SuperDataScience

DATA SCIENCE FOR BUSINESS

6 REAL-WORLD CASE STUDIES

Build 6 Practical Real-world Business Projects and Harness the Power of Data Science and Machine Learning to solve practical, real-world business problems.

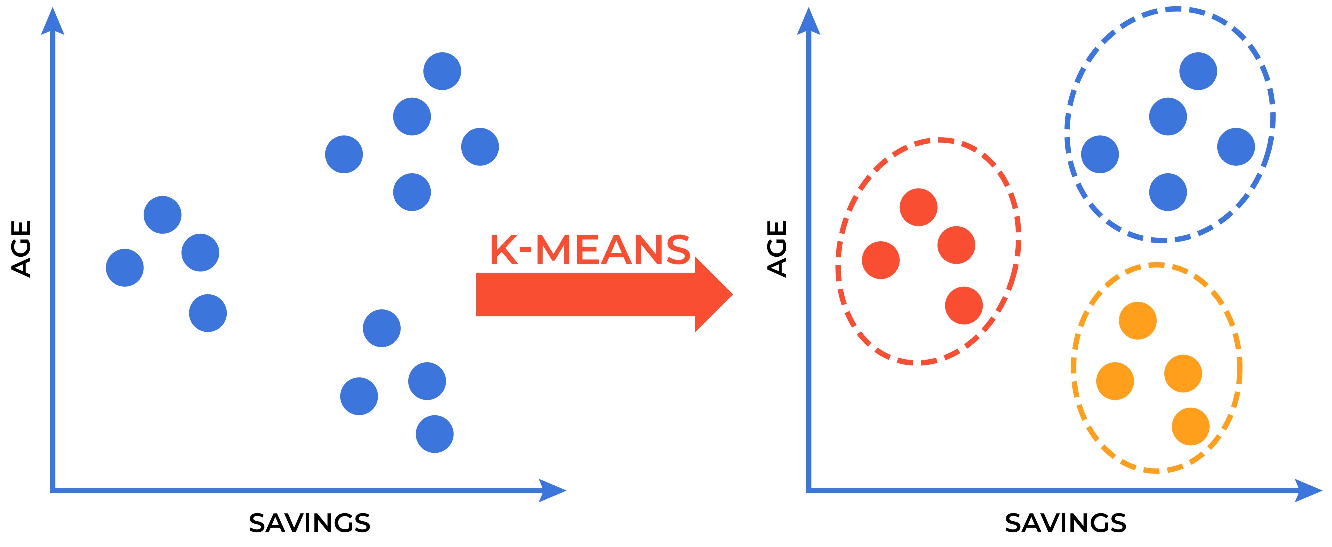
By Dr. Ryan Ahmed, Ph.D., MBA

SuperDataScience Team

1. UNSUPERVISED LEARNING (CLUSTERING) K-MEANS

A. CONCEPT

- K-means is an unsupervised learning algorithm (clustering).
- K-means works by grouping some data points together (clustering) in an unsupervised fashion.
- The algorithm groups observations with similar attribute values together by measuring the Euclidian distance between points.



B. K-MEAN ALGORITHM STEPS

1. Choose number of clusters “K”
2. Select random K points that are going to be the centroids for each cluster
3. Assign each data point to the nearest centroid, doing so will enable us to create “K” number of clusters
4. Calculate a new centroid for each cluster
5. Reassign each data point to the new closest centroid
6. Go to step 4 and repeat.

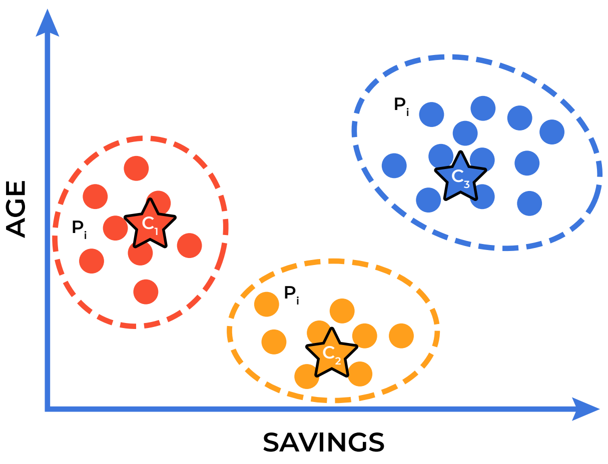
C. HOW TO CHOOSE OPTIMAL NUMBER OF K?

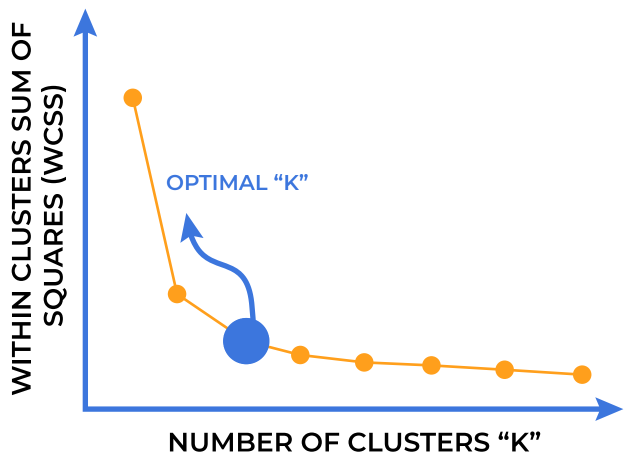
- Calculate the “Within Cluster Sum of Squares (WCSS)” for various values of K (number of clusters).
- Plot the WCSS vs. K and choose the elbow of the curve as the optimal number of clusters to use.

Within Cluster Sum of Squares (WCSS)

$$= \sum_{P_i \text{ in Cluster } 1} \text{distance}(P_i, C_1)^2$$

$$+ \sum_{P_i \text{ in Cluster } 2} \text{distance}(P_i, C_2)^2 + \sum_{P_i \text{ in Cluster } 3} \text{distance}(P_i, C_3)^2$$





D. HOW TO IMPLEMENT K-MEANS IN SCI-KIT LEARN?

```
>> k = 4  #specify the number of clusters
>> kmeans = KMeans(k)
>> kmeans.fit(X_train)
>> labels = kmeans.labels_
```

2. TIME SERIES FORECASTING

A. CONCEPT

- Predictive models attempt at forecasting future sales based on historical data while taking into account seasonality effects, demand, holidays, promotions, and competition.
- Facebook 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.

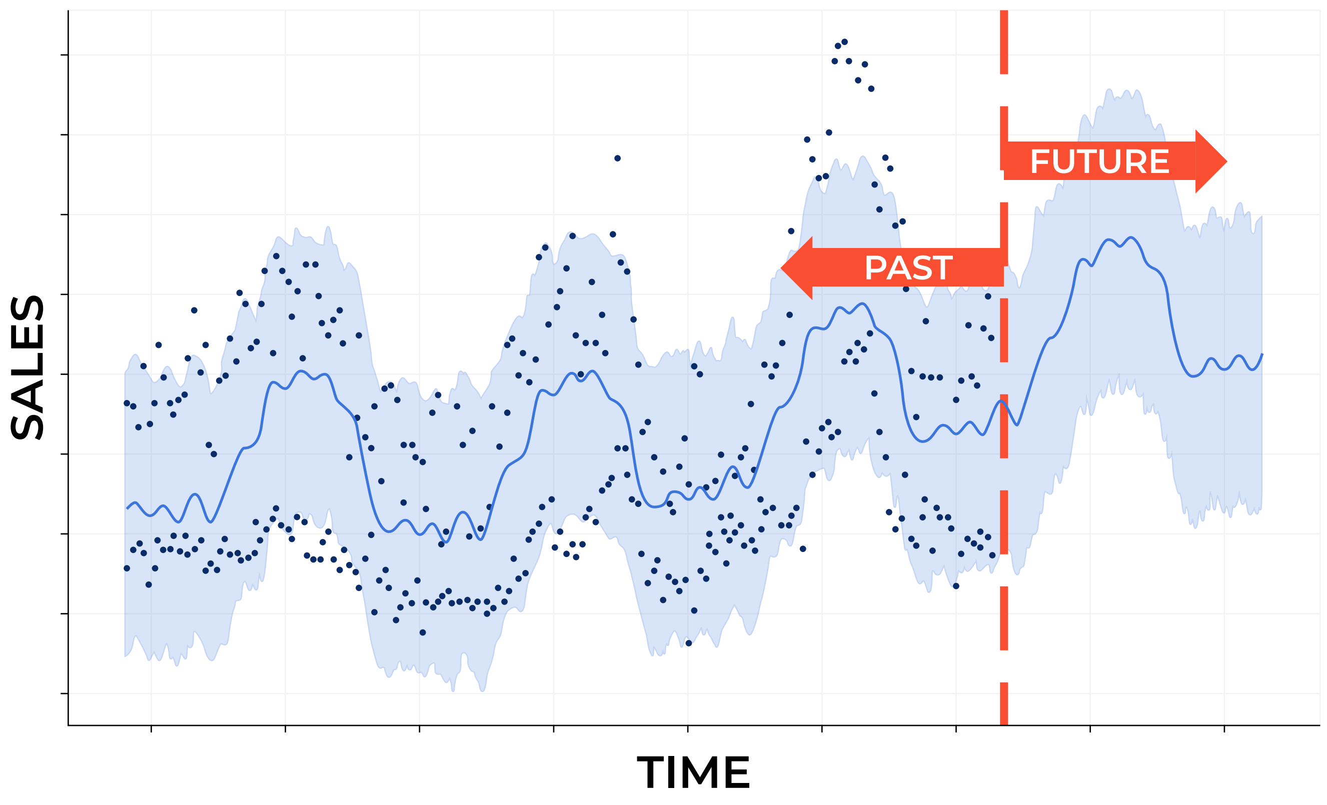
- Prophet implements an additive regression model with four elements:

  1. A piecewise linear, Prophet automatically picks up change points in the data and identifies any change in trends.
  2. A yearly seasonal component modeled using Fourier series.
  3. A weekly seasonal component.
  4. A holiday list that can be manually provided.

- Additive Regression model takes the form:

$$Y = \beta_0 + \sum_{j=1}^{p} f_j(X_j) + \epsilon$$

The functions f_j (x_j) are unknown smoothing functions fit from the data



B. FACEBOOK PROPHET

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

3. REGRESSION TASKS

A. CONCEPT

- Regression is used to predict the value of one variable Y based on another variable X.
- X is called the independent variable and Y is called the dependant variable.
- Simple Linear regression is a statistical model that examines linear relationship between two variables only.
- Multiple Linear Regression examines the relationship between more than two variables.
- Each independent variable has its own corresponding coefficient.

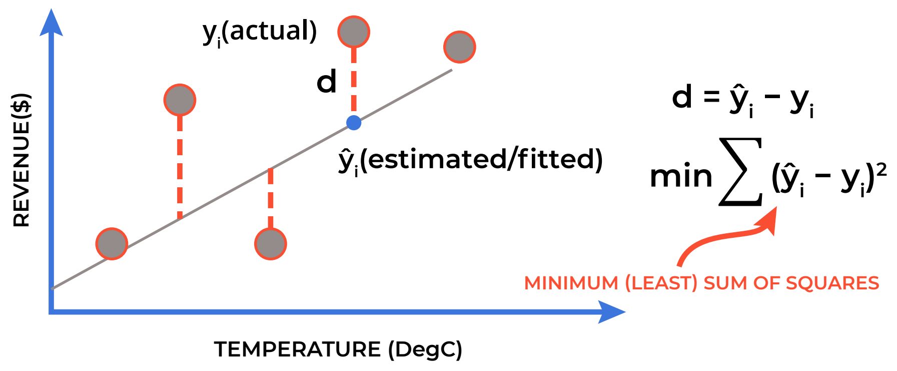
$$y = b_0 + b_1 * x_1 + b_2 * x_2 + .. + b_n x_n$$

DEPENDANT VARIABLES

INDEPENDANT VARIABLES

B. LEAST SQUARES CONCEPT

- Least squares fitting is a way to find the best fit curve or line for a set of points.
- The sum of the squares of the offsets (residuals) are used to estimate the best fit curve or line.
- Least squares method is used to obtain the coefficients m and b.


y_i(actual)
d
$\hat{y}_i$(estimated/fitted)
$d = \hat{y}_i - y_i$
$\min \sum (\hat{y}_i - y_i)^2$
MINIMUM (LEAST) SUM OF SQUARES
REVENUE($)
TEMPERATURE (DegC)

C. IMPLEMENT MULTIPLE LINEAR REGRESSION IN SCIKIT-LEARN

```
>> from sklearn.linear_model import LinearRegression
>> regressor = LinearRegression(fit_intercept =True)
>> regressor.fit(X_train,y_train)
>> print('Linear Model Coefficient (m): ', regressor.coef_)
>> print('Linear Model Coefficient (b): ', regressor.intercept_)
```

D. EVALUATE THE MODEL (MAKE PREDICTIONS USING TRAINED MODEL)

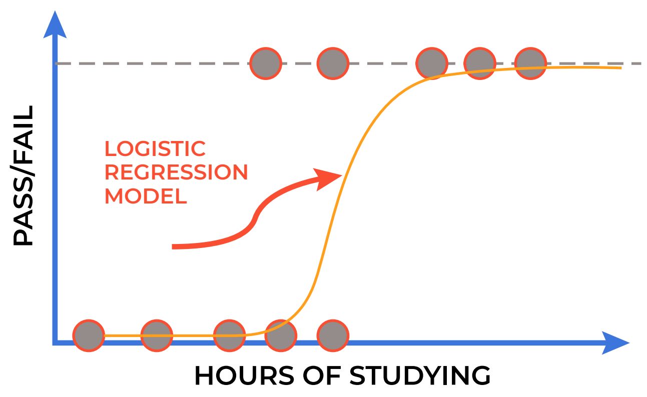
```
>> y_predict = regressor.predict( X_test)
>> y_predict
```

4. CLASSIFICATION TASKS

4.1. CLASSIFICATION USING LOGISTIC REGRESSION

A. LOGISTIC REGRESSION CONCEPT

- Logistic regression algorithm works by implementing a linear equation first with independent predictors to predict a value.
- This value is then converted into a probability that could range from 0 to 1
- Logistic regression can be used as a classification technique by setting a threshold value (Ex: 0.5) to predict binary outputs with two possible values labeled "0" or "1"
- Therefore, Logistic model output can be one of two classes: pass/fail, win/lose, healthy/sick



Linear equation:

$y = b_0 + b_1 * x$

Apply Sigmoid function:

$P(x) = \text{sigmoid}(y)$

$P(x) = \frac{1}{1+e^{-y}}$

$P(x) = \frac{1}{1+e^{-(b_0+b_1*x)}}$

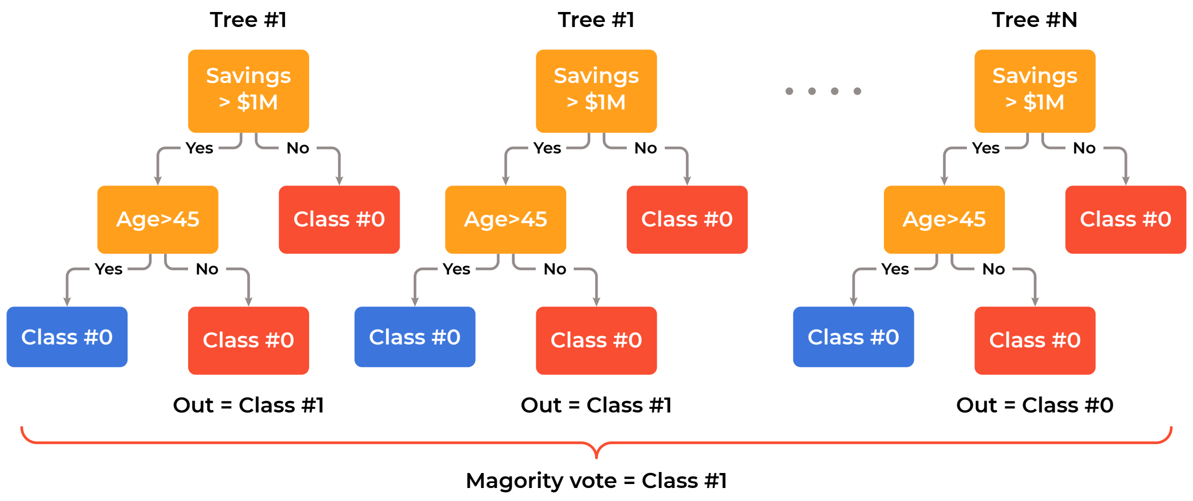
B. LOGISTIC REGRESSION IN SCI-KIT LEARN

```
>> from sklearn.linear_model import LogisticRegression
>> Logistic_Regressor = LogisticRegression(random_state = 0)
>> Logistic_Regressor.fit(X_train, y_train)
```

4.2. CLASSIFICATION USING RANDOM FOREST CLASSIFIER

A. RANDOM FOREST REGRESSION CONCEPT

- Random Forest Classifier is a type of ensemble algorithm.
- It creates a set of decision trees from randomly selected subset of training set.
- It then combines votes from different decision trees to decide the final class of the test object.
- It overcomes the issues with single decision trees by reducing the effect of noise.
- Overcomes overfitting problem by taking average of all the predictions, cancelling out biases.



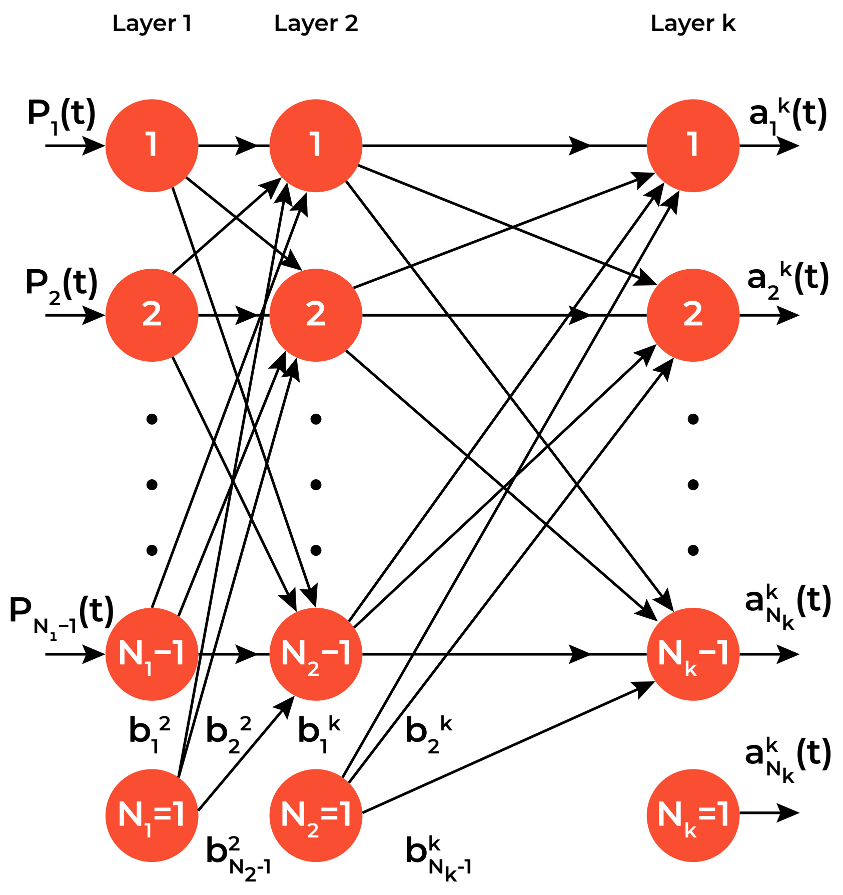
B. RANDOM FOREST CLASSIFIER IN SCI-KIT LEARN

```
>> from import  sklearn.ensemble RandomForestClassifier
>> RandomForest = RandomForestClassifier(n_estimators=250)
>> RandomForest.fit(X_train, y_train)
```

5. DEEP LEARNING AND COMPUTER VISION

A. ARTIFICIAL NEURAL NETWORKS 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:

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

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
```

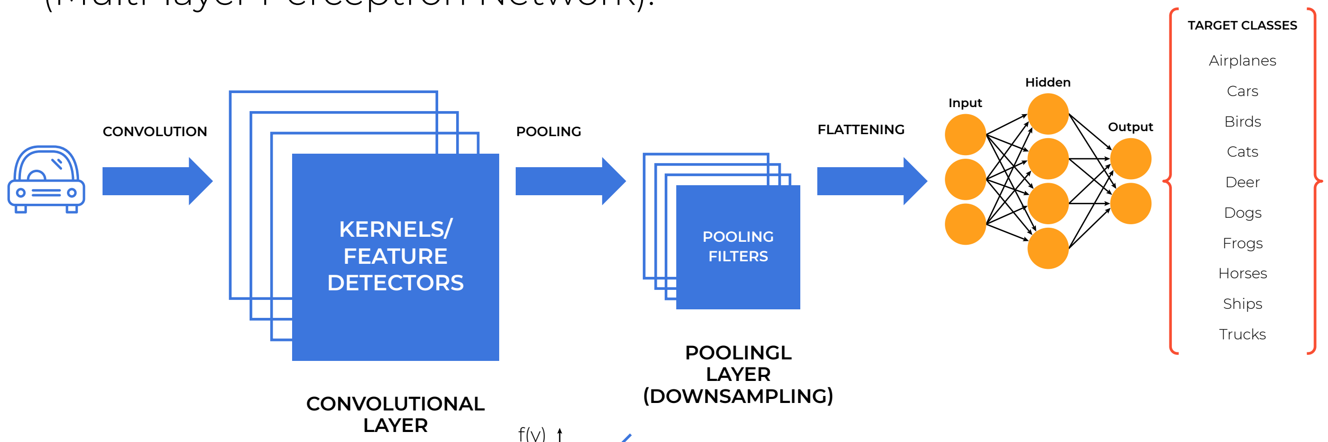
D. EVALUATE THE TRAINED ANN MODEL

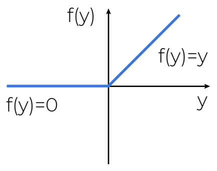
```
>> X_Testing = np.array([[input #1, input #2, input #3,.., input
#n]])
>> y_predict = model.predict(X_Testing)
```

E. HOW TO BUILD A CONVOLUTIONAL NEURAL NETWORK (CNN)

- 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).





F. 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'))
```

6. NATURAL LANGUAGE PROCESSING (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

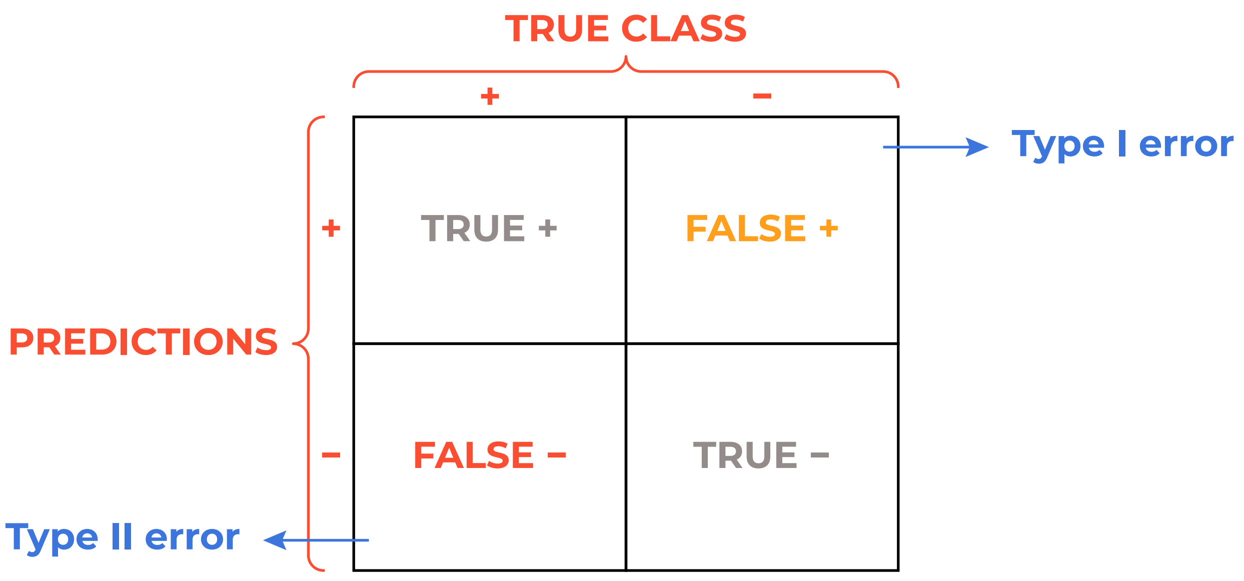
```
>> from sklearn.feature_extraction.text import CountVectorizer
>> vectorizer = CountVectorizer()
>> output = vectorizer.fit_transform(data_df])
```

7. CLASSIFICATION MODELS ASSESSMENT (CONFUSION MATRIX/CLASSIFICATION REPORT

A. CONFUSION MATRIX 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/Total TRUE Predictions = TP/ (TP+FP) (When model predicted TRUE class, how often did it get it right?)
- Recall = TP/ 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))
```

8. REGRESSION MACHINE LEARNING MODELS METRICS

A. MEAN ABSOLUTE ERROR (MAE)

- Mean Absolute Error (MAE) is obtained by calculating the absolute difference between the model predictions and the true (actual) values
- MAE is a measure of the average magnitude of error generated by the regression model
- The mean absolute error (MAE) is calculated as follows:

$$MAE = \frac{1}{n}\sum_{i=1} |y_i - \hat{y}_i|$$

- MAE is calculated by following these steps:

1. Calculate the residual for every data point
2. Calculate the absolute value (to get rid of the sign)
3. Calculate the average of all residuals

- If MAE is zero, this indicates that the model predictions are perfect

B. MEAN SQUARE ERROR (MSE)

- Mean Square Error (MSE) is very similar to the Mean Absolute Error (MAE) but instead of using absolute values, squares of the difference between the model predictions and the training dataset (true values) is being calculated.
- MSE values are generally large compared to the MAE since the residuals are being squared.
- In case of data outliers, MSE will become much larger compared to MAE
- In MSE, error increases in a quadratic fashion while the error increases in proportional fashion in MAE
- The MSE is calculated as follows:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- MAE is calculated by following these steps:

1. Calculate the residual for every data point

2. Calculate the squared value of the residuals
3. Calculate the average of all residuals

C. ROOT MEAN SQUARE ERROR (RMSE)

- Root Mean Square Error (RMSE) represents the standard deviation of the residuals (i.e.: differences between the model predictions and the true values (training data)).
- RMSE can be easily interpreted compared to MSE because RMSE units match the units of the output.
- RMSE provides an estimate of how large the residuals are being dispersed.
- The MSE is calculated as follows:

$$\text{MSE}= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- RMSE is calculated by following these steps:

1. Calculate the residual for every data point
2. Calculate the squared value of the residuals
3. Calculate the average of the squared residuals
4. Obtain the square root of the result

D. MEAN ABSOLUTE PERCENTAGE ERROR (MAPE)

- MAE values can range from 0 to infinity which makes it difficult to interpret the result as compared to the training data.
- Mean Absolute Percentage Error (MAPE) is the equivalent to MAE but provides the error in a percentage form and therefore overcomes MAE limitations.
- MAPE might exhibit some limitations if the data point value is zero (since there is division operation involved)
- The MAPE is calculated as follows:

$$\text{MAPE}=\frac{100\%}{n}\sum_{i=1}^{n}|(y_i-\hat{y}_i)/y_i|$$

E. MEAN PERCENTAGE ERROR (MPE)

- MPE is similar to MAPE but without the absolute operation
- MPE is useful to provide an insight of how many positive errors as compared to negative ones
- The MPE is calculated as follows:

$$\text{MPE}=\frac{100\%}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)/y_i$$