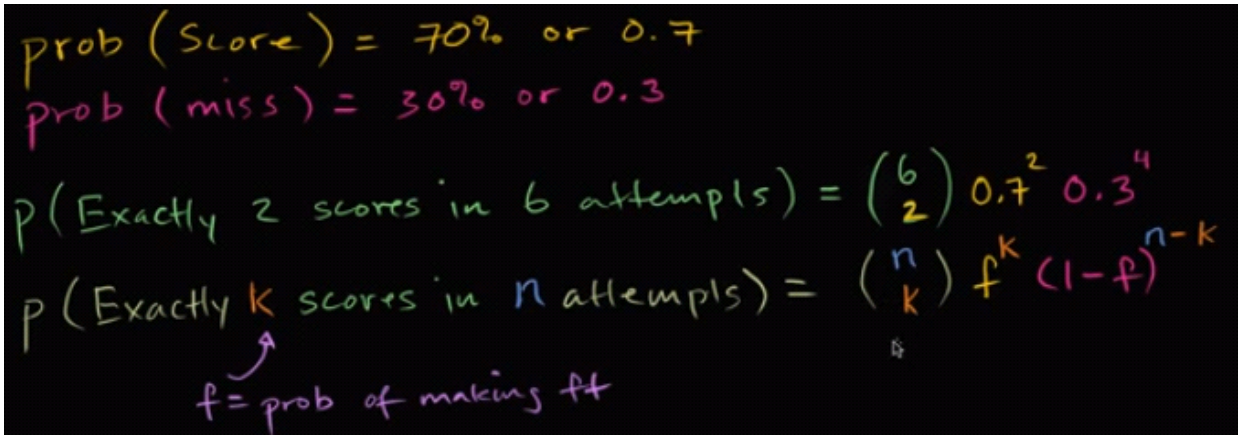
Random ; Discrete; finite distribution

Binomial distribution is the discrete version of a normal distribution. As the number of trials increases the binomial distribution will approach the normal distribution. When you have many many values you are coming closer to a continuous probability distribution function or a normal distribution.



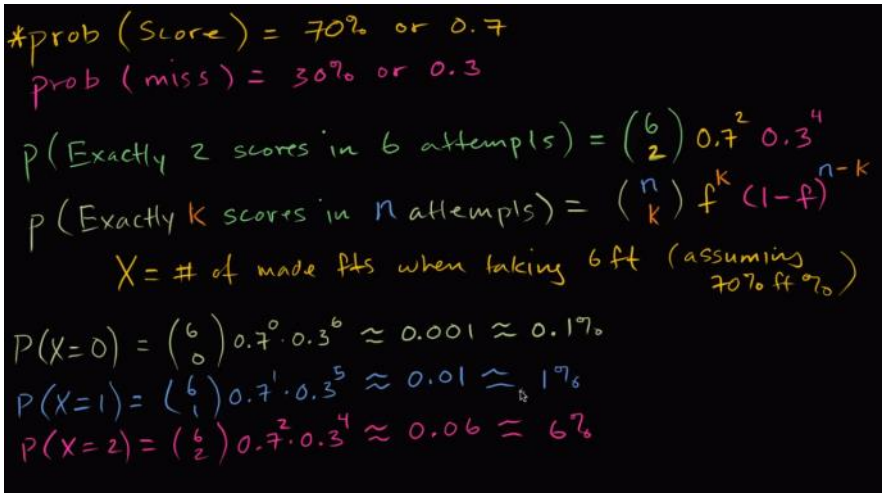

**Example**

This basically is an example of ***getting the Probability of different values of a random variable***.




So if
P(scoring) = 0.7
and
**X was # of free throws made when taking 6 shots** then,
P (X = 0) = 6C0 * 0.7^0 * 0.3^6
P(x=1) = 6C1 *0.7^1 * 0.3^5
..
..
..
P(X=6) = 6C6 * 0.7^6 * 0.3^0

As seen below



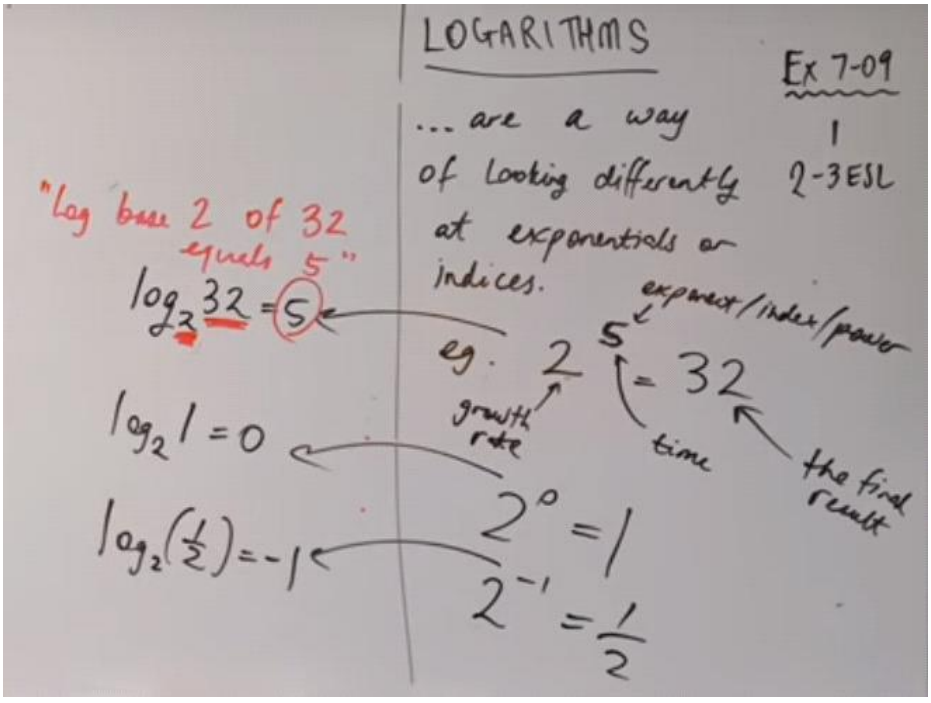

BINOMPDF and BINOMCDF functions
Basically calculating the probability

In excel it is:
*BINOM.DIST(4,7,0.35,False)*

# Logarithms

Tuesday, August 6, 2019 9:15 PM

**Logarithms are a way of looking differently at exponentials or indices**.
**A different perpective to powers or exponents**




Indices or **Powers or Exponents** (as in pic RHS) says -
**$2^5 = 32$**
so if 2 is the growth rate
5 is no of time
32 the final value
assume initial value 1.

Then what (the Index/Exponent) is conveying is "if your doubling each time, how many times to get to **32x** (i.e. 32 times what you are); 5 times"
Simillarily, " if your staying the same, how many times to get to **1x** (same or 1 times what you are); 0"
or "if you become half, how many times to become **1/2x** (i.e. half times what you are); ans is -1)

So Logarithms as a different way of looking at Exponents implies --
**$\log_2 32 = 5$**
What this tells you is if 2 is the rate you grow and 32x is where you end up, how many times did you grow; ans is 5
{its just a different perspective to exponents}
Simillarily, $\log_2 1 = 0$
if your doubling and you ended same size as before then how long were you growing for; 0.

So Different perspective to exponents, basically instead of 'if you know the power how big does the answer end up (i.e. how many time the original)' it is 'if this is the answer whats the power'

==>
$\log_5 5 = 1$

Law
$\log_a a = 1$
$\log_a p + \log_a q = \log_a(pq)$
$\log_a p + \log_a q = \log_a(p/q)$
$\log_a p^n = n\log_a p$

If you growing at the rate of 5 i.e. 5, 10, 15, 20 ….. how many times till you get to 5; Ans: 1

To continue from this video -
https://www.youtube.com/watch?v=3WIWSYVR1Q8

# Calculus

Wednesday, September 18, 2019 7:48 PM

Calculus basically helps figure out 2 problems -
1 - Calculating Area and Volume
We know area of a circle or a rectangle, but what if we find the area of a figure with no known shape. Calculus helps us figure out the area of such crazy figures. And it tells the exact area or volume.

Basically if we have the function of the curve we can use calculus to find the area underneath the curve. We use integeration to get the area.

2 - Whats the slope of a line is.
Ex - growth of a stock over time. Slope indicates the rate of growth.
Slope (we know is tan theeta)

But slope of a curve may be constantly changing.
So we calculate the exact rate of change at one precise moment.

Again we have the function of the curve. The first derivative of the function gives us the slope.

And it is a big deal to be able to calculate the slope of a curve (for ex - if its curve/linegraph of crime rate vs time). Because then we want to see things like where the slope was maximum i.e. Biggest change in crime rate. And also say the Minima of the curve, so where the curve was minimum.

Do 3Blue 1Brown - the Essence of Calculus video

# Other

Sunday, April 21, 2019 8:41 PM

**Math Websites** -

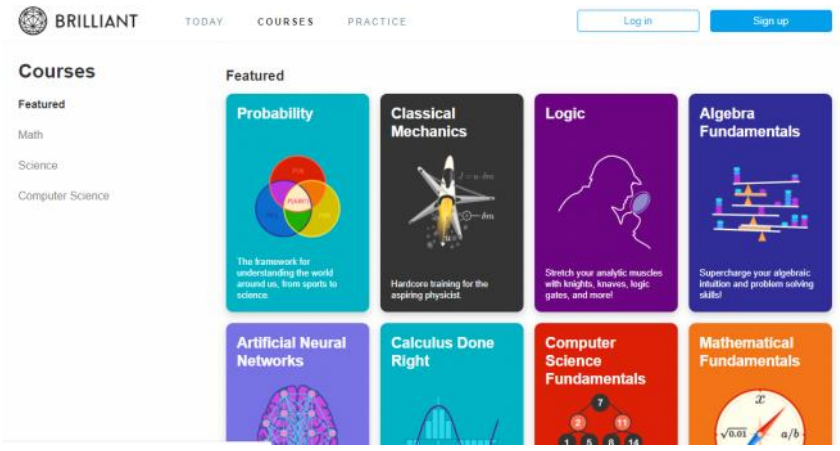
**Brilliant.org**
*Topics taught based on real/practical problems.*

Khan Academy

**Misterwootube.com / YouTube channel-Eddi woo**

https://www.mathbootcamps.com/courses-and-downloads/

https://www.youtube.com/watch?v=tzI1zWgFXow



# Python For Marketing

Saturday, January 11, 2020 11:20 PM

Saturday, January 11, 2020 11:21 PM

**Sources imported**:

Google Analytics -
- GA Keyword data
- GA Social data
- GA referral data
- GA Channel data

Google Search Console -
- Basically search query (or term/phrase searched in Google search)
(This is probably Paid Search..)
- Also Country
- & Device

Adwords PPC Data -
Table of Ad group, Campaign, CPC, CTR, Clicks, Conversion, Cost etc.

Facebook Ads Data -

Also - Competitor Site csv, Keyword Rank tracker sheet

Google Trends Data
Pytrends to import google trends data --
Let's see how used. From Pytrends API

**Some Checks** on the data usability -
Dates (parse as date columns/list of columns when importing)
NA's
Unnamed columns
Fix data types (ex - Numbers stored as Text
Currency

For Github -
Redownload all the NBs
Modify
Upload to Github

**Fixing GA Data**
Section 3 lecture 2
- 1 dataset fix demonstrated in this video wherein the url name as displayed in GA exported data needs to be formatted differently.