DSC 80: The Practice and Application of Data Science

Qirui Zheng

February 5, 2026

Acknowledgements

This set of notes follows the lectures from University of California, San Diego's undergraduate course DSC 80: The Practice and Applications of Data Science. This course is taugh by Professor Sam Lau from the department of Halıcıoğlu Data Science Institute (HDSI) Fall Quarter 2023. For any typos, please email: q7zheng@ucsd.edu.

Contents

Chapter 1

Lecture 1: Introduction, Data Science Lifecycle

1.1 Introduction: What is data science?

Data Science is the intersection across multiple fields, and is rapidly developing. From DSC 10 data science is drawing conclusions from data with the use of computation[1]. Where python is used to explore and visualize data, simulated sample data to make inference about a larger population, as well as making predictions on data about the future when given past data. Data Science in the work field is currently weighted more towards Machine Learning, cloud/containers, and data tools. In DSC 80, the goal is to be able to answer questions using data, as the answers are more ambiguous. This ambiguity is what makes data science challenging. However, data science involves people! Behind every single data point can be an living individual, thus the decision we make have the potential to impact the livelihoods of other people. Although there are many tools available to perform data analysis, it is up to the user of these tools to make the important decisions.

1.2 Data Science Lifecycle

Data science can be understood as a workflow in cycles. On the surface, it is the cycle of a hypothesis to data modeling and ending with a conclusion. The conclusion can then spark more ideas and questions. However, this cycle hides a lot of complexity, and is missing the most important part, the data. It is actually a two way loop, during the process it is important to go back to the data to obtain more, or revise the question after understanding the data.

[1]DSC 10: Principles of Data Science is a prerequisite course for DSC 80

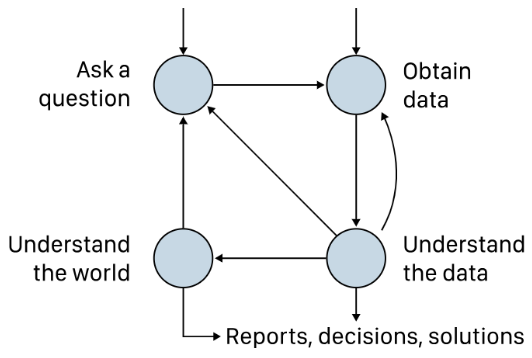


Figure 1.1: Data Science Cycle

Chapter 2

Lecture 2: DataFrame Fundamentals

2.1 Pandas review

Pandas is a library in python for working with data, it is build on top of Numpy. Numpy is another library in python that is implemented in C, the benefits of numpy is the ability to support vectorized operations. There are three main data structures in Pandas.

1. DataFrame (df): 2 dimensional table
2. Series: 1 dimensional array-like object, typically representing a column or row. Each column in a dataset is a series.
3. Index: a sequence of column or row labels

2.2 Basic Pandas Operations

The initial conditions are, df is a data frame object in python.

To obtain the fist x rows of a DataFrame

```
df.head(x)
```

To obtain the last x rows of a DataFrame.

```
df.tail(x)
```

To obtain the number of rows of column of a DataFrame, it returns a tuple in the format of (rows, columns). Notice it is an attribute of the DataFrame object, which is different from the two function calls of a DataFrame object from above.

```
df.shape
```

To sort the column of a DataFrame. Notice that the ascending can be set to $False$ to sort in descending fashion.

```
df.sort_values("column name", ascending=True)
```

To set the index of a DataFrame. Notice that the index values need to be unique identifier or name. This operation is not in place and needs to be reassigned, or the *inplace* parameter can be used for the operation to be in place(see line 3).

```
df = df.set_index("index name")

df.set_index("index name", inplace=True)
```

To get a column(s) of a DataFrame. Notice with out listing the column(s) it will return a series, but when columns are listed it returns a DataFrame object. Some common error with this query is key error, it might be caused by using the wrong column name or wring query type. Note: This method is frequently used in DSC10 with babypands

```
df.get("column name") #This only works to query a single
    column, return a series
df.get(["column name1", "column name2"]) #This work to query
    single or multiple columns, return a DataFrame object
```

To obtain all the unique values within the column

```
df["column"].unique()
```

To obtain the distribution of the count of elements in the column

```
df["column"].value_counts()
```

To obtain the mean of a column. Note that this is the numpy mean, not original python method.

```
df["column"].mean()
```

To obtain the sum in a DataFrame, Axes should be considered. Where it is either summing over the row or columns. If Axis is not specified it sums to the columns in default.

```
df.sum() #sums the columns (same as when axis=0)
df.sum(axis=1) #sumes the rows of a DataFrame
```

To obtain the number of unique values

```
df.nunique()
```

To obtain general statistical info of a column or a DataFrame. This returns the counts, mean std, IQR, min, max statistics.

```
df["column"].describe()
df.describe() #used to describe the whole DataFrame with
    respect to each column
```

To slice out rows and columns using labels, this is also flexible with slicing in expanded dimensions

```
df.loc["row label", "column label"]
df.loc[["row1", "row2"], ["column1", "column2"]]
df.loc[["row1", "row2"], :] #this can get all column or row,
    same concept as index slicing in python lists
```

2.3 Filtering - Querying

Filtering is often used to obtain certain parts of the data that satisfies given conditions. It uses the result of a boolean series from comparison made with array (series).

```
df.loc[df["column"]<x] #The following result is like filter
   that takes out all true values, in other words take out all
    values when this condition is met.
```

However, *.loc* is not always required, can use the short cut to directly query

```
df.loc[df.index.str.contains("string")]
df[df.index.str.contains("string")] #This will retrun the same
    output as the line from above
```

Note: When filtering with multiple condition. Must use & and | instead of key words, using key words causes pandas to make weird decisions.
To query for data we can also use string sequences instead of conditions.

```
df.query("string condition") #general form
df.query("A>B") = df[df.A > df.B] #where A/B are column names
df.query("kind in ["column1", "column2"] and column < x")
```

.iloc is another way of filtering for data in a DataFrame, where it uses the integer location to query for data. It is useful when the data is sorted first.

```
df.iloc[row, column] #must be integer values
df.iloc[x:y, z:w]
```

EXAMPLE: Given the DataFrame below, notice that in column1 it is the integer 1, while in column 2 it is the string "1". To keep things similar, the integer 1 will be represented as 1, while the string 1 will contain quotes as "1".

Index	**1**	**"1"**
0	fee	fo
1	fi	fum

Q1:

```
df[[1]] = ?
```

Ans:

Index	**1**
0	fee
1	fi

Q2:

```
df["1"] = ?
```

Ans: Note that although the answer is presented in tabular format here, but in python the result is a series.

0	fo
1	fum

Q3:

```
df[1] = ?
```

Ans: Note that although the answer is presented in tabular format here, but in python the result is a series.

0	fee
1	fi

Q4:

```
df[[1, 1]] = ?
```

Ans:

Index	**1**	**1**
0	fee	fee
1	fi	fi

Q5:

```
df.loc[1] = ?
```

Ans: It takes out the row of a DataFrame indexed by 1. Note that although the answer is presented in tabular format here, but in python the result is a series.

1	fi
1	fum

Q6:

```
df[1, ["1", 1]] = ?
```

Ans: This throws a type error, due to the fact tha df[1] takes out the column with name of integer 1 then thrun to locate the rows od ["1", 1] causing the error. If the the query is made using *.loc*:

```
df.loc[1, ["1", 1]] = ?
```

Ans: Note that although the answer is presented in tabular format here, but in python the result is a series.

1	fum
1	fi

Q7:

```
df.loc[1,1] = ?
```

Ans: "fi" (string fi)

2.4 DataFrame Manipulations

So far, we only looked at how to obtain data and information from a DataFrame, however, when considering the data science cycle, we also want the ability to add new data to the current DataFrame.

To assign a new column to a DataFrame. Note that the *.assign* method returns a new copy, and does not modify the original DataFrame. This assignment can be an array of the same length as the original DataFrame

```
df.assign(newColumnName = df['column1']/df['column1']) #When
    df["column1"] is treated as arrays thus allow the
    performance of arithmitc
df.assign(**{"ANYTHING (emoji or special characters)": df["
    column1"]/2}) #** is dictionary unpacking
```

To assign a column in-place, this mutates the original DataFrame.

```
df["new column"] = df["column"]/2
```

DataFrames are mutable meaning that they can be modified after created. When testing for ideas we can often use a copy of the original DataFrame

```
df_copy = df.copy()
```

When replacing values within a DataFrame. we can directly use the replace method. This method does not mutate the original DataFrame, only returns a new copy

```
df.replace({"column" : {"column key": "replaceValue"}})
```

2.5 Pandas Data Types

Since pandas is built upon numpy it can also be represented in numpy format. This allows operations to be fast since numpy is implemented in c.

1. DataFrame (df): is like a dictionary of columns, each of which is a numpy array
2. Series: it is a numpy array with index

To access the array of DataFrame or Series.

```
df.to_numpy()
```

However, pandas also have it's own datatypes. A series(column) has a numpy data type which refers to the type of value stored within the series. It's value datatype determines which operations can be applied to it. Pandas can often "guess" the correct data type for a given DataFrame, but it usually results in the wrong type. Thus, the the correct datatyples should be explicitly converted.

To show the data types of all columns, note that it is not a method but instead an attribute.

```
df.dtypes
```

To change the data type of a column, note that this method is also not mutable and should be reassigned.

```
df["column"].astype(np.int64)
```

Lastly, how to obtain a DataFrame object in pandas. It can be constructed with array like structures, but more often times there is a large amount of data and create the structure manually is infeasible. Luckily, data is often stored with in a structure (structured data). There are pandas method build in to read in the data files and create the DataFrame. For example, data can be often find in the format of *.csv* which stands for Comma-Separated Values. To read in data, the pandas function below can be used. Note that the dtype parameter is required but it does speed up CSV read time.

```
df = pd.read_csv("path", dtypes={"column": int})
```

Chapter 3

Lecture 3: Aggregating

3.1 Granularity

The concept of granularity refers to what each observation in a dataset represent. Observations can have fine granularity referring to smaller details, or coarse granularity that provides a bigger picture. In terms of a DataFrame, rows correspond to observations, while columns correspond to attributes. During the initial phase of creating a dataset, to gain more control, a finer granularity is often preferred. Since it is easy to remove details but rather hard to add details back in. But there is also the trade off between the time and cost, since a fine-grained data cost will cost more in time and money.

3.2 Groupby: Aggregation

Groupby is method of a DataFrame in pandas. It take thee steps: split, apply, and combine. First, during split, it breaks up and "groups" the rows of a DataFrame according to key. This results in only 1 group for every unique value of the key. Then. during apply, it applies a function to each of the groups. Lastly, it takes the result of the apply functions and combines it together to return a new DataFrame. This process can be parallelized which means it can work on multiple computers/threads by sending computation for each group to different processors.

To get the index of the max element of a column

```
df["column"].idxmax()
```

To get a DataFrame with only the values for the given key

```
df.get_group("key")
#This can also be achieved with query
df.query("column==key")
```

Aggregation is the process of reducing many values to one some examples are, .mean(), .max(), .median(), .sum(), or .last() functions that can be applied to the DataFrame. During aggregation, the calculation is applied to each column independently enforcing column independence. Before apply function should consider the columns that is needed to go through the aggregation and query first to avoid longer run time that the function is applied to the whole DataFrame. DataFrame groupbys have more general aggregation method, it can be single function, list of functions, or dictionary mapping column names to fact.

```
df.groupby("column").aggreate(["function1", "function2"])
df.groupby("column").agg(["function1", "function2"]) #Short
    hand for .aggreate
```

There are other groupby method, such as Transformation and Filtration. In transformation it perform operations to every value within each group. While in filtration it only keep the groups that satisfy the condition.

Transformation Example:

```
def xy(x):
    ''' function take in a series'''
    return x/x.sum()
#first way of using
xy(df["column value"])
#or it can be used with groupby
df.groupby("column group")["column value"].transform(xy)
```

Filtering method take in a function, within the function it can take in a DataFrame of series and return single boolean result DataFrame with only groups containing filter function returned True.

```
df.groupby("column").filter(lambda df: df["column"].mean()>y)
```

Groupby create index based columns, it can be multiIndex if grouped by twice, and it is easier to work with in following with .reset_index(). Note that after you filter you have to groupby again since filter undo groupbys.

```
df.groupby(["column1", "column2"]).agg("function").reset_index
    ()
#or the index can be reset during the groupby
df.groupby(["column1", "column2"], as_index = False)
```

To obtain the summary statistics of two column, a pivot table can be created.

```
df.pivot_table(index = idx_col,
                columns = columns_col,
                values= values_col,
                aggfunc = function,
                #optional parameter to full NaN values
                fill_value = 0)
#Or values can be filled with
df.pivot_table().fillna(0)
```

There are different reshaping methods to reshape a DataFrame.

1. melt: which un-pivots a DataFrame

2. pivot: it is like a pivot_table but with no aggregation

3. stack: this pivots multi-level columns to multi-indices

4. unstack: this pivots multi-indices to columns

From a pivot table we can also obtain the distribution of data. A contingency table can be creating to describe joint distribution of two categorical variable. When pivoting use $aggfunc = count$. We can then normalize the DataFrame by diving by the total count

```
    counts = df.pivot_table()
    joint = counts/counts.sum().sum()
```

Chapter 4

Lecture 4: Simpson's Paradox, Probabilities

4.1 Simpson's Paradox

Conditional probability is the probability of event A when event B is true. It is given by the formula below.

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}$$

Simpson's Paradox describe the event when grouped data and un-grouped data show opposing tend, it is purely arithmetic consequence of weighted averages. Which often occur when there is a hidden confound that influence the results, or when many conditions is applied to obtain a probability.

4.2 Joining

Join is a SQL term (merge in pandas), it is appropriate when we have two source of information about same individuals linked by a common column. (Often times it is the id of the individual). There are different types of joins.

1. inner: It keeps only matching keys
2. outer: That keep all keys in both DataFrame
3. left: keep all keys in the left hand side DataFrame whether or not they are in the right DataFrame
4. right: keep all keys in the right DataFrame whether or not they are in the left DataFrame.

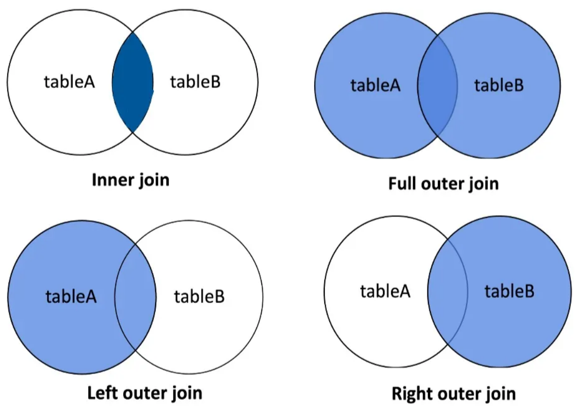


Figure 4.1: Different types of joins

To merge two DataFrames together in python

```
df1.merge(df2,
        left_on="column",
        right_on="column",
        how = "left/right/inner/outer")
```

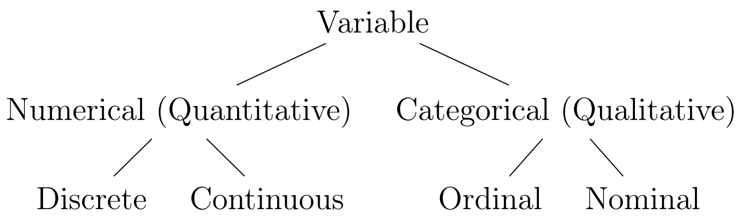
When merging if column have same name in both left and right DataFrame, parameters suffix_x or suffix_y can be added to avoid duplicate column name.

Chapter 5

Lecture 5: Exploring and Cleaning Data

5.1 Feature Types

Features can be in numbers or in words. To distinguish between these features they can be split in to two major categories, either Numerical (Quantitative), or Categorical (Qualitative).



1. Discrete Quantitative: Whole integer numbers
2. Continuous Quantitative: float point numbers
3. Ordinal: string with inherent ordering, such as t-shirt size
4. Nominal: string with no ordering, such as different class names

Note that although date is (sometimes) represented in string format, but is can be Continuous or Discrete because of the ability to perform numerical operation between two timestamps. When unsure of variable feature it is important to first ask the question if arithmetic operation be applied to the variable.

5.2 Data Cleaning

Data cleaning is the process of taking the original raw format data and transform into desired data types. It is important to check the scope of the data if it matches with the understanding of general population, and the the measurements and values are reasonable. At the same time pull out features that might be useful in a future analysis, and to check that their relationship as in agreement.

To obtain the information of a DataFrame, including memory, number of non-null values and data types

```
df.info()
```

To check for duplicated values, this returns if there is a duplicate (True/False) and is often helpful in query being a boolean series

```
df.duplicated(subset=["columns"])
```

To check for null values

```
df["column"].isna()
```

There's often multiple steps that needs to be taken to clean. To organize everything together, a pipeline can be used to chain different steps to result in the final DataFrame results. See the example below:

```
def double(df):
    return df * 2
def add_ten(df):
    return df + 10

clean_df = (pd.read_csv("path")
            .pipe(double)
            .pipe(add_ten))
```

For data cleaning and transformations with timestamps, it is uniquely accessed with the pandas Timestamp object. A date format need to be defined first, then converted to pandas timestamp.

```
date_format = "%Y-%m-%d" #or other formats "%d.%m.%Y"
pd.to_datetime(df["column"], format = date_format)
```

Panda Timestamp object have a .dt accessor for the properties of timestamps, such as .dt.day, or .dt.day_of_week.

Lastly, modifying the structure of the data adjust granularity during aggregation.

Chapter 6

Lecture 6: Hypothesis Testing

6.1 General Hypothesis Testing Setup

From the assumption that random samples look like the access frame that they were sampled from enables statistical inferences. It is true that the random samples is like access frame, but the access frame many not be from the general population.
From the random samples we have access to the test statistic $\hat{\theta}$. We wanted to know if the true parameter θ is equal to $\hat{\theta}$. The assumption is that $x_1, x_2, \ldots, x_n \sim D(\theta)$. Under this assumption the population parameter θ encodes information about population we want to infer.
Null hypothesis H_0 a statement about value of θ under the most conservative assumptions (baseline).
Alternative Hypothesis H_a is a statement about value of θ you want to show evidence in favor of.
Sample Statistics $\hat{\theta}$ is a function of data, use to estimate θ, this is generated from the sample data.
Test Statistics $\hat{T} = \hat{T}(x_1, x_2, \ldots, x_n)$, a function of data that we use to decide between H_0 and H_a.
To test the null hypothesis, a null distribution which is a distribution of the test statistic under the assumption that the null hypothesis is true is generated. Then where $\hat{\theta}$ is on the distribution and the area under it is used to decide between favoring H_o or H_a.
P-Value is the probability of seeing a result at least as extreme as the observed, under the null hypothesis. When p < 0.05 reject the null, otherwise failed to reject the null. Note that we will never accept the null hypothesis.
Empirical Distributions of Test Statistics: is generated with simulations, with greater than 10,000 number of repletion empirical distribution of test statistic will look similar to the true probability distribution of test statistic.

6.2 Generating the Empirical Distributions

To generate the null hypothesis, two probability is needed. $p = x$ the probability of of a positive event, $p = 1 - x$ the probability of a failure.

```
np.random.multinomial(count, [p,q])
```

```python
np.random.multinomial(count, [p,q], size = 100_000) #Can use
    this to adjust for how many samples to generate.
```

When putting this situation under the assumption: tossing a coin.
If test statistic is large, this means that there were many more heads than expected, or many fewer heads than expected.
If test statistic is small this means that the number of head was close to expected.
A good test statistic is one that large value correspond to one hypothesis and small value to another.

6.3 Test Statistics: Total Varation Distance (TVD)

Total variation distance (TVD) is a test statistic that describes the distance between two categorical distribution.
If A $= [a_1, a_2, \ldots, a_k]$ and B $= [b_1, b_2, \ldots, b_k]$. Both categorical distribution then TVD between A and B is defined as:

$$\text{TVD}(A, B) = \frac{1}{2}\sum_{i=1}^{k} |a_1 - b_i|$$

TVD is used to test if sample came from population:
null: Sample is random
alt: Sample is not random
To compute TVD without using a for-loop

```python
result = np.random.multinomial()
np.sum(np.abs(result - df["column"].to_numpy()), axis=1)/2
```

This assess whether an "observed sample" was drawn randomly from categorical distribution. TVD measures the distance between two categorical distribution, and to compute the TVD random samples form the population should be used to obtain general distribution, and compare with empirical distribution of TVDs.

6.4 Test Statistics: Kolmogorov-Smirnov (KS)

When performing permutation test to see of two groups came from the same distribution, one straight forward test statistics would be the difference in means. However, consider a multinational distribution with a normal distribution centered right in the middle, both groups will have similar mean thus the test statistic is not significant to tell the difference in distribution. Kolmogorov-Smirnov (KS test statistic) measures similarity between two distributions. It is defined in terms of the cumulative distribution function (CDF) defined as:

$$F_x(x) = \imath(x \leq x)$$

If $f(x)$ is a distribution then CDF $F(x)$ is the proportion of values in distribution f that are less than or equal to X. This is basically taking the integral of the distribution. The KS test statistic finds the largest difference between two CDFs.

6.5 Hypothesis Test Steps

1. pose yes/no hypothesis
2. decide on a valid test statistic help different between affirm/reject the hypothesis
3. create probability model for the data generating process that reflects the baseline
4. simulate data generating process using probability model
5. assess if observation is consistent by computing p-value

To test for permutation test, generate new data by shuffling group labels.

```
np.random.permuatiion(df["column"])
```

Chapter 7

Lecture 7: Missingness Mechanisms

The data that are collected only an approximate explanation of the "True" behavior. The confounds during data collection, and the access frame (granularity) contribute to biases when generating the data. Sometimes, the random (or stratified) sampling techniques of a data can be biased thus not able to generalized to the true population trends. These all causing the data that are collected to be imperfect. However, during the data collection data can be missing, thus resulting in NaNs in the database that we get access to. Although, it is possible to drop the NaNs during when doing feature extraction. But it decreases the amount of data and with less data it is harder to fit a model that generalizes to the true population trend. Thus, we wanted to analysis the missingness within the data and be able to impute it to keep the amount of data maximized.

7.1 Missing By Design (MD)

Missing by design (MD) describes the missingness that is rooted in the data collection process. Where the designers intentionally decide to not collect data under that feature.

7.2 Not Missing At Random (NMAR)

Not missing at random (NMAR) is the probability that a value is missing depend on the actual missing value itself. Some examples are surveying for people's income, and some may not want to answer income because their income in high. This missingness is missing because of the type of variable collected, and it is "non-ignorable". Meaning we cannot ignore the fact that the data missing is the data in and of itself.
Let φ be other factors that influence the missingness.

$$\mathbb{P}(\text{data present}|Y_{obs}, Y_{miss}, \varphi)$$

7.3 Missing At Random (MAR)

Missing at random (MAR) is the probability that a value is missing depends on other features but not the actual missing value itself. If a column is MAR, when conditioned on some set of other conditions this then transforms the missingness to be MCAR.

$$\mathbb{P}(\text{data present}|Y_{obs}, Y_{miss}, \varphi) = \mathbb{P}(\text{data present}|Y_{obs}, \varphi)$$

7.4 Missing Completely At Random (MCAR)

Missing completely at random (MCAR) is the probability that a value is missing that is completely independent of other columns, and the actual missing value. Some examples are sensor that is installed to sense the environmental changes, however the sensors are turned off (causing record to be NaNs), this status of the sensors shutdown has no pattern. This has no correlations with other columns or within itself.

$$\mathbb{P}(\text{data present}|Y_{obs}, Y_{miss}, \varphi) = \mathbb{P}(\text{data present}|\varphi)$$

7.5 Deciding on Missingness

To determine the missingness of the data, here are some questions to ask:

1. Missing by Design (MD): It it a data collection issue?
2. Not missing at random (NMAR): Is there a good reason why the missingness depends on the values themselves?
3. Missing at Random (MAR): Are other columns telling me anything about the likelihood that a value is missing?
4. Missing completely at random (MCAR): is the missingness must not depend on other columns or the values themselves

Chapter 8

Lecture 8: Imputation

When modeling from data, the missing values that exits does not provide useful guidelines in being able to extract the underlying true trend of the world, thus they need to be either removed or imputed. Imputation is the act of filling in missing data with plausible values. An approach would be listwise deletion which is dropping entire rows that contain missing values and this may delete good columns that are useful to tell the true trend. At the same time it also cost money in data collection. So the best way is to impute the data based on the goal and the missingness type.

8.1 Mean Imputation

Mean imputation is the act of filling in missing values in a column with mean of observed values in the column. This preserves the mean of the observed data, but decreases the variance of the data for all missingness types. This can also create biased estimate of the true mean if the data is not MCAR. Some way of dealing with this bias are to do within-group conditional mean imputation. Which works in groupbys on other columns.

8.2 Probabilistic Imputation

Probabilistic imputation is imputing the missing values using distribution of known data. This amples with in group and choices are in the group. However, in sampling from the known distribution it doesn't capture more extreme values, such as any value small than the current minimum of the data, or any value that is greater than the maximum of the data. It will only include the current values that exist within the dataset. This process is generated through random sampling and this randomness cause the result to be different each time.

8.3 Multiple Imputation

Multiple imputation is a combination of both imputation methods. First, with using probabilistic procedure to create m imputed versions of the data. For all versions of the imputed datasets the observed data entries are identical, and the only difference is within the imputed values. This difference reflect the uncertainty about what values to impute. Then based on these m versions of data, the parameter estimates (mean, median, standard deviations, etc) is computed. Finally, a single parameter estimate can be pooled from the m parameters. All m parameter estimates generate the probability of the true parameter estimate being X. The m parameter generates a histogram, and depicts what the true parameter is more likely going to be.

Chapter 9

Lecture 9: HTTP

HTTP stands for hyper text transfer protocol, it is a request-response portal. A request is made by the client, while a response is returned by the server. With in the response the website's information is return and is often times useful to obtain data through this format. A request can be made using curl in the command line or python.
The command-line tool for sending HTTP request is curl. The $-v$ flag is short for verbose. This returns the webpage in HTML format.

```
curl -v https://....
```

We can also query for a webpage using query string in a website url. Following the $?q =$ is the actual search filed.

```
https://www.google.com/search?q=ucsd+dsc+80+hard
```

9.1 Request in Python

Since a webpage information is returned, it would be helpful if it can be stored and later parsed into structured data. Thus these request can also be made in python by utilizing the library request. The *res* in the code below creates an object that contains a webpage's information. If simply printing the *res* object it will we the response status.

```python
import requests
res = request.get("url")
```

There are three different kinds of HTML codes.

1. Response 200: No issues
2. Response 400: 404 (representing page not found)
3. Response 500: Representing internal sever error, which is error on the server's side

Instead of manually checking if the response is successful, it can also be checked through the attribute *.ok*. This will return True if it is a response 200, and false otherwise.

```
res.ok
```

To get the HTML information contained behind the website. This will return raw format of the webpage.

```
res.text
from IPython.display import HTML #This is another library that
    can be used to display the text in more readable format
HTML(res.text)
```

The data format types returned from the request can be of many types. If it is accessible as text, it is then a HTML (webpage), but the response can also be JSON (through API), or POST (JSON), XML(Extensible Markup Language) an also be a format.

9.2 JSON

JavaScript object notation (JSON) is a lightweight format for storing and transferring data. In python JSON resembles python dictionaries. JSON need to be parsed, and not evalued as dictionaries. There are many different JSON data type:

Type	Description
String	Anything inside double quotes
Number	Any number
Boolean	true of false (all lowercase in JSON)
Null	JSON empty null (lowercase)
Array	Like python lists
Object	Collection of key-value pairs, like dictionaries, keys, strings, values can be anything

To check the python code if it runs successfully without error, the *eval*() function can be used to run the code before hand. Note to be careful when using *eval*() as it actually run python code. If there's an error in the code such as delete all, it will then delete everything in the real environment. So the *eval*() function should never be used on unfamiliar data. See example below.

```
x = 10
y = 20
expression = "x + y"
eval(expression) #This will output 30
```

9.3 API

Application programming interface (API) is a service that makes data directly available to the user in a convenient fashion. The advantages of using an API is that data are usually clean and up-to-date. It is also an representation of approval from the data provider that gives consent with users using their data. The data providers plan and regulate data usage. However, APIs don't always exist for the data is wanted. API endpoint refers to a URL of the data source that user wants to make request to. For example the Reddit API, */comment* or */hot* add end point name to the base URL of the API. From the API response it might not always be in HTML. If the response is returned in JSON:

```
res.keys() #pulls out relevent JSON keys
res.json() #returns in JSON spaced
```

Chapter 10

Lecture 10: Web Scraping

10.1 HTML Tags

Hypertext Markup Language (HTML) is the standard markup language for creating webpages. The internal representation of HTML document is under DOM (Ducument Object Model), which is a hierarchical tree structure. HTML element is an object in the DOM such as paragraph header, or title. HTML tags are markers that denote the start and end of element. Some common tags are:

Element	**Description**
<html>	document
<head>	the header
<body>	the body
<div>	a logical division of the document
<span>	an inline logical division
<p>	a paragraph
<a>	an anchor (or hyperlink)
<h1>, <h2>	header(s)
<img>	an image

Tag can have attributes, such as description of what is in the tag (alt). For the example below, it is the description that shows up when the image fails to render.

```
<img scr="_____.png" alt="description">
```

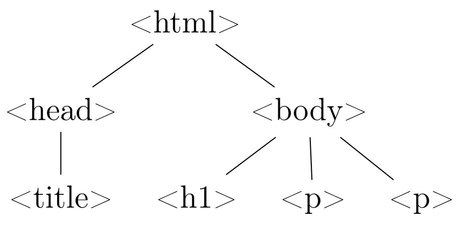
Hyperlinks have href, which is the hyperlink reference.

```
<a href="urlPath"> description </a> description
```

The <div> tag defines a division or a "section" of an HTML document, it is similar to a cell in jupyter notebook. The <div> tag contains attributes.

10.2 Document Trees

As mentioned HTML has a DOM structured, and can be represented as trees.



Request objects return raw HTML, and is often hard to parse. With the help of a python library Beautiful soup (HTML parser) traversing through the descendants of the tree structure is much easier. Descendants are attributes, and this traversing is through a depth-first traversal.

```python
import bs4
soup = bs4.BeautifulSoup(res) #Note res is the response object
```

To find the first instance of a tag. Attributes can also be passed in

```python
soup.find(tag)
soup.find("tag", attrs={key:val})
```

To find all instance of a tag

```python
soup.find_all(tag)
```

To pull out the text information within the tag.

```python
soup.find("tag").text
```

To find all attributes associated with the tag

```python
soup.find(tag).attrs
```

To get the content of the tag

```python
soup.find(tag).get("attr")
```

Chapter 11

Lecture 11: Regular Expressions

Regular Expressions (Regex) is a sequence of characters used to match patterns in strings. It is powerful and widely used, but when expressions gets long it becomes hard to read. Literals in regex represent character that has no special meaning, and special characters need to be escaped before being able to use them. For example: `\(\d{3}\)\d{3}-\d{4}`, this matches (xxx)xxx-xxxx, where x are digits that may resemble phone numbers.

11.1 Regex Operations

Concatenation in Regex is matching the string entirely. Consider the expression `AABAAB` it will only match "AABAAB" and does not match any other string.

The or operation in regex is similar to the one in python. For example: `AA|BAAB` only matches "AA", and "BAAB", and not other strings.

To match multiple instances of an element, the exact number does not need to be hard coded. Closure in regex matches if there is 0 or more occurrences of the element. For example: `AB*A` it will match the patters "AA", and "ABBBBBA" but it fails to match "AB" or "ABABA".

Regex has its own order of operations. First it considers parentheses, then closures, followed by concatenation and end with or. For example: `A(A|B)AAB`, this matches "AAAAB", "ABAAB" but not any other string.

There are special characters in Regex that enhances the flexibility and power of pattern matching. A common one is the wildcard (`.`), which has the ability to match anything. For example: `.U.U.U` matches "AUBUCU" but not "AUBBUCCCU".

To match characters, character classes can be specified. For example: `[A-Za-z][a-z]*` matches "Word", "capital" but not "camlCase" or "4illegal". Character classes can also be negated. For exampe: `[^a-z]+` matches "KING666", "177!!" but not "porch", "billy.edu".

To match at least one occurances of a character. For example: `bi(ll)+y` matches "billy", "billlly" but not "biy", "bily".

To match between i and j occurrences. For example: `m[aeiou]{1,2}m` matches "mem","maam","miem" but not "mm","mooom","meme"

Sometimes, special characters exist in the string that we may want to match. Thus, we need escape characters to be able to match them normally. Note to use the escape character normally, it needs to be escaped again. For example: `ucsd\.edu` match "ucsd.edu" but not "ucsd!edu".

To mark the beginning of the line in regex use `^`. For example: `^ark` matches "ark two", "ark one" but does not match "dark".

To mark the end of the line in regex use `$`. For example: `ark$` matches "dark", "ark and ark" but not "ark one".

To ask if there exists zero or one is `?` in regex. For example: `cat?` matches "ca" and "cat" but does not match "cart". Though it will partly match "ca" portion in "cart".

To not specify character classes there are built in ones.

1. `\d`: digits
2. `\w`: alphanumeric [A-Z][a-z][0-9](and space)
3. `\s`: whitespace
4. `\b`: word boundary (Example: `\bword\b`)

Although regex is very powerful in matching strings but it has it's own limitations. Such as it's complex syntax and the difficulty to debug. Regex should not be used to count the same number of instance, or parsing anything with complex structures. Such as HTML text structure.

Chapter 12

Lecture 12: Text Features

12.1 Count Matrix

In modeling calcuations is usually used, however when text is one of the features before it can be used in calcuation it needs to be transformed in to numbers that computers will be able to understand. The importance is pulling out text features that can be used to analyze and understand textual data. One common way is to create a count matrix of the word frequencies. The dot product is then taken between the word counts.
If $\vec{a} = [a_1, a_2, \ldots, a_n]^T$ and $\vec{b} = [b_1, b_2, \ldots, b_n]^T$ are two vectors their dot product $\vec{a} \cdot \vec{b}$ is then defined as:

$$\vec{a} \cdot \vec{b} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

The geometric interpretation of the dot product is that if $|\vec{a}|$ and $|\vec{b}|$ are the L_2 norms of $\vec{a}$ and $\vec{b}$ and θ is angle between $\vec{a}$ and $\vec{b}$ then:

$$\vec{a} \cdot \vec{b} = |\vec{a}||\vec{b}| \cos\theta$$

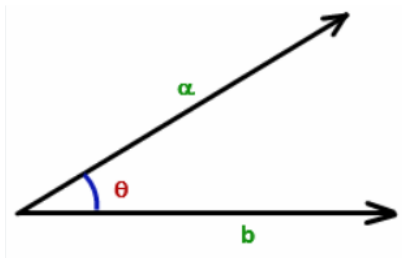


Figure 12.1: Dot Product Visual Representation

The key idea is that the more similar two unit vectors are, the larger their dot product is.

12.2 Bag of Words

Another way of extracting text features is using bag of words. This represents texts as vectors of word counts. Bag of words model defines a vector space in $\mathbb{R}^{\text{number of unique words}}$. It is named bag of words because it doesn't consider any ordering.

Cosine Similarity: is a measure of similarity between two word vector. This compute their normalized dot product. The denominator is the number of words in the element.

$$\cos\theta = \frac{\vec{a}\cdot\vec{b}}{|\vec{a}||\vec{b}|}$$

If the cosine similarity is large then the two word vectors are similar. It is important to normalize by the length of the vectors this accounts for the fact that text with more words have artificially high similarities with other texts.

Cosine Distance can be computed using cosine similarity. If the distance is small, then the two word vector are similar.

$$\text{dist}(\vec{a},\vec{b}) = 1 - \cos\theta$$

Some things to be caution of when using the bag of words model are that ordering doesn't matter. Neither does the model consider the meaning of words, it treats all words with the same level of importance.

12.3 TF-IDF

The goal of TF-IDF is to find the word that best summarizes a document. TF stands for term frequency, and IDF stands for inverse document frequency.

Term Frequency(TF): a word t in document d, denoted $tf(t,d)$, is the proportion of words in document d that are equal to t.

$$tf(t,d) = \frac{\text{number of occurrences of t in d}}{\text{total number of words in d}}$$

Inverse Document Frequency (IDF): is a word t in s set of documents $d_1, d_2, \ldots$. if a word appear in every document it is probably not a good summary of any one document. $idf(t)$ is a rarity factor of t across documents larger $idf(t)$ more rare t is.

$$idf(t) = \log\left(\frac{\text{total number of documents}}{\text{number of documents in which t appears}}\right)$$

1. if $idf(t)$ is large, t rarely found in document
2. if $idf(t)$ is small, t commonly found in documents.

Since the goal is to find a word that best summarize document d. The factors below should be considered:

1. if $tf(t,d)$ is small, then t doesn't occur very often in d, so t can't be good summary.
2. if $idf(t)$ is small, then t occur often amongst all document and so it is not a good summary of any one document.
3. if $tf(t,d)$ and $idf(t)$ are both large then t occurs often in d but rarely overall. This makes t a good summary of document d.

With these factors we can define TF-IDF as:

$$tfidf(t,d) = tf(t,d) \cdot idf(t)$$

$$= \left(\frac{\text{number of occurrences of t in d}}{\text{total number of words in d}}\right) \cdot \log\left(\frac{\text{total number of documents}}{\text{number of documents in which t appears}}\right)$$

If $tfidf(t,d)$ is large then t is a good summary of d because t occurs often in d but rarely across all document. TF-IDF is a heuristic and has no probability justification. To know if it is truly large or one particular word t, it needs to be compared with several different words.
Be caution $tfodf(t,d)$ can be 0:

1. if t appears in every document then:

$$idf(t) = \log\left(\frac{\text{number of document}}{\text{number of document}}\right) = \log(1) = 0$$

2. if t does not appear in document d then:

$$tf(t,d) = \frac{0}{len(d)} = 0$$

Chapter 13

Lecture 13: Linear Regression

13.1 Modeling

The data generating process resembles a real-word phenomena that we are interested in studying. Thus a model is a theory about the data generating process. To fit a model is having a model learn from a particular set of observations (training data). The goals of modeling:

1. Make accurate predictions regarding unseen data from data generating process

2. Make inferences about the structure of the data generating process. (i.e. being able to understand complex phenomena)

13.2 Features

A feature is a measurable property of a phenomenon being observed. In terms of DataFrames features are columns. We can also use existing columns to create features that we need. It is important to understand that "All models are wrong but some are useful". Thus, there needs to be a way to evaluate the qualify of predictions, how good is this constant prediction at predicting the information in the training data.

13.3 Evaluation

Mean Square Error: if y_i represent the ith actual value and $H(x_i)$ represent the ith predicted value then.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - H(x_i))^2$$

Root Mean Squared Error: is basically taking the square root of the MSE

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - H(x_i))^2}$$

For a simple model: $y = w_0 + w_1 x_1$, $H(x_i) = w_0 + w_1 x_i$. The lower the MSE the better the model fits training data.
Let: $H(x_i) = w_0^* + w_1^* x_i$ to be the parameter that minimizes the MSE (line of best fit)

To perform this process in sklearn:

```
from sklearn.linear_model import LinearRegression
model = LinearRegression() #init a model
model.fit(x, y)
model.intercept_ #This provides the w_0 term
model.coef_ #This provides the w_1 ... w_n term dependent on
   the number of columns in x
```

Residual Plots are used to evaluate the goodness-of-fit of a regression model. It is calculated by $y - H(x)$. On the residual plot, the x-axis is the predicted values $H(x)$, and y-axis is the residuals $y - H(x)$. This can be used in a visualization aspect to know when models are more "wrong" than other times. It resembles the proportion og vatiance in y that the linear model explains. This score can also be found in sklearn.

```
model.score #This provides the R^2
```

R^2 is the coefficient of determination which is a measure of the quality of a linear fit. Calculated by: Let $H(x_i)$ be the predicted y values, y_i be the actual y values.

$$R^2 = \frac{var(H(x_i)}{var(y_i)}$$

or

$$R^2 = correlation(pred, actual)^2$$

R^2 is between 0-1, the closer to 1 the better.

Chapter 14

Lecture 14: Feature Engineering, Pipelines

14.1 Feature Engineering

Feature Engineering is the act of finding transformations that transform data into effective quantitative variables. A feature function ϕ is a mapping from raw data to d-dimensional space:

$$\phi : \text{raw data} \longrightarrow \mathbb{R}^d$$

If two observations x_i and x_j are "similar" in the raw data space the $\phi(x_i)$ and $\phi(x_j)$ should also be "similar". A good choice of features depends on many factors such as the kind of data (quantitative, ordinal, nominal), relationships and associations(s) being modeled, as well as the model type.

One Hot Encoding is the transformation that turns a categorical feature into several binary features. If column has N unique values $A_1, A_2, \ldots, A_n$ for each unique value A_i, define following feature function:

$$\phi_i(x) = \begin{cases} 1 & \text{if } x > A_i \\ 0 & \text{if } x \neq A_i \end{cases}$$

This in sklearn:

```python
from sklearn.preprocessing import OneHotEncoder
```

It also important to consider dropping features after feature transformation. Drop a feature if it does not contain information associated with prediction task or when feature is not available at prediction time. During one-hot encoding, we can specify to drop a column. The accuracy of the model is not affected by dropping this column, while if this column is included it messes up the interpretation of coefficients of the model. To be able to make sense of the coefficients, the user should consider if they want to drop the column or not.

Transformation is to transform ordinal encoding that maps ordinal values to the positive integers in a way that preserves order. This transformation preserves "distance" between ratings. Transformation can also be quantitative known as quantitative scaling. To transform non-linear data to linear, can reference the Tukey Mosteller Bulge Diagram and pick the transformation.

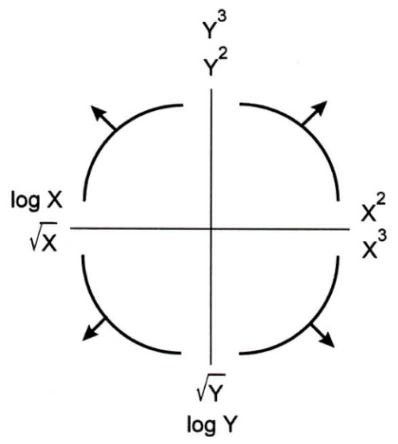


Figure 14.1: Tukey Mosteller Bulge Diagram

Transformers in Sklearn: transformers classes take in "raw" data and output "processed" data, used for crearing features. The input can be a multi-dimensional numpy array, while the output is numpy array.

Transformer Binarizer allows to map quantitative squence to a sequence of 1s and 0s.

```
binar = Binarizer(thresh) #set x=1 if x>thresh, else 0
feature = binar.transform(data)
```

Transformer StdScaler standardizes data using mean and std of the data. This requires knowledge (mean and sd) of the dataset before transforming. Thus, we can no longer use $.transform$ but instead $.fit_transform$. Let $\bar{x}$ be the mean of the data, s be the SD of the data.

$$E(x_i) = \frac{x_i - \bar{x}}{s}$$

```
stdscaler = StandardScaler()
stdscaler.fit(x) #this gains info on mean and sd of x
feature  = stdscaler.transform(x_new)
stdscaler.fit_transform(x) #or lines 2-3 can be computed with
```

Summary of the modelig process:

1. create (engineer) features to best reflect the "meaning" behind data.
2. choose a model that is appropriate to capture the relationships between features (x) and the target/response (y)
3. select a loss function and fit the model (determine W^*)
4. evaluate model (RMSE or R^2)

14.2 Pipelines

In sklearn to instantiate a pipeline, must provide a list with zero or more transformers followed by a single model. All steps must have fit methods, and all but the last last must have transform method. Once a pipeline is instantiated fit all steps, using single call to the fit method. ($pl.fit(x, y)$) To make new predictions using raw, un-transformed data *pl.predict*. The list that we provide in pipeline must be a list of tuples, first element being the name (we choose) for the step, second element is a transformer or estimator instance. For a more sophisticated pipeline using columnTransformer, a third element can be added to the tuple being a list of relevant column names.

```
pl = Pipeline([
    ("name", transeform()),
    ...
])
pl.fit(x, y)
```

Chapter 15

Lecture 15: Multicollinearity, Generalization, Cross-Validation

15.1 Multicollinearity

Multicollinearity occurs when features in regression model are highly correlated with one another. It is when a feature can be predicted using linear combination of other features. They having no meaning but are present in gestures and coefficients of the model, they also don't impact model's prediction. When a feature is redundant, it should be manually removed.

15.2 Generalization

With large amounts of data a model can predict very well on training data. To be able to generalize it to unseen data, consider the bias and variance trade off. Bias is the expected deviation between a predicted value and actual value. For a given x_i, how far $H(x_i)$ from true y_i on Average. High bias is a sign of underfitting (model is too basic). The variance of a model's prediction for a given x_i, is the variance of $H(x_i)$ across all datasets. High variance is a sign of overfitting. The observation variance is variance due to random noise in the process we are trying to model.

$$R(x) = bias^2 + variance + noise$$

Bias: how close model gets to real world model
Variance: how different model can change with slight change to a data point.
In order to assess how a model generalizes to unseen data a train-test split is often used to split the dataset in to two part. Where the train holds the majority of the data while the test holds a lower percentage. After the model is trained on the train set, it's accuracy is then evaluated with the test set to see how the model generalized to unseen data.

```
sklearn.model_selection.train_test_split
```

A model with good performance is where it's training error is similar to test error. Usually with hyperparameter tuning that can lead to the best test set performance.

15.3 Cross-Validation

There will always be a trade off between the bias and variance when fitting a model that contains parameters. Parameters are what defines the relationship between variables in a model, and it is usually learned from data. Such as the weights to linear regression. In machine learning there is also the aspect of hyperparameters where it's the parameter that we get to choose before our model is fitted to the data. To find the best set of hyperparameters that minimizes the test error a k-fold cross-validation is usually performed. It shuffles the train dataset randomly and split it into K disjoint groups. Then for each hyperparameter from each unique group it holds that unique group to be the "validation set", and let all other groups be the training set. It then trains a model using the selected hyperparameter. Then compute the average validation score (ex RMSE) for the particular hyperparameter. This can all be done through sklearn.

```
sklearn.model_selection.GridSearchCV
```

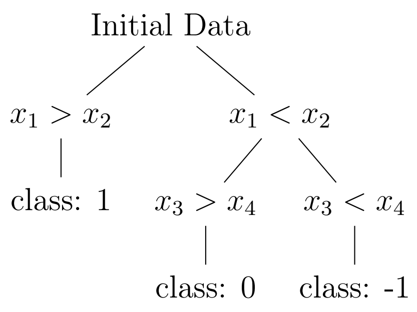
The GridSearchCV takes in un-fit instance of an estimator, and a dictionary of hyperparameter values to try. It then performs k-fold cross-validation to find the combination of hyperparameters with the best average validation performance.

Chapter 16

Lecture 16: Decision Trees, Random Forest

16.1 Decision Trees

A decision tree model in sklearn is one where the internal nodes of the tree check feature values, and the leaf nodes specify class. Decision trees can work with categorical data without one hot encoding. Thus, this makes the predictions interpretable and the decision boundaries can be arbitrarily complicated, but it also allows for multi-class classification. With in a decision tree model it's fat to fit and predict while also robust to irrelevant features. When a feature is lineally transformed it does not affect preditions. (See example below)



To decide on the best split in a decision tree a loss function is chosen and all split are tried at a point in the tree. The split that minimizes the loss function is then kept.
When evaluating classifiers the accuracy is given by:

$$acc = \frac{\text{number of data points correct}}{\text{number of total data points}}$$

However, in decision trees there are different methods of evaluating the split performance. One being entropy defined as a node is the average surprise over all classes.

$$L(x, y) = -\sum_c P_c \log_2 P_c$$

P_c is defined as the proportion of points with the label and the surprise is $-\log_2 P_c$
A decision tree is trained by recursively picking the best split, and by default there

is no "maximum depth". Thus, decision trees have the tendency to overfit due to nonlinearity, where it ask enough questions to effectively memorize the correct response. This cause decision trees to result in high variance as the tree always overfits.

16.2 Random Forest

To combat the tendency that decision tree are likely to overfit we can use an ensemble method Random Forest. Where a bunch of decision tree are trained and have them vote on a final prediction. We can bootstrap the training data also known as bagging (ensemble learning) or only use a subset of features. At each split take a random subset of m features, the rule of thumb here is that $m \approx \sqrt{d}$. When $m = d$ it is just like training a decision tree, when $m = 1$ it has high depth, same bias, and higher variance than the original decision tree.

Chapter 17

Lecture 17: Logistic Regression, Classifier Evaluation

17.1 Logistic Regression

Decision trees are not the only models that works in classification. Logistic regression is a linear classification model that uses linear regression. It models the probability of belonging to a class, given a feature vector.

$$P(y = 1|\vec{x}) = \sigma(w_0 + w_1x^{(1)} + w_2x^{(2)} + \cdots + w_dx^{(d)})$$

The sigmoid function $\sigma(t)$ is defined as:

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

The outputs to the sigmoid functions are between 0 and 1, a threshold is picked to determine the cutoff of predicting a class. Based on the setting of making predictions a class can be punished more than another to tune for more predictions of one class.

17.2 Classifier Evaluation

Although the accuracy is an important step in analysing classifier performance but in some instances we would want to see the types of error that this classifier is yielding. For example, in a medical setting we would want someone to be diagnosed with a disease even if there is uncertainty about it. Since it is better to perform more check for diagnosis then to have someone that does have the disease to walk away thinking that they don't. In a medical setting it is important to provide treatment in early phases rather than later. A confusion matrix is important in binary classification that contain information about a model's performance.

1. **True Positive (TP)**: The true class is positive and the predictor correctly predicts the positive class.

2. **False Positive (FP)**: The true class is negative but the predictor incorrectly predicts the positive class.

3. **False Negative (FN)**: The true class is positive but the predictor incorrectly predicts the negative class.

4. **True Negative (TN)**: The true class is negative and the predictor correctly predicts the negative class.

	Predicted Positive	Predicted Negative
Actual Positive	True Positives (TP)	False Negatives (FN)
Actual Negative	False Positives (FP)	True Negatives (TN)

The accuracy of a classifier model is given by:

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FN + FP}$$

We can also assess the recall of a binary classifier, defined as the proportion of actually positive instances that are correctly classified.

$$\text{recall} = \frac{TP}{TP + FN}$$

Another performance metrics is the precision of a binary classifier model, defined as the proportion of predicted positive instances that are correctly classified.

$$\text{precision} = \frac{TP}{TP + FP}$$

Combining precision PR and recall RE is the F1-score, which also measures the overall accuracy of the model:

$$\text{F1-score} = \text{harmonic mean} = 2 \cdot \frac{PR \cdot RE}{PR + RE}$$

Although the accuracy is important in evaluating classifier performance, however, it can be misleading with class in balance and does not provide any insights about where the classifier makes mistakes at making predictions.
In the medical setting, we would want a smaller number of False Negatives. Thus, when tuning for hypterparameters we can punish a model more for making False Negative predictions to increase the amount of Positive predictions overall.

Chapter 18

Lecture 18: Conclusion

This concludes the course DSC 80. The topics covered in the course is from a wide range of topics. Starting in basic operations in Pandas to Web Scraping and ending with modeling. The topics from this course are an excellent introduction to develop your own Data Science Projects. For a more in-depth description about the topics covered in this course visit the textbook, Principles and Techniques of Data Science[1].

[1]Authors: Sam Lau, Joey Gonzalez, and Deb Nolan
https://learningds.org/intro.html