# DEEP LEARNING WITH KERAS WORKSHOP

MUHAMMAD RAJABINASAB @ TARBIAT MODARES UNIVERSITY

## Chapter 1 :

## Data Processing

# IN THIS WORKSHOP WE USE:

- Python programming language.
- Keras with tensorflow backend.
- Some famous data processing and machine learning libraries such as pandas and scikit-learn.
- Jupyter Lab to work on the examples easily.

# PREREQUISITES

- Python 3
- Jupyter Lab

```
pip install jupyterlab
```

- All the libraries we need are easy to install using pip. If not, we will discuss the installation procedure.

# IN THIS CHAPTER WE LEARN TO:

- Load data into workspace
- Clean the data
- Preprocess the data

# LOADING DATA

- Loading data is the first step to do.
- There are many ways to load data in Python.
- We use Pandas library in order to load data.
- As an example, we will work on online_shoppers_intention dataset.
- You can download the data of this chapter using this link:

  https://1drv.ms/u/s!Ag8ZObnfMMPjjR-c2hLuzu--lMqS?e=zvsf9Z

# LOADING DATA

- We can load csv data using pandas read_csv method:

```
import pandas as pd
data = pd.read_csv('../data/online_shoppers_intention.csv')
```

- To verify that we loaded the data into the memory correctly:

```
data.head(20)
```

# CLEANING THE DATA

- In machine learning, we usually want to predict a **target** using some **features**.
- **Revenue** seems to be a good target. We must separate features from targets then:

```
feats = data.drop('Revenue', axis=1)
target = data['Revenue']
```

- We can verify the shapes of the target and features datasets:

```
print(f'Features table has {feats.shape[0]} rows and {feats.shape[1]}
columns')
print(f'Target table has {target.shape[0]} rows')
```

# CLEANING THE DATA

- Finally we can save the target and features datasets in order to use them later:

```
feats.to_csv('../data/OSI_feats.csv', index=False)
target.to_csv('../data/OSI_target.csv', header='Revenue', index=False)
```

# DATA PREPROCESSING

- Before using learning techniques, we must represent the data appropriately.
- All features must be represented in numerical format.
- We must combat the large variation in scale between features using normalization.

# DATA PREPROCESSING

- Examples of numerical encoding:

| Label | | is_dog |
|---|---|---|
| Cat | | 0 |
| Dog | | 1 |
| Cat | → | 0 |
| Cat | | 0 |
| Dog | | 1 |
| Dog | | 1 |

| month | | month |
|---|---|---|
| January | | 1 |
| March | | 3 |
| October | → | 10 |
| April | | 4 |
| July | | 7 |
| January | | 1 |

# DATA PREPROCESSING

- Now let's continue with previous example.
- We load features into memory:

```
%matplotlib inline
import pandas as pd
data = pd.read_csv('../data/OSI_feats.csv')
```

- We look at the first 20 rows to checkout the data:

```
data.head(20)
```

# DATA PREPROCESSING

- There are some features which are currently numerical, e.g. **BounceRates**.
- There are some categorical features, e.g. **Month**.
- There is also a binary feature, e.g. **Weekened**.

# DATA PREPROCESSING

- For numerical features, we can use **describe** function to get some information.

```
data.describe()
```

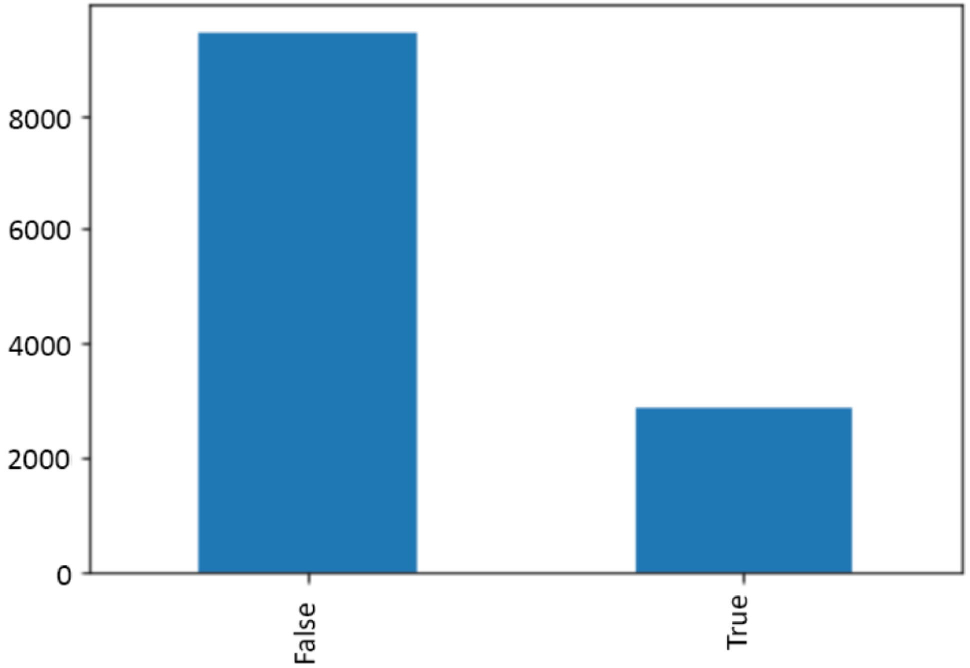
- First, we start with the binary feature, **Weekend**. We can get the distribution of each value using **value_counts** function.

```
data['Weekend'].value_counts()
```

# DATA PREPROCESSING

- We can also represent it as a bar chart:

```
data['Weekend'].value_counts().plot(kind='bar')
```



# DATA PREPROCESSING

- Now we can do the numerical encoding:

```
data['is_weekend'] = data['Weekend'].apply(lambda row: 1 if row ==
True else 0)
```

- Next, we can compare **is_weekend** and **Weekend**:

```
data[['Weekend','is_weekend']].tail()
```

- And then we can drop **Weekend**:

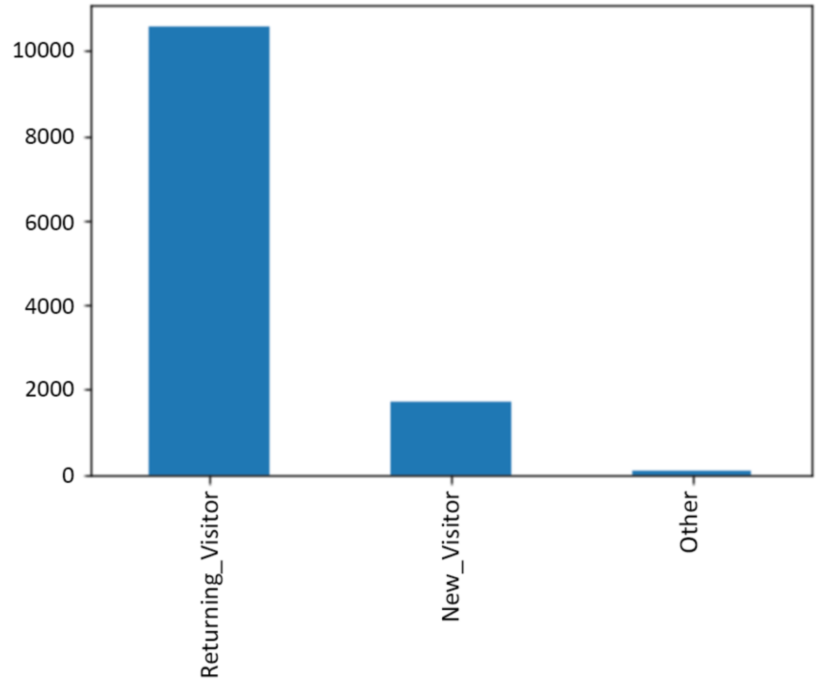
```
data.drop('Weekend', axis=1, inplace=True)
```

# DATA PREPROCESSING

- Next, we must deal with the categorical features.
- We convert categorical columns to a set of dummy columns.
- This is achieved by using **get_dummies** function.
- Let's work on an example using **VisitorType** feature.

# DATA PREPROCESSING

- First let's get the distribution and then represent it as a bar chart:




```
data['VisitorType'].value_counts()
data['VisitorType'].value_counts().plot(kind='bar')
```

# DATA PREPROCESSING

- Now we use get_dummies function to produce dummy columns:

```
colname = 'VisitorType'
visitor_type_dummies = pd.get_dummies(data[colname], prefix=colname)
pd.concat([data[colname], visitor_type_dummies], axis=1).tail(n=10)
```

- As you can see, there is some redundant information. To counter that, we drop VisitorType_Other, the dummy column with the lowest frequency:

```
visitor_type_dummies.drop('VisitorType_Other', axis=1, inplace=True)
visitor_type_dummies.head()
```

# DATA PREPROCESSING

- Finally, we add the dummy columns to the original dataset and remove the original categorical feature **Visitor_Type**:

```
data = pd.concat([data, visitor_type_dummies], axis=1)
data.drop('VisitorType', axis=1, inplace=True)
```

- If we repeat the exact same steps for the feature **Month**, our dataset will be fully numerical. Then we can save the numerical dataset for further use.

```
data.to_csv('../data/OSI_feats_e2.csv', index=False)
```

# DATA PREPROCESSING

- If we take a closer look, we will find out something interesting about OSI dataset.
- For example, lets inspect **OperatingSystems** feature:

```
import pandas as pd
data = pd.read_csv('../data/OSI_feats_e2.csv')
data['OperatingSystems'].value_counts()
```

- As you can see, there is only 8 distinct values for this feature. Representing it as a numerical value with this variation won't be a very good choice. So we deal with this feature and other features like that such as **Browsers**, **TrafficType** and **Region** as a categorical feature.

# DATA PREPROCESSING

- We create dummy variables, drop the one with the lowest frequency and add the rest to the dataset:

```
colname = 'OperatingSystems'
operation_system_dummies = pd.get_dummies(data[colname], prefix=colname)

operation_system_dummies.drop(colname+'_5', axis=1, inplace=True)
data = pd.concat([data, operation_system_dummies], axis=1)
```

- We can repeat this for **Browsers**, **TrafficType** and **Region** features.

# DATA PREPROCESSING

- Finally, we can save the processed dataset:

```
data.to_csv('../data/OSI_feats_e3.csv', index=False)
```

- Now we must work on the target dataset.

# DATA PREPROCESSING

- Let's do the same for the target dataset. First we load and checkout the data:

```
target = pd.read_csv('../data/OSI_target.csv')
target.head(n=10)
```

- As you can see, there is only one feature and it is in binary format, so:

```
target['Revenue'] = target['Revenue'].apply(lambda row: 1 if row==True
else 0)
target.head(n=10)
```

Finally, we save the target dataset as well:

```
target.to_csv('../data/OSI_target_e2.csv', index=False)
```

# DATA PREPROCESSING

- Are we done? No, we aren't.
- Let's take an even closer look.
- There are some features in the features dataset which have a high variance.
- High variance has a bad effect on different learning algorithms, especially deep learning based methods.
- We counter this issue using **normalization**.
- Features which need to be normalized are **Administrative**, **Administrative_Duration**,**Informational**, **Informational_Duration**, **ProductRelated**, **ProductRelated_Duration** and **PageValues**

# DATA PREPROCESSING

- Let's work on **PageValues** as an example.
- First, we load the feature dataset into memory. Note that we must import **preprocessing** from **sklearn** in order to do the normalization:

```
import pandas as pd
from sklearn import preprocessing
data = pd.read_csv('../data/OSI_feats_e3.csv')
```

# DATA PREPROCESSING

- Now we normalize the data using the **MinMaxScaler()**:

```
x = data[['PageValues']].values.astype(float)
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
```

- Next, we must drop the original feature and add the normalized version to the dataset:

```
PageValuesNormalized = pd.DataFrame(x_scaled)
PageValuesNormalized.columns = ['PageValuesNormalized']
data.drop('PageValues', axis=1, inplace=True)
data = pd.concat([data, PageValuesNormalized], axis=1)
```

# DATA PREPROCESSING

- We can check out the data to make sure the normalization is done correctly:

```
data.head(200)
```

- We can repeat the exact same steps to do the normalization for other features as well.

# DATA PREPROCESSING

- **Administrative**

```
x = data[['Administrative']].values.astype(float)
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
AdministrativeNormalized = pd.DataFrame(x_scaled)
AdministrativeNormalized.columns = ['AdministrativeNormalized']
data.drop('Administrative', axis=1, inplace=True)
data = pd.concat([data, AdministrativeNormalized], axis=1)
```

# DATA PREPROCESSING

- **Administrative_Duration**

```
x = data[['Administrative_Duration']].values.astype(float)
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
Administrative_DurationNormalized = pd.DataFrame(x_scaled)
Administrative_DurationNormalized.columns =
['Administrative_DurationNormalized']
data.drop('Administrative_Duration', axis=1, inplace=True)
data = pd.concat([data, Administrative_DurationNormalized], axis=1)
```

# DATA PREPROCESSING

- **Informational**

```
x = data[['Informational']].values.astype(float)
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
InformationalNormalized = pd.DataFrame(x_scaled)
InformationalNormalized.columns = ['InformationalNormalized']
data.drop('Informational', axis=1, inplace=True)
data = pd.concat([data, InformationalNormalized], axis=1)
```

# DATA PREPROCESSING

- **Informational_Duration**

```
x = data[['Informational_Duration']].values.astype(float)
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
Informational_DurationNormalized = pd.DataFrame(x_scaled)
Informational_DurationNormalized.columns =
['Informational_DurationNormalized']
data.drop('Informational_Duration', axis=1, inplace=True)
data = pd.concat([data, Informational_DurationNormalized], axis=1)
```

# DATA PREPROCESSING

- **ProductRelated**

```
x = data[['ProductRelated']].values.astype(float)
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
ProductRelatedNormalized = pd.DataFrame(x_scaled)
ProductRelatedNormalized.columns = ['ProductRelatedNormalized']
data.drop('ProductRelated', axis=1, inplace=True)
data = pd.concat([data, ProductRelatedNormalized], axis=1)
```

# DATA PREPROCESSING

- **ProductRelated_Duration**

```
x = data[['ProductRelated_Duration']].values.astype(float)
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
ProductRelated_DurationNormalized = pd.DataFrame(x_scaled)
ProductRelated_DurationNormalized.columns = 
['ProductRelated_DurationNormalized']
data.drop('ProductRelated_Duration', axis=1, inplace=True)
data = pd.concat([data, ProductRelated_DurationNormalized], axis=1)
```

# DATA PREPROCESSING

- Finally, we save the normalized dataset:

```
data.to_csv('../data/OSI_feats_e4.csv', index=False)
```

- There are other preprocessing techniques such as feature reduction and feature selection, but we do not cover them in this workshop.
- The dataset is now ready for deep learning!

# THANKS FOR YOU ATTENTION!

Feel free to ask your questions.