



SuperDataScience

[2020]

# DEEP AND MACHINE LEARNING

## PRACTICAL TUTORIAL



# 1. HOW TO DIVIDE DATASETS INTO TRAINING AND TESTING?

## A. CONCEPT

Data set is generally divided into

**75% for training and 25% for testing.**

- **Training set:** used for model training.
- **Testing set:** used for testing trained model.

- ! **Make sure that testing dataset has never been seen by the trained model before.**

### Testing



### Training



## B. DIVIDING DATASET USING SCIKIT-LEARN

```
>> from import sklearn.model_selection train_test_split
>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=0)
```

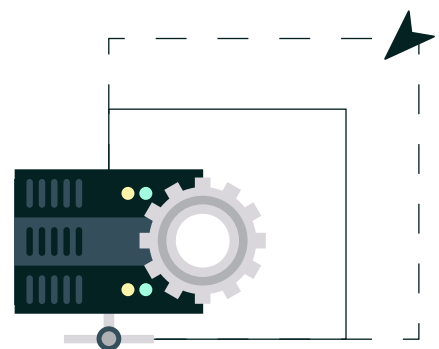
# 2. HOW TO SCALE DATA USING SCIKIT-LEARN?

## A. CONCEPT

Objective is to scale the training data to (0, 1) or (-1, 1).

This step is critical before training Artificial Neural Networks to ensure that all inputs fed to the network are within similar range (i.e.: “treated fairly”).

If normalization is ignored, large inputs will dominate smaller ones.

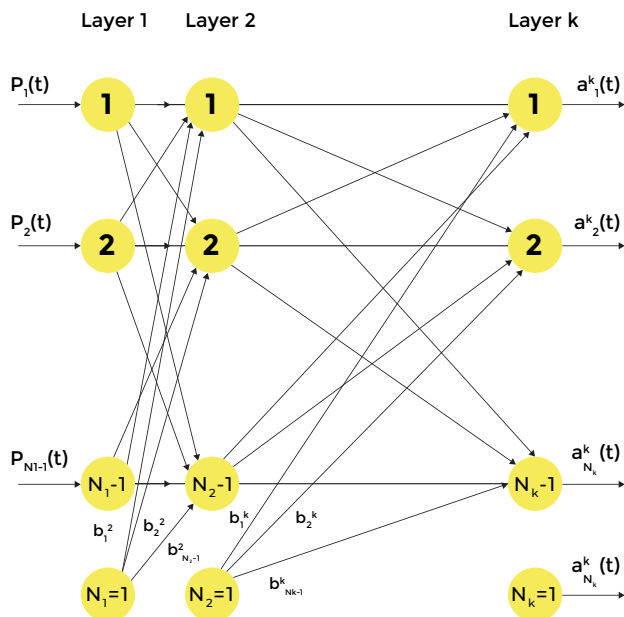


## B. SCALING DATA USING SCIKIT LEARN

```
>> from sklearn.preprocessing import MinMaxScaler
>> scaler = MinMaxScaler()
>> X_scaled = scaler.fit_transform(X)
```

# 3. HOW TO BUILD AN ARTIFICIAL NEURAL NETWORKS (ANNS) USING KERAS?

## A. CONCEPT



Artificial Neural Networks are information processing models inspired by the human brain. ANNs are built in a layered fashion where inputs are propagated starting from the input layer through the hidden layers and finally to the output.

Networks training is performed by optimizing the matrix of weights outlined below:

$$\begin{bmatrix} W_{11} & W_{12} & \dots & W_{1,N} \\ W_{21} & W_{22} & \dots & W_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ W_{m-1,1} & W_{m-1,2} & \dots & W_{m-1,N_1} \\ W_{m,1} & W_{m,2} & \dots & W_{m,N_1} \end{bmatrix}$$

## B. BUILD AN ANNs USING KERAS

```
>> import tensorflow.keras
>> from keras.models import Sequential
>> from keras.layers import Dense
>> from sklearn.preprocessing import MinMaxScaler

>> model = Sequential()
>> model.add(Dense(25, input_dim=5, activation='relu'))
>> model.add(Dense(25, activation='relu'))
>> model.add(Dense(1, activation='linear'))
>> model.summary()
```

## C. TRAIN AN ANN USING KERAS

```
>> model.compile(optimizer='adam', loss='mean_squared_error')
>> epochs_hist = model.fit(X_train, y_train, epochs=20, batch_size=25, validation_split=0.2)
```

## D. EVALUATE THE TRAINED ANN MODEL

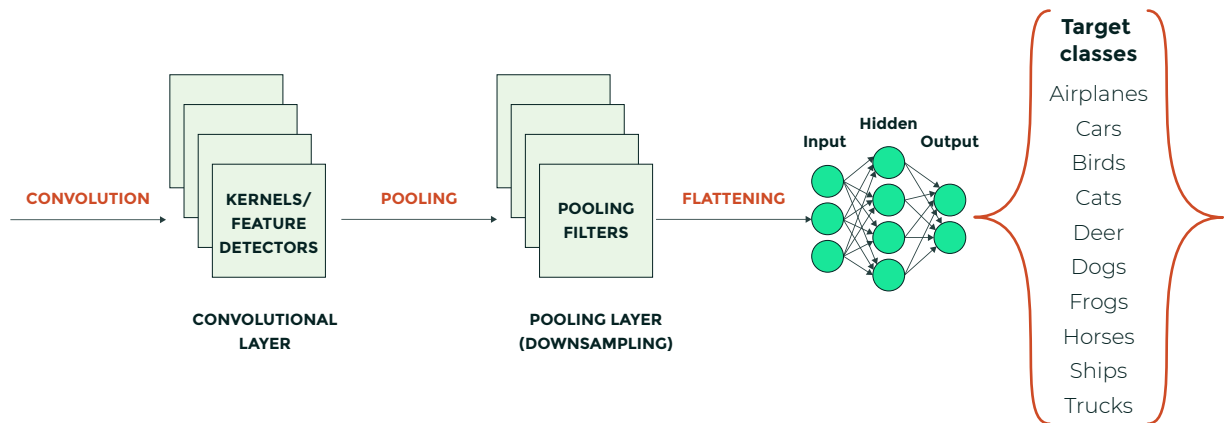
```
>> X_Testing = np.array([[input #1, input #2, input #3,..., input #n]])
>> y_predict = model.predict(X_Testing)
```

## 4. HOW TO BUILD A CONVOLUTIONAL NEURAL NETWORK (CNN)

### A. CONCEPT

**CNNs is a type of deep neural networks that are commonly used for image classification.**

CNNs are formed of (1) Convolutional Layers (Kernels and feature detectors), (2) Activation Functions (RELU), (3) Pooling Layers (Max Pooling or Average Pooling), and (4) Fully Connected Layers (Multi-layer Perceptron Network).



### B. BUILD A CNN USING KERAS:

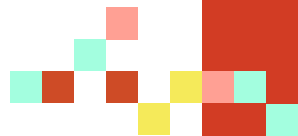
```
>> from keras.models import Sequential
>> from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D,
Dense, Flatten, Dropout
>> from keras.optimizers import Adam
>> from keras.callbacks import TensorBoard

>> cnn_model = Sequential()

>> cnn_model.add(Conv2D(filters = 32, kernel_size=(3,3), activation = 'relu',
input_shape = (32,32,3)))
>> cnn_model.add(Conv2D(filters = 32, kernel_size=(3,3), activation = 'relu'))
>> cnn_model.add(MaxPooling2D(2,2))
>> cnn_model.add(Dropout(0.3))

>> cnn_model.add(Flatten())

>> cnn_model.add(Dense(units = 512, activation = 'relu'))
>> cnn_model.add(Dense(units = 10, activation = 'softmax'))
```

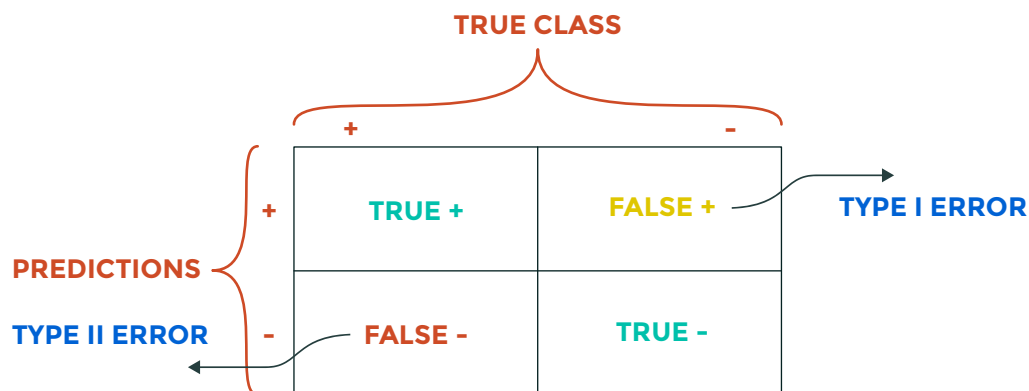


## 5. CONFUSION MATRIX/CLASSIFICATION REPORT

### A. CONCEPT

A confusion matrix is used to describe the performance of a classification model:

- **True positives (TP):** cases when classifier predicted TRUE (has a disease), and correct class was TRUE (patient has disease).
- **True negatives (TN):** cases when model predicted FALSE (no disease), and correct class was FALSE (patient does not have disease).
- **False positives (FP) (Type I error):** classifier predicted TRUE, but correct class was FALSE (patient does not have disease).
- **False negatives (FN) (Type II error):** classifier predicted FALSE (patient do not have disease), but they actually do have the disease



- **Classification Accuracy** =  $(TP+TN) / (TP + TN + FP + FN)$
- **Misclassification rate (Error Rate)** =  $(FP + FN) / (TP + TN + FP + FN)$
- **Precision** =  $TP / \text{Total TRUE Predictions} = TP / (TP+FP)$  (When model predicted TRUE class, how often did it get it right?)
- **Recall** =  $TP / \text{Actual TRUE} = TP / (TP+FN)$  (when the class was actually TRUE, how often did the classifier get it right?)

### B. CONFUSION MATRIX IN SKLEARN

```
>> from sklearn.metrics import classification_report, confusion_matrix
>> y_predict_test = classifier.predict(X_test)
>> cm = confusion_matrix(y_test, y_predict_test)
>> sns.heatmap(cm, annot=True)
```

### C. CLASSIFICATION REPORT

```
>> from sklearn.metrics import classification_report
>> print(classification_report(y_test, y_pred))
```

## 6. HOW TO PERFORM IMAGE AUGMENTATION?


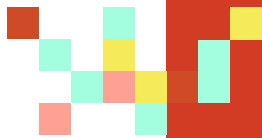
### A. CONCEPT

- **Image Augmentation is the process of artificially increasing the variations of the original images** (training data) by flipping, enlarging, rotating the original images.
- **Image augmentation is performed to artificially increase the size of the training data** therefore enhance the network generalization capability.

### B. CONFUSION MATRIX IN SKLEARN

```
>> from keras.preprocessing.image import ImageDataGenerator
>> datagen = ImageDataGenerator(
    rotation_range = 90,
    width_shift_range = 0.1,
    height_shift_range = 0.1,
    horizontal_flip = True,
    vertical_flip = False,
    rescale = None)

>> datagen.fit(X_train)
>> cnn_model.fit_generator(datagen.flow(X_train, y_train, batch_size=32), epochs=2)
```



## 7. HOW TO BUILD A CONVOLUTIONAL NEURAL NETWORK (CNN)

### A. CONCEPT

**Prophet is open source software released by Facebook's Core Data Science team.**

Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects.

### B. FACEBOOK PROPHET

```
>> from fbprophet import Prophet
>> data_df = data_df.rename(columns = {'Date': 'ds', 'Crime Count': 'y'})
>> m = Prophet()
>> m.fit(data_df)
>> future = m.make_future_dataframe(periods = 720)
>> forecast = m.predict(future)
>> figure = m.plot(forecast)
```

## 8. HOW TO USE KERAS TO BUILD A LE-NET NETWORK?

### A. CONCEPT

**LeNet-5** is a Convolutional Neural Network (CNN) that consists of **7 layers with 3 convolutions, 2 subsampling (pooling), and 1 fully connected dense layer.**

### B. LENET IN KERAS

```
>> cnn_model = Sequential()
>> cnn_model.add(Conv2D(filters = 6, kernel_size=(5,5), activation = 'relu',
input_shape = (32,32,1)))
>> cnn_model.add(AveragePooling2D())
>> cnn_model.add(Conv2D(filters = 16, kernel_size=(5,5), activation = 'relu'))
>> cnn_model.add(AveragePooling2D())
>> cnn_model.add(Flatten())
>> cnn_model.add(Dense(units = 120, activation = 'relu'))
>> cnn_model.add(Dense(units = 84, activation = 'relu'))
>> cnn_model.add(Dense(units = 43, activation = 'softmax'))
```

## 9. HOW TO PERFORM TOKENIZATION?

### A. CONCEPT

Tokenization is a common procedure in natural language processing. **Tokenization works by dividing a sentence into a set of words.** These words are then used to train a machine learning model to perform a certain task.



### B. TOKENIZATION USING SCIKIT-LEARN

```
>> from sklearn.feature_extraction.text import CountVectorizer
>> vectorizer = CountVectorizer()
>> output = vectorizer.fit_transform(data_df)]
```

# 10. HOW TO DEVELOP A SIMPLE MOVIE RECOMMENDER SYSTEM?

## A. CONCEPT

- Recommender systems are algorithms designed to help users discover movies, products, and songs by predicting the user's rating of each item and displaying similar items that they might rate high as well.
- Item-based collaborative filters work by recommending elements based on relationship between items and not people.

## B. OBTAIN CORRELATIONS USING PANDAS

```
>> titanic_correlations = pd.DataFrame(userid_movietitle_matrix.corrwith(titanic),  
columns=['Correlation']) # calculate the correlations  
>> titanic_correlations.dropna(inplace=True) # Remove NaN elements  
>> titanic_correlations.sort_values('Correlation', ascending=False) # Sort correlations vector
```

