# INTRODUCTION TO MACHINE LEARNING

Bavan Thomas
Credence - MFS

# Creating Data

```
import numpy as np
from sklearn import preprocessing

#We imported a couple of packages. Let's create some sample data and add the line to this file:

input_data = np.array([[3, -1.5, 3, -6.4], [0, 3, -1.3, 4.1], [1, 2.3, -2.9, -4.3]])
```

# Data Preprocessing Techniques

Data can be preprocessed using several techniques like -

# Mean removal

- It involves removing the mean from each feature so that it is centered on zero.
- Mean removal helps in removing any bias from the features.

```
print ("Mean = ", input_data.mean(axis = 0))
print ("StdDeviation =", input_data.std(axis = 0))
```

# Scaling

- On working with raw data, data will be in different magnitude
- So it is necessary to be scaled to a level
- This help in properly training the alogorithm
- Three Methods of scaling

***Standard Scalar*** – brings each feature to a mean = 0 & deviation = 1

***MinMax Scalar*** – brings feature in 0-1 range

***Normalizer*** – bring feature vector Euclidean len = 1

# Hands on

```
print (input_data)
data_standardized = preprocessing.scale(input_data)
print (data_standardized )

print ("Mean = ", data_standardized.mean(axis = 0))
print ("StdDeviation =", data_standardized.std(axis = 0))
```

- Observe that in the output, mean is almost 0 and the standard deviation is 1.

# Hands on

```
data_scaler = preprocessing.MinMaxScaler (feature_range = (0, 1))
data_scaled = data_scaler.fit_transform (input_data)
print ("Min Max scaled data = ", data_scaled)
```

- Observe that in the output, Feature is scaled between 0 – 1.

# Normalization

- Normalization involves adjusting the values in the feature vector so as to measure them on a common scale
- Normalization is used to ensure that data points do not get boosted due to the nature of their features.
- Two normalization method

***L1 (Least Absolute Deviations)*** : Sum of absolute values on each row = 1

***L2 (Least Squares)*** : Sum of squares on each row = 1

# Hands on

```
data_normalized_l1 = preprocessing.normalize (input_data, norm = 'l1')
print ("L1 normalized data = ", data_normalized_l1)
#(take a row, add each data avoiding sings (negatives are not mattered in absolute))

data_normalized_l2 = preprocessing.normalize (input_data, norm = 'l2')
print ("L2 normalized data = ", data_normalized_l2)
#(take a row, take square of each and add)
```

# Binarization

- Binarization is used to convert a numerical feature vector into a Boolean vector

```
data_binarized = preprocessing.Binarizer(threshold=1.4)
trns_binarized = data_binarized.transform(input_data)
print ("Binarized data =", trns_binarized)
```

# Label Encoding

- In supervised learning, we mostly come across a variety of labels which can be in the form of numbers or words
- If they are numbers, then they can be used directly by the algorithm
- the training data is usually labelled with words
- Label encoding refers to changing the word labels into numbers

# Hands on

```
input_classes = ['suzuki', 'ford', 'suzuki', 'toyota', 'ford',
'bmw']

label_encoder = preprocessing.LabelEncoder()
label_encoder.fit(input_classes)
for i, item in enumerate(label_encoder.classes_):
        print item, '-->', i
```

# Hands on

```
input_classes = ['suzuki', 'ford', 'suzuki', 'toyota', 'ford',
'bmw']

label_encoder = preprocessing.LabelEncoder ()
encoded_labels = label_encoder.transform (input_classes)

print "Labels =", input_classes
print "Encoded labels =", list (encoded_labels)
```

# Hands on

- transforming numbers back to word labels

```
decoded_labels = label_encoder.inverse_transform
(encoded_labels)

print ("Encoded labels =", encoded_labels)
print ("Decoded labels =", list(decoded_labels))
```

# Data Analysis

# Data Analysis

- We can load the data directly from the UCI Machine Learning repository
- We have an in built dataset for Iris Flower

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

# Summarizing the Dataset

- Summarizing the data can be done in many ways

- **Check dimensions of the dataset**
- **List the entire data**
- **View the statistical summary of all attributes**
- **Breakdown of the data by the class variable**

# Summarizing the Dataset

Dimensions of Dataset

```
feat_shape = iris_flower.data.shape
print(feat_shape)
```

Feature Names

```
names = iris_flower.feature_names
print(names)
```

Data

```
dataset = iris_flower.data
print(dataset)
```

# Summarizing the Dataset

Target Name / Class Name (species name)

```
print (iris_flower.target_names)
```

Target Label / Class Label (species label)

```
print (iris_flower.target)
```

Target shape / Class Shape

```
print(iris_flower.target.shape)
```

# Data Visualization

# Data Visualization

- Uses matplotlib to plot

```
from matplotlib import pyplot as plt
x_index = 0
y_index = 1

# this formatter will label the colorbar with the correct target names
formatter = plt.FuncFormatter(lambda i, *args: iris_flower.target_names[int(i)])
```

# Hands on

```
plt.figure (figsize=(5, 4))

plt.scatter (iris_flower.data[:, x_index], iris_flower.data[:, y_index], c =
iris_flower.target)

plt.colorbar(ticks=[0, 1, 2], format=formatter)

plt.xlabel(iris_flower.feature_names[x_index])
plt.ylabel(iris_flower.feature_names[y_index])

plt.tight_layout()
plt.show()
```