In [ ]:
```
#!pip install spacy
```

In [ ]:
```
#!python -m spacy download en_core_web_sm
```

In [1]:
```
import warnings
warnings.filterwarnings('ignore')

# From Example Code  https://github.com/Swathiu/Detecting-Fake-Reviews/blob/master/Deception_Detection.py
import pandas as pd
import numpy as np

from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer

from datetime import datetime
from time import time

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, pairwise_distances
from sklearn.metrics import confusion_matrix

import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm

import spacy
```

In [2]:
```
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.max_colwidth', None)
```

In [3]:
```
file_path = "C:/Users/tsaie/OneDrive/Desktop/000 Resumes & Projects/# Projects/DS3 Fake Amazon Reviews/Dataset/"
# apparel = pd.read_csv(file_path + 'amazon_reviews_us_Apparel_v1_00.tsv.gz', compression='gzip', header=0, sep='\t', qu
electronics = pd.read_csv(file_path + 'amazon_reviews_us_Electronics_v1_00.tsv.gz', compression='gzip', header=0, sep='\
print(f"The 'electronics' file has {electronics.shape[0]} rows and {electronics.shape[1]} columns")
electronics.head(3)
```

```
The 'electronics' file has 20000 rows and 15 columns
```

Out[3]:

	marketplace	customer_id	review_id	product_id	product_parent	product_title	product_category	star_rating	helpful_votes	total_vot
0	US	41409413	R2MTG1GCZLR2DK	B00428R89M	112201306	yoomall 5M Antenna WIFI RP-SMA Female to Male Extensionl Cable	Electronics	5	0	
1	US	49668221	R2HBOEM8LE9928	B000068O48	734576678	Hosa GPM-103 3.5mm TRS to 1/4" TRS Adaptor	Electronics	5	0	
2	US	12338275	R1P4RW1R9FDPEE	B000GGKOG8	614448099	Channel Master Titan 2 Antenna Preamplifier	Electronics	5	1	

In [4]:
```
electronics_small = electronics[['verified_purchase', 'review_body']]
electronics_small.head()
```

Out[4]:

	verified_purchase	review_body
0	Y	As described.
1	Y	It works as advertising.
2	Y	Works pissa
3	Y	Did not work at all.
4	Y	Works well. Bass is somewhat lacking but is present. Overall pleased with the item.

Create Balanced Dataset

- have same number of rows of verified and unverified reviews

In [5]:
```
def under_sampling(df):
    print("Under-Sampling Data")
    # Count of Reviews
    print("Verified:", sum(df['verified_purchase'] == 'Y'))
    print("Un-Verified:", sum(df['verified_purchase'] == 'N'))

    sample_size = sum(df['verified_purchase'] == 'N')

    authentic_reviews_df = df[df['verified_purchase'] == 'Y']
    fake_reviews_df = df[df['verified_purchase'] == 'N']

    authentic_reviews_us_df = authentic_reviews_df.sample(sample_size)
    under_sampled_df = pd.concat([authentic_reviews_us_df, fake_reviews_df], axis=0)

    print("Under-Sampled Verified", sum(under_sampled_df['verified_purchase'] == 'Y'))
    print("Under-Sampled Un-Verified", sum(under_sampled_df['verified_purchase'] == 'N'))


    # Graph of Data Distribution
    fig, ax = plt.subplots(figsize=(6, 4))
    sns.countplot(x='verified_purchase', data=under_sampled_df)
    plt.title("Count of Reviews")
    plt.show()
    print("Under-Sampling Complete")
    return under_sampled_df
```

In [6]:
```
electronics_equal_weight = under_sampling(electronics_small)
# electronics_equal_weight
```

```
Under-Sampling Data
Verified: 18309
Un-Verified: 1691
Under-Sampled Verified 1691
Under-Sampled Un-Verified 1691
```


```
Under-Sampling Complete
```

Data Cleaning

In [7]:
```
# Pre-processing Text Reviews
def data_cleaning(df):
    # Removing emtpy cells
    df.dropna(inplace=True)
    df['review_body_cleaned'] = df['review_body']

    # Replace HTML keywords with blank space ("&quot;", "br", "&#34")
    remove_dict = {"<br /><br />": " ", "<br />": " ", "br": " ", "&quot;": " ", "&#34": " "}
    for key, val in remove_dict.items():
        df['review_body_cleaned'] = df['review_body_cleaned'].apply(
            lambda x: x.replace(key, val))

    print("\n######## Remove HTML Keywords Complete ########")

    # Remove Punctuations and numbers
    tokenizer = RegexpTokenizer(r'\w+')
    df['review_body_cleaned'] = df['review_body_cleaned'].apply(
        lambda x: ' '.join([word for word in tokenizer.tokenize(x)]))

    remove_dict = {"0": "", "1": "", "2": "", "3": "", "4": "", "5": "", "6": "", "7": "", "8": "", "9": "",
                   "(": "", ")":""}
    for key, val in remove_dict.items():
        df['review_body_cleaned'] = df['review_body_cleaned'].apply(
            lambda x: x.replace(key, val))

    print("\n######## Remove Punctuation and Numbers Complete ########")

    # Lowercase Words
    df['review_body_cleaned'] = df['review_body_cleaned'].str.lower()

    print("\n######## Lowercase Complete ########")

    # Remove Stop Words.
    stop = stopwords.words('english')
    stop += ["can't", "i'm", "I'm", "i'd", "i've", "i'll", "that's", "there's", "they're"]

    df['review_body_cleaned'] = df['review_body_cleaned'].apply(
        lambda x: ' '.join([word for word in x.split() if word.strip() not in stop]))

    print("\n######## Remove Stop Words Complete ########")

    # Lemmatization using .lemma_
    nlp = spacy.load('en_core_web_sm', disable=['parser', 'ner'])
    df['review_body_cleaned'] = df['review_body_cleaned'].apply(
        lambda x: ' '.join([token.lemma_ for token in nlp(x)]))

    print("\n######## Data Cleaning Complete ########")

    return df
```

In [8]:
```
# Clean the dataset
electronics_cleaned = data_cleaning(electronics_equal_weight)
electronics_cleaned.head()
```

```
######## Remove HTML Keywords Complete ########

######## Remove Punctuation and Numbers Complete ########

######## Lowercase Complete ########

######## Remove Stop Words Complete ########

######## Data Cleaning Complete ########
```

Out[8]:

	verified_purchase	review_body	review_body_cleaned
14751	Y	Small and compact. Bought it as a gift for family member and they are very happy with it.	small compact buy gift family member happy
13393	Y	Overall audio quality is good. However, I feel its less on bass than my old Sennheiser CX 180. Good noise cancellation and battery life.	overall audio quality good however feel less bass old sennheiser cx good noise cancellation battery life
		Amazing!! Seriously, I want to order a backup pair in case they die as Sony isn't	amazing seriously want order backup pair case

Feature Engineering + Prepare Data for Machine Learning

Add Bigrams

In [9]:
```
# https://stackoverflow.com/questions/48331315/how-to-extract-all-the-ngrams-from-a-text-dataframe-column-in-different-or

from collections import Counter
from nltk import ngrams
from itertools import chain

def find_ngrams(input_list, n):
    return list(zip(*[input_list[i:] for i in range(n)]))


electronics_cleaned['bigrams'] = electronics_cleaned['review_body_cleaned'].map(lambda x: find_ngrams(x.split(), 2))
electronics_cleaned.head()
```

Out[9]:

	verified_purchase	review_body	review_body_cleaned	bigrams
14751	Y	Small and compact. Bought it as a gift for family member and they are very happy with it.	small compact buy gift family member happy	[(small, compact), (compact, buy), (buy, gift), (gift, family), (family, member), (member, happy)]
13393	Y	Overall audio quality is good. However, I feel its less on bass than my old Sennheiser CX 180. Good noise cancellation and battery life.	overall audio quality good however feel less bass old sennheiser cx good noise cancellation battery life	[(overall, audio), (audio, quality), (quality, good), (good, however), (however, feel), (feel, less), (less, bass), (bass, old), (old, sennheiser), (sennheiser, cx), (cx, good), (good, noise), (noise, cancellation), (cancellation, battery), (battery, life)]
5155	Y	Amazing!! Seriously, I want to order a backup pair in case they die as Sony isn't making these anymore. Sound quality is as good as beats, not too bassy, pretty balanced sound. For the price you absolutely can't beat these.	amazing seriously want order backup pair case die sony make anymore sound quality good beat bassy pretty balanced sound price absolutely beat	[(amazing, seriously), (seriously, want), (want, order), (order, backup), (backup, pair), (pair, case), (case, die), (die, sony), (sony, make), (make, anymore), (anymore, sound), (sound, quality), (quality, good), (good, beat), (beat, bassy), (bassy, pretty), (pretty, balanced), (balanced, sound), (sound, price), (price, absolutely), (absolutely, beat)]
11111	Y	Works as advertised. Have had no trouble with this product. It is just what I needed!	work advertised trouble product need	[(work, advertised), (advertised, trouble), (trouble, product), (product, need)]
3932	Y	This little thing is unbelievable. Easy to set up..easy to understand. Battery life is amazing,quality is on a different level for a Bluetooth speaker. Very loud..and it does have some bass to it which I think is pretty cool. You can always change the colors if you want . Overall...I'm very pleased. :)	little thing unbelievable easy set easy understand battery life amazing quality different level bluetooth speaker loud bass think pretty cool always change color want overall please	[(little, thing), (thing, unbelievable), (unbelievable, easy), (easy, set), (set, easy), (easy, understand), (understand, battery), (battery, life), (life, amazing), (amazing, quality), (quality, different), (different, level), (level, bluetooth), (bluetooth, speaker), (speaker, loud), (loud, bass), (bass, think), (think, pretty), (pretty, cool), (cool, always), (always, change), (change, color), (color, want), (want, overall), (overall, please)]

In [10]:
```
electronics_un_verified = electronics_cleaned[electronics_cleaned['verified_purchase'] == 'N']
electronics_verified = electronics_cleaned[electronics_cleaned['verified_purchase'] == 'Y']

electronics_un_verified.tail(1)
```

Out[10]:

	verified_purchase	review_body	review_body_cleaned	bigrams
19952	N	I use them at work for my chargers. Keeps cords looking organized and I love it.	use work charger keep cord look organize love	[(use, work), (work, charger), (charger, keep), (keep, cord), (cord, look), (look, organize), (organize, love)]

In [11]:
```
verified_bigrams = electronics_verified['bigrams'].tolist()
verified_bigrams = list(chain(*verified_bigrams))

verified_bigram_counts = Counter(verified_bigrams)
verified_bigram_counts.most_common(20)
```

Out[11]:
```
[(('work', 'great'), 118),
 (('sound', 'quality'), 74),
 (('work', 'well'), 63),
 (('sound', 'great'), 47),
 (('work', 'perfectly'), 45),
 (('great', 'sound'), 41),
 (('work', 'fine'), 40),
 (('great', 'price'), 40),
 (('great', 'product'), 33),
 (('sound', 'good'), 30),
 (('good', 'quality'), 26),
 (('can', 'not'), 26),
 (('easy', 'use'), 25),
 (('highly', 'recommend'), 25),
 (('ear', 'bud'), 23),
 (('good', 'sound'), 22),
 (('battery', 'life'), 21),
 (('good', 'product'), 21),
 (('would', 'recommend'), 20),
 (('good', 'price'), 19)]
```

In [12]:
```
un_verified_bigrams = electronics_un_verified['bigrams'].tolist()
un_verified_bigrams = list(chain(*un_verified_bigrams))

un_verified_bigram_counts = Counter(un_verified_bigrams)
un_verified_bigram_counts.most_common(20)
```

Out[12]:
```
[(('sound', 'quality'), 265),
 (('bluetooth', 'speaker'), 133),
 (('good', 'sound'), 99),
 (('work', 'well'), 98),
 (('work', 'great'), 94),
 (('exchange', 'honest'), 93),
 (('receive', 'product'), 88),
 (('listen', 'music'), 88),
 (('battery', 'life'), 84),
 (('honest', 'review'), 83),
 (('sound', 'good'), 80),
 (('sound', 'great'), 74),
 (('great', 'sound'), 69),
 (('easy', 'use'), 66),
 (('unbiased', 'review'), 60),
 (('honest', 'unbiased'), 56),
 (('highly', 'recommend'), 55),
 (('product', 'discount'), 52),
 (('can', 'not'), 50),
 (('quality', 'sound'), 49)]
```

Let's call the bigrams in vertified reviews "gold_bigrams" and the bigrams in unverified reviews "fake_bigrams"

count = number of gold/fake_bigrams in a review

percent = number of gold/fake_bigrams as a percentage of total number of bigrams in a review.

simple score = sum of the gold/fake_bigrams' popularity scores (calculated using the bigram's count in the Counter)

normalized score = simple score / total bigram count

In [14]:
```
def get_bigram_count(bigrams, bigram_dict):
    count = 0
    for bigram in bigrams:
        if bigram in bigram_dict.keys():
            count += 1
    return count

def get_bigram_simple_score(bigrams, bigram_dict):
    score = 0
    for bigram in bigrams:
        if bigram in bigram_dict.keys():
            score += bigram_dict[bigram]
    return score
```

In [15]:
```
electronics_equal_weight['bigram_count'] = electronics_equal_weight['bigrams'].apply(
    lambda x: len(x))
```

In [16]:
```
# fake

fake_bigram_dict = dict(un_verified_bigram_counts) # fake_bigram_dict = dict(un_verified_bigram_counts.most_common(30))

electronics_equal_weight['fake_bigram_count'] = electronics_equal_weight['bigrams'].apply(
    lambda x: get_bigram_count(x, fake_bigram_dict))

electronics_equal_weight['fake_bigram_percent'] = electronics_equal_weight['fake_bigram_count'] / electronics_equal_weigl

electronics_equal_weight['fake_bigram_simple_score'] = electronics_equal_weight['bigrams'].apply(
    lambda x: get_bigram_simple_score(x, fake_bigram_dict))

electronics_equal_weight['fake_bigram_normalized_score'] = electronics_equal_weight['fake_bigram_simple_score'] / electr
```

In [17]:
```
# gold

gold_bigram_dict = dict(verified_bigram_counts) # gold_bigram_dict = dict(verified_bigram_counts.most_common(30))

electronics_equal_weight['gold_bigram_count'] = electronics_equal_weight['bigrams'].apply(
    lambda x: get_bigram_count(x, gold_bigram_dict))

electronics_equal_weight['gold_bigram_percent'] = electronics_equal_weight['gold_bigram_count'] / electronics_equal_weigl

electronics_equal_weight['gold_bigram_simple_score'] = electronics_equal_weight['bigrams'].apply(
    lambda x: get_bigram_simple_score(x, gold_bigram_dict))

electronics_equal_weight['gold_bigram_normalized_score'] = electronics_equal_weight['gold_bigram_simple_score'] / electr
```

In [18]:
```
# Fill all the NaN values with zero
electronics_equal_weight = electronics_equal_weight.fillna(0)
electronics_equal_weight.tail(1)
```

Out[18]:

	verified_purchase	review_body	review_body_cleaned	bigrams	bigram_count	fake_bigram_count	fake_bigram_percent	fake_bigram_simple_
19952	N	I use them at work for my chargers. Keeps cords looking organized and I love it.	use work charger keep cord look organize love	[(use, work), (work, charger), (charger, keep), (keep, cord), (cord, look), (look, organize), (organize, love)]	7	7	1.0	

Use Machine Learning to Make Predictions for Verified VS. Unverified

LABELS:

- 1 = verified review
- 0 = unverified review

In [19]:
```
def semi_supervised_learning(df, model, algorithm, threshold=0.8, iterations=40):
    df = df.copy()

    df_unlabled = df[['fake_bigram_count', 'fake_bigram_percent', 'fake_bigram_simple_score', 'fake_bigram_normalized_sc
             'gold_bigram_count', 'gold_bigram_percent', 'gold_bigram_simple_score', 'gold_bigram_normalized_score']]

    df['verified_purchase'] = df['verified_purchase'].apply(lambda x: 1 if x == 'Y' else 0)
    print("Training " + algorithm + " Model")
    labels = df['verified_purchase']

    train_data, test_data, train_label, test_label = train_test_split(df_unlabled, labels, test_size=0.25, random_state=

    test_data_copy = test_data.copy()
    test_label_copy = test_label.copy()

    all_labeled = False

    current_iteration = 0

    pbar = tqdm(total=iterations)

    while not all_labeled and (current_iteration < iterations):
        current_iteration += 1
        model.fit(train_data, train_label)

        probabilities = model.predict_proba(test_data)
        pseudo_labels = model.predict(test_data)

        indices = np.argwhere(probabilities > threshold)

        for item in indices:
            train_data.loc[test_data.index[item[0]]] = test_data.iloc[item[0]]
            train_label.loc[test_data.index[item[0]]] = pseudo_labels[item[0]]
        test_data.drop(test_data.index[indices[:, 0]], inplace=True)
        test_label.drop(test_label.index[indices[:, 0]], inplace=True)

        print("--" * 20)

        if len(test_data) == 0:
            print("Exiting loop")
            all_labeled = True
        pbar.update(1)

    pbar.close()
    predicted_labels = model.predict(test_data_copy)

    print(algorithm + ' Model Results')
    print('--' * 20)
    print('Accuracy Score : ' + str(accuracy_score(test_label_copy, predicted_labels)))
    print('Precision Score : ' + str(precision_score(test_label_copy, predicted_labels, pos_label=1)))
    print('Recall Score : ' + str(recall_score(test_label_copy, predicted_labels, pos_label=1)))
    print('F1 Score : ' + str(f1_score(test_label_copy, predicted_labels, pos_label=1)))
    print('Confusion Matrix : \n' + str(confusion_matrix(test_label_copy, predicted_labels)))
    plot_confusion_matrix(test_label_copy, predicted_labels, classes=[1, 0],
                          title=algorithm + ' Confusion Matrix').show()


def plot_confusion_matrix(y_true, y_pred, classes, title=None, cmap=plt.cm.Blues):
    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           xticklabels=classes,
           yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
```

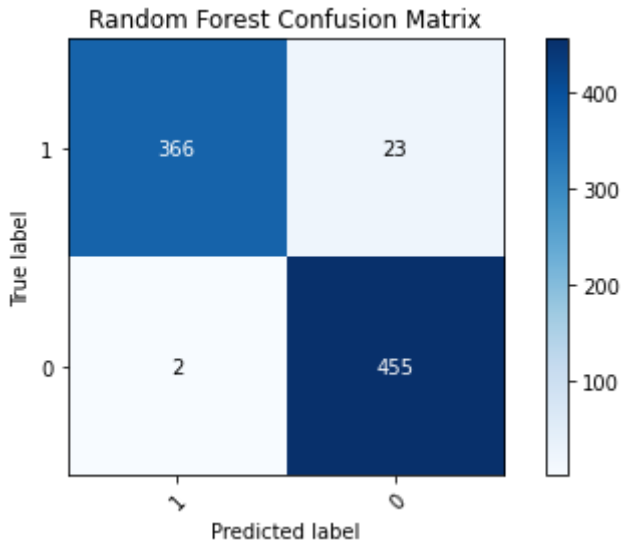
```
    return plt
```

ML Method #1: RandomForestClassifier

In [20]:
```
start_time = time()
rf = RandomForestClassifier(random_state=42, criterion='entropy', max_depth=14, max_features='auto', n_estimators=500)
semi_supervised_learning(electronics_equal_weight, model=rf, threshold=0.7, iterations=15, algorithm='Random Forest')
end_time = time()

print("Time taken : ", end_time - start_time)
```

```
Training Random Forest Model
  7%|▊                                                                | 1/15 [00:02<00:40,  2.86s/it]
----------------------------------------
 13%|█▍                                                               | 2/15 [00:04<00:24,  1.89s/it]
----------------------------------------
 20%|██▏                                                              | 3/15 [00:05<00:19,  1.60s/it]
----------------------------------------
 27%|██▉                                                               | 4/15 [00:06<00:16,  1.46s/i
t]
----------------------------------------
 33%|███▋                                                              | 5/15 [00:07<00:13,  1.40s/i
t]
----------------------------------------
 40%|████▍                                                             | 6/15 [00:09<00:12,  1.36s/i
t]
----------------------------------------
 47%|█████▏                                                            | 7/15 [00:10<00:10,  1.35s/i
t]
----------------------------------------
 53%|█████▉                                                            | 8/15 [00:11<00:09,  1.30s/i
t]
----------------------------------------
 60%|██████▋                                                           | 9/15 [00:12<00:07,  1.28s/i
t]
----------------------------------------
 67%|███████▎                                                         | 10/15 [00:14<00:06,  1.27s/i
t]
----------------------------------------
 73%|████████                                                         | 11/15 [00:15<00:05,  1.32s/i
t]
----------------------------------------
 80%|████████▊                                                        | 12/15 [00:16<00:04,  1.34s/it]
----------------------------------------
 87%|█████████▌                                                       | 13/15 [00:18<00:02,  1.30s/it]
----------------------------------------
 93%|██████████▎                                                      | 14/15 [00:19<00:01,  1.31s/it]
----------------------------------------
100%|███████████| 15/15 [00:20<00:00,  1.40s/it]
----------------------------------------
Random Forest Model Results
----------------------------------------
Accuracy Score : 0.9704491725768322
Precision Score : 0.9518828451882845
Recall Score : 0.9956236323851203
F1 Score : 0.9732620320855615
Confusion Matrix :
[[366  23]
 [  2 455]]
```




```
Time taken :  21.176746129989624
```

ML Method #2: GaussianNB

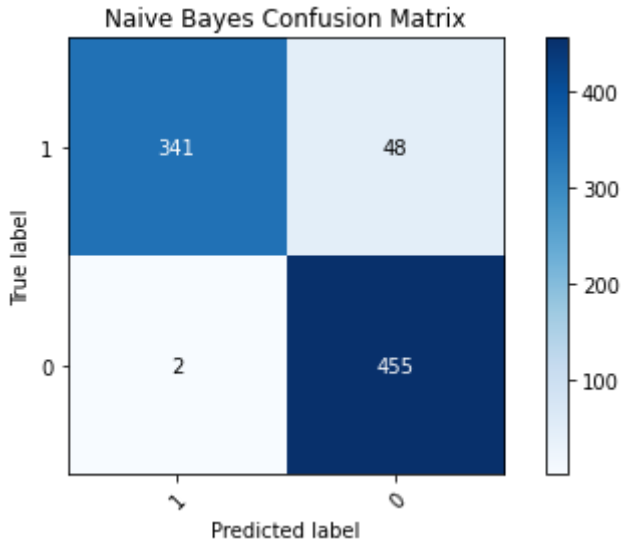
In [21]:
```
start_time = time()
nb = GaussianNB()
semi_supervised_learning(electronics_equal_weight, model=nb, threshold=0.7, iterations=15, algorithm='Naive Bayes')
end_time = time()

print("Time taken : ", end_time - start_time)
```

```
Training Naive Bayes Model

100%|██████████████████████████████████████████████████████████████████████████████| 15/15 [00:01<00:00,  9.30it/s]

----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
Naive Bayes Model Results
----------------------------------------
Accuracy Score : 0.9408983451536643
Precision Score : 0.904572564612326
Recall Score : 0.9956236323851203
F1 Score : 0.9479166666666666
Confusion Matrix :
[[341  48]
 [  2 455]]
```




```
Time taken :  1.7593953609466553
```

END