NLP Project

Fake News Detection

Presented by:

(Team ID: T23)

Name	ID	Section
Mina Edwar Dawood Elias	2021170565	7
Nada AbdElmoneem Ahmed	2021170583	7
Nardine Guirguis Edward Amin	2021170576	7
Youssef Emad El-din Ibrahim	2021170640	8
Youssef Hany Ezzat Aly	2021170653	8
Dalia AbdElazim Mohamed	2021170179	3

Table of Contents

1.Data Preprocessing

1.1 Data Loading

We started our project by loading the dataset which is news.csv a comma separated values file into our environment in a pandas data frame (df) using the method read_csv().

1.2 Data Cleaning

1.2.1 Handling Missing, Duplicates Values

We assured of dropping all nulls and duplicates.

```
Before Unnamed: 0       219
title                   610
text                    867
label                  1040
```

```
After Unnamed: 0      0
title                 0
text                  0
label                 0
```

1.2.2 Handling Outliers

We dropped the rows where it had an empty text.

```
### rows that doesn't have text in it########
c=0
listOfIndcies=[]
for row in df['text']:
    if row == ' ':
        listOfIndcies=(df.index[df['text']==row].tolist())
        c+=1
print(c)
```

```
36
```

```
df = df.drop(labels=listOfIndcies,axis=0)
```

```
c=0
for row in df['text']:
    if row == ' ':
        c+=1
print(c)
```

```
0
```

```
df=df.reset_index(drop=True)
```

1.3 Processing Numerical Columns

Column (Unnamed: 0): In this column we found that each row contains useless numbers for our task or other formats so we just converted them to be an index for each row.

	Unnamed: 0
0	8476
1	10294
2	3608
3	10142
4	875
...	...
6294	4490
6295	8062
6296	8622
6297	4021
6298	4330

```
unique_ids = df['Unnamed: 0'].unique()
id_mapping = {original_id: new_id for new_id, original_id in enumerate(unique_ids)}
df['Unnamed: 0'] = df['Unnamed: 0'].map(id_mapping)
df.rename(columns={'Unnamed: 0': "ID"},inplace=True)
df
```

	ID
0	0
1	1
2	2
3	3
4	4
...	...
6294	6294
6295	6295
6296	6296
6297	6297
6298	6298

1.4 Handling Categorical Features

Column(label): Our target variable we saw that it has two values Fake or Real so we mapped its values to 0 and 1.

label
FAKE
FAKE
REAL
FAKE
REAL

```
df.label = df.label.astype(str)
df.label = df.label.str.strip()
dict = { 'REAL' : '1' , 'FAKE' : '0'}
df['label'] = df['label'].map(dict)
```

label
0
0
1
0
1

1.5 Handling text and title Features

We started by converting the title and text fields in each row to lowercase to avoid same word different treatment. We then defined three functions as the following:

Function 1: To remove puncutation, stopwords and to tokenize

```
tokenizer = RegexpTokenizer(r'\w+')
def remove_punctuation_tokenize_and_remove_stopwords(text):
    # Remove_punctuation
    cleaned_text =re.sub(r'[^\w\s]', '', str(text))
    tokenized_text = tokenizer.tokenize(cleaned_text)
    #remove_stopwords
    stop_words = set(stopwords.words('english'))
    filtered_text = [word for word in tokenized_text if word.lower() not in stop_words]
    return filtered_text
```

Function 2: To map between the tree bank and the word net

```
def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return 'a'  # Adjective
    elif tag.startswith('V'):
        return 'v'  # Verb
    elif tag.startswith('N'):
        return 'n'  # Noun
    elif tag.startswith('R'):
        return 'r'  # Adverb
    else:
        return None
```

Function 3: To lemmatize we used here pos tag to lemmatize accurately

```
lemmatizer = WordNetLemmatizer()

def pos_tag_and_lemmatize(tokenized_words):
    pos_tagged_words = pos_tag(tokenized_words)
    lemmatized_words = [lemmatizer.lemmatize(word, get_wordnet_pos(tag)) if get_wordnet_pos(tag)
                        else word for word, tag in pos_tagged_words]
    return lemmatized_words
```

Now we used the functions to toxenize the text column and then lemmatizing it and we did the same to the title column resulting in a new 4 columns:

	ID	title	text	label	tokenized_text	lemmatized_text	tokenized_title	lemmatized_title
0	0	you can smell hillary's fear	daniel greenfield, a shillman journalism fello...	0	[daniel, greenfield, shillman, journalism, fel...	[daniel, greenfield, shillman, journalism, fel...	[smell, hillarys, fear]	[smell, hillarys, fear]
1	1	watch the exact moment paul ryan committed pol...	google pinterest digg linkedin reddit stumbleu...	0	[google, pinterest, digg, linkedin, reddit, st...	[google, pinterest, digg, linkedin, reddit, st...	[watch, exact, moment, paul, ryan, committed, ...	[watch, exact, moment, paul, ryan, commit, pol...
2	2	kerry to go to paris in gesture of sympathy	u.s. secretary of state john f. kerry said mon...	1	[us, secretary, state, john, f, kerry, said, m...	[us, secretary, state, john, f, kerry, say, mo...	[kerry, go, paris, gesture, sympathy]	[kerry, go, paris, gesture, sympathy]
3	3	bernie supporters on twitter erupt in anger ag...	— kaydee king (@kaydeeking) november 9, 2016 t...	0	[kaydee, king, kaydeeking, november, 9, 2016, ...	[kaydee, king, kaydeeking, november, 9, 2016, ...	[bernie, supporters, twitter, erupt, anger, dn...	[bernie, supporter, twitter, erupt, anger, dnc...
4	4	the battle of new york: why this primary matters	it's primary day in new york and front-runners...	1	[primary, day, new, york, frontrunners, hillar...	[primary, day, new, york, frontrunners, hillar...	[battle, new, york, primary, matters]	[battle, new, york, primary, matter]
...	...	...	...	...	...	...	...	...

1.6 Feature Selection

- We selected only two columns from the dataframe to apply tf_idf on them which are “lemmatized_text” and “lemmatized_title”

```
X = df.drop(['title','text','tokenized_text','tokenized_title','label','ID'], axis=1)
```

```
X
```

	lemmatized_text	lemmatized_title
0	[daniel, greenfield, shillman, journalism, fel...	[smell, hillarys, fear]
1	[google, pinterest, digg, linkedin, reddit, st...	[watch, exact, moment, paul, ryan, commit, pol...
2	[us, secretary, state, john, f, kerry, say, mo...	[kerry, go, paris, gesture, sympathy]
3	[kaydee, king, kaydeeking, november, 9, 2016, ...	[bernie, supporter, twitter, erupt, anger, dnc...
4	[primary, day, new, york, frontrunners, hillar...	[battle, new, york, primary, matter]
...	...	...

- We then combined both in one column “combined_text” to start applying tf idf on this column.

```
####### Applying tf_idf#####
X['combined_text'] = X['lemmatized_text'].apply(lambda x: ' '.join(x)) + ' ' + X['lemmatized_title'].apply(lambda x: ' '.join(x))
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(X['combined_text'])

print(tfidf_matrix)
```

```
  (0, 24541)    0.01931317300642821
  (0, 12977)    0.01503765807506309
```

1.7 Train Test Split

- Then we started to split the data to 80% for training and 20% for testing by using the train_test_split and giving it test_size=0.2.

```
######splitting data to tarin and test######
tfidf_x_train, tfidf_x_test, y_train, y_test = train_test_split(tfidf_matrix, Y, test_size=0.2, random_state=0)
```

Finally, we are ready to use the data in our machine learning models.

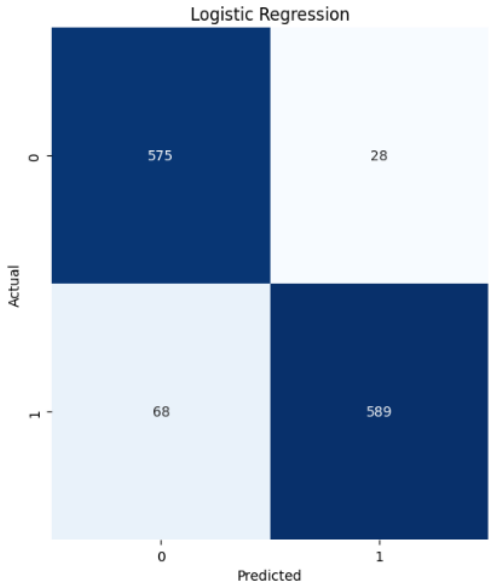
2.Models

2.1 Logistic Regression

2.1.1 Accuracy

```
Train Accuracy is:  95.31653105774956
Test Accuracy is:  92.38095238095238
```

2.1.2 Confusion Matrix and report



Logistic Regression:

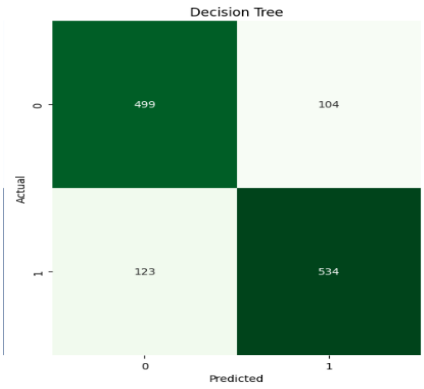
	precision	recall	f1-score	support
0	0.89	0.95	0.92	603
1	0.95	0.90	0.92	657
accuracy			0.92	1260
macro avg	0.92	0.93	0.92	1260
weighted avg	0.93	0.92	0.92	1260

2.2 Decision Tree

2.2.1 Accuracy

```
Train Accuracy is:  95.31653105774956
Test Accuracy is:  81.98412698412699
```

2.2.2 Confusion Matrix and report



Decision Tree:

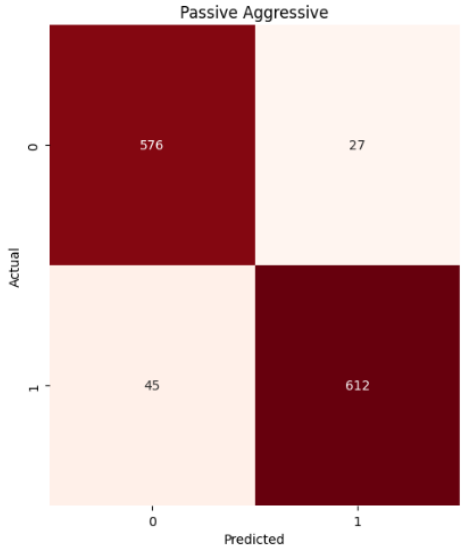
	precision	recall	f1-score	support
0	0.80	0.83	0.81	603
1	0.84	0.81	0.82	657
accuracy			0.82	1260
macro avg	0.82	0.82	0.82	1260
weighted avg	0.82	0.82	0.82	1260

2.3 Passive Aggressive

2.3.1 Error/Accuracy

```
Train Accuracy is:  100.0
Test Accuracy is:  94.28571428571428
```

2.3.2 Confusion Matrix and report



Passive Aggressive:

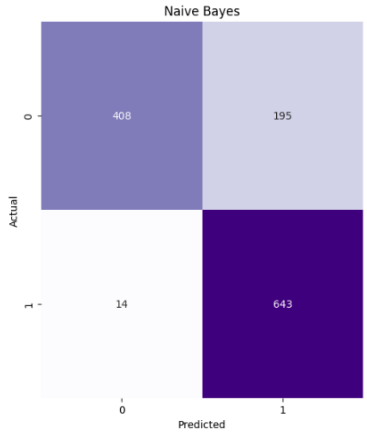
	precision	recall	f1-score	support
0	0.93	0.96	0.94	603
1	0.96	0.93	0.94	657
accuracy			0.94	1260
macro avg	0.94	0.94	0.94	1260
weighted avg	0.94	0.94	0.94	1260

2.4 Naïve Bayes

2.4.1 Error/Accuracy

```
Train Accuracy is:  88.76761262155189
Test Accuracy is:  83.41269841269842
```

2.4.2 Confusion Matrix and report



Naive Bayes:

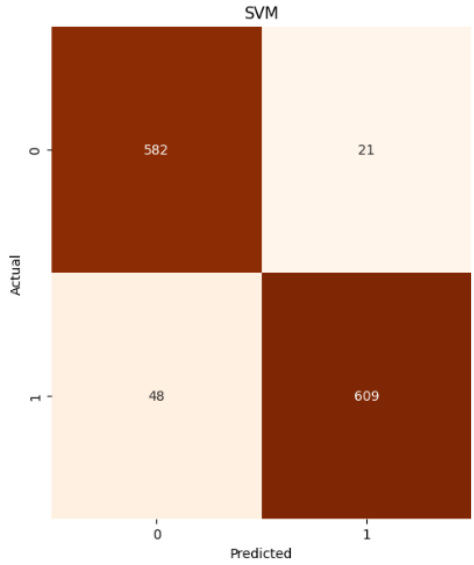
	precision	recall	f1-score	support
0	0.97	0.68	0.80	603
1	0.77	0.98	0.86	657
accuracy			0.83	1260
macro avg	0.87	0.83	0.83	1260
weighted avg	0.86	0.83	0.83	1260

2.5 SVC

2.5.1 Error/Accuracy

```
Train Accuracy is:  99.04743004564398
Test Accuracy is:  94.52380952380952
```

2.5.2 Confusion Matrix and report

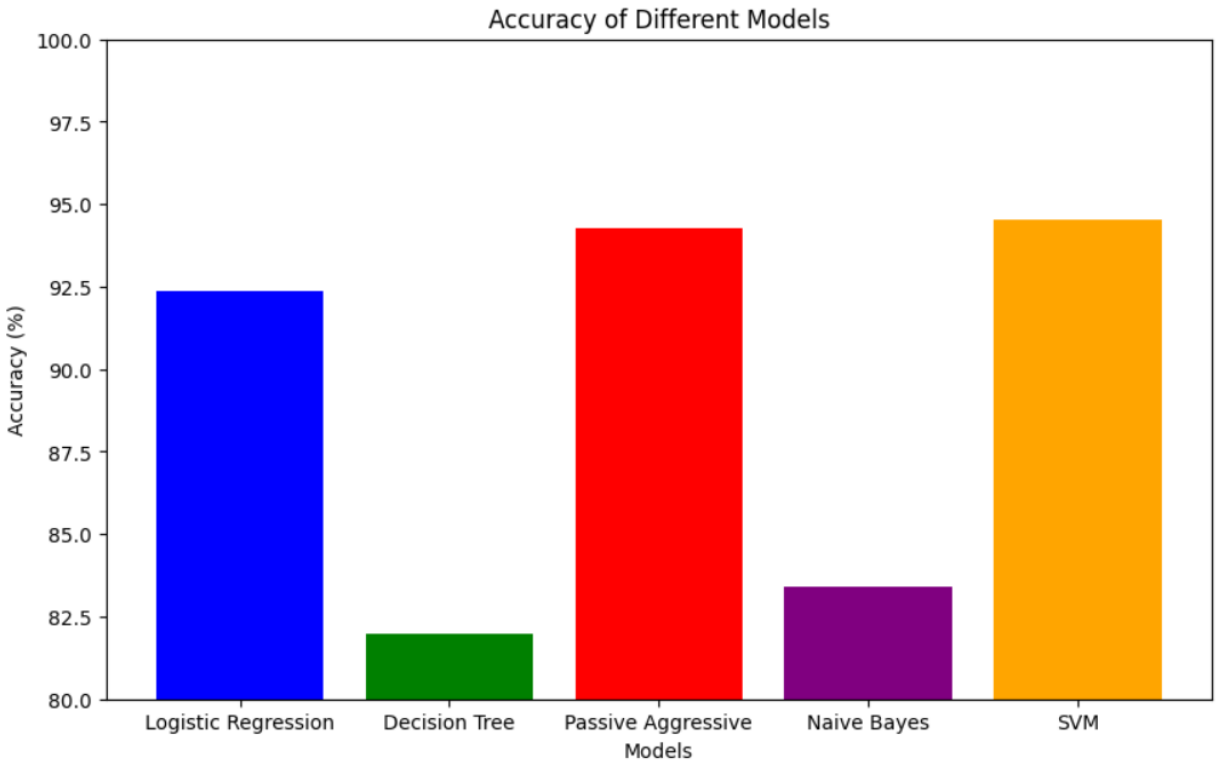


Support Vector Machine:

	precision	recall	f1-score	support
0	0.92	0.97	0.94	603
1	0.97	0.93	0.95	657
accuracy			0.95	1260
macro avg	0.95	0.95	0.95	1260
weighted avg	0.95	0.95	0.95	1260

3.Conclusion

	Model	Train Accuracy (%)	Test Accuracy (%)
0	Logistic Regression	95.316531	92.380952
1	Decision Tree	100.000000	81.984127
2	Passive Aggressive	100.000000	94.285714
3	Naive Bayes	88.767613	83.412698
4	SVM	99.047430	94.523810



4.Saving Models

```
############Saving the models#########
lr_model_path = 'logistic_regression_model.joblib'
dt_model_path = 'decision_tree_model.joblib'
pa_model_path = 'passive_aggressive_model.joblib'
nb_model_path = 'naive_bayes_model.joblib'
sv_model_path = 'support_vector_machine_model.joblib'

dump(lr, lr_model_path)
dump(dt, dt_model_path)
dump(pa, pa_model_path)
dump(nb, nb_model_path)
dump(sv, sv_model_path)

print("Models saved successfully!")
```

```
Models saved successfully!
```

Now we can load and use the loaded models:

```
############Loading Models###############
try:
    lr = load('logistic_regression_model.joblib')
    dt = load('decision_tree_model.joblib')
    pa = load('passive_aggressive_model.joblib')
    nb = load('naive_bayes_model.joblib')
    sv = load('support_vector_machine_model.joblib')

    print("Models loaded successfully!")
except FileNotFoundError:
    print("Saved models not found. Training models from scratch...")

    lr = LogisticRegression()
    lr.fit(tfidf_x_train, y_train)

    dt = DecisionTreeClassifier()
    dt.fit(tfidf_x_train, y_train)

    pa = PassiveAggressiveClassifier(max_iter=150)
    pa.fit(tfidf_x_train, y_train)

    nb = MultinomialNB()
    nb.fit(tfidf_x_train, y_train)

    sv = SVC(kernel='linear')
    sv.fit(tfidf_x_train, y_train)

    print("Models trained successfully!")
```

```
Models loaded successfully!
```