import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

import random

import seaborn as sns

import matplotlib.pyplot as plt

# Machine Learning & nltk library

import nltk

from nltk.tokenize import sent\_tokenize, word\_tokenize

from nltk.classify.scikitlearn import SklearnClassifier

from nltk.classify import ClassifierI

from sklearn.naive\_bayes import MultinomialNB, GaussianNB, BernoulliNB

from sklearn.linear\_model import LogisticRegression, SGDClassifier

from sklearn.svm import SVC, LinearSVC

from sklearn.metrics import confusion\_matrix

from statistics import mode

import pickle

df = pd.read\_csv("/kaggle/input/sms-spam-collection-dataset/spam.csv", encoding='latin-1')

df.dropna(how="any", inplace=True, axis=1)

df.columns = ['label', 'message']

df.head()

print(df.shape)

print(df['label'].value\_counts())

df['label'].value\_counts().plot(kind='bar')

# Make a bag of word and count the frequncy of the word

df = df.reset\_index()

all\_words = []

for index, row in df.iterrows():

for word in word\_tokenize(row['message']):

all\_words.append(word.lower())

all\_words = nltk.FreqDist(all\_words)

all\_words.most\_common(50)

# choose most frequent 3,000 words

word\_features = list(all\_words.keys())[:3000]

# Make a documents that contains tokenized words and labels

documents = []

for index, row in df.iterrows():

documents.append((word\_tokenize(row['message']),row['label']))

# Make a feature set that shows which word was in the word\_features

def find\_features(document):

words = set(document)

features = {}

for w in word\_features: # 3,000 frequent words

features[w] = (w in words)

return features

featuresets = [(find\_features(message), label) for (message, label) in documents]

# shuffle with the same seed

random.Random(4).shuffle(featuresets)

# divide training set and test set

training\_set = featuresets[:5000]

testing\_set = featuresets[5000:]

# sklearn.naive\_bayes : NaiveBayesClassifier, MultinomialNB,BernoulliNB

classifier = nltk.NaiveBayesClassifier.train(training\_set)

MNB\_classifier = SklearnClassifier(MultinomialNB())

MNB\_classifier.train(training\_set)

BernoulliNB = SklearnClassifier(BernoulliNB())

BernoulliNB.train(training\_set)

# sklearn.linear\_model : LogisticRegression, SGDClassifier

LogisticRegression = SklearnClassifier(LogisticRegression())

LogisticRegression.train(training\_set)

SGDClassifier = SklearnClassifier(SGDClassifier())

SGDClassifier.train(training\_set)

# sklearn.svm : SVC, LinearSVC, NuSVC

SVC = SklearnClassifier(SVC())

SVC.train(training\_set)

LinearSVC = SklearnClassifier(LinearSVC())

LinearSVC.train(training\_set)

# Build a VoteClassifier Class

class VoteClassifier(ClassifierI):

def \_\_init\_\_(self,\*classifiers):

self.\_classifiers = classifiers

def classify(self, features):

votes = []

for c in self.\_classifiers:

v = c.classify(features)

votes.append(v)

return mode(votes)

def confidence(self, features):

votes = []

for c in self.\_classifiers:

v = c.classify(features)

votes.append(v)

choice\_votes = votes.count(mode(votes))

confidence\_val = choice\_votes / len(votes)

return confidence\_val

voted\_classifier = VoteClassifier(classifier,

MNB\_classifier,

BernoulliNB,

LogisticRegression,

SGDClassifier,

SVC,

LinearSVC)

print("voted\_classifier Accuracy percent: ", (nltk.classify.accuracy(voted\_classifier, testing\_set)\*100))

print("Classification: ", voted\_classifier.classify(testing\_set[0][0]), 'confidence %', voted\_classifier.confidence(testing\_set[0][0]) \* 100)

models = [classifier, MNB\_classifier, BernoulliNB,LogisticRegression,SGDClassifier,SVC,LinearSVC, voted\_classifier]

models\_name = ['classifier', 'MNB\_classifier', 'BernoulliNB','LogisticRegression','SGDClassifier','SVC','LinearSVC', 'voted\_classifier']

accuracy\_dict = {}

for idx, model in enumerate(models):

accuracy\_dict[models\_name[idx]] = round(nltk.classify.accuracy(model, testing\_set)\*100,2)

print(accuracy\_dict)

df = pd.DataFrame(accuracy\_dict, index=[0])

df.plot(kind = 'bar', ylim = [96,100])

{'classifier': 97.55, 'MNB\_classifier': 97.2, 'BernoulliNB': 97.03, 'LogisticRegression': 98.25, 'SGDClassifier': 96.85, 'SVC': 97.38, 'LinearSVC': 97.73, 'voted\_classifier': 98.25}

<AxesSubplot:>

#Generate the confusion matrix

models = [classifier,MNB\_classifier,BernoulliNB,LogisticRegression,SGDClassifier,SVC,LinearSVC,voted\_classifier]

matrixs = []

for model in models:

y\_pred = []

y\_test = []

for sample in testing\_set:

y\_pred.append(model.classify(sample[0]))

y\_test.append(sample[1])

matrixs.append(confusion\_matrix(y\_test, y\_pred))

group\_names = ['True Neg','False Pos','False Neg','True Pos']

fig, axs = plt.subplots(nrows = 3, ncols=3)

for idx,matrix in enumerate(matrixs):

group\_counts = ['{0:0.0f}'.format(value) for value in

matrix.flatten()]

group\_percentages = ['{0:.2%}'.format(value) for value in

matrix.flatten()/np.sum(matrix)]

labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in

zip(group\_names,group\_counts,group\_percentages)]

labels = np.asarray(labels).reshape(2,2)

sns.heatmap(matrix, annot=labels, fmt='', cmap='Blues', ax = axs[idx//3][idx%3])

plt.tight\_layout()

# save in the pickle

save\_classifier = open("voted\_classifier.pickle", "wb")

pickle.dump(voted\_classifier, save\_classifier)

save\_classifier.close()

classifier\_f = open("voted\_classifier.pickle", "rb")

voted\_classifier = pickle.load(classifier\_f)

classifier\_f.close()

print("Pickled voted\_classifier percent: ", (nltk.classify.accuracy(voted\_classifier, testing\_set)\*100))

# we check the voted\_classifier method, confidence, using the first dataset in training set

print("Classification: ", voted\_classifier.classify(testing\_set[1][0]), 'confidence %', voted\_classifier.confidence(testing\_set[1][0])\* 100)