**Project Development Phase**

**Model Performance Test**

|  |  |
| --- | --- |
| Date | 10 November 2022 |
| Team ID | Team-592275 |
| Project Name | Gilded Emotions: Unearthing Market Sentiments In Gold News |
| Maximum Marks | 10 Marks |

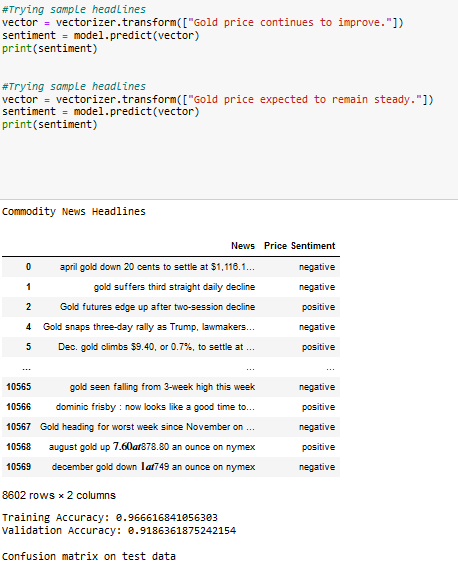
**Model Performance Testing:**

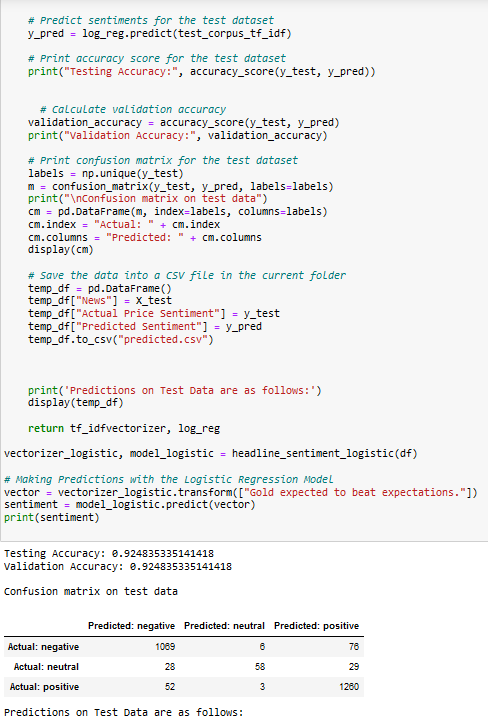
Project team shall fill the following information in model performance testing template.

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No.** | **Parameter** | **Values** | **Screenshot** |
|  | Model Summary | **Model: SVM with TF-IDF Vectorizer** | [SC-1] in the next page |
|  | Accuracy | Training Accuracy: 0.966616841056303  Validation Accuracy: 0.9186361875242154 |  |
| 3. | Confidence Score (Only Yolo Projects) | Class Detected -   Confidence Score - |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No.** | **Parameter** | **Values** | **Screenshot** |
|  | Model Summary | **Model:Logistic Regression model with TF-IDF Vectorizer** | [SC-2] in the next page |
|  | Accuracy | Testing Accuracy: 0.924835335141418  Validation Accuracy: 0.924835335141418 |  |
| 3. | Confidence Score (Only Yolo Projects) | Class Detected -   Confidence Score - |  |

**[SC-1]Model: SVM with TF-IDF Vectorizer [SC-2] Model:Logistic Regression model with TF-IDF Vectorizer**





Validation Accuracy: 0.9186361875242154

Validation Accuracy: 0.924835335141418

Testing Accuracy: 0.924835335141418

Training Accuracy: 0.966616841056303

**Model: SVM with TF-IDF Vectorizer**

#Load packages

import numpy as np

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, f1\_score, recall\_score, precision\_score, confusion\_matrix

from sklearn.svm import SVC

import re

df = pd.read\_csv("gold-dataset-sinha-khandait.csv")

#Let us ignore the news headlines that do not have any price movement information in it, i.e. drop rows with "Price Sentiment" as 'none'

df = df[df["Price Sentiment"] != 'none']

print("Commodity News Headlines")

display(df[["News","Price Sentiment"]])

#The following piece of code is used to clean the headlines

def cleaner(impure\_data):

temp\_list = []

for item in impure\_data:

#finding words which start with @

item = re.sub('@\S+', '', item)

#finding words which start with http

item = re.sub('http\S+\s\*', '', item)

#finding special characters, but not "emoji"

item = re.sub('[%s]' % re.escape("""!"#$%&'()\*+,-./:;<=>?@[\]^\_`{|}~"""), '', item)

temp\_list.append(item)

return temp\_list

# Function to calculate accuracy

def calculate\_accuracy(model, X, y):

y\_pred = model.predict(X)

accuracy = accuracy\_score(y, y\_pred)

return accuracy

# Updated headline\_sentiment function

def headline\_sentiment(df):

headlines = df["News"]

polarity = df["Price Sentiment"].tolist()

# Cleaning headlines i.e. removing @mentions, http(s) links and special characters such as punctuations

clean\_headline = cleaner(headlines)

# Initializing tf-idf vectorizer

tf\_idfvectorizer = TfidfVectorizer(sublinear\_tf=True, use\_idf=True)

# Splitting the data into train and test dataset in 70:30 ratio at random

X\_train, X\_test, y\_train, y\_test = train\_test\_split(clean\_headline, polarity, test\_size=0.3)

train\_corpus\_tf\_idf = tf\_idfvectorizer.fit\_transform(X\_train)

test\_corpus\_tf\_idf = tf\_idfvectorizer.transform(X\_test)

# Using SVC package to initialize a classifier with Linear kernel and other default parameters

SVM\_L = SVC(kernel='linear')

# Fitting the sparse matrix in the classifier with their respective sentiments

SVM\_L.fit(train\_corpus\_tf\_idf, y\_train)

# Calculate training accuracy

training\_accuracy = calculate\_accuracy(SVM\_L, train\_corpus\_tf\_idf, y\_train)

# Calculate validation accuracy

validation\_accuracy = calculate\_accuracy(SVM\_L, test\_corpus\_tf\_idf, y\_test)

# This prints accuracy score for the test dataset

print("Training Accuracy:", training\_accuracy)

print("Validation Accuracy:", validation\_accuracy)

# This prints confusion matrix for the test dataset

labels = np.unique(y\_test)

m = confusion\_matrix(y\_test, SVM\_L.predict(test\_corpus\_tf\_idf), labels=labels)

print("\nConfusion matrix on test data")

cm = pd.DataFrame(m, index=labels, columns=labels)

cm.index = "Actual: " + cm.index

cm.columns = "Predicted: " + cm.columns

display(cm)

# Saving the data into a csv file in the current folder

temp\_df = pd.DataFrame()

temp\_df["News"] = X\_test

temp\_df["Actual Price Sentiment"] = y\_test

temp\_df["Predicted Sentiment"] = SVM\_L.predict(test\_corpus\_tf\_idf)

temp\_df.to\_csv("predicted.csv")

print('Predictions on Test Data are as follows:')

display(temp\_df)

return tf\_idfvectorizer, SVM\_L

# Testing the model

vectorizer, model = headline\_sentiment(df)

# Looking at the confusion matrix, it is clear that the performance on neutral will be poor.

# Positive and negative headlines are likely to be identified correctly

# Testing the model

vector = vectorizer.transform(["Gold expected to beat expectations."])

sentiment = model.predict(vector)

print(sentiment)

#Trying sample headlines

vector = vectorizer.transform(["The price of gold continues declining."])

sentiment = model.predict(vector)

print(sentiment)

#Trying sample headlines

vector = vectorizer.transform(["Gold price continues to improve."])

sentiment = model.predict(vector)

print(sentiment)

#Trying sample headlines

vector = vectorizer.transform(["Gold price expected to remain steady."])

sentiment = model.predict(vector)

print(sentiment)

**Model : Logistic Regression model with TfidfVectorizer**

import numpy as np

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, confusion\_matrix

from sklearn.linear\_model import LogisticRegression

import re

df = pd.read\_csv("gold-dataset-sinha-khandait.csv")

df = df[df["Price Sentiment"] != 'none']

# Data Preprocessing

def cleaner(impure\_data):

temp\_list = []

for item in impure\_data:

# Apply cleaning steps (remove mentions, links, special characters, etc.)

item = re.sub('@\S+', '', item)

item = re.sub('http\S+\s\*', '', item)

item = re.sub('[%s]' % re.escape("""!"#$%&'()\*+,-./:;<=>?@[\]^\_`{|}~"""), '', item)

temp\_list.append(item)

return temp\_list

# Create a Logistic Regression Model

def headline\_sentiment\_logistic(df):

headlines = df["News"]

polarity = df["Price Sentiment"].tolist()

# Clean headlines

clean\_headline = cleaner(headlines)

# Initialize tf-idf vectorizer

tf\_idfvectorizer = TfidfVectorizer(sublinear\_tf=True, use\_idf=True)

# Split data into train and test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(clean\_headline, polarity, test\_size=0.3)

# Transform headlines using tf-idf

train\_corpus\_tf\_idf = tf\_idfvectorizer.fit\_transform(X\_train)

test\_corpus\_tf\_idf = tf\_idfvectorizer.transform(X\_test)

# Initialize Logistic Regression model

log\_reg = LogisticRegression()

# Fit the model with training data

log\_reg.fit(train\_corpus\_tf\_idf, y\_train)

# Predict sentiments for the test dataset

y\_pred = log\_reg.predict(test\_corpus\_tf\_idf)

# Print accuracy score for the test dataset

print("Testing Accuracy:", accuracy\_score(y\_test, y\_pred))

# Calculate validation accuracy

validation\_accuracy = accuracy\_score(y\_test, y\_pred)

print("Validation Accuracy:", validation\_accuracy)

# Print confusion matrix for the test dataset

labels = np.unique(y\_test)

m = confusion\_matrix(y\_test, y\_pred, labels=labels)

print("\nConfusion matrix on test data")

cm = pd.DataFrame(m, index=labels, columns=labels)

cm.index = "Actual: " + cm.index

cm.columns = "Predicted: " + cm.columns

display(cm)

# Save the data into a CSV file in the current folder

temp\_df = pd.DataFrame()

temp\_df["News"] = X\_test

temp\_df["Actual Price Sentiment"] = y\_test

temp\_df["Predicted Sentiment"] = y\_pred

temp\_df.to\_csv("predicted.csv")

print('Predictions on Test Data are as follows:')

display(temp\_df)

return tf\_idfvectorizer, log\_reg

vectorizer\_logistic, model\_logistic = headline\_sentiment\_logistic(df)

# Making Predictions with the Logistic Regression Model

vector = vectorizer\_logistic.transform(["Gold expected to beat expectations."])

sentiment = model\_logistic.predict(vector)

print(sentiment)