

Problem Statement

In today's world everything is online and so as news which is the basic source of what is happening around the world. Online news articles provide very useful information but internet is also filled with fake and misinformation news which even led to misleading people in wrong way. So in order to solve this problem our team of four people combined with the help of each other has develop a **A.I. based fake news detection system** which detect a news being real or fake.

DATA UNDERSTANDING

Data understanding relies on problem statement. Data related to the this Fake News Detection System is collected from "kaggle". A snapshot of raw data is been shown below.

```
In [3]: train.head()
```

Out[3]:

	id		title	author	text	label
0	0		House Dem Aide: We Didn't Even See Comey's Let...	Darrell Lucas	House Dem Aide: We Didn't Even See Comey's Let...	1
1	1		FLYNN: Hillary Clinton, Big Woman on Campus - ...	Daniel J. Flynn	Ever get the feeling your life circles the rou...	0
2	2		Why the Truth Might Get You Fired	Consortiumnews.com	Why the Truth Might Get You Fired October 29, ...	1
3	3		15 Civilians Killed In Single US Airstrike Hav...	Jessica Purkiss	Videos 15 Civilians Killed In Single US Aistr...	1
4	4		Iranian woman jailed for fictional unpublished...	Howard Portnoy	Print 'nAn Iranian woman has been sentenced to...	1

Data Preparation

Once the data has been collected, it must be transformed into a usable subset unless it is determined that more data is needed. Since we will be using NLP so the missing data was replaced by a single space and a column is added which is concatenation of two column in the same train dataset those are title , author and text.

```
In [5]: print(train.isnull().sum())
print('*****')
print(test.isnull().sum())

id      0
title   558
author  1957
text     39
label    0
dtype: int64
*****
id      0
title   122
author  503
text      7
dtype: int64

In [6]: test=test.fillna(' ')
train=train.fillna(' ')
test['total']=test['title']+' '+test['author']+test['text']
train['total']=train['title']+' '+train['author']+train['text']
#test.head()
```

We used word cloud to see if we could interpreter the world which are used the most in a fake and real news using the WORDCLOUD library.

```
In [73]: wordcloud = WordCloud(width = 800, height = 800,
                                background_color = 'white',
                                stopwords = stopwords,
                                min_font_size = 10).generate(real_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



figure shows keywords used in real news

Text Cleaning and Preprocessing:-

Removing punctuations and symbols in the the articles as the these are not the factors that will affect the the news being real or fake. We used Regex to remove those and replaced with a blank. The 're' library for this which uses regular expressions to do the task in text processing.

Tokenizing the sentences using 'nltk' library to use the words in article as an array. Once the tokenizing is done the **stopwords were removed** from the array as stopwords are not useful in this.

After removing the stopwords **Lemmatization of words** was done using WordNetLemmatizer() library in nltk. It is done to get the root words so that for predict efficiently. E.g. lemmatization covert cities into city, mice into mouse.

```
In [69]: lemmatizer=WordNetLemmatizer()
for index,row in train.iterrows():
    filter_sentence = ''

    sentence = row['total']
    sentence = re.sub(r'[^\w\s]','',sentence) #cleaning

    words = nltk.word_tokenize(sentence) #tokenization

    words = [w for w in words if not w in stop_words] #stopwords removal

    for word in words:
        filter_sentence = filter_sentence + ' ' + str(lemmatizer.lemmatize(word)).lower()

    train.loc[index,'total'] = filter_sentence
```

APPLYING NLP TECHNIQUES:-

Now that we have got all the words in articles to be used for training the model but before that we will use some NLP techniques to use our data in modeling.

Using **CountVectorizer** on previously processed data to get the frequency of the words in the articles.

Using **TF-IDF Vectorizer** which stands for term frequency—inverse document frequency. It is used to find how relevant a term in a given document.

TF-IDF Vectorizer

```
In [111]: def vectorize_text(features, max_features):
vectorizer = TfidfVectorizer( stop_words='english',
                             decode_error='strict',
                             analyzer='word',
                             ngram_range=(1, 2),
                             max_features=max_features,
                             #aux_df=0.5 # Verwendet im ML-Kurs unter Preprocessing
                             )
feature_vec = vectorizer.fit_transform(features)
return feature_vec.toarray()

In [112]: tfidf_features = vectorize_text(['hello how are you doing','hi i am doing fine'],30)

In [113]: tfidf_features

Out[113]: array([[0.44943642, 0.          , 0.          , 0.6316672 , 0.6316672 ,
                  0.          , 0.          ],
                [0.33517574, 0.47107781, 0.47107781, 0.          , 0.          ,
                  0.47107781, 0.47107781]])
```

Applying

```
In [117]: #Feature extraction using count vectorization and tfidf.
count_vectorizer = CountVectorizer()
count_vectorizer.fit_transform(X_train)
freq_term_matrix = count_vectorizer.transform(X_train)
tfidf = TfidfTransformer(norm="l2")
tfidf.fit(freq_term_matrix)
tf_idf_matrix = tfidf.transform(freq_term_matrix)

In [118]: tf_idf_matrix

Out[118]: <20800x220387 sparse matrix of type '<class 'numpy.float64'>'
          with 5987666 stored elements in Compressed Sparse Row format>
```

Modelling

For modeling we used two Machine Learning Algorithms to fit the model with train data and best performing algorithm was used for the deployment of the model.

Logistic Regression

```
In [40]: logreg = LogisticRegression(C=1e5)
logreg.fit(X_train, y_train)
pred = logreg.predict(X_test)
print('Accuracy of Logistic classifier on training set: {:.2f}'
      .format(logreg.score(X_train, y_train)))
print('Accuracy of Logistic classifier on test set: {:.2f}'
      .format(logreg.score(X_test, y_test)))
from sklearn.naive_bayes import MultinomialNB
cm = confusion_matrix(y_test, pred)
cm
```

Accuracy of Logistic classifier on training set: 1.00
Accuracy of Logistic classifier on test set: 0.98

```
Out[40]: array([[2494,  70],
               [ 45, 2591]], dtype=int64)
```

MultinomialNB

```
In [41]: from sklearn.naive_bayes import MultinomialNB

NB = MultinomialNB()
NB.fit(X_train, y_train)
pred = NB.predict(X_test)
print('Accuracy of NB classifier on training set: {:.2f}'
      .format(NB.score(X_train, y_train)))
print('Accuracy of NB classifier on test set: {:.2f}'
      .format(NB.score(X_test, y_test)))
cm = confusion_matrix(y_test, pred)
cm
```

Accuracy of NB classifier on training set: 0.88
Accuracy of NB classifier on test set: 0.83

```
Out[41]: array([[2558,  6],
               [ 853, 1783]], dtype=int64)
```

Acti
Go to

As shown in figure above the Logistic regression has higher accuracy on test data i.e 98% while MultinomialNB has 88% accuracy. So we will use logistic Regression to deploy the model.

Pipelining for the deployment of model:

Pipelining the process that we have done till now and saving the pipeline so that we can use this in the deployment part.

Pipeline

```
In [42]: #Assigning the variables again as once transformed vectors can't be transformed again using pipeline.  
X_train = train['total']  
Y_train = train['label']
```

```
In [43]: from sklearn.pipeline import Pipeline
from sklearn.externals import joblib
from sklearn import linear_model
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
In [44]: pipeline = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer(norm='l2')),
    ('clf', linear_model.LogisticRegression(C=1e5)),
])
```

```
In [45]: pipeline.fit(X_train, Y_train)
```

```
Out[45]: Pipeline(memory=None,
               steps=[('vect', CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                                                dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                                                lowercase=True, max_df=1.0, max_features=None, min_df=1,
                                                ngram_range=(1, 1), preprocessor=None, stop_words=None,
                                                strip_accents='12', random_state=None,
                                                solver='liblinear', tol=0.0001, verbose=0, warm_start=False))])
```

```
In [46]: pipeline.predict(["flynn hillary clinton big woman campus breitbart daniel j flynnnever get feeling life circle roundabout rather
```

```
Out[46]: array([0], dtype=int64)
```

```
In [47]: #saving the pipeline
filename = 'pipeline.sav'
joblib.dump(pipeline, filename)
```

```
out[47]: ['pipeline.sav']
```

```
In [48]: filename = './pipeline.sav'
```

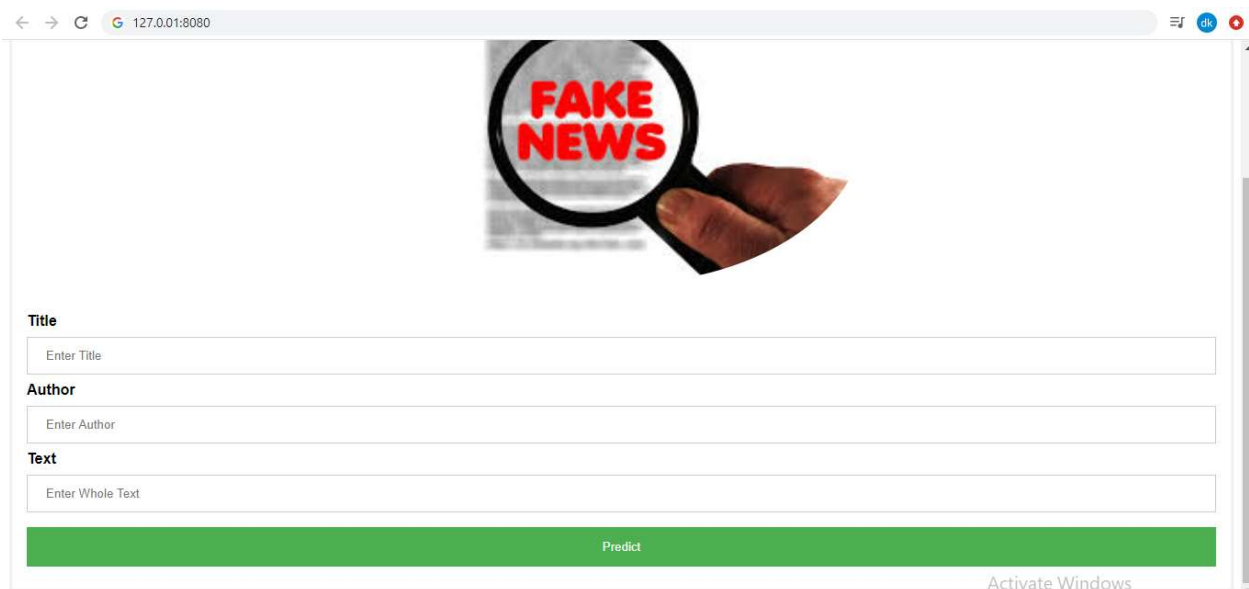
Model Deployment

In the deployment step, the model is used on new data outside of the scope of the dataset and by new users. The new interactions at this phase might reveal the new variables and needs for the dataset and model.

For deployment we will be using our local server to deploy the model using Flask which is a web development framework used for small apps.

```
app.py > Flask
1  from flask import Flask, abort, jsonify, request, render_template
2  from sklearn.externals import joblib
3  from feature import *
4  import json
5
6  pipeline = joblib.load('./pipeline.sav')
7
8  app = Flask(__name__)
9
10 @app.route('/')
11 def home():
12     return render_template('index.html')
13
14
15 @app.route('/api', methods=['POST'])
16 def get_delay():
17
18     result=request.form
19     query_title = result['title']
20     query_author = result['author']
21     query_text = result['maintext']
22     print(query_text)
23     query = get_all_query(query_title, query_author, query_text)
24     user_input = {'query':query}
25     pred = pipeline.predict(query)
26     print(pred)
27     dic = {1:'real',0:'fake'}
28     return f'<html><body><h1>{dic[pred[0]]}</h1> <form action="/"> <button type="submit">back </button> </form></body></html>'
29
30
31 if __name__ == '__main__':
32     app.run(port=8080, debug=True)
```

Running app.py will start the server and the following page will be displayed on localhost:8080



127.0.0.1:8080

FAKE NEWS

Title

Enter Title

Author

Enter Author

Text

Enter Whole Text

Predict