Shashank Kurakula

March 6, 2023

Spam_detection report

The dataset was downloaded from the Kaggle website https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset?resource=download.

I imported the necessary libraries numpy, and pandas to read the dataset using encoding “ISO-8859-1”. The shape is (5572,2).

We do the following steps:

1. Data Cleaning
2. Exploratory Data Analysis
3. Text Processing
4. Model Building
5. Evaluation and improvements
6. Pickle files
7. Deployment

Data Cleaning:

We are dropping 2,3,4 columns as it has missing values and less number of values using drop from pandas. Since the column names are not specified we rename the columns using df. rename().

The target column has non-numerical values ie spam and ham, we need to convert the column into numbers (0 for ham, 1 for spam) using Label Encoder which comes from sklearn. preprocessing. We do object.fit_transform(target) and put it back into the target of our dataset.

Now we check for null values using df.isnull().sum() and check for duplicates using df.duplicated().sum(). In our data set we have 403 duplicated.

We remove it using df.drop_duplicates(keep='first') and check the shape.

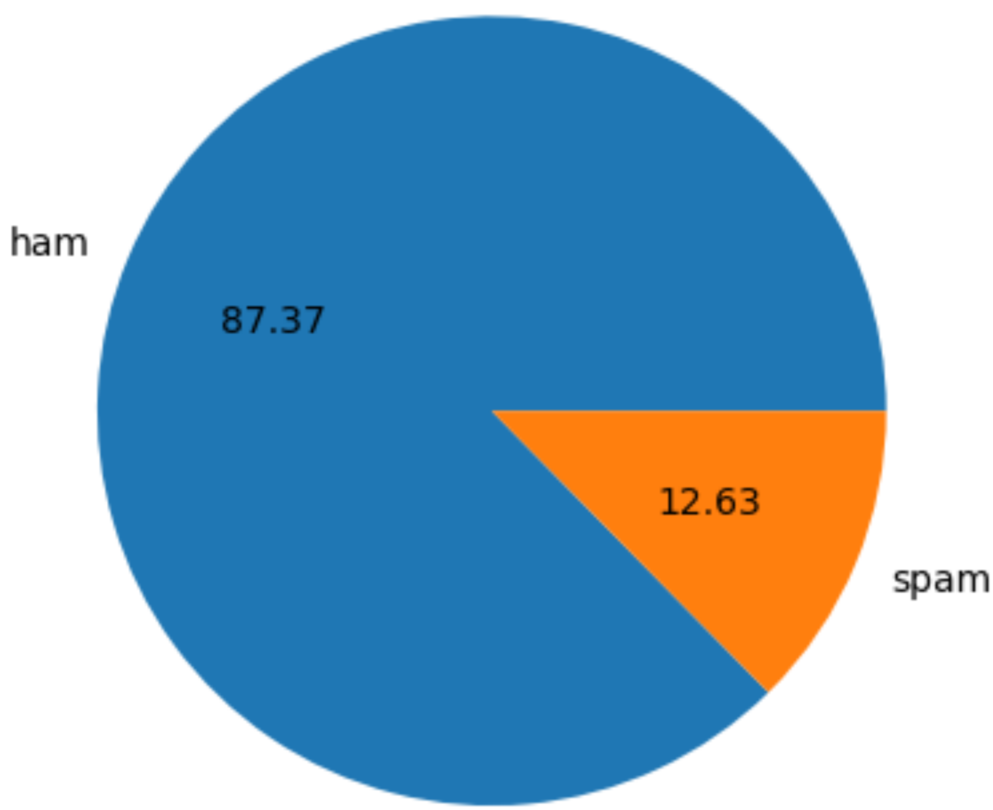
EDA:

We check our target variable for how many spam and ham using df['target'].value_counts().

We got ham as 4516 and spam as 653.

To visualize we plot a pie chart, using

plt.pie(plt.pie(df['target'].value_counts(), labels= ['ham','spam'], autopct='%0.2f'))



The data is imbalanced here.

Text Processing:

We use the nltk library for text processing and we download Punkt

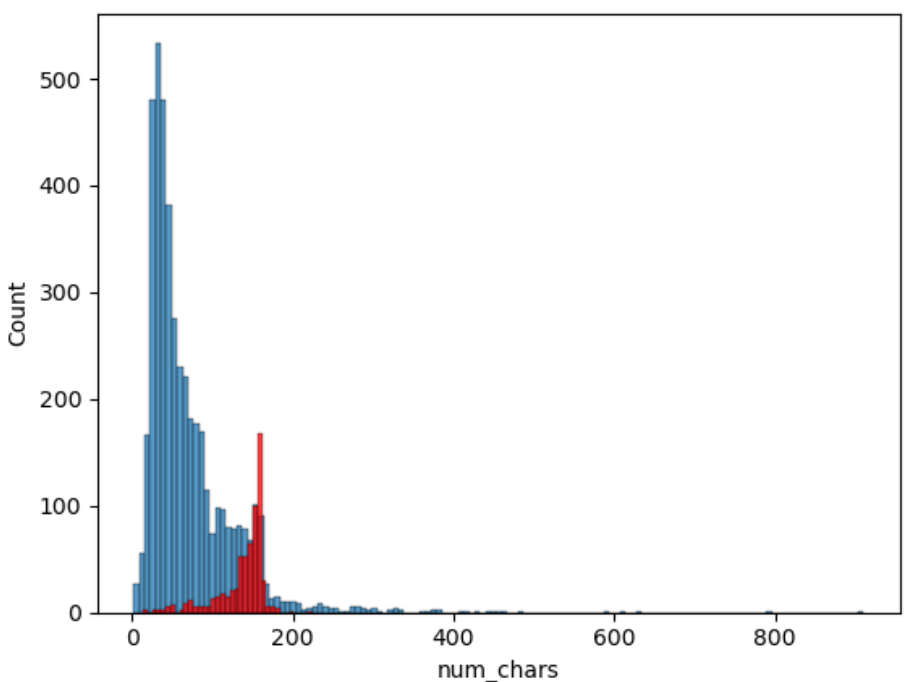
We count the number of chars, number of words, and number of sentences in each msg.

We use apply and lambda functions to perform on each msg in the dataset.

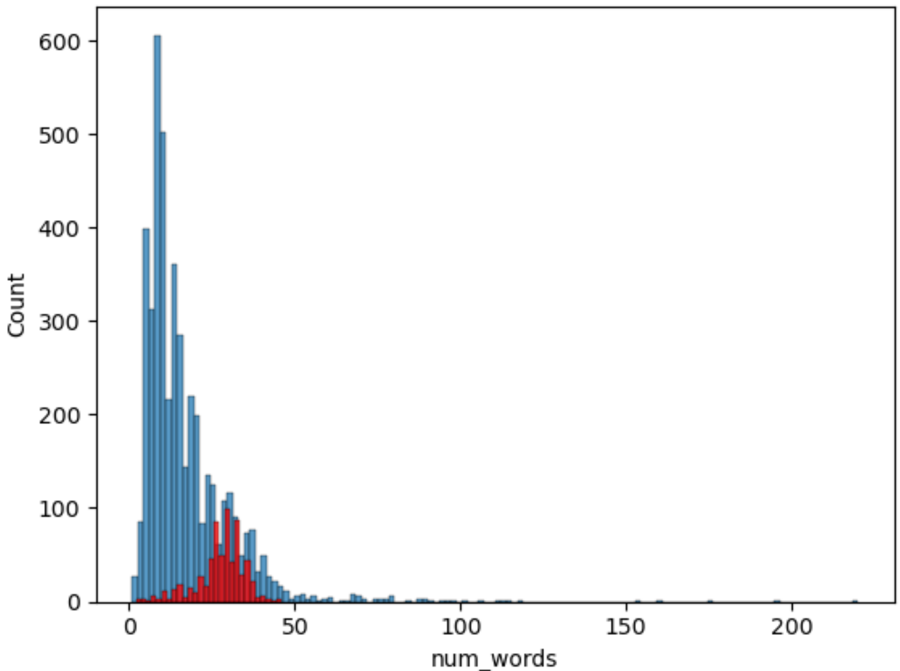
For the number of chars we use apply(len) which gives the number of chars in the msg.

To get words and sentences we use nltk.word_tokenize() and nltk.sent_tokenzie().

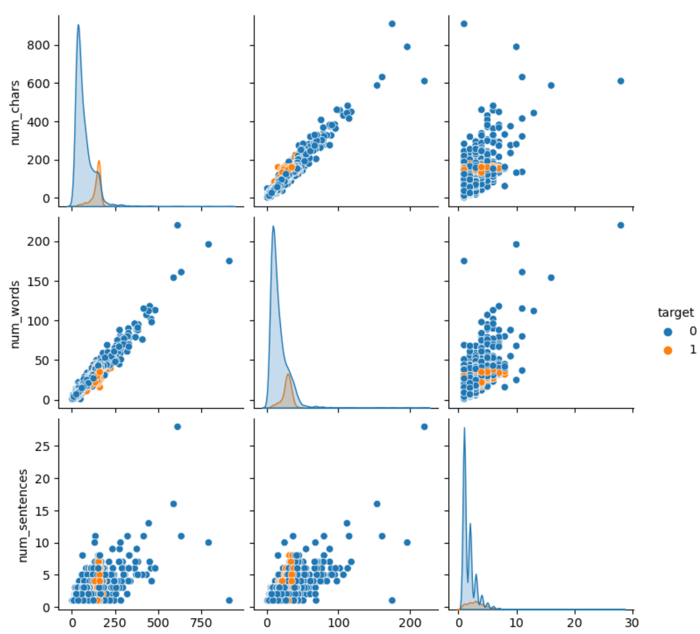
We check the summary of the ham and spam and look at the mean closer.



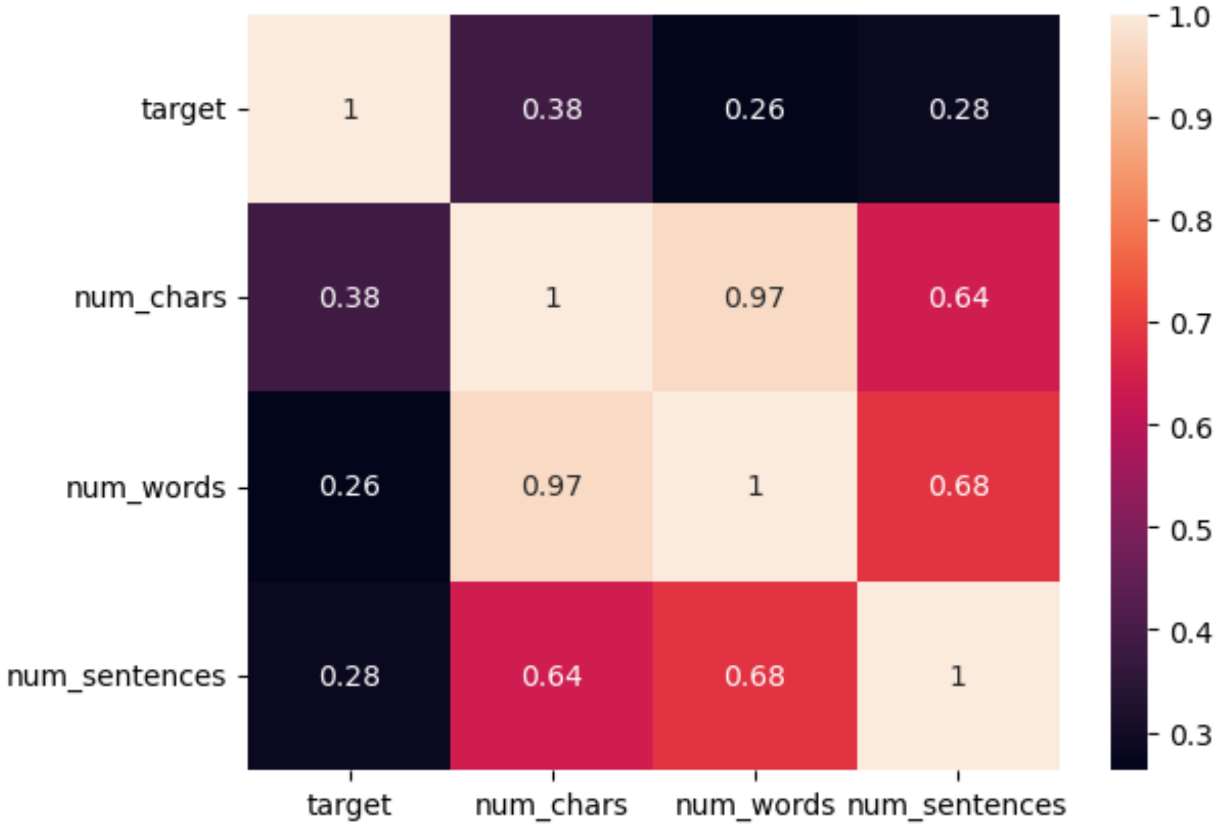
For better understanding, we visualize using seabird. For chars, we have more in ham and less in spam. Same for spam too.



We use sis.pairplot(df, hue=‘target') for further understanding.



We plot a heat map for checking the correlation.



We choose the num of chars for our model since it has a 0.38 correlation with the target variable.

Data Preprocessing:

In this we do :

- Lower case
- Tokenization
- Removing special char
- Remove stop words and punctuation (is, of ,the etc)
- Stemming (lemetization)(go, goes, going is all the same)

We write an important method that will also be used in the app.py file for input data processing.

```
def transform_msg(msg):
    msg = msg.lower()
    msg = nltk.word_tokenize(msg)
    y = []
    for i in msg:
        if i.isalnum():
            y.append(i)
    msg = y[:]
    y.clear()
    for i in msg:
        if i not in stopwords.words('english') and i not in string.punctuation:
            y.append(i)
    msg = y[:]
    y.clear()
    for i in msg:
        y.append(ps.stem(i))
    return " ".join(y)
```

First, we convert into lowercase, tokenize for words, remove special characters.

For removing punctuation and stop words we use stopwords.words('English') and string. punctuation.

For Lemetization we use PorterStemmer from nltk.stem.porter and create an object of it.

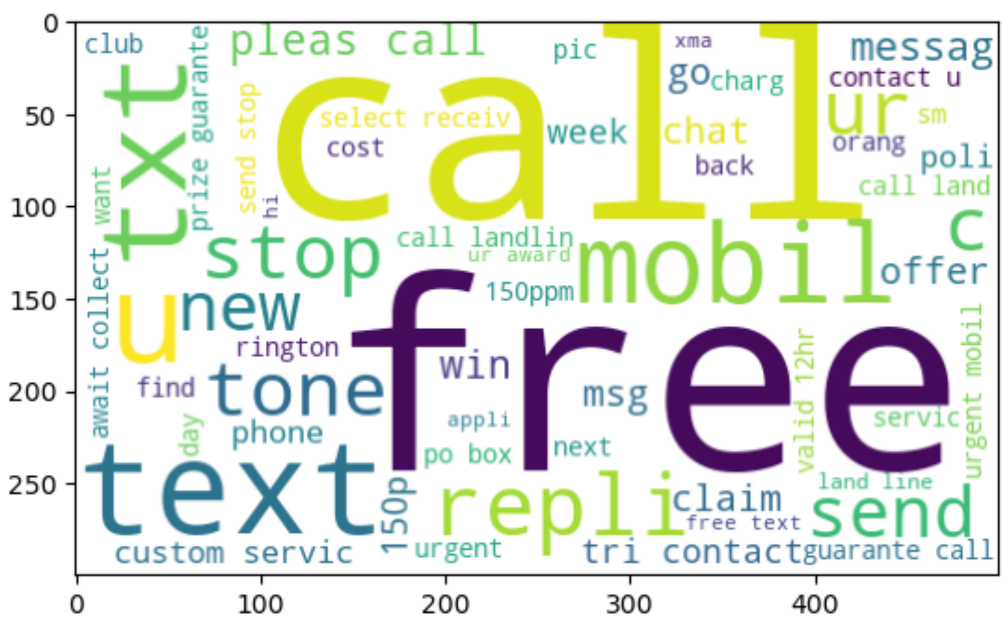
Now we apply the transform_msg() to our df[‘msg’] column and add it to our data set.

We use WordCloud to get the most freq words in ham and spam.

Ex:

wc = WordCloud(width=500,height=300, min_font_size=10, background_color='white')

spam_wc = wc.generate(df[df['target']==1]['transformed_msg'].str.cat(sep=' ‘))



Model Building:

We know that Naive Bayes Classifier works better for text data.

For this we need to convert the data in numerical format.

Hence we can use bag of words or TFIDF vectorizer.

We do tfidf.fit_transform().to array()

Now we split the data in train test split.

We train our model using all three GaussianNB,MultinomialNB,BernoulliNB and check the accuracy and precision score of our model with the test data.

Evaluation and improvements:

Multinominal NB worked better in our case with 95.7% accuracy and a precision score of 1.

I tested with all the classification algorithms to improve the accuracy but it turns out that Multinominal NB is the best.

The algorithms that I have tested with are :

from sklearn.linear_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import AdaBoostClassifier

from sklearn.ensemble import BaggingClassifier

from sklearn.ensemble import ExtraTreesClassifier

from sklearn.ensemble import GradientBoostingClassifier

from xgboost import XGBClassifier.

I have done hyper parameter tuning for my model using these values:

svc = SVC(kernel='sigmoid', gamma=1.0)

knc = KNeighborsClassifier()

mnb = MultinomialNB()

dtc = DecisionTreeClassifier(max_depth=5)

lrc = LogisticRegression(solver='liblinear', penalty = 'l1')

rfc = RandomForestClassifier(n_estimators=50,random_state=5)

abc = AdaBoostClassifier(n_estimators=50, random_state=5)

bc = BaggingClassifier(n_estimators=50, random_state=5)

etc = ExtraTreesClassifier(n_estimators=50, random_state=5)

gbc = GradientBoostingClassifier(n_estimators=50, random_state=5)

xgb = XGBClassifier(n_estimators=50, random_state=5)

The results were:

for KNN

Accuracy 0.90715667311412

Precision 1.0

for MNB

Accuracy 0.9574468085106383

Precision 1.0

for DTC

Accuracy 0.9429400386847195

Precision 0.8053097345132744

for LR

Accuracy 0.9506769825918762

Precision 0.9425287356321839

for RFC

Accuracy 0.9671179883945842

Precision 0.9795918367346939

for AdaBost

Accuracy 0.9680851063829787

Precision 0.9439252336448598

for BC

Accuracy 0.9700193423597679

Precision 0.907563025210084

for ETC

Accuracy 0.971953578336557

Precision 0.9805825242718447

for GBC

Accuracy 0.9487427466150871

Precision 0.9032258064516129

for XGB

Accuracy 0.9709864603481625

Precision 0.9454545454545454

	Algo	Accuracy	Precision
1	KNN	0.907157	1.000000
2	MNB	0.957447	1.000000
8	ETC	0.971954	0.980583
5	RFC	0.967118	0.979592
0	SVC	0.969052	0.961538
10	XGB	0.970986	0.945455
6	AdaBost	0.968085	0.943925
4	LR	0.950677	0.942529
7	BC	0.970019	0.907563
9	GBC	0.948743	0.903226
3	DTC	0.942940	0.805310

Pickle files:

pickle.dump(tfidf,open('vectorizer.pkl','wb'))

pickle.dump(mnb,open(‘mnbmodel.pkl','wb'))

Deployment:

I deployed in Streamlit since it is easy.