Platform used for coding: Google Colab

!pip install pandas numpy scikit-learn matplotlib

#importing the necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.preprocessing import StandardScaler

from google.colab import drive

from keras.applications.vgg16 import VGG16, preprocess\_input

from keras.preprocessing import image

import os

drive.mount('/content/drive')

# Load data

data\_path = '/content/drive/My Drive/Mini-Project/instagram\_data.csv'

data = pd.read\_csv(data\_path)

# Print original image paths to see their current format

print("Original image paths:")

print(data['image\_path'].head())

# Define the base path where images are stored

base\_image\_path = '/content/drive/My Drive/Mini-Project/insta\_data/'

# Clean up the image\_path in case it includes unnecessary relative paths

data['image\_path'] = data['image\_path'].apply(lambda x: os.path.basename(x)) # This takes just the file name, removing any directory structure

# Append the base path to the cleaned image names

data['image\_path'] = data['image\_path'].apply(lambda x: os.path.join(base\_image\_path, x))

# Print the corrected paths

print("Corrected image paths:")

print(data['image\_path'].head())

# Verify the full content of image paths to check for repeated extensions

for path in data['image\_path'].head(10):

print(path)

# Check if the files actually exist

data['file\_exists'] = data['image\_path'].apply(os.path.exists)

# Print paths that do not point to an existing file

print("Paths to non-existent files:")

print(data[data['file\_exists'] == False]['image\_path'])

# Optionally, you might decide to drop entries where images do not exist

# data = data[data['file\_exists'] == True]

data.dropna(inplace=True) #data cleaning

#calculating quantiles for 'likes'

q\_low = data['likes'].quantile(0.01)

q\_high = data['likes'].quantile(0.99)

#filtering data based on 'likes' quantiles

data = data[(data['likes'] > q\_low) & (data['likes'] < q\_high)]

#if 'posted\_time' column exists before proceeding

if 'posted\_time' in data.columns:

#converting 'posted\_time' to datetime if it exists

data['posted\_time'] = pd.to\_datetime(data['posted\_time']) #feature engineering

#extract day of week and hour of the day from timestamp

data['day\_of\_week'] = data['posted\_time'].dt.dayofweek

data['hour\_of\_day'] = data['posted\_time'].dt.hour

else:

#if 'posted\_time' is not present

data['day\_of\_week'] = 0 # Example: Set a default value

data['hour\_of\_day'] = 0 # Example: Set a default value

#scaling the numerical features

scaler = StandardScaler()

data.loc[:, ['follower\_count\_at\_t', 'no\_of\_comments']] = scaler.fit\_transform(data[['follower\_count\_at\_t', 'no\_of\_comments']])

#verifying the full content of image paths to check for repeated extensions

#for path in data['image\_path'].head(10):

# print(path)

# Print the first few entries of the image\_path to understand its content

#print(data['image\_path'].head())

#verifying the full content of image paths to check for repeated extensions

for path in data['image\_path'].head(10):

print(path)

# Load the model

base\_model = VGG16(weights='imagenet', include\_top=False)

print("Model loaded.")

# Function to extract features from an image

def extract\_features(image\_path, model):

print(f"Trying to load image: {image\_path}") # Ensure this line is uncommented to see the paths

if not os.path.exists(image\_path):

print(f"File not found: {image\_path}")

return np.zeros((model.output\_shape[-1],)) # Using model.output\_shape[-1] to match the expected feature size

try:

img = image.load\_img(image\_path, target\_size=(224, 224))

img\_array = image.img\_to\_array(img)

expanded\_img\_array = np.expand\_dims(img\_array, axis=0)

preprocessed\_img = preprocess\_input(expanded\_img\_array)

features = model.predict(preprocessed\_img)

flattened\_features = features.flatten()

return flattened\_features

except Exception as e:

print(f"Error processing image {image\_path}: {e}")

return np.zeros((model.output\_shape[-1],))

# Check a few paths to ensure they're correct

print("Sample image paths:")

print(data['image\_path'].head(5))

# Extract features for each image using the actual paths

features\_arrays = data['image\_path'].apply(lambda x: extract\_features(x, base\_model))

features\_matrix = np.array(features\_arrays.tolist())

# Normalizing the target variable (likes) and other numeric features

data['likes\_log'] = np.log(data['likes'] + 1)

scaler = StandardScaler()

data[['follower\_count\_at\_t', 'no\_of\_comments']] = scaler.fit\_transform(data[['follower\_count\_at\_t', 'no\_of\_comments']])

print("Feature extraction and scaling complete.")

# Prepare the feature set

X\_numeric = data[['follower\_count\_at\_t', 'no\_of\_comments', 't']] # Update or remove 't' as needed if it's a timestamp

X\_combined = np.hstack((X\_numeric, features\_matrix)) # Combine numeric features with image features

# Prepare target variable

y = data['likes\_log']

# Split data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_combined, y, test\_size=0.2, random\_state=42)

# Initialize and train a RandomForest model

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = model.predict(X\_test)

y\_pred\_exp = np.exp(y\_pred) - 1 # Convert back from log scale to original scale for evaluation

mse = mean\_squared\_error(np.exp(y\_test) - 1, y\_pred\_exp)

r2 = r2\_score(np.exp(y\_test) - 1, y\_pred\_exp)

print(f"Mean Squared Error: {mse}")

print(f"R^2 Score: {r2}")

# Visualizing the improved predictions

plt.scatter(np.exp(y\_test) - 1, y\_pred\_exp)

plt.xlabel('Actual Likes')

plt.ylabel('Predicted Likes (exp-transformed)')

plt.title('Actual vs Predicted Likes with RandomForest')

plt.show()

OUTPUTS:

~ **Mean Squared Error (MSE): 7,119,221,703.048768**

~ **R² Score: 0.7408933389266661**

A graph with blue dots

Description automatically generated

What the code does:

**Objective**

The code is designed to predict the number of 'likes' an Instagram post might receive. It uses a combination of numerical data (like follower count and number of comments) and visual cues extracted from post images.

**Steps in the Code**

1. **Data Loading and Preparation**
   * **Load Data**: The code starts by loading a dataset that contains information about Instagram posts, including details like the number of likes, comments, follower count at the time of posting, and the path to the images.
   * **Clean Data**: It removes entries with missing data and filters out extreme values in the number of likes to focus on more typical posts.
2. **Feature Engineering**
   * **Time Features**: It extracts time-based features like the day of the week and hour of the day from the posting time, assuming these factors might influence the number of likes.
   * **Normalize Data**: It scales numerical features such as follower counts and comments to standardize their ranges, making them more comparable and suitable for modeling.
3. **Image Processing**
   * **Model Loading**: A pre-trained model called VGG16, known for its ability to recognize visual patterns, is loaded to help in extracting meaningful features from post images.
   * **Extract Features**: For each image, the model analyzes and converts visual information into a set of features (numerical data that represents the image).
   * **Error Handling**: If an image cannot be found or an error occurs during processing, it is handled gracefully by inserting placeholder values.
4. **Model Training**
   * **Combine Features**: Numerical data from the dataset and extracted image features are combined to form a comprehensive set of inputs for training.
   * **Prepare Data for Prediction**: The 'likes' data is transformed to make it suitable for regression analysis.
   * **Split Data**: The combined dataset is divided into training and testing subsets to enable model validation.
5. **Predictive Modeling**
   * **Random Forest Model**: A type of predictive model known for its robustness and accuracy in handling complex datasets with mixed types of features.
   * **Model Fitting and Evaluation**: The model is trained on the training subset and then used to predict the number of likes on the testing subset. The predictions are compared to the actual number of likes to evaluate the model’s accuracy.
6. **Performance Evaluation**
   * **Metrics Computation**: Metrics like Mean Squared Error (MSE) and R² Score are computed to quantify how well the model predicts the number of likes. MSE measures average prediction error squared, while R² indicates the proportion of variance in the number of likes that is predictable from the features.
7. **Visualization**
   * **Plot Results**: A scatter plot is generated to visually compare the predicted number of likes against the actual likes, providing a visual assessment of the model's accuracy.

**Explanation of Impact**

* **Practical Use**: This model helps in understanding what factors contribute to the popularity of Instagram posts and can be used to optimize content for higher engagement.
* **Business Insight**: Marketers and content creators can use insights from the model to tailor their strategies, potentially increasing their reach and engagement on social platforms like Instagram.

The code essentially turns raw data from Instagram posts into actionable insights by leveraging both numerical analytics and sophisticated image processing techniques, making it a powerful tool for social media analysis.