FAKE PROFILE DETECTION MODEL USING MACHINE LEARNING ALGORITHM

INTRODUCTION:

In today's digital age, the rise of social media platforms has significantly reshaped how individuals interact, communicate and engage with each other. However, with this newfound connectivity comes the persuasive issue of fake profiles created with malicious intent. These profiles can propagate misinformation, pose threats to user's privacy and security. Machine learning, a subset of artificial intelligence empowers systems to learn patterns and make predictions from data without explicit programming.

ABSTRACT:

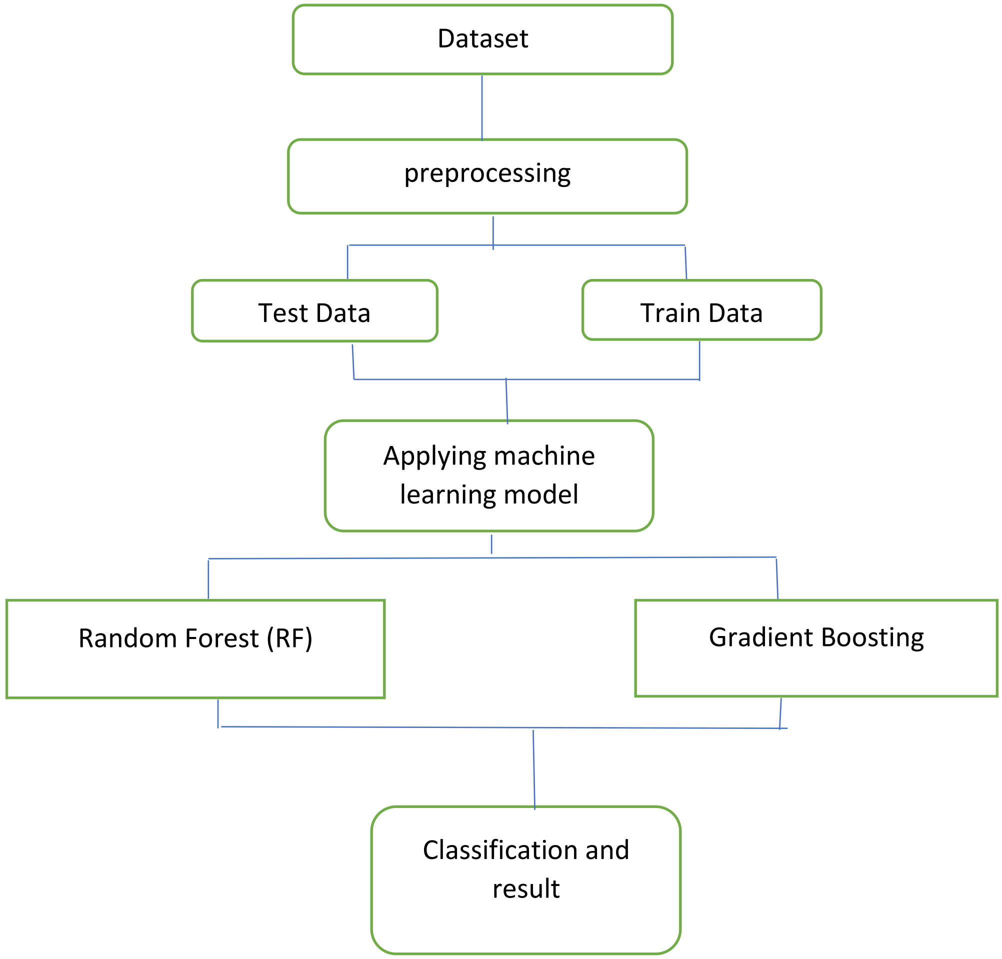
Social media platforms has led to a surge in fake profiles posting significant threats to user's trust and security.
Traditional methods are inadequate, prompting the exploration of machine learning as a solution.
Our project examines the landscape of fake profile detection using machine learning techniques. Various machine learning algorithms are scrutinized for their efficacy in finding out fake profiles. This project aims to provide a comprehensive understanding of machine learning's role in combating fake profiles an upholding the integrity of social media platforms.
The dataset collected from Kaggle is been used in the project.

OBJECTIVE:

The objective encompasses several key goals:

1. **Classification of profiles:** The primary objective is to classify user profiles as either genuine or fake. By analyzing various features such as profile completeness, profile picture quality and network connectivity, machine learning model can assign probability scores or labels to profiles indicating the likelihood of it being fake.
2. **Model training and Evaluation:** The goal is to develop machine learning models to detect genuine and fake users. This involves training models on labeled datasets containing of both real and fake profiles. Evaluation metrics such as accuracy, precision and frequency are used to assess the performance of the models.
3. **Scalability and efficiency:** Another objective is to design algorithms that can scale to handle large volumes of user data efficiently.
4. **Adaptability to Evolving Tactics:** Fake profile creators evolve their strategies to evade detection. Hence, the objective is to develop new machine learning models that can adapt to new tactics and patterns employed by the malicious people.
5. **Integration with platform policies:** Machine learning-based fake profile detection systems should align with the policies and guidelines of the online platform where they are deployed. The objective is to ensure that the detection algorithms comply with user consent requirements, privacy regulations and certain platform rules governing the use of user data.

BLOCK DIAGRAM:



CODE:

In1:

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

In 2:

```
from sklearn.datasets  import load_files
def load_train_data():
```

```
    ''' Load data from csv files, Train data
        Splits data into X (feature matrix) and y (labels)

        returns: X_train, y_train

    '''
    train_data = pd.read_csv('/content/train.csv', header = 0)

    X_train = train_data.drop(columns='fake')
    y_train = train_data['fake']

    return X_train, y_train
```

In 3:

```
from sklearn.datasets import load_files
def load_test_data():
    ''' Load data from csv files, Train and Test data
        Splits data into X (feature matrix) and y (labels)

        returns: X_test, y_test

    '''
    test_data = pd.read_csv('/content/test.csv', header = 0)

    X_test = test_data.drop(columns='fake')
    y_test = test_data['fake']
return X_test, y_test
```

In:4

```
from sklearn.model_selection import cross_validate
def get_classifier_cv_score(model, X, y, scoring='accuracy', cv=7):
    '''Calculate train and validation score of classifier (model) using
cross-validation
```

```
        model (sklearn classifier): Classifier to train and evaluate
        X (numpy.array or pandas.DataFrame): Feature matrix
        y (numpy.array or pandas.Series): Target vector
        scoring (str): a scoring string accepted by
sklearn.metrics.cross_validate()
        cv (int): number of cross-validation folds see
sklearn.metrics.cross_validate()

        returns: mean training score, mean validation score

    '''
    scores = cross_validate(model, X, y, cv=cv, scoring=scoring,
return_train_score=True)
    train_scores = scores['train_score']
    val_scores = scores['test_score']

    train_mean = np.mean(train_scores)
    val_mean = np.mean(val_scores)

    return train_mean, val_mean
```

In 5:

```
def print_grid_search_result(grid_search):
    '''Prints best parameters and mean training and validation scores of a
grid search object.

        grid_search (sklearn GridSearchCV): Fitted GridSearchCV object

        scores are printed with 3 decimal places.

    '''

     #TODO: implement function body

    print(grid_search.best_params_)

    best_train =
grid_search.cv_results_["mean_train_score"][grid_search.best_index_]
    print("best mean_train_score: {:.3f}".format(best_train))
```

```
    best_test =
grid_search.cv_results_["mean_test_score"][grid_search.best_index_]
    print("best mean_test_score: {:.3f}".format(best_test))
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(y_actual, y_pred, labels, title=''):
    '''Creates a heatmap plot of the confusion matrix.

        y_actual (pandas.DataSeries or numpy.Array): Ground truth label
vector
        y_pred (pandas.DataSeries or numpy.Array): Predicted label vector
        labels (list(str)): Class names used for plotting (ticklabels)
        title (str): Plot title

        uses sklearn.metrics.confusion_matrix

    '''
    data = confusion_matrix(y_actual, y_pred)
    ax = sns.heatmap(data,
                     annot=True,
                     cbar=False,
                     fmt='d',
                     xticklabels = labels,
                     yticklabels = labels)
    ax.set_title(title)
    ax.set_xlabel("predicted values")
    ax.set_ylabel("actual values")
```

In 6:

```
X_data, y_data = load_train_data()
print(X_data.info())
```

Out1:

```
Output:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 576 entries, 0 to 575
Data columns (total 11 columns):
 #   Column                 Non-Null Count  Dtype
```

```
---  ------                     --------------  -----
 0   profile pic                576 non-null    int64
 1   nums/length username       576 non-null    float64
 2   fullname words             576 non-null    int64
 3   nums/length fullname       576 non-null    float64
 4   name==username             576 non-null    int64
 5   description length         576 non-null    int64
 6   external URL               576 non-null    int64
 7   private                    576 non-null    int64
 8   #posts                     576 non-null    int64
 9   #followers                 576 non-null    int64
 10  #follows                   576 non-null    int64
dtypes: float64(2), int64(9)
memory usage: 49.6 KB
None
```

In 7:

```
X_data.head()
```

Out 2:

```
Output:
```

	profile pic	nums/length username	fullname words	nums/length fullname	name==username	description length	external URL	private	#posts	#followers	#follows
0	1	0.27	0	0.0	0	53	0	0	32	1000	955
1	1	0.00	2	0.0	0	44	0	0	286	2740	533
2	1	0.10	2	0.0	0	0	0	1	13	159	98
3	1	0.00	1	0.0	0	82	0	0	679	414	651
4	1	0.00	2	0.0	0	0	0	1	6	151	126

In 8:

```
print("Size: ",X_data.shape, ", Type: ", type(X_data))
print("Size: ",y_data.shape, ", Type: ", type(y_data))
```

Out 3:

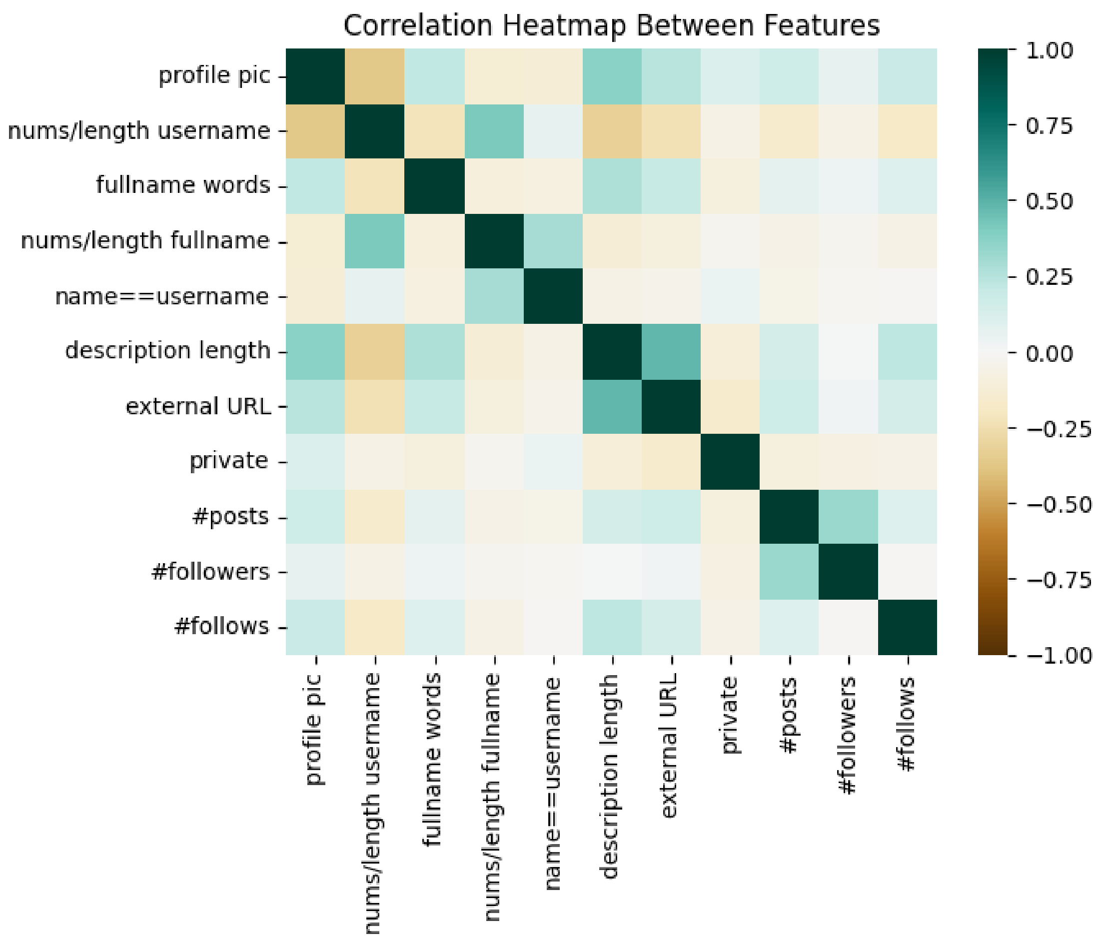
```
output:

Size:  (576, 11) , Type:  <class 'pandas.core.frame.DataFrame'>
Size:  (576,) , Type:  <class 'pandas.core.series.Series'>
```

In 9:

```
data_corr = X_data.corr(method='pearson')
ax = sns.heatmap(data_corr, vmin=-1, vmax=1, cmap='BrBG')
ax.set_title("Correlation Heatmap Between Features")
```

Out: 4

```
Text(0.5, 1.0, 'Correlation Heatmap Between features’)
```



In 10:

```
print(X_data.isnull().sum())
```

Out5:

```
output:
```

```
profile pic             0
nums/length username    0
fullname words          0
nums/length fullname    0
name==username          0
description length      0
external URL            0
private                 0
#posts                  0
#followers              0
#follows                0
      dtype: int64
```

In 11:

```
unique, freq = np.unique(y_data, return_counts = True)

for i, j in zip(unique, freq):
    print("Label: ", i, ", Frequency: ", j)
```

Out 6:

```
output:

Label:  0 , Frequency:  288
Label:  1 , Frequency:  288
```

In 12:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data,
test_size=0.2, random_state=37)
```

In 13:

```
print(X_train.shape)
print(y_train.shape)
```

Out 7:

```
output:
(460, 11)
(460,)
```

In 14:

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
```

```python
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB

model_list = [LogisticRegression(max_iter=600),
              SVC(),
              GaussianNB(),
              RandomForestClassifier(random_state=55),
              GradientBoostingClassifier(random_state=56)]
```

```python
train_scores = []
val_scores = []

for model in model_list:
    train, val = get_classifier_cv_score(model, X_train,
y_train,'average_precision')
    train_scores.append(train)
    val_scores.append(val)

models_score = sorted(list(zip(val_scores, train_scores, model_list)),
reverse=True)

print("-------------------------------------")
for val, train, model in models_score:
    print("Model: {} ".format(model.__class__.__name__))

    print("train_score: {:.3f}".format(train))

    print("validation_score: {:.3f}".format(val))

    print("-------------------------------------")
```

Out 8:

```
output:
```

Model: RandomForestClassifier
train_score: 1.000
validation_score: 0.984

Model: GradientBoostingClassifier
train_score: 1.000
validation_score: 0.979

Model: LogisticRegression
train_score: 0.981
validation_score: 0.975

Model: SVC
train_score: 0.929
validation_score: 0.928

Model: GaussianNB
train_score: 0.804
validation_score: 0.821

In 15:

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
import os
model = RandomForestClassifier(random_state=55)
parameters = {'n_estimators': [300, 500, 700, 1000],
              'max_depth': [7, 9, 11, 13]}
#if run on own computer
#grid1 = GridSearchCV(model, parameters, cv=7, scoring='average_precision'
, n_jobs=num_cpu, return_train_score=True)
grid1 = GridSearchCV(model, parameters, cv=7, scoring='average_precision',
return_train_score=True)
```

In 16:

```
grid1.fit(X_train, y_train)
```

Out 9:

```
output:
```

```
GridSearchCV
GridSearchCV(cv=7, estimator=RandomForestClassifier(random_state=55),
             param_grid={'max_depth': [7, 9, 11, 13],
                         'n_estimators': [300, 500, 700, 1000]},
             return_train_score=True, scoring='average_precision')
estimator: RandomForestClassifier
RandomForestClassifier(random_state=55)
RandomForestClassifier
RandomForestClassifier(random_state=55)
```

In 17:

```
print_grid_search_result(grid1)
```

Out 10:

```
output:

{'max_depth': 7, 'n_estimators': 700}
best mean_train_score: 1.000
best mean_test_score: 0.985
```

In 18:

```
model = GradientBoostingClassifier(max_depth=5, random_state=56)

parameters = {'n_estimators': [50, 100, 200],
              'learning_rate': [0.001, 0.01, 0.1, 1.0, 10.0]}

#if run on own computer
#grid2 = GridSearchCV(model, parameters, cv=7,
scoring='average_precision', n_jobs=num_cpu, return_train_score=True)
grid2 = GridSearchCV(model, parameters, cv=7, scoring='average_precision',
return_train_score=True)
```

In 19:

```
grid2.fit(X_train, y_train)
```

Out 11:

output:

```
GridSearchCV
GridSearchCV(cv=7,
             estimator=GradientBoostingClassifier(max_depth=5, random_state=56),
             param_grid={'learning_rate': [0.001, 0.01, 0.1, 1.0, 10.0],
                         'n_estimators': [50, 100, 200]},
             return_train_score=True, scoring='average_precision')
estimator: GradientBoostingClassifier
GradientBoostingClassifier(max_depth=5, random_state=56)
GradientBoostingClassifier
GradientBoostingClassifier(max_depth=5, random_state=56)
```

In 20:

```
print_grid_search_result(grid2)
```

Out 12:

output:

```
{'learning_rate': 1.0, 'n_estimators': 50}
best mean_train_score: 1.000
best mean_test_score: 0.981
```

In 21:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

pipeline = Pipeline([('preprocessing', StandardScaler()), ('classifier',
grid1.best_estimator_)])
pipeline.fit(X_train, y_train)
```

In 22:

```
print("Test score: {:.3f}".format(pipeline.score(X_test, y_test)))
```

Out 13:

output:

```
Test score: 0.922
```

In 23:

```
X_final, y_final = load_test_data()

print("Test score: {:.3f}".format(pipeline.score(X_final, y_final)))
```

Out 14:

```
output:
Test score: 0.908
```

In 24:

```
from sklearn.metrics import classification_report
y_pred = pipeline.predict(X_final)
print(classification_report(y_final, y_pred, target_names=["genuine",
"fake"]))
```

Out 15:

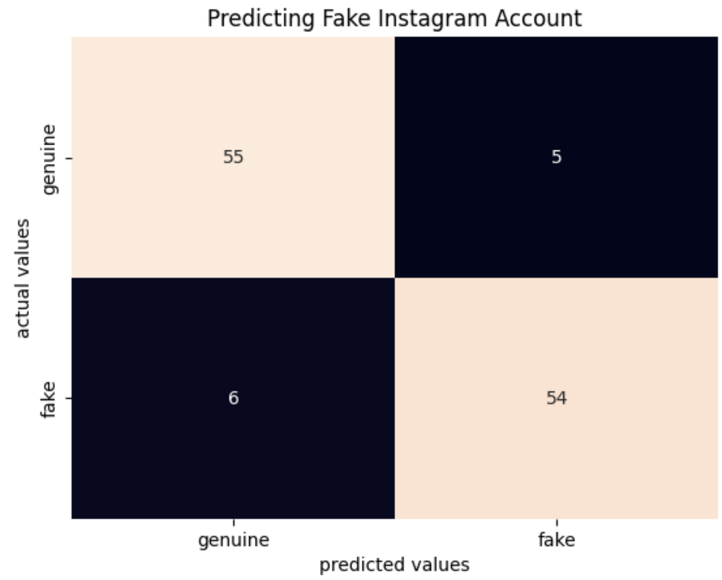
```
output:


              precision    recall  f1-score   support

     genuine       0.90      0.92      0.91        60
        fake       0.92      0.90      0.91        60

    accuracy                           0.91       120
   macro avg       0.91      0.91      0.91       120
weighted avg       0.91      0.91      0.91       120
```

In 25:

```
labels = ["genuine", "fake"]
title = "Predicting Fake Instagram Account"
plot_confusion_matrix(y_final, y_pred, labels, title)
```

Out 16:

```
output:
```



In 26:

```
from sklearn.ensemble import RandomForestClassifier
import numpy as np
import pandas as pd

# Define feature names and the features array
feature_names = ['profile pic', 'nums/length username', 'fullname words',
'fullname words', 'nums/length fullname', 'name==username', 'description
length', 'external URL', 'private', '#posts', '#followers', '#follows']
features = np.array([[1, 1, 100000.0, 100000.0, 0.0, 0.0, 0.0,
0,1,0.0,1,0.0]])

# Convert features array to DataFrame
features_df = pd.DataFrame(features, columns=feature_names)

# Assuming labels is a numpy array containing corresponding labels
labels = np.array([1])  # Assuming a single label for the provided
features

# Create and fit a Random Forest Classifier
rf_classifier = RandomForestClassifier()
rf_classifier.fit(features_df, labels)

# Now, you can make predictions using the fitted classifier
predictions = rf_classifier.predict(features_df)
```

```python
# Output the predictions
if predictions[0] == 1:
    print("Prediction: Genuine Profile")
else:
    print("Prediction: Fake Profile")
```

Out 17:

```
output:

Prediction: Genuine Profile
```

APPLICATIONS:

Some potential real world applications of fake profile detection are listed below:

1.**Social media platforms:**

- Social media platforms like Facebook ,Twitter ,and LinkedIn could integrate a fake profile detection system to identify and remove fraudulent accounts.
- The system could automatically flag suspicious profiles, reducing the spread of fake news, scams and misinformation.

2. **E-Commerce Platforms:**

- E-commerce websites such as Amazon, Alibaba could employ fake profile detection to identify and remove fake profiles. This could also help to avoid fraudulent transactions protecting both the buyers and sellers.

3.**Financial Services:**

- Financial institutions and payment processors could use fake profile detection to mitigate fraudulent activities in peer -to -peer payments and e-commerce transactions.

4. **Online Gaming communities:**
    - Online gaming platforms use this to identify hackers and bots.
    - This could help to maintain a fair and safe gaming environment

ADVANTAGES:

1.Enhanced Security:

Fake profile detection system provides security by identifying and removing fraudulent accounts thereby reducing the risk of scam and cyber attacks.

2.Improved Trust:

By effectively filtering out fake profiles and fraudulent activities, these systems help built trust among the users leading to a safer environment.

3.Reduced Operational cost:

Automated fake detection can lead to significant cost savings.

4.Enhanced User Experience:

Removing fake profiles improves the quality of interactions on online platform resulting in a positive user experience for genuine user.

DISADVANTAGES:

1.False positives:

Fake profile detection systems may occasionally misclassify genuine users as fake, leading to false positives and unintended consequences such as account suspensions or bans.

2.Adversarial Attacks:

Fraudsters may attempt to evade detection by continuously adapting their tactics, making it challenging for detecting systems to keep up with evolving techniques.

3. Privacy concerns:

Collecting and analyzing user data for fake profile detection purposes raises privacy concerns, if sensitive information is involved.

4. Complexity and Maintenance:

Developing and maintaining effective fake profile detection systems requires expertise in machine learning, data management and cybersecurity making it complex and requires maintenance.

REFERENCES:

Here are some references on fake profile detection:

❖ Alowibdi, J. S., Buy, U. A., & Yu, P. S. (2014). "Detecting Fake Profiles in Social Media Websites." In Proceedings of the 47th Hawaii International Conference on System Sciences.

- Wang, Z., Wilson, C., Wang, X., Zhao, X., & Mohanlal, M. (2013). "Social Turing Tests: Crowdsourcing Sybil Detection." In
- Proceedings of the 22nd International Conference on World Wide Web.
- Ferrara, E., Varol, O., Davis, C., Menczer, F., & Flammini, A. (2016). "The Rise of Social Bots." In Communications of the ACM.