AIROBOTICA SERVICES PVT LTD.

INTERNSHIP REPORT

ON

FAKE NEWS DETECTION

Submitted By

Poulami Bakshi **1GA17EC152**

Under the Guidance Of

SHAIK MD RASOOL
DIRECTOR,
AIROBOTICA SERVICES PVT. LTD

30th December,2020-24th January,2021

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without mentioning the people who made it possible. With deep gratitude, I acknowledge all those guidance and encouragement, which served as bacon of light and crowned our efforts with success. I thank each one of them for their valuable support.

I express my sincere thanks to Mr. SHAIK MD RASOOL, DIRECTOR, AIROBOTICA Services Pvt. Ltd, for providing necessary facilities and motivation to carry out internship work successfully.

I would like to express my sincere gratitude towards DR. AJAY B. GADICHA, the trainer for providing encouragement and inspiration throughout the internship.

Poulami Bakshi

1GA17EC152

ABSTRACT

This is to verify that the project work entitled **"Fake News Detection"** is a bonafide work carried out by **Poulami Bakshi** bearing the USN number 1GA17EC152 in partial fulfillment of the requirement for gaining soft skill in the field of Machine Learning after completing my Bachelor of Engineering in Electronics and Electrical from KIIT university of, Bhubaneswar, Orissa during the year 2016 – 2020.

.....

DIRECTOR

Mr. SHAIK MDR

AIROBOTICA SERVICES PVT LTD

ABOUT COMPANY

About AIROBOTICA!

BUILDING AN INNOVATIVE TOMORROW!

AIROBOTICA,
is a Bangalore (the Silicon Valley of India, Make in India) based firm that strives to foster Artificial Intelligence by bridging the unexplored realm of innovative opportunities in Aerospace Technology, IOT, IIOT R&D in Nano, Micro Satellite research based Company We believe in the power of Futuristic technology, where Artificial Intelligence will establish its hold to solve the complex problems and to open up new avenues for businesses of all sizes, across all verticals. Hence, our solutions are highly Industry and Requirement specific. Using AI, ML RPA, Space, IOT and SAP ERP services, we will help you promote agile and cost-effective business, administrative and marketing operations.

Our aim is to help enterprises create an automated environment to achieve accuracy. Our enterprise-friendly applications can be configured and managed within an IT governed framework and operating model, thereby unlocking the possibilities of becoming the Future Leaders.

Our company specializes in providing solutions for Small, Medium and Large Enterprises via our products and services.

Fake News Detection Using Machine Learning Ensemble Methods.

Abstract:

The advent of the World Wide Web and the rapid adoption of social media platforms (such as Facebook, Twitter, and Instagram) paved the way for information dissemination that has never been witnessed in human history before. With the current usage of social media platforms, consumers are creating and sharing more information than ever before, some of which are misleading with no relevance to reality. Automated classification of a text article as misinformation or disinformation is a challenging task. Even an expert in a particular domain has to explore multiple aspects before giving a verdict on the truthfulness of an article. In this work, we propose to use a machine learning ensemble approach for the automated classification of news articles. Our study explores different textual properties that can be used to distinguish fake content from real. By using those properties, we train a combination of different machine learning algorithms using various ensemble methods and evaluate their performance on real-world datasets. The experimental evaluation confirms the superior performance of our proposed ensemble learner approach in comparison to individual learners.

Introduction:

The advent of the World Wide Web and the rapid adoption of social media platforms (such as Facebook, Twitter, and Instagram) paved the way for information dissemination that has never been witnessed in human history before. Besides other use cases, news outlets benefitted from the widespread use of social media platforms by providing updated news in near real-time to their subscribers. The news media evolved from newspapers, tabloids, and magazines to a digital form such as online news platforms, blogs, social media feeds, and other digital media formats. It became easier for consumers to acquire the latest news at their fingertips. Facebook referrals account for 70% of traffic to news websites. These social media platforms in their current state are extremely powerful and useful for their ability to allow users to discuss and share ideas and debate over issues such as democracy, education, and health. However, such platforms are also used with a negative perspective by certain entities commonly for monetary gain and in other cases for creating biased opinions, manipulating mindsets, and spreading satire or absurdity. The phenomenon is commonly known as fake news.

There has been a rapid increase in the spread of fake news in the last decade, most prominently observed in the 2016 US elections. Such proliferation of sharing articles online that do not conform to facts has led to many problems not just limited to politics but covering various other domains such as sports, health, and also science. One such area affected by fake news is the financial markets, where a rumor can have disastrous consequences and may bring the market to a halt.

Our ability to take a decision relies mostly on the type of information we consume; our world view is shaped on the basis of the information we digest. There is increasing evidence that consumers have reacted absurdly to the news that later proved to be fake. One recent case is the spread of the novel coronavirus, where fake reports spread over the Internet about the origin, nature, and behavior of the virus. The situation worsened as more people read about the fake content online. Identifying such news online is a daunting task.

Fortunately, there are a number of computational techniques that can be used to mark certain articles as fake on the basis of their textual content. The majority of these techniques use fact checking websites such as “PolitiFact” and “Snopes.” There are a number of repositories maintained by researchers that contain lists of websites that are identified as ambiguous and fake. However, the problem with these resources is that human expertise is required to identify articles/websites as fake. More importantly, the fact-checking websites contain articles from particular domains such as politics and are not generalized to identify fake news articles from multiple domains such as entertainment, sports, and technology.

The World Wide Web contains data in diverse formats such as documents, videos, and audios. News published online in an unstructured format (such as news, articles, videos, and audios) is relatively difficult to detect and classify as this strictly requires human expertise. However, computational techniques such as natural language processing (NLP) can be used to detect anomalies that separate a text article that is deceptive in nature from articles that are based on facts. Other techniques involve the analysis of propagation of fake news in contrast with real news. More specifically, the approach analyzes how a fake news article propagates differently on a network relative to a true article. The response that an article gets can be differentiated at a theoretical level to classify the article as real or fake. A more hybrid approach can

also be used to analyze the social responsibility of an article along with exploring the textual features to examine whether an article is deceptive in nature or not.

A number of studies have primarily focused on the detection and classification of fake news on social media platforms such as Facebook and Twitter. At a conceptual level, fake news has been classified into different types; the knowledge is then expanded to generalize machine learning (ML) models for multiple domains. The study by Ahmed et al. included extracting linguistic features such as n-grams from textual articles and training multiple ML models including K-nearest neighbor (KNN), support vector machine (SVM), logistic regression (LR), linear support vector machine (LSVM), decision tree (DT), and stochastic gradient descent (SGD), achieving the highest accuracy (almost 90%) with SVM and logistic regression. According to the research, as the number of increased in -grams calculated for a particular article, the overall accuracy decreased. The phenomenon has been observed for learning models that are used for classification achieved better accuracies with different models by combining textual features with auxiliary information such as using social engagements on social media. The authors also discussed the social and psychological theories and how they can be used to detect false information online. Further, the authors discussed different data mining algorithms for model constructions and techniques shared for features extraction. These models are based on knowledge such as writing style, and social contexts such as stance and propagation.

The Problem:

The problem is not only hackers, going into accounts, and sending false information. The bigger problem here is what we call “Fake News”. A fake are those news stories that are false: the story itself is fabricated, with no verifiable facts, sources, or quotes. When someone (or something like a bot) impersonates someone or a reliable source to false spread information, that can also be considered fake news. In most cases, the people creating this false information have an agenda, that can be political, economic or to change the behavior or thought about a topic.

There are countless sources of fake news nowadays, mostly coming from programmed bots, that can’t get tired (they’re machines hehe) and continue to spread false information 24/7.

The tweets in the introduction are just basic examples of this problem, but much more serious studies in the past 5 years, have demonstrated big correlations between the spread of false information and elections, the popular opinion or feelings about different topics.

The problem is real and hard to solve because the bots are getting better are tricking us. Is not simple to detect when the information is true or not all the time, so we need better systems that help us understand the patterns of fake news to improve our social media, communication and to prevent confusion in the world.

Purpose:

In this short article, I’ll explain several ways to detect fake news using collected data from different articles. But the same techniques can be applied to different scenarios. In this article, I'll explain the Python code to load, clean, and analyze data. Then we will do some machine learning models to perform a classification task (fake or not).

Collection of Data:

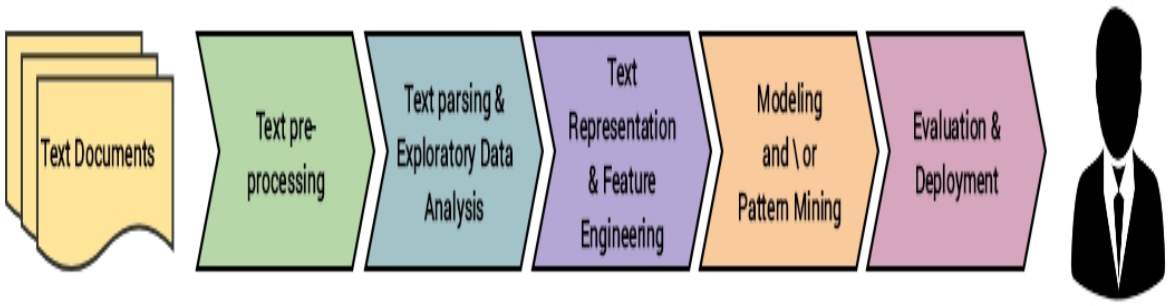
The Datasets has been collected from *https://data-flair.training/* and
the drive link for the datasets:-

https://drive.google.com/file/d/1er9NJTLUA3qnRuyhfzuN0XUsoIC4a-_q/view

Tools/Skills Used:

1. Python programing
2. Jupyter Notebook
3. Pandas
4. Numpy
5. Matplotlib
6. Seaborn
7. Exploratory Data Analytics
8. Feature Engineering
9. Data Visualization
10. Sciklearn
11. Machine Learning Algorithm
12. Natural Language Processing(NLTK)

Workflow:



Solving the problem with Python

Data reading

```
In [1]: import pandas as pd
        df=pd.read_csv('C:\\Users\\nEW u\\Desktop\\DataSEts\\news.csv',encoding='ISO-8859-1')
        df.head()
```

Out[1]:

	Unnamed: 0	title	text	label
0	8476	You Can Smell Hillaryâ□□s Fear	Daniel Greenfield, a Shillman Journalism Fello...	FAKE
1	10294	Watch The Exact Moment Paul Ryan Committed Pol...	Google Pinterest Digg Linkedin Reddit Stumbleu...	FAKE
2	3608	Kerry to go to Paris in gesture of sympathy	U.S. Secretary of State John F. Kerry said Mon...	REAL
3	10142	Bernie supporters on Twitter erupt in anger ag...	â□□ Kaydee King (@KaydeeKing) November 9, 2016...	FAKE
4	875	The Battle of New York: Why This Primary Matters	It's primary day in New York and front-runners...	REAL

We are using the Pandas library to load the CSV file in the Jupyter Notebook. Pandas is one of the tools in Machine Learning which is used for data cleaning and analysis. It has features that are used for exploring, cleaning, transforming, and visualizing from data.

head():- Returns the first 5 rows of the Dataframe. To override the default, you may insert a value between the parenthesis to change the number of rows returned. Example: df. head(10) will return 10 rows.

Data Cleaning:

It is very important to clean the data because it contains many unwanted columns unwanted data outliers null values nan columns and many more. Data cleaning refers to identifying and correcting errors in the dataset that may negatively impact a predictive model. Data cleaning is used to refer to all kinds of tasks and activities to detect and repair errors in the data.
Hence, using some basics like:
df.columns, df.isnull().sum(), df.drop().

```
In [122]: # to check the column labels of the DataFrame.
          df.columns

Out[122]: Index(['Unnamed: 0', 'title', 'text', 'label'], dtype='object')

In [123]: #droping the unwanted columns from the Datasets
          df.drop(['Unnamed: 0','title'], axis='columns', inplace=True)

In [124]: #checking the no. of rows and columns in the Datasets
          #there are 6335 rows or records and 2 rows
          df.shape

Out[124]: (6335, 2)

In [125]: #checking the columns labels after dropping the unwanted columns
          df.columns

Out[125]: Index(['text', 'label'], dtype='object')
```

```
In [4]: #checking is there any null values in the Datasets
        df.isnull().sum()

Out[4]: text     0
        label    0
        dtype: int64
```

Analyzing the dependent features:

In this Dataset, the target feature is 'label' it contains two unique values ['FAKE', 'REAL'] for analyzing it we are using a function like:-

1. **nunique**() function return number of unique elements in the object. It returns a scalar value which is the count of all the unique values in the Index. By default, the NaN values are not included in the count.
2. **unique()** function is used to find the unique elements of an array.
3. **value_counts()** function returns object containing counts of unique values. The resulting object will be in descending order so that the first element is the most frequently-occurring element. Excludes NA values by default.

```
In [126]: #df['label'] is the target columns
          #nunique() function return number of unique elements in the object.
          df['label'].nunique()

Out[126]: 2

In [127]: #unique() function is used to find the unique elements of an array
          df['label'].unique()

Out[127]: array(['FAKE', 'REAL'], dtype=object)

In [128]: #value_counts() function returns object containing counts of unique values
          df['label'].value_counts()

Out[128]: REAL    3171
          FAKE    3164
          Name: label, dtype: int64
```

From the above, we can conclude that the target feature is of object type with two unique features FAKE and REAL. As we all know that object type data cannot be pass to the model hence, converting it to numeric datatype by using LabelEncoder.

sklearn.preprocessing.LabelEncoder:

Sklearn provides a very efficient tool for encoding the levels of categorical features into numeric values. **LabelEncoder** encode labels with a value between 0 and n_classes-1 where n is the number of distinct labels. If a label repeats it assigns the same value as assigned earlier.
These are transformers that are not intended to be used on features, only on supervised learning targets.

```
In [5]: #Encode target labels with a value between 0 and n_classes-1.
        from sklearn.preprocessing import LabelEncoder
        scaler=LabelEncoder()
        df['label']=scaler.fit_transform(df['label'])
```

```
In [6]: df['label'].unique()
Out[6]: array([0, 1], dtype=int64)
```

```
In [7]: df.head()
```

Out[7]:

	text	label
0	Daniel Greenfield, a Shillman Journalism Fello...	0
1	Google Pinterest Digg Linkedin Reddit Stumbleu...	0
2	U.S. Secretary of State John F. Kerry said Mon...	1
3	â□□ Kaydee King (@KaydeeKing) November 9, 2016...	0
4	It's primary day in New York and front-runners...	1

Hence, we convert the target feature object datatypes to numeric datatypes where 0 is for FAKE and 1 is for REAL.

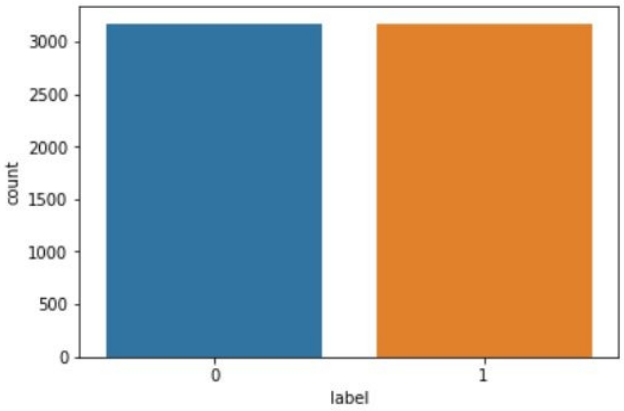
Data Visualization:

Matplotlib and Seaborn are the two libraries that are been used in this model. **Matplotlib** is mainly deployed for basic plotting. Visualization using Matplotlib generally consists of bars, pies, lines, scatter plots, and so on. **Seaborn**, on the other hand, provides a variety of visualization patterns. It uses fewer syntax and has easily interesting default themes.

This model applying countplot() from seaborn to show the visualizing count of target feature.

countplot(): Show the counts of observations in each categorical bin using bars.
A **count plot** can be thought of as a histogram across a categorical, instead of quantitative, variable. The basic API and options are identical to those for barplot(), so you can compare counts across nested variables.

```
In [8]: #show the visualizing count of target feature.
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline

        sns.countplot(df['label'])

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x21f5b3858d0>
```



Text cleaning using NLP(Natural Language Processing):

First of taking the copy of the original Dataset so that while cleaning the text the original text doesn't affect.

```
In [10]: df['text'].head(4)

Out[10]: 0    Daniel Greenfield, a Shillman Journalism Fello...
         1    Google Pinterest Digg Linkedin Reddit Stumbleu...
         2    U.S. Secretary of State John F. Kerry said Mon...
         3    â Kaydee King (@KaydeeKing) November 9, 2016...
         Name: text, dtype: object

In [11]: #Now let's clean the text first of all make the copy of the original Dataset so that dosen't affect the original data
         message=df.copy()
```

Natural language processing is defined as “the application of computational techniques to the analysis and synthesis of natural language and speech”. To perform these computational tasks, we first need to convert the language of text into a language that the machine can understand.

In this problem statement Fake news classifier, I am going to describe some of the most common steps involved in preparing text data for natural language processing.

1)Normalization: One of the key steps in processing language data is to remove noise so that the machine can more easily detect the patterns in the data. Text data contains a lot of noise, this takes the form of special characters such as hashtags, punctuation, and numbers. All of which are difficult for computers to understand if they are present in the data. We need to, therefore, process the data to remove these elements.
Additionally, it is also important to apply some attention to the casing of words. If we include both upper case and lower case versions of the same words then the computer will see these as different entities, even though they may be the same.

2)Stop words: Stop words are commonly occurring words that for some computational processes provide little information or in some cases introduce unnecessary noise and therefore need to be removed. This is particularly the case for text classification tasks.
There are other instances where the removal of stop words is either not advised or needs to be more carefully considered. This includes any situation where the meaning of a piece of text may be lost by the removal of a stop word. For example, if we were building a chatbot and removed the word"**not**" from the phrase "**i am not happy**" then the reverse meaning may be interpreted by the algorithm. This would be particularly important for use cases such as chatbots or sentiment analysis.
The Natural Language Toolkit (NLTK) python library has built-in methods for removing stop words.

Let's take a single paragraph for example message['text'][1] and check how the cleaning is going on after that it will be done in whole datasets.

```
In [12]: message['text'][1]
```

```
Out[12]: 'Google Pinterest Digg Linkedin Reddit Stumbleupon Print Delicious Pocket Tumblr \nThere are two fundamental truths in this wor
ld: Paul Ryan desperately wants to be president. And Paul Ryan will never be president. Today proved it. \nIn a particularly st
aggering example of political cowardice, Paul Ryan re-re-re-reversed course and announced that he was back on the Trump Train a
fter all. This was an aboutface from where he was a few weeks ago. He had previously declared he would not be supporting or def
ending Trump after a tape was made public in which Trump bragged about assaulting women. Suddenly, Ryan was appearing at a pro-
Trump rally and boldly declaring that he already sent in his vote to make him President of the United States. It was a surreal
moment. The figurehead of the Republican Party dosed himself in gasoline, got up on a stage on a chilly afternoon in Wisconsin,
and lit a match. . @SpeakerRyan says he voted for @realDonaldTrump : â\x80\x9cRepublicans, it is time to come homeâ\x80\x9d htt
ps://t.co/VyTT49YvoE pic.twitter.com/wCvSCg4a5I \nâ\x80\x94 ABC News Politics (@ABCPolitics) November 5, 2016 \nThe Democratic
Party couldnâ\x80\x99t have asked for a better moment of film. Ryanâ\x80\x99s chances of ever becoming president went down to z
ero in an instant. In the wreckage Trump is to leave behind in his wake, those who cravenly backed his campaign will not recove
r. If Ryanâ\x80\x99s career manages to limp all the way to 2020, then the DNC will have this tape locked and loaded to be used
in every ad until Election Day. \nThe ringing endorsement of the man he clearly hates on a personal level speaks volumes about
his own spinelessness. Ryan has postured himself as a â\x80\x9cprincipledâ\x80\x9d conservative, and one uncomfortable with Tru
mpâ\x80\x99s unapologetic bigotry and sexism. However, when push came to shove, Paul Ryan â\x80\x93 like many of his colleagues
â\x80\x93 turned into a sniveling appeaser. After all his lofty tak about conviction, his principles were a house of cards and
collapsed with the slightest breeze. \nWhatâ\x80\x99s especially bizarre is how close Ryan came to making it through unscathed.
For months the Speaker of the House refused to comment on Trump at all. His strategy seemed to be to keep his head down, preten
d Trump didnâ\x80\x99t exist, and hope that nobody remembered what happened in 2016. Now, just days away from the election, he
screwed it all up. \nIf 2016â\x80\x99s very ugly election has done any good itâ\x80\x99s by exposing the utter cowardice of the
Republicans who once feigned moral courage. A reality television star spit on them, hijacked their party, insulted their wives,
and got every last one of them to kneel before him. What a turn of events. \nFeatured image via Twitter'
```

```
In [13]: para='''Google Pinterest Digg Linkedin Reddit Stumbleupon Print Delicious Pocket Tumblr \nThere are two fundamental truths
in this world: Paul Ryan desperately wants to be president. And Paul Ryan will never be president. Today proved it. \nIn a
particularly staggering example of political cowardice, Paul Ryan re-re-re-reversed course and announced that he was back on
the Trump Train after all. This was an aboutface from where he was a few weeks ago. He had previously declared he would not
be supporting or defending Trump after a tape was made public in which Trump bragged about assaulting women. Suddenly, Ryan
was appearing at a pro-Trump rally and boldly declaring that he already sent in his vote to make him President of the United
States. It was a surreal moment. The figurehead of the Republican Party dosed himself in gasoline, got up on a stage on a
chilly afternoon in Wisconsin, and lit a match. . @SpeakerRyan says he voted for @realDonaldTrump : â\x80\x9cRepublicans,
it is time to come homeâ\x80\x9d https://t.co/VyTT49YvoE pic.twitter.com/wCvSCg4a5I \nâ\x80\x94 ABC News Politics
(@ABCPolitics) November 5, 2016 \nThe Democratic Party couldnâ\x80\x99t have asked for a better moment of film.
Ryanâ\x80\x99s chances of ever becoming president went down to zero in an instant. In the wreckage Trump is to
leave behind in his wake, those who cravenly backed his campaign will not recover. If Ryanâ\x80\x99s career manages to
limp all the way to 2020, then the DNC will have this tape locked and loaded to be used in every ad until Election Day.
\nThe ringing endorsement of the man he clearly hates on a personal level speaks volumes about his own spinelessness.
Ryan has postured himself as a â\x80\x9cprincipledâ\x80\x9d conservative, and one uncomfortable with Trumpâ\x80\x99s
unapologetic bigotry and sexism. However, when push came to shove, Paul Ryan â\x80\x93 like many of his colleagues
â\x80\x93 turned into a sniveling appeaser. After all his lofty tak about conviction, his principles were a house of
cards and collapsed with the slightest breeze. \nWhatâ\x80\x99s especially bizarre is how close Ryan came to making it
through unscathed. For months the Speaker of the House refused to comment on Trump at all. His strategy seemed to be to
keep his head down, pretend Trump didnâ\x80\x99t exist, and hope that nobody remembered what happened in 2016. Now, just
days away from the election, he screwed it all up. \nIf 2016â\x80\x99s very ugly election has done any good itâ\x80\x99s
by exposing the utter cowardice of the Republicans who once feigned moral courage. A reality television star spit on them,
hijacked their party, insulted their wives, and got every last one of them to kneel before him. What a turn of events.
\nFeatured image via Twitter'''
```

3)Stemming: Stemming is the process of reducing words to their root form. For example, the words "**rain**", "**raining**" and "**rained**" have very similar, and in many cases, the same meaning. The process of stemming will reduce these to the root form of "rain". This is again a way to reduce noise and the dimensionality of the data.
The NLTK library also has methods to perform the task of stemming. The code below uses the PorterStemmer to stem the words in my example above. As you can see from the output all the words now become "rain".

In [14]:
```
import nltk
import re

from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()
corpus = []

review=re.sub(r'http\S+',' ',para) #removing all the link releted text
review = re.sub('[^a-zA-Z]', ' ', review)# removing all the element except a-z and A-Z
review = review.lower()#lowering the text
review = review.split()

#removing all the stopwords and then stemming the text
review = [ps.stem(word) for word in review if not word in stopwords.words('english')]
review = ' '.join(review)
corpus.append(review)
```

In [15]:
```
corpus
```

Out[15]:
```
['googl pinterest digg linkedin reddit stumbleupon print delici pocket tumblr two fundament truth world paul ryan desper want p
resid paul ryan never presid today prove particularli stagger exampl polit cowardic paul ryan revers cours announc back trump t
rain aboutfac week ago previous declar would support defend trump tape made public trump brag assault women suddenli ryan appea
r pro trump ralli boldli declar alreadi sent vote make presid unit state surreal moment figurehead republican parti dose gasoli
n got stage chilli afternoon wisconsin lit match speakerryan say vote realdonaldtrump republican time come home pic twitter com
wcvscg abc news polit abcpolit novemb democrat parti ask better moment film ryan chanc ever becom presid went zero instant wrec
kag trump leav behind wake cravenli back campaign recov ryan career manag limp way dnc tape lock load use everi ad elect day ri
ng endors man clearli hate person level speak volum spineless ryan postur principl conserv one uncomfort trump unapologet bigot
ri sexism howev push came shove paul ryan like mani colleagu turn snivel appeas lofti tak convict principl hous card collaps sl
ightest breez especi bizarr close ryan came make unscath month speaker hous refus comment trump strategi seem keep head pretend
trump exist hope nobodi rememb happen day away elect screw ugli elect done good expos utter cowardic republican feign moral cou
rag realiti televis star spit hijack parti insult wive got everi last one kneel turn event featur imag via twitter']
```

It seems that 80% of the text is cleaned.

Now let's see whats happening with Lemmatization

4) Lemmatization: The goal of lemmatization is the same as for stemming, in that it aims to reduce words to their root form. However, stemming is known to be a fairly crude method of doing this. Lemmatization, on the other hand, is a tool that performs full morphological analysis to more accurately find the root, or “lemma” for a word. Again NLTK can be used to perform this task.

In [17]:
```
from nltk.stem import WordNetLemmatizer
lem=WordNetLemmatizer()

corpus_lem=[]
review=re.sub(r'http\S+',' ',para)#removing all the link releted text
review = re.sub('[^a-zA-Z]', ' ', review)# removing all the element except a-z and A-Z
review = review.lower()#lowering the text
review = review.split()

#removing all the stopwords and then lemmatizing the text
review=[lem.lemmatize(word) for word in review if word not in set(stopwords.words('english'))]
review=' '.join(review)
corpus_lem.append(review)

corpus_lem
```

Out[17]:
```
['google pinterest digg linkedin reddit stumbleupon print delicious pocket tumblr two fundamental truth world paul ryan despera
tely want president paul ryan never president today proved particularly staggering example political cowardice paul ryan revers
ed course announced back trump train aboutface week ago previously declared would supporting defending trump tape made public t
rump bragged assaulting woman suddenly ryan appearing pro trump rally boldly declaring already sent vote make president united
state surreal moment figurehead republican party dosed gasoline got stage chilly afternoon wisconsin lit match speakerryan say
voted realdonaldtrump republican time come home pic twitter com wcvscg abc news politics abcpolitics november democratic party
asked better moment film ryan chance ever becoming president went zero instant wreckage trump leave behind wake cravenly backed
campaign recover ryan career manages limp way dnc tape locked loaded used every ad election day ringing endorsement man clearly
hate personal level speaks volume spinelessness ryan postured principled conservative one uncomfortable trump unapologetic bigo
try sexism however push came shove paul ryan like many colleague turned sniveling appeaser lofty tak conviction principle house
card collapsed slightest breeze especially bizarre close ryan came making unscathed month speaker house refused comment trump s
trategy seemed keep head pretend trump exist hope nobody remembered happened day away election screwed ugly election done good
exposing utter cowardice republican feigned moral courage reality television star spit hijacked party insulted wife got every l
ast one kneel turn event featured image via twitter']
```

Now apply all this cleaning process in whole datasets.

In [19]:
```
#stemming and cleaning
corpus_stem = []
for i in range(0, len(message)):
    review=re.sub(r'http\S+',' ',message['text'][i])
    review = re.sub('[^a-zA-Z]', ' ', review)
    review = review.lower()
    review = review.split()

    review = [ps.stem(word) for word in review if not word in stopwords.words('english')]
    review = ' '.join(review)
    corpus_stem.append(review)
```

In [20]:
```
len(message)
```

Out[20]: 6335

In [101]:
```
df2=pd.DataFrame(df['label'],index=None)
```

In [103]:
```
df2['stemming_text']=corpus_stem
```

In [104]:
```
df2.head(4)
```

Out[104]:

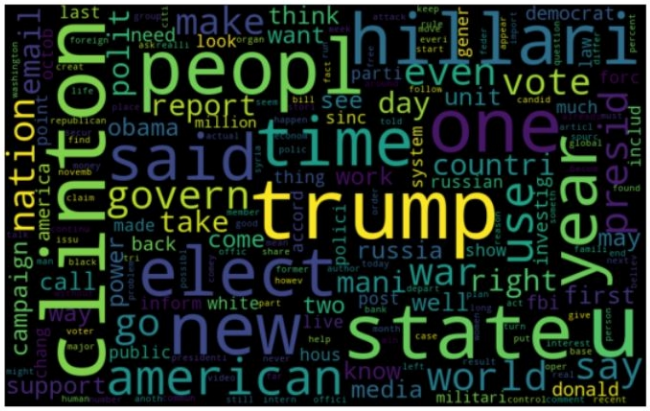
	label	stemming_text
0	0	daniel greenfield shillman journal fellow free...
1	0	googl pinterest digg linkedin reddit stumbleup...
2	1	u secretari state john f kerri said monday sto...
3	0	kayde king kaydeek novemb lesson tonight dem l...

In [21]:
```
#lammetizing and cleaning

corpus_lemmetize = []
for i in range(0, len(message)):
    review=re.sub(r'http\S+',' ',message['text'][i])
    review = re.sub('[^a-zA-Z]', ' ', review)
    review = review.lower()
    review = review.split()

    review = [lem.lemmatize(word) for word in review if word not in set(stopwords.words('english'))]
    review = ' '.join(review)
    corpus_lemmetize.append(review)
```

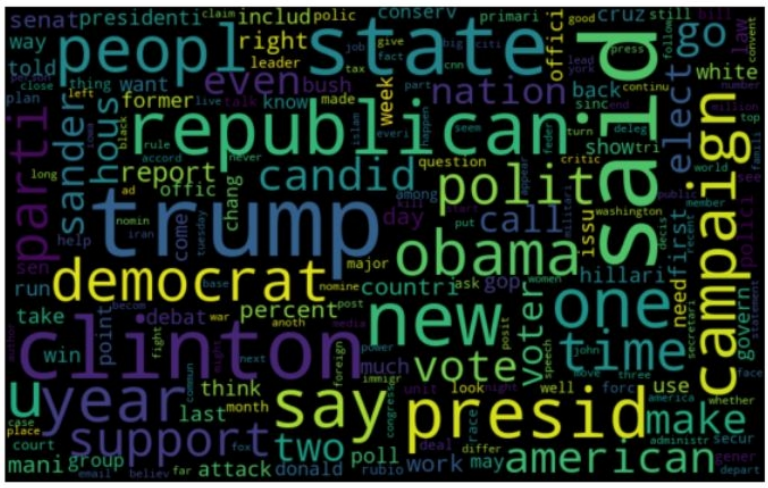
Let's check now what are the most common fake word used for fake news are what are the most commonly used real word for real news.
For the visualization of most words, I'm using the word cloud library.

Word Cloud is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance. Significant textual data points can be highlighted using a word cloud. Word clouds are widely used for analyzing data from social network websites.

In [106]:
```
#most common fake words used
from wordcloud import WordCloud
fake_text = df2[df2['label'] == 0]
all_words = ' '.join([text for text in fake_text.stemming_text])
wordcloud = WordCloud(width= 800, height= 500,
                      max_font_size = 110,
                      collocations = False).generate(all_words)
plt.figure(figsize=(10,7))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```



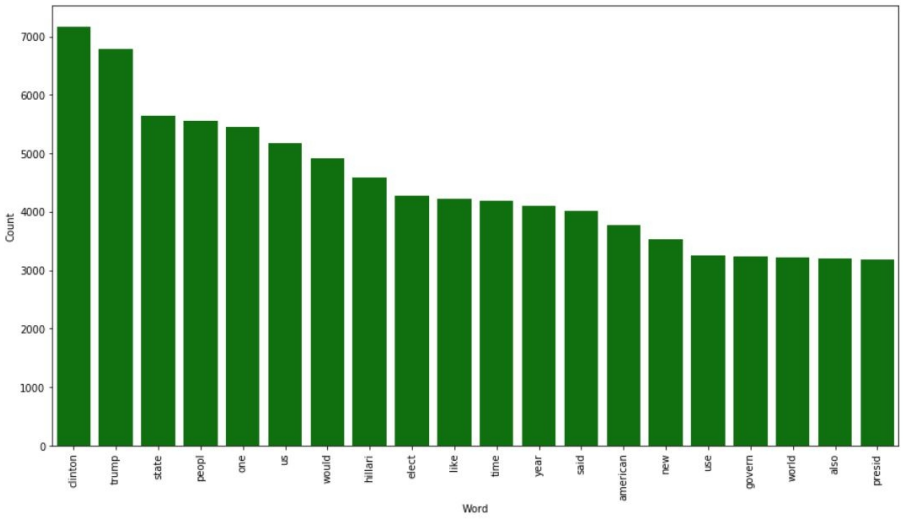
In [107]:
```
#most common real words used
from wordcloud import WordCloud
fake_text = df2[df2['label'] == 1]
all_words = ' '.join([text for text in fake_text.stemming_text])
wordcloud = WordCloud(width= 800, height= 500,
                      max_font_size = 110,
                      collocations = False).generate(all_words)
plt.figure(figsize=(10,7))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```



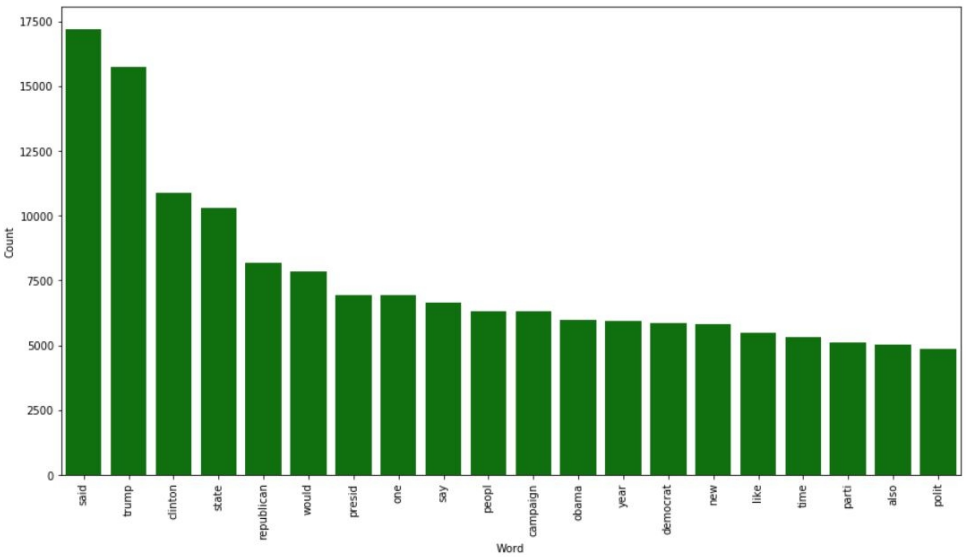
Now let's see the frequency of the most commonly used word for fake and real news.

In [115]:
```
from nltk import tokenize
token_space = tokenize.WhitespaceTokenizer()
def counter(text, column_text, quantity):
    all_words = ' '.join([text for text in text[column_text]])
    token_phrase = token_space.tokenize(all_words)
    frequency = nltk.FreqDist(token_phrase)
    df_frequency = pd.DataFrame({"Word": list(frequency.keys()),
                                 "Frequency": list(frequency.values())})
    df_frequency = df_frequency.nlargest(columns = "Frequency", n = quantity)
    plt.figure(figsize=(15,8))
    ax = sns.barplot(data = df_frequency, x = "Word", y = "Frequency", color = 'green')
    ax.set(ylabel = "Count")
    plt.xticks(rotation='vertical')
    plt.show()
```

In [116]:
```
#frequency of most fake words
counter(df2[df2['label'] == 0], 'stemming_text', 20)
```



In [117]:
```
#frequency of most real word
counter(df2[df2['label'] == 1], 'stemming_text', 20)
```



Hence the cleaning of the text is done now the text data is not be understood by the machine therefore it is important to convert the text data to some integers, or floating-point values. And here comes the term of feature extraction i.e CountVectorizer.

CountVectorizer(): To use textual data for predictive modeling, the text must be parsed to remove certain words – this process is called **tokenization**. These words need to then be encoded as integers, or floating-point values, for use as inputs in machine learning algorithms. This process is called **feature extraction (or vectorization)**.
Scikit-learn's CountVectorizer is used to convert a collection of text documents to a vector of term/token counts. It also enables the pre-processing of text data before generating the vector representation. This functionality makes it a highly flexible feature representation module for text.

Applying CountVectorizer in both Stemming and Lemmatizing corpus.

```
In [24]: #countvectoriser with stemming

         from sklearn.feature_extraction.text import CountVectorizer
         CV=CountVectorizer(max_features=5000, ngram_range=(1,3))
         x_stem=CV.fit_transform(corpus_stem).toarray()
         x_stem

Out[24]: array([[0, 0, 0, ..., 0, 0, 0],
                [0, 1, 1, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                ...,
                [0, 0, 0, ..., 1, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0]], dtype=int64)

In [25]: print(x_stem.shape)

         (6335, 5000)

In [26]: #countvectorizer with lemmetizing

         from sklearn.feature_extraction.text import CountVectorizer
         CV=CountVectorizer(max_features=5000)
         x_lem=CV.fit_transform(corpus_lemmetize).toarray()
         x_lem

Out[26]: array([[0, 0, 0, ..., 0, 0, 0],
                [0, 0, 1, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                ...,
                [0, 0, 0, ..., 1, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0]], dtype=int64)

In [27]: print(x_lem.shape)

         (6335, 5000)
```

There is also another term **TF-IDF Vectorizer.**
TF-IDF: TF-IDF is an abbreviation for Term Frequency Inverse Document Frequency. This is a very common algorithm to transform the text into a meaningful representation of numbers which is used to fit machine algorithms for prediction. Count Vectorizer gives the number of the frequency concerning index of vocabulary whereas *tf-idf* considers overall documents of the weight of words.
Applying TF-IDF Vectorizer in this problem statement.

```
In [28]: #TF-IDF for stemming

         from sklearn.feature_extraction.text import TfidfVectorizer
         tf_stem=TfidfVectorizer()
         x_tf_stem=tf_stem.fit_transform(corpus_stem)
         print(x_tf_stem.shape)

         (6335, 43312)

In [129]: #TF-IDF for lemmatizing

          from sklearn.feature_extraction.text import TfidfVectorizer
          tf_stem=TfidfVectorizer()
          x_tf_lem=tf_stem.fit_transform(corpus_lemmetize)
          print(x_tf_lem.shape)

          (6335, 57599)
```

Let’s check the target feature(y):

```
In [31]: y=df['label']
         y[:5]

Out[31]: 0    0
         1    0
         2    1
         3    0
         4    1
         Name: label, dtype: int32
```

Modeling:

Let's know few terms for modeling:

1)train_test_split() is a function in **Sklearn model selection** for splitting data arrays into **two subsets**: for training data and for testing data. With this function, you don't need to divide the dataset manually.
By default, Sklearn **train_test_split** will make random partitions for the two subsets. However, you can also specify a random state for the operation.

2) The **multinomial Naive Bayes** classifier is suitable for classification with discrete features (e.g., word counts for text classification). The **multinomial** distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work. Using **MultinomialNB** in this problem. It is another useful Naïve Bayes classifier. It assumes that the features are drawn from a simple Multinomial distribution. The Scikit-learn
provides **sklearn.naive_bayes.MultinomialNB** to implement the Multinomial Naïve Bayes algorithm for classification.

3) A **confusion matrix** is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. The confusion matrix itself is relatively simple to understand, but the related terminology can be confusing. Confusion Matrix is a useful machine learning method that allows you to measure Recall, Precision, Accuracy, and AUC-ROC curve. Below given is an example to know the terms True Positive, True Negative, False Negative, and True Negative. True Positive: You projected positive and its turns out to be true.

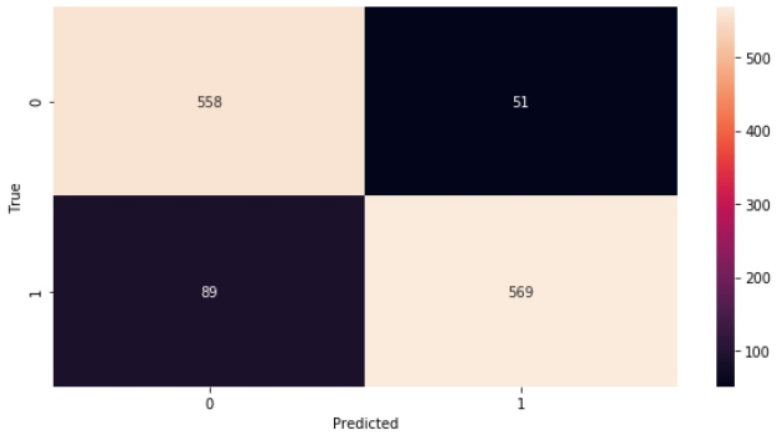
4) The **classification report** is about key metrics in a classification problem. You'll have precision, recall, f1-score, and support for each class you're trying to find. The recall means "how many of this class you find over the whole number of elements of this class".

Now I'm using CountVectorizer with the stemming text and MultinomialNB() and let's check the accuracy.

```
In [136]: from sklearn.model_selection import train_test_split
          X_train, X_test, y_train, y_test= train_test_split(x_stem,y,test_size=0.2)

In [137]: X_train.shape, X_test.shape, y_train.shape, y_test.shape

Out[137]: ((5068, 5000), (1267, 5000), (5068,), (1267,))

In [138]: from sklearn.naive_bayes import MultinomialNB
          model=MultinomialNB().fit(X_train,y_train)

In [139]: model.score(X_test,y_test)

Out[139]: 0.8895027624309392

In [140]: from sklearn.metrics import confusion_matrix
          y_pred=model.predict(X_test)
          cm=confusion_matrix(y_test,y_pred)
          cm

Out[140]: array([[558,  51],
                 [ 89, 569]], dtype=int64)

In [141]: from sklearn.metrics import classification_report
          print(classification_report(y_test, y_pred))

                        precision    recall  f1-score   support

                     0       0.86      0.92      0.89       609
                     1       0.92      0.86      0.89       658

              micro avg       0.89      0.89      0.89      1267
              macro avg       0.89      0.89      0.89      1267
           weighted avg       0.89      0.89      0.89      1267
```

Now let’s plot the heatmap() of the confusion_matrix for the visualization of the confusion_matrix.

```
In [142]: import matplotlib.pyplot as plt
          import seaborn as sns
          %matplotlib inline
          plt.figure(figsize=(10,5))
          sns.heatmap(cm, annot=True, fmt='d')
          plt.xlabel('Predicted')
          plt.ylabel('True')

Out[142]: Text(69,0.5,'True')
```



By using CountVectorizer and Stemming I got an accuracy of 87%.

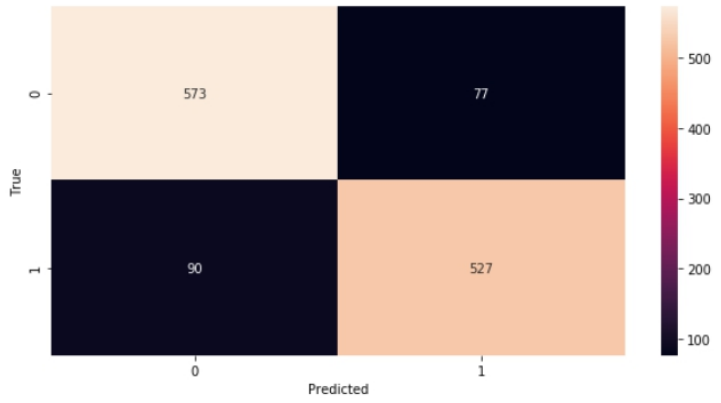
Now Let's check with CountVectorizer and Lemmetizing.

```
In [151]: from sklearn.model_selection import train_test_split
          X_train, X_test, y_train, y_test= train_test_split(x_lem,y,test_size=0.2)

In [152]: X_train.shape, X_test.shape, y_train.shape, y_test.shape

Out[152]: ((5068, 5000), (1267, 5000), (5068,), (1267,))

In [153]: from sklearn.naive_bayes import MultinomialNB
          model=MultinomialNB().fit(X_train,y_train)

In [154]: model.score(X_test,y_test)

Out[154]: 0.8681925808997633

In [155]: from sklearn.metrics import confusion_matrix
          y_pred=model.predict(X_test)
          cm=confusion_matrix(y_test,y_pred)
          cm

Out[155]: array([[573,  77],
                 [ 90, 527]], dtype=int64)

In [156]: from sklearn.metrics import classification_report
          print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.86	0.88	0.87	650
1	0.87	0.85	0.86	617
micro avg	0.87	0.87	0.87	1267
macro avg	0.87	0.87	0.87	1267
weighted avg	0.87	0.87	0.87	1267

Now let's plot the heatmap() of the confusion_matrix for the visualization of the confusion_matrix.

```
n [157]: plt.figure(figsize=(10,5))
         sns.heatmap(cm, annot=True, fmt='d')
         plt.xlabel('Predicted')
         plt.ylabel('True')

ut[157]: Text(69,0.5,'True')
```



By using CountVectorizer and Lemmetizing I got an accuracy of 86%.

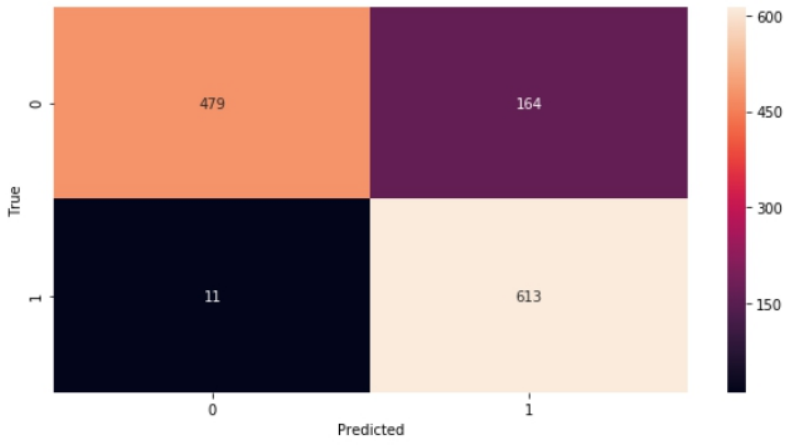
Now let’s check by using TF-IDF and Stemming

```
In [158]: from sklearn.model_selection import train_test_split
          X_train, X_test, y_train, y_test= train_test_split(x_tf_stem,y,test_size=0.2)

In [159]: X_train.shape, X_test.shape, y_train.shape, y_test.shape

Out[159]: ((5068, 43312), (1267, 43312), (5068,), (1267,))

In [160]: from sklearn.naive_bayes import MultinomialNB
          model=MultinomialNB().fit(X_train,y_train)

In [161]: model.score(X_test,y_test)

Out[161]: 0.861878453038674

In [162]: from sklearn.metrics import confusion_matrix
          y_pred=model.predict(X_test)
          cm=confusion_matrix(y_test,y_pred)
          cm

Out[162]: array([[479, 164],
                 [ 11, 613]], dtype=int64)

In [163]: from sklearn.metrics import classification_report
          print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.98	0.74	0.85	643
1	0.79	0.98	0.88	624
micro avg	0.86	0.86	0.86	1267
macro avg	0.88	0.86	0.86	1267
weighted avg	0.88	0.86	0.86	1267

Now let’s plot the heatmap() of the confusion_matrix for the visualization of the confusion_matrix.

```
In [164]: plt.figure(figsize=(10,5))
          sns.heatmap(cm, annot=True, fmt='d')
          plt.xlabel('Predicted')
          plt.ylabel('True')

Out[164]: Text(69,0.5,'True')
```



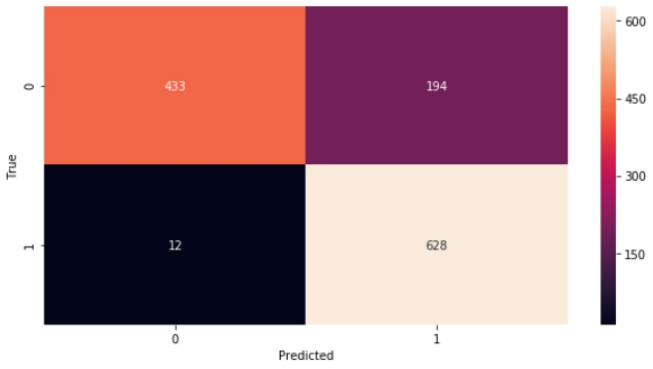
By using TF-IDF and Stemming I got an accuracy of 86%.

Now let’s check by using TF-IDF and Lemmetizing

```
In [165]: from sklearn.model_selection import train_test_split
          X_train, X_test, y_train, y_test= train_test_split(x_tf_lem,y,test_size=0.2)

In [166]: x_tf_lem.shape
Out[166]: (6335, 57599)

In [168]: from sklearn.naive_bayes import MultinomialNB
          model=MultinomialNB().fit(X_train,y_train)

In [169]: model.score(X_test,y_test)
Out[169]: 0.8374112075769534

In [170]: from sklearn.metrics import confusion_matrix
          y_pred=model.predict(X_test)
          cm=confusion_matrix(y_test,y_pred)
          cm
Out[170]: array([[433, 194],
                 [ 12, 628]], dtype=int64)

In [171]: from sklearn.metrics import classification_report
          print(classification_report(y_test, y_pred))
                        precision    recall  f1-score   support

                     0       0.97      0.69      0.81       627
                     1       0.76      0.98      0.86       640

              micro avg       0.84      0.84      0.84      1267
              macro avg       0.87      0.84      0.83      1267
           weighted avg       0.87      0.84      0.83      1267
```

Now let’s plot the heatmap() of the confusion_matrix for the visualization of the confusion_matrix.

```
In [172]: plt.figure(figsize=(10,5))
          sns.heatmap(cm, annot=True, fmt='d')
          plt.xlabel('Predicted')
          plt.ylabel('True')
Out[172]: Text(69,0.5,'True')
```



By using TF-IDF and Lemmetizing I got an accuracy of 83%.

Hence, we can go through that with Lememtizing and CountVectorizer as it is giving almost 86%, with Stemming and CountVectorizer it is giving 88% and TF-IDF with stemming is also giving 86%.

Let's check the accuracy after applying hyperparameter in MultinomialNB

```
In [173]: from sklearn.model_selection import train_test_split
          X_train, X_test, y_train, y_test= train_test_split(x_stem,y,test_size=0.2)

          classifier=MultinomialNB(alpha=0.1)

          from sklearn import metrics
          import numpy as np
          previous_score=0
          for alpha in np.arange(0,1,0.1):
              sub_classifier=MultinomialNB(alpha=alpha)
              sub_classifier.fit(X_train,y_train)
              y_pred=sub_classifier.predict(X_test)
              score = metrics.accuracy_score(y_test, y_pred)
              if score>previous_score:
                  classifier=sub_classifier
              print("Alpha: {}, Score : {}".format(alpha,score))

          C:\Users\Public\anacondainstallation\lib\site-packages\sklearn\naive_bayes.py:480: UserWarning: alpha too small will result in
          numeric errors, setting alpha = 1.0e-10
            'setting alpha = %.1e' % _ALPHA_MIN)

          Alpha: 0.0, Score : 0.8808208366219415
          Alpha: 0.1, Score : 0.8792423046566693
          Alpha: 0.2, Score : 0.8784530386740331
          Alpha: 0.30000000000000004, Score : 0.8784530386740331
          Alpha: 0.4, Score : 0.8784530386740331
          Alpha: 0.5, Score : 0.8784530386740331
          Alpha: 0.6000000000000001, Score : 0.8784530386740331
          Alpha: 0.7000000000000001, Score : 0.8784530386740331
          Alpha: 0.8, Score : 0.877663772691397
          Alpha: 0.9, Score : 0.877663772691397
```

Hence, it's better to select alpha=0.0, as it is giving maximum accuracy 88%.

With some others algorithm:

Now let's apply with some other classification algorithim like Now let's check with other classifier algorithm like DecisionTreeClassifier, RandamForestClassifier, SVM, LogisticRegression by using GridSearchCV and crossvalidation.

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression).

The **Random Forest Classifier** is a set of decision trees from randomly selected subset of training set. It aggregates the votes from different decision trees to decide the final class of the test object.

Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. The decision tree splits the nodes on all available variables and then selects the split which results in the most homogeneous sub-nodes.

SVC()-The objective of a Linear **SVC** (Support Vector Classifier) is to fit the data you provide, returning a "best fit" hyperplane that divides or categorizes your data. From there, after getting the hyperplane, you can then feed some features to your classifier to see what the "predicted" class is.

GridSearchCV is a library function that is a member of sklearn's model_selection package. It helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. So, in the end, you can select the best parameters from the listed hyperparameters.

cross_val_score()-A cross-validation generator to use. If int, determines the number of folds in StratifiedKFold if y is binary or multiclass and estimator is a classifier, or the number of folds in KFold otherwise. If None, it is equivalent to cv=3.

In [62]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import  DecisionTreeClassifier
from sklearn.svm import SVC

from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.metrics import classification_report,confusion_matrix
```

In [63]:
```
model_params = {
    'svm': {
        'model': SVC(gamma='auto'),
        'params' : {
            'C': [1,10,20,25,30,40],
            'kernel': ['rbf','linear']
        }
    },
    'random_forest': {
        'model': RandomForestClassifier(),
        'params' : {
            'n_estimators': [1,5,10,15,20,25,30]
        }
    },
    'logistic_regression' : {
        'model': LogisticRegression(solver='liblinear'),
        'params': {
            'C': [1,5,10,15,20,25]
        }
    },
    'decision_tree': {
        'model': DecisionTreeClassifier(),
        'params':{
            'criterion':['gini','entropy']
        }
    }
}
```

In [64]:
```
scores = []

X_train, X_test, y_train, y_test= train_test_split(x_stem,y,test_size=0.2)

for model_name, mp in model_params.items():
    clf =  GridSearchCV(mp['model'], mp['params'], cv=5, return_train_score=False)
    clf.fit(X_train,y_train)
    scores.append({
        'model': model_name,
        'best_score': clf.best_score_,
        'best_params': clf.best_params_
    })

df_score = pd.DataFrame(scores,columns=['model','best_score','best_params'])
df_score
```

Out[64]:

	model	best_score	best_params
0	svm	0.897790	{'C': 40, 'kernel': 'rbf'}
1	random_forest	0.885754	{'n_estimators': 30}
2	logistic_regression	0.908642	{'C': 1}
3	decision_tree	0.815706	{'criterion': 'entropy'}

Hence, we can conclude that this problem statement Fake_News_Classifier work with Logistic_Regression with accuracy of 90%, SVM with accuracy of 90%, and also with Random_Forest_Classifier 88%.

Conclusion:

The task of classifying news manually requires in-depth knowledge of the domain and expertise to identify anomalies in the text. In this research, we discussed the problem of classifying fake news articles using machine learning models and ensemble techniques. The data we used in our work is collected from the World Wide Web and contains news articles from various domains to cover most of the news rather than specifically classifying political news. The primary aim of the research is to identify patterns in text that differentiate fake articles from true news. We extracted different textual features from the articles using an LIWC tool and used the feature set as an input to the models. The learning models were trained and parameter-tuned to obtain optimal accuracy. Some models have achieved comparatively higher accuracy than others. We used multiple performance metrics to compare the results for each algorithm. The ensemble learners have shown an overall better score on all performance metrics as compared to the individual learners.

Fake news detection has many open issues that require the attention of researchers. For instance, to reduce the spread of fake news, identifying key elements involved in the spread of news is an important step. Graph theory and machine learning techniques can be employed to identify the key sources involved in the spread of fake news. Likewise, real-time fake news identification in videos can be another possible future direction.