

Introduction to Machine Learning

Homework 1

Due: Thursday, 9/12, 5pm

Instructor:

Dr. Tom Arodz

HW1 - Data description

- It involves a simple dataset with 2 classes (car vs. SUV) and 2 features (time 0-60mph, horsepower), 10 samples in total

	A	B	C	D	E
1	SampleName	ZeroToSixty	PowerHP	IsCar	IsSUV
2	c1	8.2	173	1	0
3	c2	8.4	176	1	0
4	c3	8.4	158	1	0
5	c4	9	142	1	0
6	c5	8.8	152	1	0
7	s1	8.7	189	0	1
8	s2	9.1	173	0	1
9	s3	8.6	194	0	1
10	s4	8.7	201	0	1
11	s5	8.7	198	0	1

normalized

	A	B	C	D	E
1	SampleName	ZeroToSixty	PowerHP	IsCar	IsSUV
2	c1	-1.6686	-0.12868	1	0
3	c2	-0.94312	0.019796	1	0
4	c3	-0.94312	-0.87105	1	0
5	c4	1.23331	-1.6629	1	0
6	c5	0.507833	-1.16799	1	0
7	s1	0.145095	0.663182	0	1
8	s2	1.596048	-0.12868	0	1
9	s3	-0.21764	0.910638	0	1
10	s4	0.145095	1.257077	0	1
11	s5	0.145095	1.108603	0	1

- Your task is to analyze the process of training a perceptron (the vector version, next slide)
 - $f(\mathbf{x}_i) = \text{sign}(\mathbf{w}_1 \mathbf{x}_i^1 + \mathbf{w}_2 \mathbf{x}_i^2)$
 - $\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} + c [\mathbf{y}_i - f(\mathbf{x}_i)] \mathbf{x}_i$
- Python libraries to be used:
 - Pandas (reading in a csv file)
 - Numpy (storing vectors, doing the math with them)
 - Matplotlib (plotting diagrams of training progress)
 - ML libraries (e.g. sklearn, pytorch, tensorflow, others) not allowed

HW1 - model training

- Data: Two classes (y_i is -1 or +1),
two features, i.e., 2D vector: $x_i = (x_i^1 \text{ and } x_i^2)$

Predictive model: $f(x_i) = \text{sign}(w^T x)$

with two trainable parameters $w = (w_1, w_2)$

- Set initial value of w_1 and w_2 (random, or a guess)
- Repeat training epochs:
 - Loop over all 10 training samples once (one epoch):
 - Present a sample x_i and predict $f(x_i) = \text{sign}(w^T x)$
 - Compare true class y_i with predicted class $f(x_i)$
 - If prediction is right, go to next sample ($i=i+1$)
 - If prediction is wrong (incl. $f(x)=0$), update w
 - $w_{\text{new}} = w_{\text{old}} + c[y_i - f(x_i)]x_i$
 - Calculate current error rate:
 - Start "error count" at 0
 - Loop over all 10 training samples:
 - Present a sample x_i and predict $f(x_i) = \text{sign}(w^T x)$
 - Compare true class y_i with predicted class $f(x_i)$
 - If wrong (or $f(x_i)=0$), increase "error count"
 - "Error rate" = "error count" / 10 (i.e. /#samples)

HW1 - overall task

- It involves a simple dataset with 2 classes (car vs. SUV) and 2 features (time 0-60mph, horsepower), 10 samples in total
- Your task is to train a perceptron (the vector version)
 - $f(\mathbf{x}_i) = \text{sign}(\mathbf{w}^T \mathbf{x}) = \text{sign}(w_1 x_i^1 + w_2 x_i^2)$
 - $\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} + c[y_i - f(\mathbf{x}_i)] \mathbf{x}_i$
- Specifically:
 - Use pandas to import **carSUV_normalized.csv** file (file is in Canvas)
 - Extract numpy 2D (10 x 2) array X of features from pandas
 - Extract classes (10 x 1 array Y) from pandas dataframe,
 - Convert them from 0/1 to -1/1
 - Run the perceptron algorithm as described above,
 - One epoch: presenting each of the 10 samples and updating the weights if needed after each sample
 - Perform some number of epochs, until the model no longer improves
 - Use "error rate" as the measure of model quality:
 - At the end of each epoch, do prediction with current weights for each of the 10 samples, count how many are wrong, divide by total number of training samples
- Write functions for *loading data*, *calculating error rate for single w*, *training the model*, and (for 691 only) *calculating error rates for a whole spectrum of w's*



Specific functions to code

- **Upload your submission through Canvas / Gradescope**
- Upload **a single .py python file** (not a .ipynb notebook) with code for:
 1. Function for importing the csv file using pandas:
 - `def read_csv_convert_to_numpy(fileName='carSUV_normalized.csv'):`
 - `#your code`
 - `return numpy_x, numpy_y`
 2. Function for calculating error rate for a given w and dataset:
 - `def calc_error_rate_for_single_vector_w(w, numpy_x, numpy_y):`
 - `#your code`
 - `return error_rate`
 3. Function for training the model:
 - `def train_and_evaluate(numpy_x, numpy_y, n_epochs = 20, c = 0.01):`
 - `#your code`
 - `return w;`
 4. (691 only) Function for evaluating combinations of w1 & w2 in given ranges
 - `def function_error_rate_2D(w1_range, w2_range, numpy_x, numpy_y):`
 - `#your code`
 - `return error_rates_all_ws`
- See H01_Stub.py file in Canvas for stub code and more description

Expected outcome (438 & 691)

- Calling importing data function like this:
 - `numpy_x, numpy_y = read_csv_convert_to_numpy(fileName='carSUV_normalized.csv');`
- And it should produce the following `numpy_x` (results from `print(numpy_x)`)
- And `numpy_y` (results from `print(numpy_y)`)

```
[[ -1.66859537 -0.12867714]           [[ 1]
[ -0.94311912  0.01979648]           [ 1]
[ -0.94311912 -0.87104523]           [ 1]
[  1.23330962 -1.66290453]           [ 1]
[  0.50783338 -1.16799247]           [ 1]
[  0.14509525  0.66318216]           [-1]
[  1.59604775 -0.12867714]           [-1]
[ -0.21764288  0.91063819]           [-1]
[  0.14509525  1.25707664]           [-1]
[  0.14509525  1.10860302]]           [-1]]
```

- Internally, inside your function, if you print the dataframe you import using `read_csv`, with `index_col='SampleName'`, then `print(df)` should result in:

	ZeroToSixty	PowerHP	IsCar	IsSUV
SampleName				
c1	-1.668595	-0.128677	1	0
c2	-0.943119	0.019796	1	0
c3	-0.943119	-0.871045	1	0
c4	1.233310	-1.662905	1	0
c5	0.507833	-1.167992	1	0
s1	0.145095	0.663182	0	1
s2	1.596048	-0.128677	0	1
s3	-0.217643	0.910638	0	1
s4	0.145095	1.257077	0	1
s5	0.145095	1.108603	0	1



Expected outcome (438 & 691)

- Calling the function to calculate error rate for given weights like this:
 - `numpy_x, numpy_y = read_csv_convert_to_numpy(fileName='carSUV_normalized.csv');`
 - `np.random.seed(3) # to fix randomness`
 - `random_w = np.random.randn(2,1)`
 - `print("Random weights array shape",random_w.shape)`
 - `print("Random weights values\n",random_w)`
 - `error_rate_random_weights = calc_error_rate_for_single_vector_w(random_w, numpy_x, numpy_y)`
 - `print("Error rate for random weights",error_rate_random_weights)`
- Should produce the following output:

```
Random weights array shape (2, 1)
```

```
Random weights values
```

```
[[1.78862847]
```

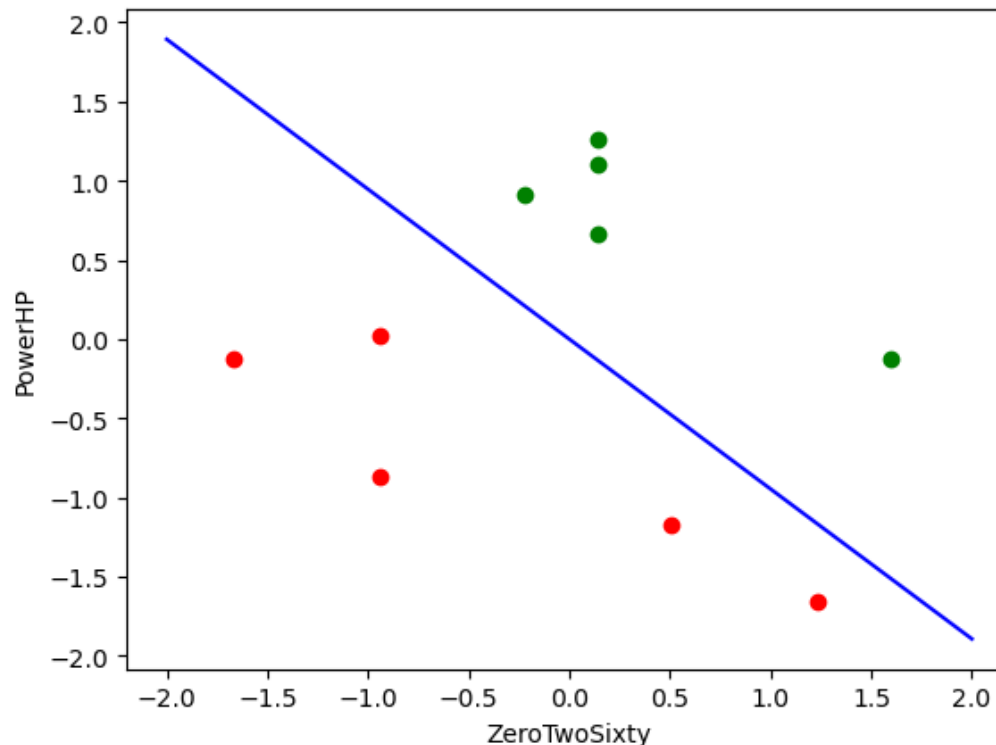
```
[0.43650985]]
```

```
Error rate for random weights 0.8
```

Expected outcome (438 & 691)

- Calling these functions like this:
 - `np.random.seed(8)` # to fix randomness
 - `numpy_x, numpy_y = read_csv_convert_to_numpy(fileName='carSUV_normalized.csv');`
 - `trained_w = train_and_evaluate(numpy_x, numpy_y, n_epochs = 20, c = 0.01);`
 - `print(trained_w)`
 - `plot_trained_w_and_dataset(numpy_x, numpy_y, trained_w);`
- Should produce the following *trained_w* values below, and plot:

```
[[ -0.02051863]  
[ -0.02167551]]
```

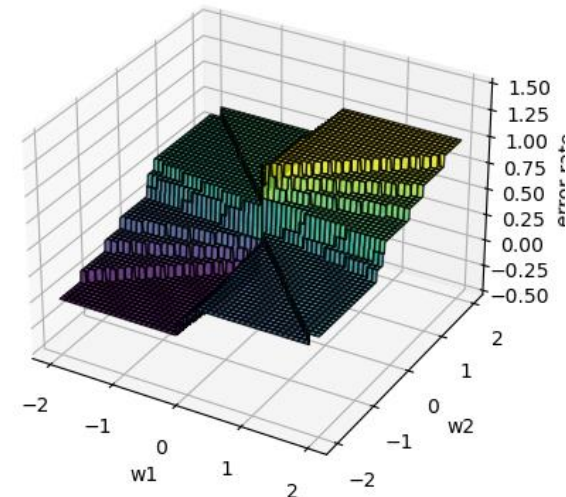
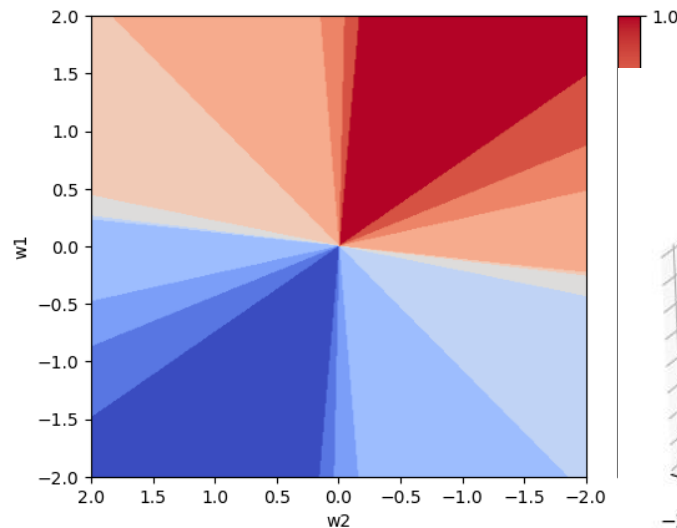


Expected outcome (691 only)

- Another call, like this:
 - `w1_range = np.arange(w1min,w1max, 0.01)`
 - `w2_range = np.arange(w2min,w2max, 0.01)`
 - `error_rates_all_ws = function_error_rate_2D(w1_range,w2_range,numpy_x, numpy_y)`
 - `print(error_rates_all_ws)`
 - `ax = plot_function_on_grid(function_error_rate_2D, numpy_x, numpy_y);`
 - `plt.show()`
 - `ax = plot3D_function_on_grid(function_error_rate_2D, numpy_x, numpy_y);`
 - `plt.show()`

- Should produce values and figures like this:

- `[[0. 0. 0. ... 0.4 0.4 0.4]`
- `[0. 0. 0. ... 0.4 0.4 0.4]`
- `[0. 0. 0. ... 0.4 0.4 0.4]`
- `...`
- `[0.6 0.6 0.6 ... 1. 1. 1.]`
- `[0.6 0.6 0.6 ... 1. 1. 1.]`
- `[0.6 0.6 0.6 ... 1. 1. 1.]`



Python – brief intro

Anaconda: Python 3 distribution tailored for machine learning and data science programming

← → ↻ 🏠 <https://www.anaconda.com/download#downloads>



[Enterprise](#)

[Pricing](#)

[Resources](#)

[About](#)

Anaconda Installers



Windows

Python 3.11

📄 64-Bit Graphical Installer (898.6 MB)



Mac

Python 3.11

📄 64-Bit Graphical Installer (610.5 MB)



Linux

Python 3.11

📄 64-Bit (x86) Installer (1015.6 MB)


Anaconda Navigator (Windows)

Anaconda Navigator


File Help

 ANACONDA.NAVIGATOR

 Home

 Environments

 Learning

 Community

Documentation

Anaconda Blog

All applications

on

base (root)

Channels



DataSpell

DataSpell is an IDE for exploratory data analysis and prototyping machine learning models. It combines the interactivity of Jupyter notebooks with the intelligent Python and R coding assistance of PyCharm in one user-friendly environment.

Install



Anaconda Notebooks

Cloud-hosted notebook service from Anaconda. Launch a preconfigured environment with hundreds of packages and store project files with persistent cloud storage.

Launch



CMD.exe Prompt

0.1.1

Run a cmd.exe terminal with your current environment from Navigator activated

Launch



JupyterLab

3.6.3

An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Launch



Notebook

6.5.4

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Launch



Powershell Prompt

0.0.1

Run a Powershell terminal with your current environment from Navigator activated

Launch



Qt Console

5.4.2

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Launch



Spyder

5.4.3

Scientific Python Development Environment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Launch



VS Code

1.77.3

Streamlined code editor with support for development operations like debugging, task running and version control.

Launch



Datalore

Kick-start your data science projects in seconds in a pre-configured environment. Enjoy coding assistance for Python, SQL, and R in Jupyter notebooks and benefit from no-code automations. Use Datalore online for free.

Launch



IBM Watson Studio Cloud

IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.

Launch

ORACLE
Cloud Infrastructure

Oracle Data Science Service

OCI Data Science offers a machine learning platform to build, train, manage, and deploy your machine learning models on the cloud with your favorite open-source tools

Launch



console_shortcut_miniconda

0.1.1



Glueviz

1.2.4

Multidimensional data visualization across files. Explore relationships within and among related datasets.



Orange 3

3.34.0

Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.



powershell_shortcut_miniconda

0.0.1



PyCharm Professional

A Full-Ridged IDE by JetBrains for both Scientific and Web Python development. Supports HTML, JS, and SQL.



RStudio

1.1.456

A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

Coding in python, option 1

Spyder: a Matlab-like layout, with panels for plots and inspecting data

The screenshot displays the Spyder Python IDE interface. The top menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. The toolbar contains icons for file operations, running, and debugging. The main window is divided into three panels:

- Editor:** Displays a Python script named `temp.py` with the following content:

```
1 # -*- coding: utf-8 -*-
2 """
3 Spyder Editor
4
5 This is a temporary script file.
6 """
7 import numpy as np;
8
9 x=1
10 y=np.random.randn(2,3)
```
- Variable Explorer:** Shows the current state of variables in the workspace.

Nam	Type	Size	Value
x	int	1	1
y	Array of float64	(2, 3)	[[0.17282472 0.07036175 -1.1... [1.73357759 -0.66901029 -1.1...
- Console:** Shows the IPython prompt and the output of the executed script:

```
Python 3.11.4 | packaged by Anaconda, Inc. | (main, Jul 5 2023, 13:38:37) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.12.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/tarodz/.spyder-py3/temp.py',
wdir='C:/Users/tarodz/.spyder-py3')

In [2]: |
```

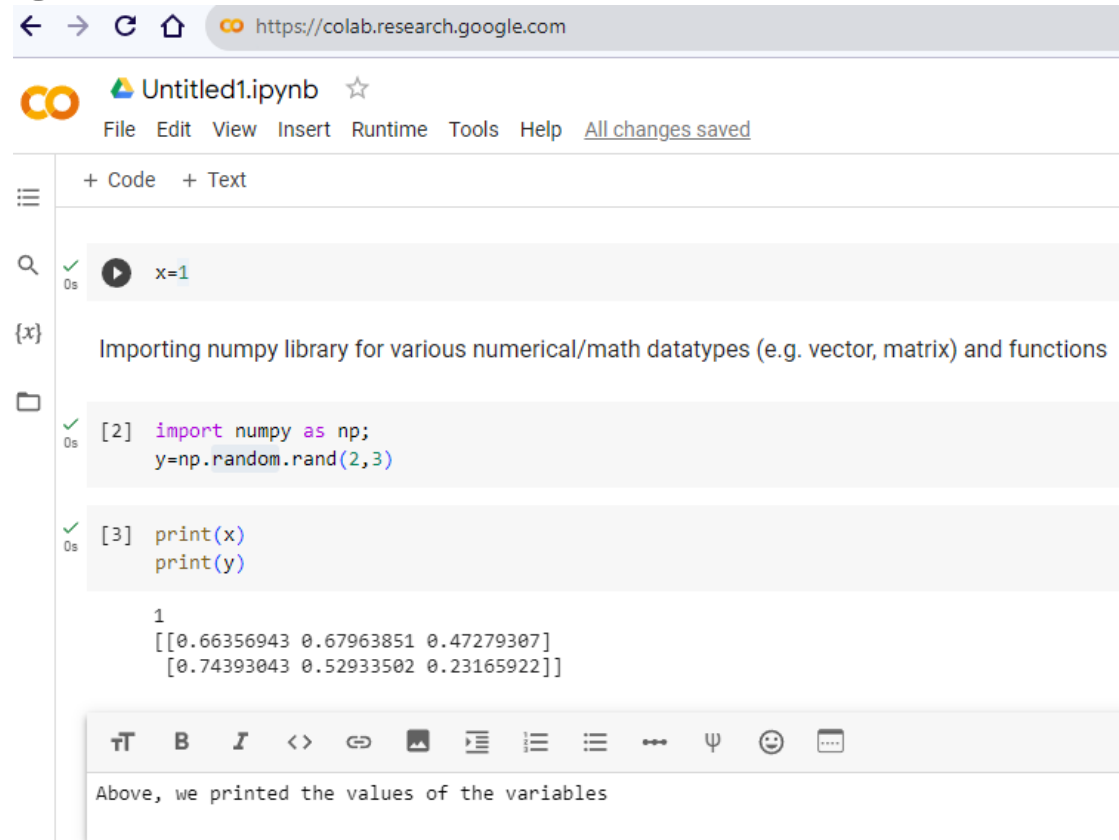
The status bar at the bottom indicates the environment is conda (Python 3.11.4) and shows various settings like Completions: conda, LSP: Python, Line 8, Col 1, UTF-8, CRLF, RW, and Mem 61%.

Coding in python, option 2

Jupyter-notebook: combines text and code, runs in the browser

Can run locally
on your machine
(e.g. on Windows:
from Navigator)

Can run online
e.g. google colab:
<https://colab.research.google.com/>



The screenshot shows the Google Colab web interface. The browser address bar displays <https://colab.research.google.com>. The notebook is titled "Untitled1.ipynb". The menu bar includes File, Edit, View, Insert, Runtime, Tools, Help, and All changes saved. The interface shows a code cell with the following code:

```
x=1
```

```
Importing numpy library for various numerical/math datatypes (e.g. vector, matrix) and functions
```

```
[2] import numpy as np;
     y=np.random.rand(2,3)
```

```
[3] print(x)
     print(y)
```

The output of the code cell is:

```
1
[[0.66356943 0.67963851 0.47279307]
 [0.74393043 0.52933502 0.23165922]]
```

The bottom of the interface shows a toolbar with various icons for text formatting, code execution, and other notebook functions. Below the toolbar, a text box contains the message: "Above, we printed the values of the variables".

Python intro

- Interpreted, no separate compilation step
- Execution starts from line one (like a script)
- Variables do not need declaration and don't need type specification
- `[a,b,c]` is a list, `(a,b,c)` is a tuple
- There is automatic garbage collection
- Has standard concepts like functions (can return multiple values), loops, etc.
- Most of useful functionality is provided in libraries (e.g., numpy)
 - Libraries need to be imported, and can be given a shortcut name

```
import numpy as np;

x=1.1

y=0
for index in range(1,5):
    y=y+index

z='one'

def make2Dvector(w1,w2):
    w=np.zeros((2,1));
    w[0]=w1
    w[1]=w2
    return w

w=make2Dvector(0.1, -0.2)

print(x,type(x))
print(y,type(y))
print(z,type(z))
print(w,type(w))

1.1 <class 'float'>
10 <class 'int'>
one <class 'str'>
[[ 0.1]
 [-0.2]] <class 'numpy.ndarray'>
```

Pandas

- Provides data import and manipulation functionality (a bit like R)
- Main data type is “DataFrame”
 - Basically, a 2D array of data, but with column names, row names, various data types in the same array

Spyder (Python 3.11)

File Edit Search Source Run Debug Consoles Projects Tools View Help



C:\Users\tarodz\Downloads\491H1\p2.py

p2.py x

```
1 # -*- coding: utf-8 -*-
2 """
3 Simple python3 code
4 """
5 import pandas as pd;
6
7 df=pd.read_csv('carSUV.csv',index_col='SampleName')
8
9
10 print(df)
```



Nam	Type	Size	Value
df	DataFrame	(10, 4)	Column names: ZeroToSixty, PowerHP, IsCar, IsSUV

SampleName	ZeroToSixty	PowerHP	IsCar	IsSUV
c1	8.2	173	1	0
c2	8.4	176	1	0
c3	8.4	158	1	0
c4	9.0	142	1	0
c5	8.8	152	1	0
s1	8.7	189	0	1
s2	9.1	173	0	1
s3	8.6	194	0	1
s4	8.7	201	0	1
s5	8.7	198	0	1

Help Variable Explorer Plots Files

Console 5/A x

```
wdir='C:/Users/tarodz/Downloads/491H1'
ZeroToSixty PowerHP IsCar IsSUV
SampleName
c1      8.2      173      1      0
c2      8.4      176      1      0
c3      8.4      158      1      0
c4      9.0      142      1      0
c5      8.8      152      1      0
s1      8.7      189      0      1
s2      9.1      173      0      1
s3      8.6      194      0      1
s4      8.7      201      0      1
s5      8.7      198      0      1
```

In [2]:

IPython Console History

SampleName	ZeroToSixty	PowerHP	IsCar	IsSUV
c1	8.2	173	1	0
c2	8.4	176	1	0
c3	8.4	158	1	0
c4	9.0	142	1	0
c5	8.8	152	1	0
s1	8.7	189	0	1
s2	9.1	173	0	1
s3	8.6	194	0	1
s4	8.7	201	0	1
s5	8.7	198	0	1



Pandas

- Provides data import and manipulation functionality (a bit like R)
- Main data type is “DataFrame”
 - Basically, a 2D array of data, but with column names, row names, various data types in the same array

```
import pandas as pd;
```

```
df=pd.read_csv('carSUV.csv',index_col='SampleName')
```

```
#can do all kinds of data analytics
```

```
print(df.mean())
```

printed:

ZeroToSixty	8.66
PowerHP	175.60
IsCar	0.50
IsSUV	0.50

```
# extract raw data as numpy matrix
```

```
myDataAsNumpyMatrix = df.to_numpy();
```


Numpy

- Vectors and matrix variables, reshaping (e.g., transposing) them
- Arithmetic and functions on vectors
- Much faster than doing math in raw python

```
import numpy as np;
```

```
# a two-by-one matrix (a vector)
w = np.ones((2,1))
print("shape of w",w.shape)
```

```
x = np.array( [[0.1,],[0.2,]])
```

```
print("x before",x)
x = 2*x-0.1
print("x after",x)
```

x before $\begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}$
x after $\begin{bmatrix} 0.1 \\ 0.3 \end{bmatrix}$

```
wT = np.transpose(w)
```

```
wTx = np.matmul(wT,x)
```

```
# compare with
wElementwiseX = np.multiply(w,x)
```

```
#squeeze() removes "useless" dimensions
# 1-by-1 matrix becomes just one number
wTxAsNumber = np.squeeze(wTx)
```

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \leftarrow \text{2D Vector (2x1)}$$

$$\mathbf{x} = \begin{bmatrix} x^1 \\ x^2 \end{bmatrix} \leftarrow \text{2D Vector (2x1)}$$

$$\mathbf{w}^T = [w_1, w_2] \leftarrow \text{Transposed vector (1x2)}$$

Matrix multiplication

$$\mathbf{w}^T \mathbf{x} = [w_1, w_2] \begin{bmatrix} x^1 \\ x^2 \end{bmatrix} = w_1 x^1 + w_2 x^2$$

Name	Type	Size	Value
w	Array of float64	(2, 1)	$\begin{bmatrix} 1. \\ 1. \end{bmatrix}$
wElementwiseX	Array of float64	(2, 1)	$\begin{bmatrix} 0.1 \\ 0.3 \end{bmatrix}$
wT	Array of float64	(1, 2)	$\begin{bmatrix} 1. & 1. \end{bmatrix}$
wTx	Array of float64	(1, 1)	$\begin{bmatrix} 0.4 \end{bmatrix}$
wTxAsNumber	Array of float64	1	0.4
x	Array of float64	(2, 1)	$\begin{bmatrix} 0.1 \\ 0.3 \end{bmatrix}$



Matplotlib

Creates plots from data

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import Normalize

def plot_trained_w_and_dataset(numpy_x, numpy_y, w):

    samples_class1 = numpy_y.flatten()==1
    samples_class0 = numpy_y.flatten()=-1
    plt.scatter(numpy_x[samples_class1,0], numpy_x[samples_class1,1], c='red')
    plt.scatter(numpy_x[samples_class0,0], numpy_x[samples_class0,1], c='green')
    plt.xlabel('ZeroTwoSixty')
    plt.ylabel('PowerHP')

    x1_line = np.linspace(-2, 2, 100)
    x2_line = (-w[0] * x1_line) / w[1]

    # Create a blue line based on the equation
    plt.plot(x1_line, x2_line, c='blue')
    plt.show()
```



Matplotlib

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import Normalize

def plot3D_function_on_grid(function_to_plot, numpy_x, numpy_y):

    # Create a meshgrid
    w1min,w1max = -2.0, 2.0
    w2min,w2max = -2.0, 2.0
    w1_range = np.arange(w1min,w1max, 0.01)
    w2_range = np.arange(w2min,w2max, 0.01)

    error_rates_values_for_W1W2 = function_to_plot(w1_range, w2_range, numpy_x, numpy_y)
    W1, W2 = np.meshgrid(w1_range, w2_range)

    # Create a figure and a 3D axis
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    # Create the surface plot
    surface = ax.plot_surface(W1, W2, error_rates_values_for_W1W2,
                              cmap='viridis', alpha=0.8, edgecolor='black')

    # Add labels and title
    ax.set_xlabel('w1')
    ax.set_ylabel('w2')
    ax.set_zlabel('error rate')
    ax.set_zlim(-0.5, 1.5)
    return ax;
```



Matplotlib

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import Normalize

def plot_function_on_grid(function_to_plot, numpy_x, numpy_y):

    # Create a meshgrid
    w1min, w1max = -2.0, 2.0
    w2min, w2max = -2.0, 2.0

    w1_range = np.arange(w1min, w1max, 0.01)
    w2_range = np.arange(w2min, w2max, 0.01)

    error_rates_values_for_W1W2 = function_to_plot(w1_range, w2_range, numpy_x, numpy_y)
    W1, W2 = np.meshgrid(w1_range, w2_range)

    # Create a figure and a 3D axis
    fig = plt.figure()
    ax = fig.add_subplot(111)

    img = ax.imshow(error_rates_values_for_W1W2, origin='lower', cmap='coolwarm',
                    extent=[w1max, w1min, w2min, w2max], aspect='auto')
    # 'coolwarm' goes from blue (low) to red (high)

    ax.set_xlabel('w2')
    ax.set_ylabel('w1')
    cbar = fig.colorbar(img) # Add a color bar to show the mapping of values to colors
    cbar.set_label('error rate')
    plt.show()

    return ax;
```