**Fake Profile Detection**

**Batch Number: CSD-G13**

**Under the Supervision of Prof. Tintu Vijayan**

**School of Computer Science & Engineering, Presidency University**

**Abstract**

This report investigates methods for detecting fake profiles using various machine-learning techniques. The study employs algorithms such as Random Forest, Gradient Boosting, and XGBoost, utilizing features extracted from user profiles to classify them as genuine or fake. The methodology includes data preprocessing steps, feature extraction, and model evaluation through accuracy, precision, and recall metrics.

**Introduction**

Fake profiles pose significant challenges on social media platforms, impacting user trust and data integrity. Accurately detecting these profiles is crucial for maintaining a safe online environment. This report explores machine learning techniques to identify fake profiles based on user attributes such as follower counts, statuses, and engagement metrics.

**Algorithm Details**

1. **Objective:**
   * The goal of the code is to compare the distributions of various numerical features between genuine and fake user profiles. This visualization helps in understanding how these features differ across the two groups.
2. **Key Components:**
   * **Matplotlib Library:** Used for creating visualizations, specifically histograms, to compare the distributions of user profile features.
   * **DataFrames (df1 and df2):** Represent the datasets for genuine users and fake users, respectively, containing the numerical features of interest.
3. **Process:**
   * Define a list of numerical features to be compared: statuses\_count, followers\_count, friends\_count, favourites\_count, and listed\_count.
   * Create a figure with subplots to accommodate histograms for each feature.
   * For each feature, plot histograms for both genuine and fake users, using different colors and transparency levels for clarity.
   * Set titles and legends for each subplot to enhance readability.
   * Adjust the layout for better spacing and display the final visualizations.

**SOURCE CODE DETAILS**

# Compare distributions for statuses\_count, followers\_count, friends\_count, favourites\_count, and listed\_count.

import matplotlib.pyplot as plt

# List of numerical features to compare

numerical\_features = ['statuses\_count', 'followers\_count', 'friends\_count', 'favourites\_count', 'listed\_count']

# Create a figure for subplots

plt.figure(figsize=(15, 10))

# Loop through the numerical features to create histograms

for i, feature in enumerate(numerical\_features, 1):

plt.subplot(2, 3, i)

plt.hist(df1[feature], bins=30, alpha=0.5, label='Genuine Users')

plt.hist(df2[feature], bins=30, alpha=0.5, label='Fake Users')

plt.title(feature)

plt.legend()

# Adjust layout and display the plots

plt.tight\_layout()

plt.show()

**ALGORITHM DETAILS**

1. **Objective:**
   * The goal of the code is to visualize the distribution of follower counts for genuine and fake user profiles using boxplots. This helps in identifying potential outliers in the data.
2. **Key Components:**
   * **Seaborn Library:** Utilized for creating aesthetically pleasing and informative statistical graphics, specifically boxplots in this case.
   * **DataFrames (df1 and df2):** Represent datasets for genuine users and fake users, containing the follower counts to be analyzed.
3. **Process:**
   * Create a figure to hold the boxplot.
   * Use Seaborn's boxplot function to plot follower counts for both genuine and fake users in a single visualization.
   * Set the x-ticks to label the two groups appropriately.
   * Add a title to the plot for context.
   * Display the boxplot to visualize the distribution and identify outliers.

**SOURCE CODE DETAILS**

# Boxplots for Outliers: Identify outliers in follower counts for both datasets.

import seaborn as sns

import matplotlib.pyplot as plt

# Create a figure for the boxplot

plt.figure(figsize=(12, 6))

# Create a boxplot for followers count

sns.boxplot(data=[df1['followers\_count'], df2['followers\_count']], palette='Set2')

# Set x-ticks for the two groups

plt.xticks([0, 1], ['Genuine Users', 'Fake Users'])

# Add a title to the plot

plt.title('Boxplot of Followers Count')

# Display the plot

plt.show()

**ALGORITHM DETAILS**

1. **Objective:**
   * The goal of the code is to visualize the correlation between various numerical features for both genuine and fake user profiles using heatmaps. This helps in understanding the relationships between different features.
2. **Key Components:**
   * **Seaborn Library:** Used for creating the heatmaps, providing a visual representation of the correlation matrices.
   * **Matplotlib Library:** Utilized for setting up the figure and displaying the plots.
   * **DataFrames (df1 and df2):** Contain the datasets for genuine and fake users, respectively, including the numerical features to be analyzed.
3. **Process:**
   * Create a figure for the heatmap of genuine users and compute the correlation matrix for the specified numerical features.
   * Use Seaborn's heatmap function to plot the correlation matrix, with annotations to show correlation values and a color map for visual clarity.
   * Set a title for the heatmap for context.
   * Repeat the process for the fake users dataset to generate a separate heatmap.
   * Display the heatmaps to visualize correlations.

**SOURCE CODE DETAILS**

# Create a correlation heatmap for Genuine Users

plt.figure(figsize=(12, 8))

correlation\_matrix\_df1 = df1[numerical\_features].corr()

sns.heatmap(correlation\_matrix\_df1, annot=True, cmap='coolwarm')

plt.title('Correlation Heatmap for Genuine Users')

plt.show()

# Create a correlation heatmap for Fake Users

plt.figure(figsize=(12, 8))

correlation\_matrix\_df2 = df2[numerical\_features].corr()

sns.heatmap(correlation\_matrix\_df2, annot=True, cmap='coolwarm')

plt.title('Correlation Heatmap for Fake Users')

plt.show()

**ALGORITHM DETAILS**

1. **Objective:**
   * The goal of the code is to visualize the trend of account creation over time for both genuine and fake user profiles. This helps in understanding the temporal patterns of user registration.
2. **Key Components:**
   * **Pandas Library:** Used for handling and manipulating date-time data, as well as for resampling the data to monthly frequency.
   * **Matplotlib Library:** Utilized for plotting the time series data.
   * **DataFrames (df1 and df2):** Contain the datasets for genuine and fake users, including the creation dates of their accounts.
3. **Process:**
   * Convert the created\_at column in both DataFrames to a datetime format for accurate time series analysis.
   * Create a figure for the plot.
   * Set the created\_at column as the index and resample the data to a monthly frequency, counting the number of accounts created each month.
   * Plot the resampled data for both genuine and fake users, using different labels for clarity.
   * Add a title and labels to the plot, and include a legend to differentiate between the two groups.
   * Display the plot to visualize the trends in account creation over time.

**SOURCE CODE DETAILS**

**# Convert 'created\_at' to datetime format for both datasets**

**df1['created\_at'] = pd.to\_datetime(df1['created\_at'])**

**df2['created\_at'] = pd.to\_datetime(df2['created\_at'])**

**# Create a figure for the time series plot**

**plt.figure(figsize=(15, 6))**

**# Resample and plot the number of accounts created over time**

**df1.set\_index('created\_at').resample('M').size().plot(label='Genuine Users')**

**df2.set\_index('created\_at').resample('M').size().plot(label='Fake Users')**

**# Add title and labels**

**plt.title('Accounts Created Over Time')**

**plt.ylabel('Number of Accounts')**

**plt.legend()**

**# Display the plot**

**plt.show()**

**ALGORITHM DETAILS**

1. **Objective:**
   * **The goal of the code is to visualize and compare the distributions of three key features—statuses count, followers count, and friends count—between genuine and fake user profiles using boxplots. This helps in identifying differences and potential outliers in these features.**
2. **Key Components:**
   * **Seaborn Library: Used for creating the boxplots, providing a clear visual representation of the distributions.**
   * **Matplotlib Library: Utilized for setting up the figure and managing subplots.**
   * **DataFrames (df1 and df2): Contain datasets for genuine and fake users, including the features of interest.**
3. **Process:**
   * **Create a figure for the subplots.**
   * **For each feature (statuses count, followers count, friends count), create a subplot:**
     + **Use Seaborn's boxplot function to plot the data for both genuine and fake users.**
     + **Set x-ticks to label the two groups.**
     + **Add a title to each subplot for context.**
   * **Adjust the layout for better spacing and display the final visualizations.**

**SOURCE CODE DETAILS**

**# Create boxplots for statuses\_count, followers\_count, and friends\_count**

**plt.figure(figsize=(15, 5))**

**# Subplot for statuses\_count**

**plt.subplot(1, 3, 1)**

**sns.boxplot(data=[df1['statuses\_count'], df2['statuses\_count']], palette='Set2')**

**plt.xticks([0, 1], ['Genuine Users', 'Fake Users'])**

**plt.title('Statuses Count Distribution')**

**# Subplot for followers\_count**

**plt.subplot(1, 3, 2)**

**sns.boxplot(data=[df1['followers\_count'], df2['followers\_count']], palette='Set1')**

**plt.xticks([0, 1], ['Genuine Users', 'Fake Users'])**

**plt.title('Followers Count Distribution')**

**# Subplot for friends\_count**

**plt.subplot(1, 3, 3)**

**sns.boxplot(data=[df1['friends\_count'], df2['friends\_count']], palette='Set2')**

**plt.xticks([0, 1], ['Genuine Users', 'Fake Users'])**

**plt.title('Friends Count Distribution')**

**# Adjust layout and display the plots**

**plt.tight\_layout()**

**plt.show()**

**ALGORITHM DETAILS**

1. **Objective:**
   * **The purpose of the code is to read user profile data from two CSV files (one for genuine users and one for fake users) and prepare a combined dataset for analysis. It also creates a corresponding label array indicating the type of user (0 for fake users and 1 for genuine users).**
2. **Key Components:**
   * **Pandas Library: Used for reading CSV files and manipulating data.**
   * **CSV Files:**
     + **users.csv: Contains data for genuine users.**
     + **fusers.csv: Contains data for fake users.**
3. **Process:**
   * **Read the CSV files into Pandas DataFrames.**
   * **Concatenate the two DataFrames to create a single DataFrame containing all user profiles.**
   * **Create a label list where each entry corresponds to the user type (0 for fake users and 1 for genuine users).**
   * **Return the combined DataFrame and the labels.**

**SOURCE CODE DETAILS**

**import pandas as pd**

**def read\_datasets():**

**""" Reads users profile from csv files """**

**users = pd.read\_csv("/content/users.csv") # Read genuine users**

**fake\_users = pd.read\_csv("/content/fusers.csv") # Read fake users**

**# Combine genuine and fake users into a single DataFrame**

**x = pd.concat([users, fake\_users])**

**# Create labels: 0 for fake users, 1 for genuine users**

**y = len(fake\_users) \* [0] + len(users) \* [1]**

**return x, y**

**ALGORITHM DETAILS**

1. **Objective:**
   * **The purpose of the code is to predict the gender of individuals based on their first names using the gender\_guesser library. The function maps gender predictions to numeric codes for further analysis.**
2. **Key Components:**
   * **Pandas Library: Used for data manipulation and handling Series.**
   * **Gender Guesser Library: A library that predicts gender based on names.**
   * **Mapping Dictionary: Converts gender predictions into numeric values.**
3. **Process:**
   * **Initialize a gender detector that is case insensitive.**
   * **Extract the first name from the input Series of names.**
   * **Apply the gender detector to predict gender for each first name.**
   * **Map the predicted genders to numeric codes using a predefined dictionary:**
     + **Female: -2**
     + **Mostly Female: -1**
     + **Unknown: 0**
     + **Mostly Male: 1**
     + **Male: 2**
   * **Convert the mapped values to numeric, coercing errors to NaN, and fill NaN values with 0.**
   * **Return the numeric gender codes.**

**SOURCE CODE DETAILS**

**import pandas as pd**

**import gender\_guesser.detector as gender**

**def predict\_sex(name):**

**sex\_predictor = gender.Detector(case\_sensitive=False) # Initialize gender detector**

**first\_name = name.str.split(' ').str.get(0) # Extract first name**

**sex = first\_name.apply(sex\_predictor.get\_gender) # Predict gender**

**sex\_dict = {**

**'female': -2,**

**'mostly\_female': -1,**

**'unknown': 0,**

**'mostly\_male': 1,**

**'male': 2**

**}**

**# Convert to numeric, handling errors by coercing to NaN and then filling NaN with 0**

**sex\_code = pd.to\_numeric(sex.map(sex\_dict), errors='coerce').fillna(0).astype(int)**

**return sex\_code**

**ALGORITHM DETAILS**

1. **Objective:**
   * **The purpose of the code is to extract relevant features from a DataFrame containing user data. It encodes language and gender information and selects specific columns for further analysis.**
2. **Key Components:**
   * **Pandas Library: Used for data manipulation and handling DataFrames.**
   * **NumPy Library: Utilized to find unique languages in the dataset.**
   * **Function predict\_sex: Used to predict the gender of users based on their names.**
3. **Process:**
   * **Create a list of unique languages and their corresponding indices.**
   * **Construct a dictionary that maps each language to a unique integer code.**
   * **Map the language codes to the DataFrame and create a new column lang\_code.**
   * **Use the predict\_sex function to generate a sex\_code based on user names.**
   * **Select specific feature columns to retain in the DataFrame.**
   * **Return the modified DataFrame containing only the selected features.**

**SOURCE CODE DETAILS**

**import numpy as np**

**def extract\_features(x):**

**# Create a mapping of unique languages to integer codes**

**lang\_list = list(enumerate(np.unique(x['lang'])))**

**lang\_dict = {name: i for i, name in lang\_list}**

**# Map languages to codes and create a new column**

**x.loc[:, 'lang\_code'] = x['lang'].map(lambda x: lang\_dict[x]).astype(int)**

**# Predict sex and create a new column**

**x.loc[:, 'sex\_code'] = predict\_sex(x['name'])**

**# Define the columns to retain**

**feature\_columns\_to\_use = [**

**'statuses\_count',**

**'followers\_count',**

**'friends\_count',**

**'favourites\_count',**

**'listed\_count',**

**'sex\_code',**

**'lang\_code'**

**]**

**# Filter the DataFrame to include only the selected columns**

**x = x.loc[:, feature\_columns\_to\_use]**

**return x**

**ALGORITHM DETAILS**

1. **Objective:**
   * **The purpose of the code is to visualize the performance of a classification model by plotting a confusion matrix, which shows the true vs. predicted classifications.**
2. **Key Components:**
   * **Matplotlib Library: Used for plotting the confusion matrix.**
   * **NumPy Library: Utilized for handling numerical operations, such as creating tick marks.**
3. **Process:**
   * **Define the target names for the classes (e.g., 'Fake' and 'Genuine').**
   * **Display the confusion matrix using plt.imshow() with a specified color map.**
   * **Set the title and color bar for the plot.**
   * **Create tick marks for the x and y axes based on the number of target classes.**
   * **Label the axes appropriately (true label and predicted label).**
   * **Adjust layout for better visualization.**

**SOURCE CODE DETAILS**

**import matplotlib.pyplot as plt**

**import numpy as np**

**def plot\_confusion\_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):**

**target\_names = ['Fake', 'Genuine'] # Define class names**

**plt.imshow(cm, interpolation='nearest', cmap=cmap) # Plot the confusion matrix**

**plt.title(title) # Set the title**

**plt.colorbar() # Add a color bar**

**tick\_marks = np.arange(len(target\_names)) # Create tick marks**

**plt.xticks(tick\_marks, target\_names, rotation=45) # Set x-axis ticks**

**plt.yticks(tick\_marks, target\_names) # Set y-axis ticks**

**plt.tight\_layout() # Adjust layout**

**plt.ylabel('True label') # Label for y-axis**

**plt.xlabel('Predicted label') # Label for x-axis**

**ALGORITHM DETAILS**

1. **Objective:**
   * **The purpose of the code is to visualize the performance of a binary classification model using the Receiver Operating Characteristic (ROC) curve, which illustrates the trade-off between the true positive rate and false positive rate.**
2. **Key Components:**
   * **Matplotlib Library: Used for plotting the ROC curve.**
   * **Scikit-learn Library: Utilized for calculating the ROC curve and AUC (Area Under the Curve).**
   * **Inputs:**
     + **y\_test: True labels for the test set.**
     + **y\_pred: Predicted probabilities or scores for the positive class.**
3. **Process:**
   * **Calculate the false positive rate, true positive rate, and thresholds using the roc\_curve function.**
   * **Print the false positive and true positive rates for reference.**
   * **Compute the AUC using the auc function.**
   * **Plot the ROC curve with labels and a diagonal line representing random guessing.**
   * **Set axis limits and labels for clarity.**
   * **Display the plot.**

**SOURCE CODE DETAILS**

**import matplotlib.pyplot as plt**

**from sklearn.metrics import roc\_curve, auc**

**def plot\_roc\_curve(y\_test, y\_pred):**

**# Calculate false positive rate and true positive rate**

**false\_positive\_rate, true\_positive\_rate, thresholds = roc\_curve(y\_test, y\_pred)**

**print("False Positive rate: ", false\_positive\_rate)**

**print("True Positive rate: ", true\_positive\_rate)**

**# Calculate AUC**

**roc\_auc = auc(false\_positive\_rate, true\_positive\_rate)**

**# Plot ROC curve**

**plt.title('Receiver Operating Characteristic')**

**plt.plot(false\_positive\_rate, true\_positive\_rate, 'b', label='AUC = %0.2f' % roc\_auc)**

**plt.legend(loc='lower right')**

**plt.plot([0, 1], [0, 1], 'r--') # Diagonal line**

**plt.xlim([-0.1, 1.2]) # Set x-axis limits**

**plt.ylim([-0.1, 1.2]) # Set y-axis limits**

**plt.ylabel('True Positive Rate') # Label for y-axis**

**plt.xlabel('False Positive Rate') # Label for x-axis**

**plt.show() # Display the plot**

**ALGORITHM DETAILS**

1. **Objective:**
   * **The purpose of the code is to train a feedforward neural network classifier using input features and target labels, and to make predictions on the dataset.**
2. **Key Components:**
   * **Neurolab Library: Used for creating and training neural networks.**
   * **Pandas Library: Utilized for data manipulation and handling DataFrames.**
   * **NumPy Library: Used for numerical operations and array manipulations.**
3. **Process:**
   * **Remove features with zero variance from the input DataFrame.**
   * **Convert the input features (X) and target labels (y) into appropriate formats for training.**
   * **Define the minimum and maximum values for each input feature to normalize the data.**
   * **Create a feedforward neural network with one hidden layer containing 5 neurons and one output layer.**
   * **Train the neural network using the input data and target labels for a specified number of epochs.**
   * **Make predictions on the input data after training.**
   * **Convert the predictions into binary classes (0 or 1) based on a threshold.**

**SOURCE CODE DETAILS**

**import neurolab as nl**

**import os # Import os module**

**import numpy as np**

**import pandas as pd**

**def train(X, y):**

**""" Trains and predicts dataset with a Neural Network classifier """**

**# Remove features with zero variance**

**X = X.loc[:, X.var() != 0]**

**# Create input and target arrays**

**input\_data = X.values.astype(float) # Convert input data to float**

**target\_data = np.array([y]).reshape(-1, 1) # Reshape target data for neurolab**

**# Define minimum and maximum values for each input feature**

**min\_max\_values = [[min(X.iloc[:, i]), max(X.iloc[:, i])] for i in range(X.shape[1])]**

**# Create a new feedforward neural network**

**fnn = nl.net.newff(min\_max\_values, [5, 1], [nl.trans.TanSig(), nl.trans.PureLin()])**

**# Train the neural network**

**error = fnn.train(input\_data, target\_data, epochs=100, show=10, goal=0.01)**

**# Make predictions on the input data**

**predictions = fnn.sim(input\_data)**

**# Convert predictions to binary classes (0 or 1)**

**predicted\_classes = [1 if pred[0] >= 0.5 else 0 for pred in predictions]**

**return target\_data.flatten(), predicted\_classes**

**ALGORITHM DETAILS**

1. **Objective:**
   * **The purpose of the code is to calculate the percentage error between predicted values and actual values, providing a measure of the accuracy of predictions.**
2. **Key Components:**
   * **NumPy Library: Utilized for numerical operations and array manipulations.**
3. **Process:**
   * **Compute the absolute difference between predicted values (y\_pred) and actual values (y\_actual).**
   * **Sum the absolute errors and divide by the number of actual values to obtain the mean absolute error.**
   * **Multiply the result by 100 to express the error as a percentage.**

**SOURCE CODE DETAILS**

**import numpy as np**

**def percentError(y\_pred, y\_actual):**

**"""**

**Calculates the percentage error between predicted and actual values.**

**"""**

**# Calculate percentage error**

**error = np.sum(np.abs(y\_pred - y\_actual)) / len(y\_actual) \* 100 # Ensure error calculation is correct**

**return error**

**ALGORITHM DETAILS**

1. **Objective:**
   * **The purpose of the code is to train a Gradient Boosting classifier on a given dataset and make predictions on a test set.**
2. **Key Components:**
   * **Scikit-learn Library:**
     + **GradientBoostingClassifier: Used for implementing the Gradient Boosting algorithm.**
     + **train\_test\_split: Used for splitting the dataset into training and testing sets.**
   * **Inputs:**
     + **X: Feature matrix.**
     + **y: Target labels.**
3. **Process:**
   * **Split the dataset into training and testing sets (80% training, 20% testing).**
   * **Initialize the Gradient Boosting model with specified parameters (number of estimators, learning rate, max depth, and random state).**
   * **Fit the model on the training data.**
   * **Predict the target labels for the test data.**
   * **Return the actual and predicted values.**

**SOURCE CODE DETAILS**

**from sklearn.ensemble import GradientBoostingClassifier**

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

**from sklearn.metrics import accuracy\_score**

**# Function for training using Gradient Boosting**

**def train\_gradient\_boosting(X, y):**

**""" Trains and predicts dataset with Gradient Boosting Classifier """**

**# Split the dataset into training and testing sets**

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

**# Initialize the Gradient Boosting model**

**gb\_model = GradientBoostingClassifier(n\_estimators=100, learning\_rate=0.1, max\_depth=3, random\_state=42)**

**# Fit the model**

**gb\_model.fit(X\_train, y\_train)**

**# Make predictions on the test set**

**y\_pred = gb\_model.predict(X\_test)**

**return y\_test, y\_pred**

**# Usage**

**y\_test, y\_pred = train\_gradient\_boosting(X, y)**

**print('Gradient Boosting Classification Accuracy:', accuracy\_score(y\_test, y\_pred))**

**ALGORITHM DETAILS**

1. **Objective:**
   * **The purpose of the code is to train an XGBoost classifier on a given dataset and make predictions on a test set.**
2. **Key Components:**
   * **XGBoost Library: Used for implementing the XGBoost algorithm.**
   * **Scikit-learn Library:**
     + **train\_test\_split: Used for splitting the dataset into training and testing sets.**
     + **accuracy\_score: Used for evaluating the accuracy of the model.**
   * **Inputs:**
     + **X: Feature matrix.**
     + **y: Target labels.**
3. **Process:**
   * **Split the dataset into training and testing sets (80% training, 20% testing).**
   * **Initialize the XGBoost model with specified parameters (disabling label encoder and setting evaluation metric).**
   * **Fit the model on the training data.**
   * **Predict the target labels for the test data.**
   * **Return the actual and predicted values.**

**SOURCE CODE DETAILS**

**import xgboost as xgb**

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

**from sklearn.metrics import accuracy\_score**

**# Function for training using XGBoost**

**def train\_xgboost(X, y):**

**""" Trains and predicts dataset with XGBoost Classifier """**

**# Split the dataset into training and testing sets**

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

**# Initialize the XGBoost model**

**xgb\_model = xgb.XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss')**

**# Fit the model**

**xgb\_model.fit(X\_train, y\_train)**

**# Make predictions on the test set**

**y\_pred = xgb\_model.predict(X\_test)**

**return y\_test, y\_pred**

**# Usage**

**y\_test, y\_pred = train\_xgboost(X, y)**

**print('XGBoost Classification Accuracy:', accuracy\_score(y\_test, y\_pred))**

**Conclusion**

Detecting fake profiles is essential for ensuring the integrity of social media platforms. By leveraging machine learning techniques, this research aims to enhance the accuracy of fake profile detection systems, thereby improving user experience and safety.

**References**

* **Benevenuto, F., Santos, M. M., & Almeida, J. M.** (2010). "Detecting Spammers on Twitter." In *Proceedings of the 2010 IEEE International Conference on Social Computing (SocialCom)*.
  + [Link to Paper](https://ieeexplore.ieee.org/document/5501459)
* **Feng, Y., Huang, Z., & Wang, Y.** (2019). "Fake Profile Detection on Social Media: A Survey." *IEEE Access*, 7, 152292-152308.
  + [Link to Paper](https://ieeexplore.ieee.org/document/8850887)
* **Kumar, S., & Singh, A.** (2018). "A Survey on Fake Profile Detection in Social Media." *International Journal of Computer Applications*, 182(14), 1-6.
  + [Link to Journal](https://www.ijcaonline.org/archives/volume182/number14/30288-2018911793)