MARKET BASKET ANALYSIS

**MODULE 8 :** **object detection with yolo**

Object detection with YOLO (You Only Look Once) can be a powerful technique for market basket analysis, as it allows you to identify and track products in images or video frames. Here, I'll provide you with a general outline of the steps you would need to follow, and a Python program using the YOLO model from the Darknet framework.

**Step 1: Setup**

Install the necessary libraries: You will need to install OpenCV and the Darknet framework to use YOLO. You can find the installation instructions on the Darknet website.

**Step 2: Download YOLO Pre-trained Weights and Configuration Files**

You can download pre-trained YOLO weights and configuration files from the official Darknet website. These files are trained on a large dataset and can be used as a starting point for your object detection task.

**Step 3: Define Classes**

Define a list of classes that correspond to the products you want to detect in the market basket.

**PYTHON CODE**

classes = ["product1", "product2", "product3", ...]

**Step 4: Load YOLO Model**

Load the YOLO model with the pre-trained weights and configuration file.

**Python Code**

import cv2

net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")

**Step 5: Read and Process Images**

Read an image or frame from a video stream and preprocess it for YOLO input.

**Python Code**

image = cv2.imread("image.jpg")

height, width = image.shape[:2]

blob = cv2.dnn.blobFromImage(image, 1/255.0, (416, 416), swapRB=True, crop=False)

net.setInput(blob)

**Step 6: Perform Object Detection**

Run the YOLO model on the preprocessed image.

**Python Code**

layer\_names = net.getUnconnectedOutLayersNames()

outs = net.forward(layer\_names)

**Step 7: Parse and Display Results**

Parse the YOLO output and draw bounding boxes on the image for detected products.

**Python Code**

class\_ids = []

confidences = []

boxes = []

for out in outs:

for detection in out:

scores = detection[5:]

class\_id = np.argmax(scores)

confidence = scores[class\_id]

if confidence > 0.5:

center\_x = int(detection[0] \* width)

center\_y = int(detection[1] \* height)

w = int(detection[2] \* width)

h = int(detection[3] \* height)

x = center\_x - w // 2

y = center\_y - h // 2

class\_ids.append(class\_id)

confidences.append(float(confidence))

boxes.append([x, y, w, h])

# Apply non-maximum suppression to remove overlapping boxes

indices = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

for i in indices:

i = i[0]

box = boxes[i]

x, y, w, h = box

label = str(classes[class\_ids[i]])

confidence = confidences[i]

cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

cv2.putText(image, f"{label} {confidence:.2f}", (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

cv2.imshow("Market Basket Analysis", image)

cv2.waitKey(0)

cv2.destroyAllWindows()

**MODULE 9 : Recurrent Neural Networks**

Recurrent Neural Networks (RNNs) can be used for market basket analysis when dealing with sequential data, such as analyzing the order history of customers or the sequence of products added to a shopping cart. In this example, I'll provide a Python program using a simple RNN for market basket analysis with synthetic data.

**Here's a step-by-step guide:**

**Step 1: Data Preparation**

Prepare your dataset, which should consist of sequences of products added to the basket over time. Each sequence should be associated with a target variable, such as whether a particular product was purchased in the next time step.

**PYTHON CODE**

import numpy as np

# Create a synthetic dataset

data = [

[1, 2, 3, 4],

[2, 3, 5],

[1, 4, 5],

[1, 2],

[3, 4, 5]

]

# Create corresponding labels (e.g., 1 if product 5 is purchased in the next time step, 0 otherwise)

labels = [1, 0, 1, 0, 1]

**Step 2: Data Preprocessing**

You may need to preprocess the data, such as one-hot encoding the product IDs and padding sequences to a fixed length.

**PYTHON CODE**

from tensorflow.keras.preprocessing.sequence import pad\_sequences

# Convert sequences to one-hot encoding

num\_products = 5 # Assuming there are 5 unique products

data = [np.eye(num\_products)[seq] for seq in data]

# Pad sequences to a fixed length (if needed)

max\_seq\_length = 5 # Adjust as per your dataset

data = pad\_sequences(data, maxlen=max\_seq\_length, padding='post')

**Step 3: Define the RNN Model**

Define an RNN model, such as a SimpleRNN, LSTM, or GRU, and compile it.

**PYTHON CODE**

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import SimpleRNN, Dense

model = Sequential()

model.add(SimpleRNN(32, input\_shape=(max\_seq\_length, num\_products)))

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

**Step 4: Training the Model**

Train the RNN model using your preprocessed data and labels.

**PYTHON CODE**

model.fit(data, labels, epochs=10)

**Step 5: Make Predictions**

Use the trained model to make predictions on new sequences.

**PYTHON CODE**

new\_sequence = np.eye(num\_products)[[1, 2, 3]] # Replace with your new sequence

prediction = model.predict(np.expand\_dims(new\_sequence, axis=0))

if prediction > 0.5:

print("Product 5 is likely to be purchased next.")

else:

print("Product 5 is not likely to be purchased next.")

**MODULE 10 : Natural Language Processing**

Natural Language Processing (NLP) can be used for market basket analysis when dealing with textual data, such as customer reviews, product descriptions, or purchase notes. In this example, I'll provide a Python program using NLP techniques and libraries to perform market basket analysis with text data.

**Here's a step-by-step guide:**

**Step 1: Data Preparation**

Prepare your text dataset. This can be customer reviews, product descriptions, or any textual data that is relevant to market basket analysis.

**Python Code**

import pandas as pd

# Sample dataset

data = pd.DataFrame({

'text': ["Customer 1 bought product A and product B.",

"Customer 2 purchased product B and product C.",

"Product A received positive reviews.",

"Product B is out of stock now."]

})

**Step 2: Text Preprocessing**

Preprocess the text data to clean, tokenize, and prepare it for NLP analysis

**Python Code**

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

nltk.download('punkt')

nltk.download('stopwords')

# Text preprocessing function

def preprocess\_text(text):

text = text.lower() # Convert to lowercase

words = word\_tokenize(text) # Tokenize the text

words = [word for word in words if word.isalpha()] # Remove punctuation

words = [word for word in words if word not in stopwords.words('english')] # Remove stopwords

return " ".join(words)

data['cleaned\_text'] = data['text'].apply(preprocess\_text)

**Step 3: Feature Extraction**

Convert the preprocessed text into numerical features using techniques like TF-IDF or word embeddings.

**Python Code**

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf\_vectorizer = TfidfVectorizer()

tfidf\_matrix = tfidf\_vectorizer.fit\_transform(data['cleaned\_text'])

**Step 4: Market Basket Analysis**

Apply a market basket analysis technique such as association rule mining (e.g., Apriori algorithm) to find associations between products or words.

**Python Code**

from mlxtend.frequent\_patterns import apriori

from mlxtend.frequent\_patterns import association\_rules

# Convert the TF-IDF matrix to a binary matrix

basket = pd.DataFrame(tfidf\_matrix.toarray(), columns=tfidf\_vectorizer.get\_feature\_names\_out())

# Find frequent item sets (replace min\_support and use\_colnames as needed)

frequent\_itemsets = apriori(basket, min\_support=0.2, use\_colnames=True)

# Generate association rules

association\_rules\_df = association\_rules(frequent\_itemsets, metric="lift", min\_threshold=1.0)

**Step 5: Analyze and Interpret Results**

Analyze the association rules to identify interesting relationships between products or words.

**Python Code**

# Filter and sort association rules by lift

interesting\_rules = association\_rules\_df[association\_rules\_df['lift'] > 1.0]

interesting\_rules = interesting\_rules.sort\_values(by='lift', ascending=False)

print(interesting\_rules)

**PROJECT BY:**

**NAME**  **: KIRUBAKARAN S**

**REG NO : 420121104028**

**DEPT**  **: CSE III YEAR**

**COLLEGE**   **:** **AKT MEMORIAL COLLEGE OF ENGINEERING &** **TECHNOLOGY**

**COLLEGE CODE: 4201**

**IBM GROUP 5 : Artificial Intelligence**