Machine Learning

* Machine learning: Taking data and turning it into insights
* Analyzing examples from past to predict future
* Python libraries: pandas, numpy, matplotlib, scikit-learn
  + pandas: Reading and manipulating data
  + numpy: Computations of numerical data
  + matplotlib: Graphing data
  + scikit-learn: For ML models
* Supervised and unsupervised learning
  + Supervised
    - Have known target based on past data
    - There are classification and regression problems
      * Regression (prediction): Predicting numerical value
      * Classification: Predicting class of something
  + Unsupervised: No past data
  + Hybrid: mix of supervised and unsupervised
    - unsupervised is used to find a target output, then supervised is used to test the model
  + reinforcement learning
    - rewards when the model is right
    - demerits when the model is incorrect
* Popular techniques
  + Logistic Regression
  + Decision Trees
  + Random Forests
  + Neural Networks

Statistics

* Data -> Statistics -> Decisions

Data science machine learning steps

* identify required data
* pre-process data (normalization of data) -> unit standard distribution, zero mean
* create training and test set -> takes up to 80% time
* select appropriate algorithms (linear regression, logistic regression, etc) -> takes up to 85% time
* train and build the model
* evaluate with test data -> takes up to 100% time

Dividing data

* Data is divided to 2 sets: Training and testing
* Ex: 80% data for training, 20% data for testing when you are confident about the model. (70:30 split when you are unsure about the model)

Overfitting data

* Model performs well on training data, not on testing data
* Need to gather more data to reduce noise in training data to use regularization

Underfitting data

* Model doesn’t perform well on training and testing data

Average

* Mean: Standard average, Layman’s average
* Median: Value in middle, 50th percentile
* Mean and median are called measures of central tendency: where the center of our data is

Standard deviation and variance

* Measures how dispersed out data is
* Root of variance = Standard deviation
* Low standard deviation: values tend to be close to mean
* High standard deviation: the opposite
* players = [180, 172, 178, 185, 190, 195, 192, 200, 210, 190]

mean = sum(players)/len(players)

stdvar = (sum((v-mean)\*\*2 for v in players)/len(players))\*\*0.5

low, high = mean-stdvar, mean+stdvar

count = len([v for v in players if low <= v <=

 high])

print(count)

numpy: Numerical Python

* numpy functions: mean, median, percentile, std, var
* lists are used to store data
* Use np.array() to create numpy array
* numpy arrays : nd arrays -> n-dimentional arrays -> can have multiple dimensions
* x = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print(x[1][2])

* o/p: 2nd row 3rd column element
* ndim: no of dimensions of the array
* size: total no of elements of array
* shape: tuple of integers that indicate the number of elements stored along each dimension of the array
* x = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print(x.ndim) # 2

print(x.size) # 9

print(x.shape) # (3, 3)

* Numpy arrays are faster and more compact than lists
* import numpy as np

x = np.array([1, 2, 3, 4])

print(x[0])

* numpy arrays are homogeneous: can have only one data type
* import numpy as np

data = [15, 16, 18, 19, 22, 24, 29, 30, 34]

print("mean:", np.mean(data))

print("median:", np.median(data))

print("50th percentile (median):", np.percentile(data, 50))

print("25th percentile:", np.percentile(data, 25))

print("75th percentile:", np.percentile(data, 75))

print("standard deviation:", np.std(data))

print("variance:", np.var(data))

* o/p:

mean: 23.0

median: 22.0

50th percentile (median): 22.0

25th percentile: 18.0

75th percentile: 29.0

standard deviation: 6.342099196813483

variance: 40.22222222222222

* np.append() -> add
* np.delete() -> remove
* np.sort() -> sort
* import numpy as np

x = np.array([2, 1, 3])

x = np.append(x, 4)

x = np.delete(x, 0)

x = np.sort(x)

print(x)

* o/p: [1 3 4]
* np.arange() -> create an array that contains a range of evenly spaced intervals (similar to python range)
* import numpy as np

x = np.arange(2, 10, 3)

print(x)

* o/p: [2 5 8]
* import numpy as np

data = np.array([1000, 2500, 1400, 1800, 900, 4200, 2200, 1900, 3500])

new\_house = int(input())

data = np.append(data, new\_house)

print(np.sort(data))

* x = np.arange(2, 8, 2)

x = np.append(x, x.size)

x = np.sort(x)

print(x[1])

* o/p: 3

Reshape

* shape: no of rows, columns in a tuple
* reshape: change the shape of array
* import numpy as np

x = np.arange(1, 7)

z = x.reshape(3, 2)

print(z)

* o/p:

[[1 2]

[3 4]

[5 6]]

* no of elements has to be same to reshape
* Reshape can take 2d array and make 1d array
* import numpy as np

x = np.array([[1, 2], [3, 4], [5, 6]])

z = x.reshape(6)

print(z)

* o/p: [1 2 3 4 5 6]
* Here, result is a flat array containing 6 elements
* Use flatten() to achieve the same thing
* import numpy as np

data = np.array([1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0])

row = int(input())

data = data.reshape(6, 5)

print(data[row])

Indexing and slicing

* np arrays can be indexed and sliced
* import numpy as np

x = np.arange(1, 10)

print(x[0:2])

print(x[5:])

print(x[:2])

print(x[-3:])

* o/p:

[1 2]

[6 7 8 9]

[1 2]

[7 8 9]

Conditions

* import numpy as np

x = np.arange(1, 10)

print(x[x<4])

* o/p: [1 2 3]
* Use & (and), | (or) to combine conditions
* import numpy as np

x = np.arange(1, 10)

print(x[(x>5) & (x%2==0)])

* o/p: [6 8]
* condition can be assigned to variable
* Ex: y = (x>5) & (x%2==0)
* import numpy as np

arr = np.arange(1, 100)

print(arr[(arr%3 == 0) & (arr%5 == 0)])

Array operations

pandas

* Table of data or pandas data object: Dataframe
* head returns first 5 rows
* import pandas as pd

df = pd.read\_csv('https://sololearn.com/uploads/files/titanic.csv')

print(df.head())

* o/p:

Survived Pclass Sex Age Siblings/Spouses Parents/Children Fare

0 0 3 male 22.0 1 0 7.2500

1 1 1 female 38.0 1 0 71.2833

2 1 3 female 26.0 0 0 7.9250

3 1 1 female 35.0 1 0 53.1000

4 0 3 male 35.0 0 0 8.0500

Summarizing data

* count: Number of rows that have value

import pandas as pd

pd.options.display.max\_columns = 6

df = pd.read\_csv('https://sololearn.com/uploads/files/titanic.csv')

print(df.describe())

* o/p:

Survived Pclass Age Siblings/Spouses Parents/Children \

count 887.000000 887.000000 887.000000 887.000000 887.000000

mean 0.385569 2.305524 29.471443 0.525366 0.383315

std 0.487004 0.836662 14.121908 1.104669 0.807466

min 0.000000 1.000000 0.420000 0.000000 0.000000

25% 0.000000 2.000000 20.250000 0.000000 0.000000

50% 0.000000 3.000000 28.000000 0.000000 0.000000

75% 1.000000 3.000000 38.000000 1.000000 0.000000

max 1.000000 3.000000 80.000000 8.000000 6.000000

Fare

count 887.00000

mean 32.30542

std 49.78204

min 0.00000

25% 7.92500

50% 14.45420

75% 31.13750

max 512.32920

Selecting single column

* import pandas as pd

df = pd.read\_csv('https://sololearn.com/uploads/files/titanic.csv')

col = df['Fare']

print(col)

* o/p:

0 7.2500

1 71.2833

2 7.9250

3 53.1000

4 8.0500

...

882 13.0000

883 30.0000

884 23.4500

885 30.0000

886 7.7500

Name: Fare, Length: 887, dtype: float64

* Here, single column is printed
* This result is called a pandas series

Selecting multiple columns

* Use double square brackets for multi columns
* import pandas as pd
* df = pd.read\_csv('https://sololearn.com/uploads/files/titanic.csv')
* small\_df = df[['Age',  'Sex', 'Survived']]
* print(small\_df.head())
* o/p:

Age Sex Survived

0 22.0 male 0

1 38.0 female 1

2 26.0 female 1

3 35.0 female 1

4 35.0 male 0

* Here, head method is used to print just 5 rows

Creating a column

* import pandas as pd

df = pd.read\_csv('https://sololearn.com/uploads/files/titanic.csv')

print(df['Sex'] == 'male')

* o/p:

0 True

1 False

2 False

3 False

4 True

...

882 True

883 False

884 False

885 True

886 True

Name: Sex, Length: 887, dtype: bool

* Here, boolean pandas series is created based on sex
* We can use this to create our new column ‘male’
* import pandas as pd

df = pd.read\_csv('https://sololearn.com/uploads/files/titanic.csv')

df['male'] = df['Sex'] == 'male'

print(df.head())

* o/p:

Survived Pclass Sex ... Parents/Children Fare male

0 0 3 male ... 0 7.2500 True

1 1 1 female ... 0 71.2833 False

2 1 3 female ... 0 7.9250 False

3 1 1 female ... 0 53.1000 False

4 0 3 male ... 0 8.0500 True

[5 rows x 8 columns]

numpy

* To manipulate lists and tables of numerical data
* List or table of data: numpy array
* numpy is used to build pandas

Converting from pandas series to numpy array

* Use values attribute
* import pandas as pd

df = pd.read\_csv('https://sololearn.com/uploads/files/titanic.csv')

print(df['Fare'].values)

* o/p:

[ 7.25 71.2833 7.925 53.1 8.05 8.4583 51.8625 21.075

11.1333 30.0708 16.7 26.55 8.05 31.275 7.8542 16.

29.125 13. 18. 7.225 26. 13. 8.0292 35.5

21.075 31.3875 7.225 263. 7.8792 7.8958 27.7208 146.5208

7.75 10.5 82.1708 52. 7.2292 8.05 18. 11.2417

9.475 21. 41.5792 7.8792 8.05 15.5 7.75 21.6792

14.1083 11.5 25.9292 69.55 13. 13. 13.8583 50.4958

9.5 11.1333 7.8958 52.5542 5. 9. 24. 7.225

9.8458 7.8958 7.8958 83.1583 26. 7.8958 10.5167 10.5

7.05 29.125 13. 30. 23.45 30. 7.75 ]

* This is a 1-dimensional array

Converting from pandas dataframe to numpy array

* import pandas as pd

df = pd.read\_csv('https://sololearn.com/uploads/files/titanic.csv')

print(df[['Pclass', 'Fare', 'Age']].values)

* o/p:

[[ 3. 7.25 22. ]

[ 1. 71.2833 38. ]

[ 3. 7.925 26. ]

...

[ 3. 23.45 7. ]

[ 1. 30. 26. ]

[ 3. 7.75 32. ]]

* This is a 2-dimensional array: 2d numpy array
* We can select single element from numpy array
* import pandas as pd

df = pd.read\_csv('https://sololearn.com/uploads/files/titanic.csv')

arr = df[['Pclass', 'Fare', 'Age']].values

print(arr[0, 1])

* o/p: 7.25
* Here, 1st row & 2nd column element is returned
* import pandas as pd

df = pd.read\_csv('https://sololearn.com/uploads/files/titanic.csv')

arr = df[['Pclass', 'Fare', 'Age']].values

print(arr[0])

* o/p: [ 3. 7.25 22. ]
* Here, 1st row is returned
* import pandas as pd

df = pd.read\_csv('https://sololearn.com/uploads/files/titanic.csv')

arr = df[['Pclass', 'Fare', 'Age']].values

print(arr[:,2])

* Here, 3rd column is returned
* import pandas as pd

df = pd.read\_csv('https://sololearn.com/uploads/files/titanic.csv')

arr = df[['Pclass', 'Fare', 'Age']].values

print(arr[0, 1])

print(arr[0])

print(arr[:,2])

* o/p:

1st row, 2nd column element

1st row

3rd column

Numpy shape attribute

* Returns size of numpy array
* import pandas as pd

df = pd.read\_csv('https://sololearn.com/uploads/files/titanic.csv')

arr = df[['Pclass', 'Fare', 'Age']].values

print(arr.shape)

* o/p: (887, 3)
* Here: 887 rows and 3 columns
* This can also be used for pandas dataframes

Masking

* Mask is a boolean array (True/False)
* arr[:, 2] < 18 returns boolean value of this condition
* import pandas as pd

df = pd.read\_csv('https://sololearn.com/uploads/files/titanic.csv')

# take first 10 values for simplicity

arr = df[['Pclass', 'Fare', 'Age']].values[:10]

mask = arr[:, 2] < 18

print(arr[mask])

* o/p:

[[ 3. 21.075 2. ]

[ 2. 30.0708 14. ]]

* print(arr[arr[:, 2] < 18]) will give the same output

Summing and counting

* import pandas as pd

df = pd.read\_csv('https://sololearn.com/uploads/files/titanic.csv')

arr = df[['Pclass', 'Fare', 'Age']].values

print((arr[:, 2] < 18).sum())

* o/p: 130
* Here, sum of array of boolean True values is returned
* import numpy as np

vac\_nums = [0,0,0,0,0,

            1,1,1,1,1,1,1,1,

            2,2,2,2,

            3,3,3

            ]

vac\_nums = np.array(vac\_nums)

mean = np.sum(vac\_nums)/20

var = np.sum((vac\_nums - mean)\*\*2)/20

print(var)

Scatter plot

* matplotlib is used to plot data
* 1st arg is x axis, 2nd arg is y axis
* xlabel and ylabel can be used to label axes
* Use color code with c parameter
* import matplotlib.pyplot as plt

plt.scatter(df[‘Fare’], df[‘Pclass’], c = df[‘Survived’])

plt.xlabel(‘Fare’)

plt.ylabel(‘Pclass’)

* Here, color code depends on if survived or not survived

Line

* Draw a line between 2 points
* Use plot function to draw a line
* plt.plot([0, 80], [85, 5])
* Here, line is drawn from (0, 85) to (80, 5)
* Don’t mix this up
* Ex: Line drawn from (10, 0) to (100, 3)

plt.plot([10, 100], [0, 3])

Classification

* under supervised learning
* Supervised learning uses past data
* Thing we’re trying to predict: Target or output feature
* Unsupervised data don’t have a known target
* Supervised learning includes classification and regression (target is numerical)
* Note: Logistic regression isn’t a regression problem. It’s classification problem
* Logistic regression is used to find the best line mathematically
* Remaining columns with supervised data: Features or predictors
* We use features to make our prediction

Equation for the line

* 0 = ax + by + c
* if we get a positive value when we apply data points to the right side, it’s a positive prediction
* Good line: separates 2 classes

regression

* predicting a real number
* linear regression: y = m\*x + c -> response = intercept + constant \* explanatory

Pearson’s coefficient

* coeff. of relation (r): Pearson’s coeff p(x,y) = covariance(x,y)/(std deviation of x \* std deviation of y)
* covariance has units, coefficient don’t
* possibilities: no correlation (r = 0), negative (-1 < r < 0), positive (1 > r > 0)
* for negative and positive r -> linear model
* zero r -> nonlinear model

best fit line

* distance between a point and the line -> error (residual) in prediction
* LDM -> Linear Deterministic Model (perfectly fit)
* LPM -> Linear Probabilistic Model (has uncertainty, has errors)

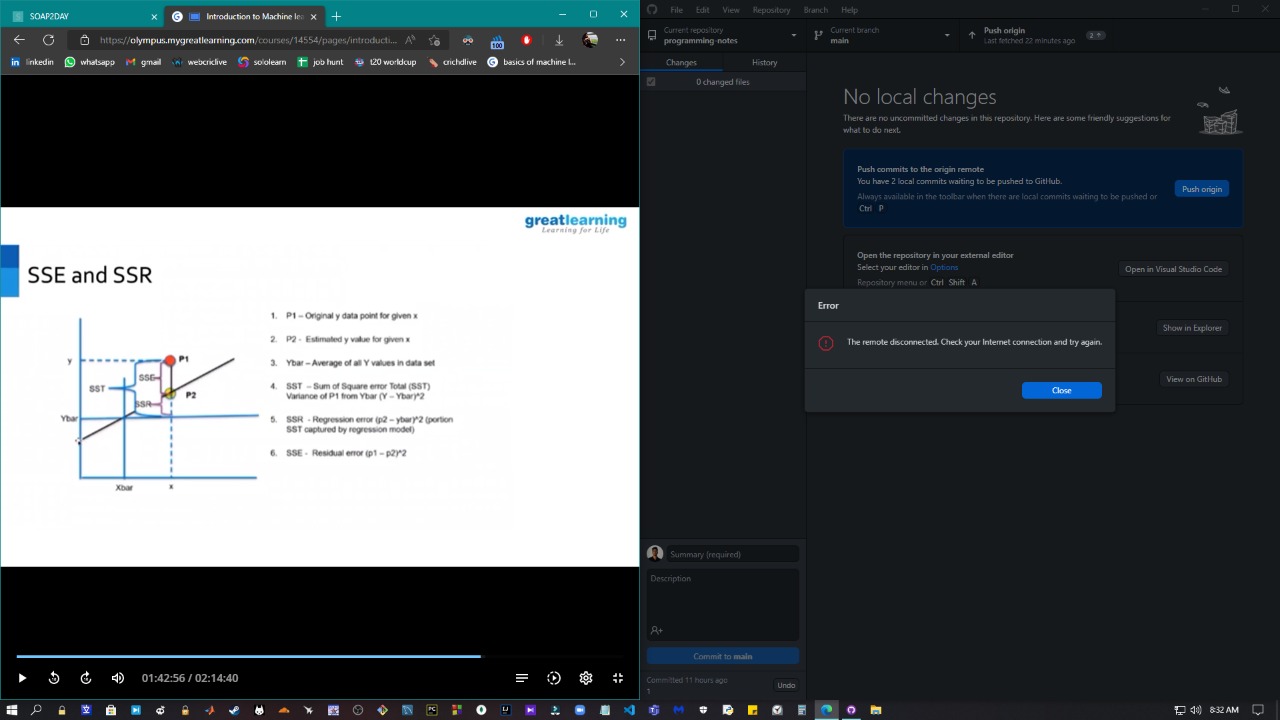
Coefficient of Determinant (R2)

* value between 0 and 1
* determines the fitness of a linear model -> helps to find the best fit line
* when points are closer to line -> coeff of determinant tends to 1 -> the better model

adjusted R-squared

* Modified version of R-squared
* If new data improves model more than expected -> adjusted R-squared increases
* If old data improves model less than expected -> adjusted R-squared decreases

SSE and SSR



* Ybar
  + avg value of y variable
  + Ybar intersects the regression line only at one point
* SST = SSE + SSR
* SST -> sum of square error
* SSR
  + regression error
  + difference between Ybar and predicted values
  + SSR can’t be controlled
* SSE
  + residual error
  + difference between actual and predicted values
  + SSE can be controlled
  + getting a best fit line reduces SSE

**case study: shelf space vs weekly sales**

|  |  |
| --- | --- |
| shelf space (x) | weekly sales (y) |
| 6 | 526 |
| 3 | 421 |
| 6 | 581 |
| 9 | 630 |
| 3 | 412 |
| 9 | 560 |
| 6 | 434 |
| 3 | 443 |
| 9 | 590 |
| 6 | 570 |
| 3 | 346 |
| 9 | 672 |

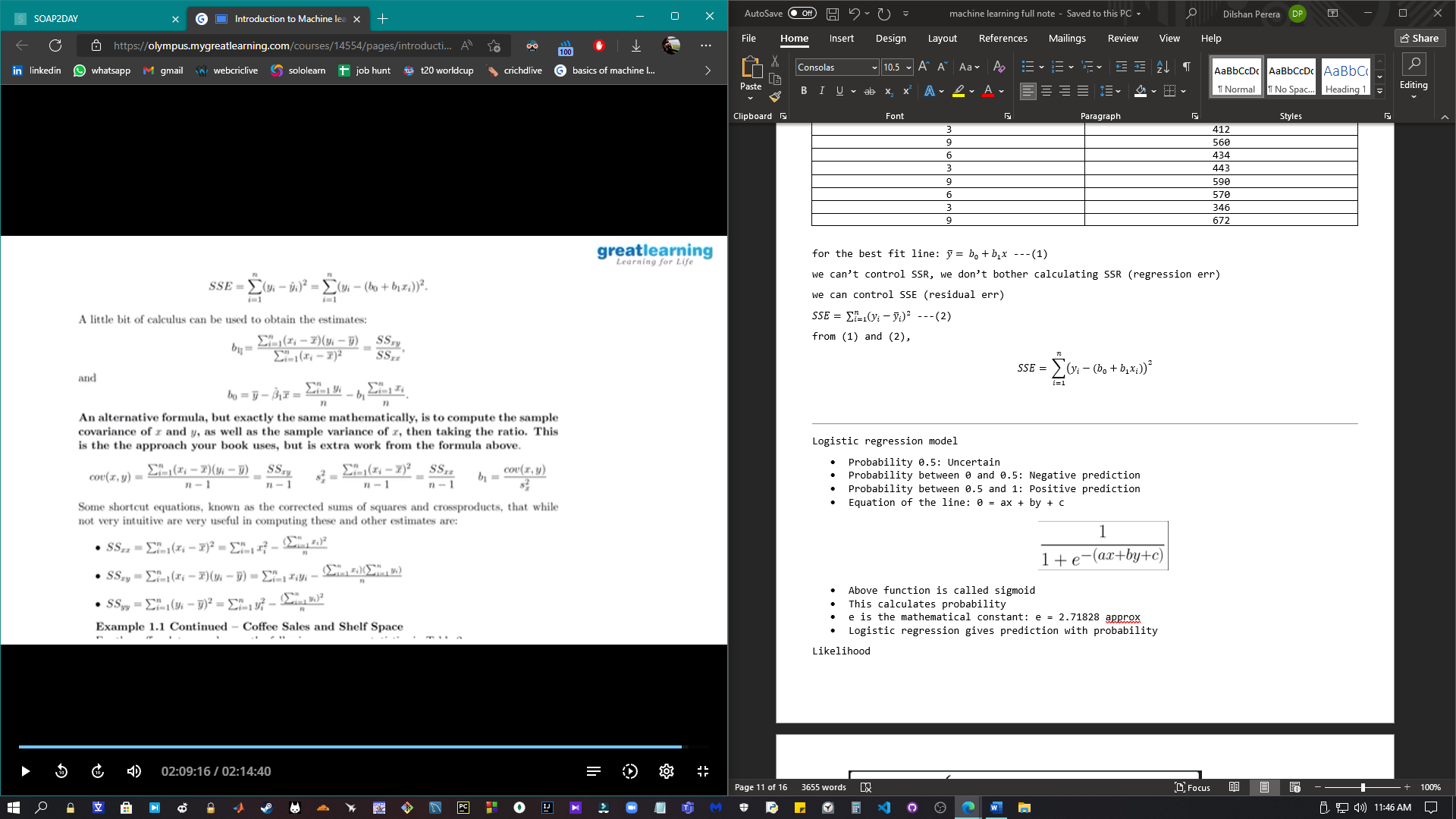
for the best fit line: ---(1)

we can’t control SSR, we don’t bother calculating SSR (regression err)

we can control SSE (residual err)

---(2)

from (1) and (2),



Remember above 3 formulas

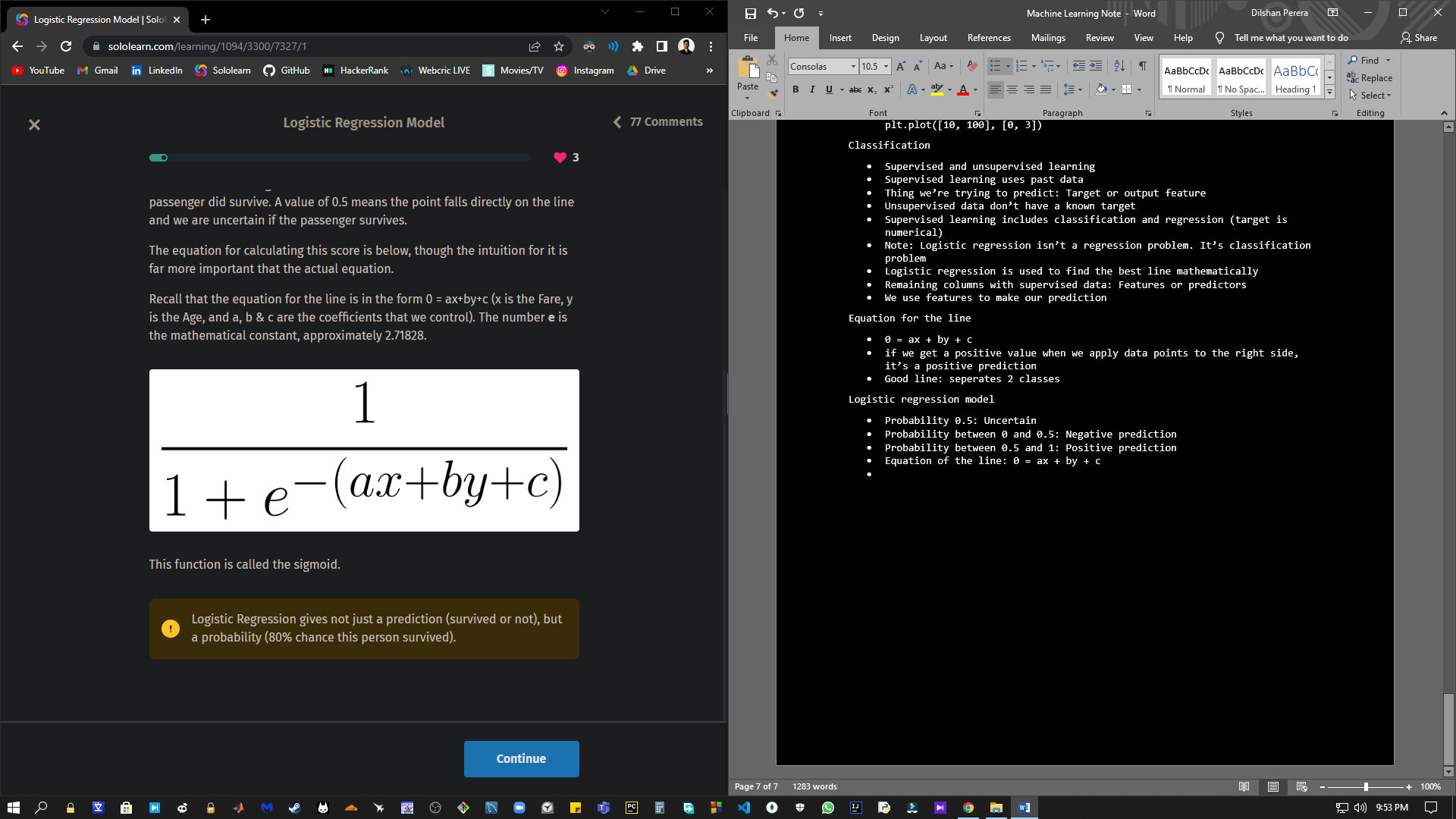
Using these formulas,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| week | shelf space (x) | weekly sales (y) | x2 | xy | y2 |
| 1 | 6 | 526 | 36 | 3156 | 276676 |
| 2 | 3 | 421 | 9 | 1263 | 177241 |
| 3 | 6 | 581 | 36 | 3486 | 337561 |
| 4 | 9 | 630 | 81 | 5670 | 396900 |
| 5 | 3 | 412 | 9 | 1236 | 169744 |
| 6 | 9 | 560 | 81 | 5040 | 313600 |
| 7 | 6 | 434 | 36 | 2604 | 188356 |
| 8 | 3 | 443 | 9 | 1329 | 196249 |
| 9 | 9 | 590 | 81 | 5310 | 348100 |
| 10 | 6 | 570 | 36 | 3420 | 324900 |
| 11 | 3 | 346 | 9 | 1038 | 119716 |
| 12 | 9 | 672 | 81 | 6048 | 451584 |
|  | 72 | 6185 | 504 | 39600 | 3300627 |

This is the best fit line

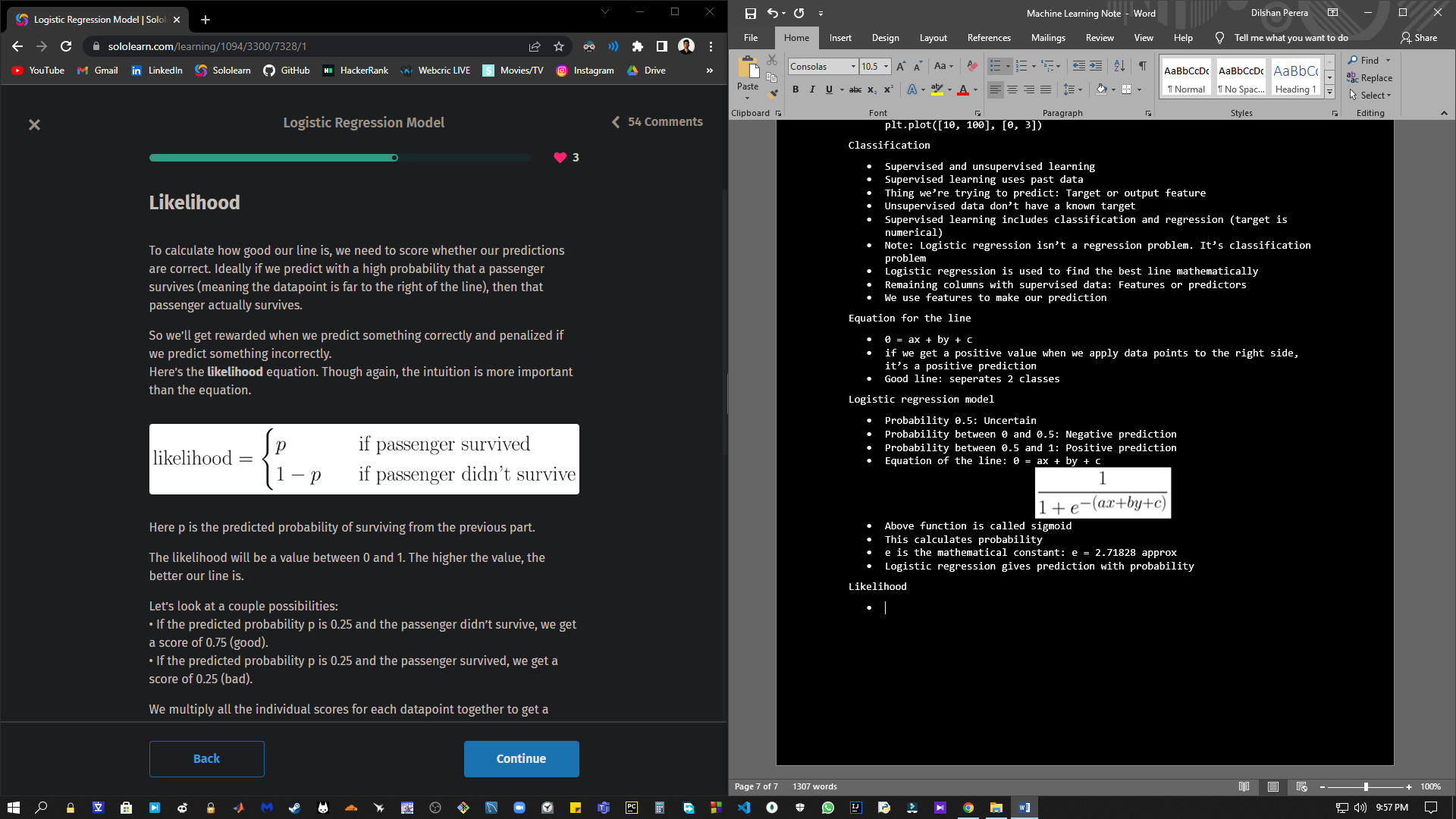
**Logistic regression model**

* Probability 0.5: Uncertain
* Probability between 0 and 0.5: Negative prediction
* Probability between 0.5 and 1: Positive prediction
* Equation of the line: 0 = ax + by + c

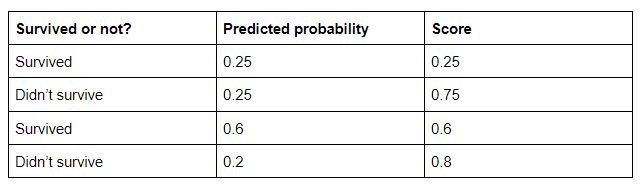


* Above function is called sigmoid
* This calculates probability
* e is the mathematical constant: e = 2.71828 approx
* Logistic regression gives prediction with probability

**Likelihood**



* Better likelihood: better prediction



* To get total score: multiply all 4 scores
* 0.25\*0.75\*0.6\*0.8 = 0.09
* Likelihood is a very low value: Model predicting everything perfectly at high likelihood

Scikit-learn

* Python module
* Short term: sklearn
* Find code samples at sckit-learn.org
* Mainly used for machine learning algorithms, build models

Building logistic regression model

* Before building models use pandas to prep data
* First, use all columns numerical
* import pandas as pd

df = pd.read\_csv('https://sololearn.com/uploads/files/titanic.csv')

df['male'] = df['Sex'] == 'male'

X = df[['Pclass', 'male', 'Age', 'Siblings/Spouses', 'Parents/Children', 'Fare']].values

y = df['Survived'].values

print(X)

print(y)

* Standard practice: Call 2d array features as X, 1d array of target as y
* Now let’s build logistic regression model using Sklearn
* First, instantiate the class
* Use fit method to build the model
* Use coef\_ & intercept\_ attributes to see coefficients
* import pandas as pd

from sklearn.linear\_model import LogisticRegression

df = pd.read\_csv('https://sololearn.com/uploads/files/titanic.csv')

X = df[['Fare', 'Age']].values

y = df['Survived'].values

model = LogisticRegression()

model.fit(X, y)

print(model.coef\_, model.intercept\_)

* o/p: [[ 0.01615949 -0.01549065]] [-0.51037152]
* Hence, the equation of line: 0 = 0.0161594x + (-0.01549065y) + (-0.51037152)
* We only used 2 features. Using more features gives more accurate model

Making predictions with the model

* Let’s use all the features instead of 2
* Use predict method to make predictions
* import pandas as pd

from sklearn.linear\_model import LogisticRegression

df = pd.read\_csv('https://sololearn.com/uploads/files/titanic.csv')

df['male'] = df['Sex'] == 'male'

X = df[['Pclass', 'male', 'Age', 'Siblings/Spouses', 'Parents/Children', 'Fare']].values

y = df['Survived'].values

model = LogisticRegression()

model.fit(X, y)

print(model.predict([[3, True, 22.0, 1, 0, 7.25]]))

print(model.predict(X[:5]))

print(y[:5])

* o/p:

[0]

[0 1 1 1 0]

[0 1 1 1 0]

* All the predictions are correct

Score the model: accuracy score

* Counting correctly predicted datapoints: accuracy score
* Use numpy sum to count correct ones
* import pandas as pd

from sklearn.linear\_model import LogisticRegression

df = pd.read\_csv('https://sololearn.com/uploads/files/titanic.csv')

df['male'] = df['Sex'] == 'male'

X = df[['Pclass', 'male', 'Age', 'Siblings/Spouses', 'Parents/Children', 'Fare']].values

y = df['Survived'].values

model = LogisticRegression()

model.fit(X, y)

y\_pred = model.predict(X)

print((y == y\_pred).sum())

* o/p: 714
* Use shape to get total number of passengers and find the accuracy
* import pandas as pd

from sklearn.linear\_model import LogisticRegression

df = pd.read\_csv('https://sololearn.com/uploads/files/titanic.csv')

df['male'] = df['Sex'] == 'male'

X = df[['Pclass', 'male', 'Age', 'Siblings/Spouses', 'Parents/Children', 'Fare']].values

y = df['Survived'].values

model = LogisticRegression()

model.fit(X, y)

y\_pred = model.predict(X)

print((y == y\_pred).sum() / y.shape[0])

* o/p: 0.8049605411499436
* Hence, accuracy of model: 80.5%
* Sklearn has already implemented this calculation
* Use score method to get that
* import pandas as pd

from sklearn.linear\_model import LogisticRegression

df = pd.read\_csv('https://sololearn.com/uploads/files/titanic.csv')

df['male'] = df['Sex'] == 'male'

X = df[['Pclass', 'male', 'Age', 'Siblings/Spouses', 'Parents/Children', 'Fare']].values

y = df['Survived'].values

model = LogisticRegression()

model.fit(X, y)

y\_pred = model.predict(X)

print(model.score(X, y))

* o/p: 0.8049605411499436
* Returns the same value
* Here, prediction for X is done and matched with y

Logistic regression with breast cancer dataset

* We don’t need to read\_csv
* This dataset is built in sklearn
* To load data use this:

from sklearn.datasets import load\_breast\_cancer

cancer\_data = load\_breast\_cancer()

* load\_breast\_cancer() is similar to python dictionary
* Use this to see available keys: print(cancer\_data.keys())
* o/p: dict\_keys(['data', 'target', 'frame', 'target\_names', 'DESCR', 'feature\_names', 'filename'])
* To get detailed description of the dataset: print(cancer\_data[‘DESCR’])
* o/p: Long description about the dataset
* import pandas as pd

from sklearn.datasets import load\_breast\_cancer

cancer\_data = load\_breast\_cancer()

print(cancer\_data.keys())

print(cancer\_data['DESCR'])

* Useful data from the DESCR:

Number of Instances (Number of datapoints): 569

Number of Attributes (Number of features): 30 numeric, predictive attributes and the class

class (target): WDBC-Malignant (Cancerous), WDBC-Benign (not cancerous)

Class distribution: 212-Malignant, 357-Benign

Measurements: 10 (radius, texture, ...)\*3 (mean, std error, worst) = 30 total features

* Process of figuring out additional features to calculate: feature engineering

Loading brest\_cancer data to pandas

* First take feature and target data from cancer\_data object
* import pandas as pd

from sklearn.datasets import load\_breast\_cancer

cancer\_data = load\_breast\_cancer()

print(cancer\_data['data'].shape)

* o/p: (569, 30)
* cancer\_data[‘data’] is a numpy array of data
* Shape is used to check the rows and columns of the array
* We want columns names to put this in pandas
* import pandas as pd

from sklearn.datasets import load\_breast\_cancer

cancer\_data = load\_breast\_cancer()

print(cancer\_data['feature\_names'])

* o/p:

['mean radius' 'mean texture' 'mean perimeter' 'mean area'

'mean smoothness' 'mean compactness' 'mean concavity'

'mean concave points' 'mean symmetry' 'mean fractal dimension'

'radius error' 'texture error' 'perimeter error' 'area error'

'smoothness error' 'compactness error' 'concavity error'

'concave points error' 'symmetry error' 'fractal dimension error'

'worst radius' 'worst texture' 'worst perimeter' 'worst area'

'worst smoothness' 'worst compactness' 'worst concavity'

'worst concave points' 'worst symmetry' 'worst fractal dimension']

* Now, let’s create pandas dataframe
* import pandas as pd

from sklearn.datasets import load\_breast\_cancer

cancer\_data = load\_breast\_cancer()

df = pd.DataFrame(cancer\_data['data'],columns=cancer\_data['feature\_names'])

print(df.head())

* o/p:

mean radius mean texture ... worst symmetry worst fractal dimension

0 17.99 10.38 ... 0.4601 0.11890

1 20.57 17.77 ... 0.2750 0.08902

2 19.69 21.25 ... 0.3613 0.08758

3 11.42 20.38 ... 0.6638 0.17300

4 20.29 14.34 ... 0.2364 0.07678

* Here, we have 30 columns in dataframe
* head method only returns 5 rows
* Now we have feature data, we need to get target data in our dataframe: Use target key
* import pandas as pd

from sklearn.datasets import load\_breast\_cancer

cancer\_data = load\_breast\_cancer()

print(cancer\_data['target'])

* o/p:

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0

1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1

1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1

1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0

1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1

1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1

1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1

1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0

0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1

1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1

0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1

1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0

1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 0 0 0 0 0 0 1]