FAKE INSTAGRAM PROFILE DETECTION USING FEEDFORWARD NEURAL NETWORK

GUIDE

Dr. Kalaivani J

Associate Professor,

Department of Computing Technologies

PANEL HEAD

Dr. Manoranjitham T

Associate Professor,

Department of Computing Technologies

FACULTY ADVISER

Dr. Viji D

Assistant Professor

Department of Computing Technologies

STUDENT DETAILS

1. SAYAN KUMAR BAG [RA2011003010951]
2. SHIVANSH SINGH [RA2011003010950]

CODE

```
import pandas as pd
import numpy as np

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

import tensorflow as tf
from tensorflow import keras
```

```
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import Accuracy

from sklearn import metrics
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report,accuracy_score,roc_curve,confusion_matrix
train_data_path = 'datasets/Insta_Fake_Profile_Detection/train.csv'
test_data_path = 'datasets/Insta_Fake_Profile_Detection/test.csv'

pd.read_csv(train_data_path)

# Load the training dataset
instagram_df_train=pd.read_csv(train_data_path)
instagram_df_train
# Load the testing data
instagram_df_test=pd.read_csv(test_data_path)
instagram_df_test
instagram_df_train.head()
instagram_df_train.tail()

# Getting dataframe info
instagram_df_train.info()

# Get the statistical summary of the dataframe
instagram_df_train.describe()

# Checking if null values exist
instagram_df_train.isnull().sum()
# Get the number of unique values in the "profile pic" feature
instagram_df_train['profile pic'].value_counts()

# Get the number of unique values in "fake" (Target column)
instagram_df_train['fake'].value_counts()
```

```
instagram_df_test.info()

instagram_df_test.describe()

instagram_df_test.isnull().sum()

instagram_df_test['fake'].value_counts()
# Correlation plot
plt.figure(figsize=(20, 20))
cm = instagram_df_train.corr()
ax = plt.subplot()
sns.heatmap(cm, annot = True, ax = ax)
plt.show()

# Preparing Data to Train the Model

# Training and testing dataset (inputs)
X_train = instagram_df_train.drop(columns = ['fake'])
X_test = instagram_df_test.drop(columns = ['fake'])

print(X_train,X_test)



# Training and testing dataset (Outputs)
y_train = instagram_df_train['fake']
y_test = instagram_df_test['fake']

print(y_train,y_test)



# Scale the data before training the model
from sklearn.preprocessing import StandardScaler, MinMaxScaler
scaler_x = StandardScaler()
X_train = scaler_x.fit_transform(X_train)
X_test = scaler_x.transform(X_test)
```

```
y_train = tf.keras.utils.to_categorical(y_train, num_classes = 2)
y_test = tf.keras.utils.to_categorical(y_test, num_classes = 2)

print(y_train,y_test)



# print the shapes of training and testing datasets
X_train.shape, X_test.shape, y_train.shape, y_test.shape

Training_data = len(X_train)/( len(X_test) + len(X_train) ) * 100

Testing_data = len(X_test)/( len(X_test) + len(X_train) ) * 100

print(Training_data, Testing_data)



# Building and Training Deep Training Model
import tensorflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# Adjusted Neural Network Architecture
model = Sequential()
model.add(Dense(100, input_dim=11, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(200, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(100, activation='relu'))
model.add(Dense(2, activation='softmax'))

# Hyperparameter Tuning
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# Data Preprocessing
scaler_x = StandardScaler()
X_train_scaled = scaler_x.fit_transform(X_train)
X_test_scaled = scaler_x.transform(X_test)

# Regularization
from tensorflow.keras import regularizers

model = Sequential()
model.add(Dense(100, input_dim=11, activation='relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dense(200, activation='relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.5))
model.add(Dense(200, activation='relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.5))
model.add(Dense(100, activation='relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dense(2, activation='softmax'))



model.summary()

model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])

epochs_hist = model.fit(X_train, y_train, epochs = 25,  verbose = 1, validation_split = 0.1)

predicted = model.predict(X_test)

predicted_value = []
test = []
for i in predicted:
    predicted_value.append(np.argmax(i))

for i in y_test:
    test.append(np.argmax(i))
```

```
print(classification_report(test, predicted_value))

plt.figure(figsize=(10, 10))
cm=confusion_matrix(test, predicted_value)
sns.heatmap(cm, annot=True)
plt.show()

# Access the Performance of the model

print(epochs_hist.history.keys())

plt.plot(epochs_hist.history['loss'])
plt.plot(epochs_hist.history['val_loss'])

plt.title('Model Loss Progression During Training/Validation')
plt.ylabel('Training and Validation Losses')
plt.xlabel('Epoch Number')
plt.legend(['Training Loss', 'Validation Loss'])
plt.show()
plt.plot(epochs_hist.history['accuracy'])
plt.plot(epochs_hist.history['val_accuracy'])
plt.title('Model Accuracy Progression During Training/Validation')
plt.ylabel('Training and Validation Losses')
plt.xlabel('Epoch Number')
plt.legend(['Training Acc', 'Validation Acc'])
plt.show()

dicts = {
    'Accuracy' : epochs_hist.history['accuracy'],
    'Validation_Accuracy' : epochs_hist.history['val_accuracy'],
    'Loss' : epochs_hist.history['loss'],
    'Validation Loss' : epochs_hist.history['val_loss']

}
model_training_progress = pd.DataFrame(dicts)
```

```
model_training_progress

print(model_training_progress)

def get_avg(lst):
    return sum(lst) / len(lst)

print("Accuracy : ", get_avg(model_training_progress['Accuracy']) * 100)
print("Validation Accuracy : ", get_avg(model_training_progress['Validation_Accuracy']) * 100)
print("Loss : ", get_avg(model_training_progress['Loss']) * 100)
print("Validation Loss : ", get_avg(model_training_progress['Validation Loss']) * 100)
```

SOFTWARE USED

The software tools used in the project are :

1. Visual Studio Code (VSCode):

- VSCode is an integrated development environment (IDE) developed by Microsoft. It provides a wide range of features for coding, debugging, and version control.
- This project utilized VSCode as the primary IDE for writing, editing, and running your Python code.

2. Python:

- Python is a high-level programming language known for its simplicity and readability.
- The entire project, including data manipulation, visualization, and building neural network models, was implemented using Python.

3. Pandas:

- Pandas is a Python library used for data manipulation and analysis. It provides data structures and functions to efficiently handle structured data.
- This project utilized Pandas to load and manipulate the training and testing datasets.

4. NumPy:

- NumPy is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.
- NumPy was used in this project for numerical operations and data transformation.

5. Matplotlib and Seaborn:

- Matplotlib and Seaborn are Python libraries used for data visualization. They offer a wide range of functions to create static, interactive, and publication-quality plots.

- This project utilized Matplotlib and Seaborn to visualize the data distributions, correlation among features, and training progress of the neural network model.

6. **TensorFlow and Keras:**
   - TensorFlow is an open-source machine learning framework developed by Google for building and training neural network models.
   - Keras is a high-level neural networks API that runs on top of TensorFlow, providing a more user-friendly interface for building and training deep learning models.
   - This project used TensorFlow and Keras to define, compile, and train the neural network model for fake Instagram profile detection.
7. **Scikit-learn:**
   - Scikit-learn is a machine learning library for Python that provides simple and efficient tools for data mining and data analysis.
   - Scikit-learn is used for data preprocessing, scaling the features, and evaluating the performance of the trained model using classification metrics such as accuracy, precision, recall, and F1-score.

.

EXECUTION

1. Importing Libraries: Initially, necessary libraries such as pandas, numpy, matplotlib, seaborn, and TensorFlow are imported. These libraries are crucial for data manipulation, visualization, and building neural network models.

2. Loading Data: Two CSV files, one for training data (`train.csv`) and one for testing data (`test.csv`), are loaded into pandas DataFrames (`instagram_df_train` and `instagram_df_test`, respectively).

3. Data Exploration:

- Basic exploration functions like `info()`, `describe()`, and `isnull().sum()` are applied to both training and testing datasets to understand their structure, summary statistics, and presence of missing values.
- Checking the distribution of target classes (`'fake'`) in both datasets.
- Visualizing correlation among features using a heatmap.

4. Data Preparation:

- Separating features (X) and target labels (y) from both training and testing datasets.
- Scaling the features using StandardScaler.
- Converting target labels to categorical using one-hot encoding.

5. Building the Neural Network Model:

- Defining a sequential model architecture using Keras.

- Adding Dense layers with ReLU activation functions.
- Incorporating dropout layers for regularization to prevent overfitting.
- Compiling the model with appropriate loss function, optimizer, and metrics.

6. Training the Model:

- Fitting the model to the training data with a batch size and number of epochs.
- Monitoring the training process by observing loss and accuracy metrics.
- Validating the model's performance using a validation split.

7. Model Evaluation:

- Predicting the target labels for the test dataset.
- Calculating classification metrics such as accuracy, precision, recall, and F1-score using `classification_report`.
- Visualizing the confusion matrix to understand the model's performance on each class.

8. Visualizing Training Progress:

- Plotting the training and validation loss across epochs to observe the model's convergence.
- Plotting the training and validation accuracy across epochs to observe the model's learning progress.

9. Model Performance Summary:

- Constructing a DataFrame to summarize the training progress including accuracy, validation accuracy, loss, and validation loss for each epoch.
- Calculating the average values of these metrics over epochs to assess the overall performance of the model.

10. Printing Results: Displaying the average accuracy, validation accuracy, loss, and validation loss.