Mini-Project 2: Text Classification

Jack Wei
McGill University
yi.wei4@mail.mcgill.ca

Grey Yuan
McGill University
linwei.yuan@mail.mcgill.ca

Yu Wu
McGill University
yu.wu4@mail.mcgill.ca

Abstract

The main goal of this project is to classify the comments in Reddit into five distinct categories: Telegram, LinkedIn, Instagram, viber, and Facebook. In order to classify the dataset more accurately, the text fit and transform tool in scikit-learn called vectorizer is utilized before Bernoulli Naive Bayes(BNB) model is implemented from scratch. Experimenting using different prepossessing techniques and parameters is also executed to test its impact to different models' training and validation accuracy, where the best validation accuracy reaches 90.04%. After using our proposed pipeline to select the fitted models, we continue to try out stacking using scikit-learn to increase accuracy even further, and conduct evaluation and possible improvement. After our investigation, ensemble methods using our best models (SVM, random forest, and LR) result in a validation accuracy of 90.05% (84.2% on Kaggle).

1 Introduction

Text Analysis is a widely used machine learning technique which could obtain valuable insights from unstructured text data. In this report, we performed text classification on comments from "Reddit.com" which are organized into five social media forums known as subreddits, including "Telegram", "Instagram", "LinkedIn", "Viber", and "Facebook". Provided with a training dataset of 1749 comments with labels consisting of the subreddits which they originated from, we experimented with various supervised classification models. The performance of the models is then compared as part of our model selection process.

The main task of this project is to classify reddit comments and thereby identify the subreddits which they originate from. In order to do so, the text data was first vectorized. Numerous encoding schemes are available when vectorizing bag-of-words features including one-hot, non-binary, tf-idf weighting and its variants; we discuss their relative performances in section 4 [1]. As part of the project, we implemented a Bernoulli Naive Bayes classifier from scratch. The base model supports multi-class classification by one-vs-all method. The empirically high performance of the Naive Bayes classifier in document classification motivates its role as a baseline for comparison with other models [2]. We then identify the standard classifiers for text classification based on literature and make comparisons by 10-fold cross validation. The models chosen are Multinomial Naives Bayes (MNB), K-Nearest-Neighbours (KNN), Support Vector Machine (SVM) with linear kernel, Random Forest and Logistic Regression [3], [4], [5].

The detailed plan for model optimization is articulated in section 3.1 in which a pipeline is devised for determining optimal vectorizing methods and hyper-parameters for each model. The best classifier is the stacking classifier of LR, SVM and RF models which produced the highest validation accuracy of

Preprint. Under review.

90.05%. We then compare the preprocessing approaches of stemming and lemmatizing. The result indicates no advantage of using such preproccessing measures. Kaggle competition accuracy which measures the predictability of our model for unseen data is 84.2%.

2 Dataset

The training dataset contains 1749 comments from reddit.com and their corresponding labels. As the unprocessed data are human-readable text embedded with complex information, we transformed the text into simple bags-of-word encoding. There are multiple ways of vectorizing the data such binary and non-binary encoding as well as applying TF-IDF weights, the effects of which will be discussed in 3.1 . The test set includes 698 comments without labels for evaluation of our proposed model.

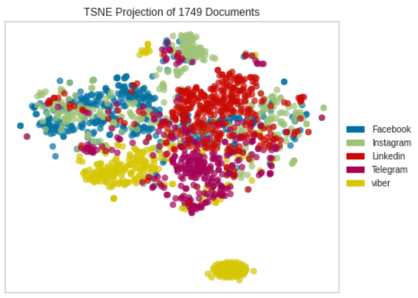


Figure 1: Clustering behavior

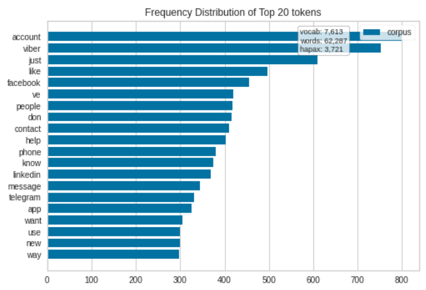


Figure 2: Frequency distribution

As the topic of discussion varies across subreddits, we believe that the encoded data exhibits clustering behaviors and is therefore separable in the feature space. The clustering behavior of the training data is verified in Figure 1 in which the high-dimensional data is visualized through t-SNE [6], [7]. It is observed that "Facebook" and "Instagram" are intertwined, which would be challenging for a classifier to separate them. Also, "Telegram" and "viber" also have some portion of overlapping, while "Telegram" has more distinct clusters. This observation is consistent with intuition as "Instagram" and "Facebook" share the same mother company Meta that may lead to similar comments being discussed; similarly, "Telegram" and "viber" are both messaging tools so they will have similar comments.

The labels "Telegram", "LinkedIn", "Instagram", "viber", and "Facebook" are encoded with 0, 1, 2, 3, and 4 respectively. We have discovered that the total number of words in the training data after vectorizing is 8352. After the reducing English stop words, applying stemming (excluding stemmed stop words), and lemmatizing (excluding lemmatized stop words), the total word count is reduced to 8073, 4716, and 6355 respectively.

The effects of urls, numbers, punctuation, mis-spelled words, and duplicate comments on the models' validation accuracy are also tested. According to our experiments, we found that they contribute little to overall model performance. The frequency distribution of the most common words shows that common English stop words take most of the Top 20 words. Hence we drop English stop words to minimize its effect. In Figure 2, the frequency distribution after stemming stop words is shown.

3 Proposed Approach

3.1 Feature and Model Optimization Pipeline

The proposed approach to determine the optimal feature and model pairs is a pipeline. The first step is to select the best features that can produce the highest validation accuracy for each model listed in 3.2. We categorize the preprocessing methods to 3 test groups: TG1 - vectorization (TF-IDF binary and TF-IDF non-binary); TG2 - stop words and tokenizer (stemming, lemmatization, and English stop words); TG3 - max feature and n-gram (max features from 3000 to 8000 and n-gram from combinations of uni-gram, 2-gram, and 3-gram).

According to prior research and experimentation, TF-IDF transformation, which tends to assign higher weights to more discriminating words, is generally preferred in text classification [8], [9].

Therefore TF-IDF transformation is used by default for later test of TG3. The TF-IDF vectorizer also supports binary and non-binary TF (term frequency) transformation. As we suspect that individual models react differently to binary and non-binary TF, the optimized vectorizer will be determined later with grid search methods. Another assumption we made is that the best model performing on the non-stemmed, non-lemmatized features would out-perform other models on processed features regardless. Therefore, we decided to compare TG2 after the best model is chosen.

Third, with the assumption made, TG3 is investigated by finding the best max feature and n-gram combination that give the highest 10-fold cross validation accuracy. Next, we employ grid search to determine the optimal hyperparameters and simultaneously comparing the vectorizers listed in TG1 for each model. Finally, the best model is selected by the highest cross validation accuracy with the chosen variables from TG1, TG2, and TG3 respectively.

3.2 Models

Bernoulli Naive Bayes: Bernoulli Naive Bayes (BNB) is a generative learning binary classifier that assumes features are conditionally independent and models class distribution based on such assumption. In the model that we implement, we use constant Laplace smoothing such that non-occurring dictionary words in a document are assigned a prior probability of 0.5. We then use One-vs-Rest method to make predictions based on the highest-class probability out of five BNB models for multi-class classifications.

Multinomial Naive Bayes: In contrast to BNB that can only model feature as occurring or non-occurring binary encoding, multinomial Naive Bayes (MNB) is capable of distinguishing the frequency of a word's occurrence and perform multi-class classification. As such, the comparison of MNB and BNB would provide with us significant insights into the structure of the dataset. The hyperparameter we choose to tune is the Laplace smoothing parameter.

K-nearest Neighbors: K-nearest Neighbors (KNN) is a non-linear, instance-based learning classification algorithm. The classification of a data point is determined by the majority voting of the labels of its k nearest neighbours. This would in theory, model the clusters of classes in the feature space. The number of nearest neighbors k could be adjusted and higher k would prevent over-fitting.

Linear Support Vector Machine: Linear support vector machine (SVM) is a SVM with linear kernel that constructs an optimal hyperplane in the feature space for binary classification. Mutli-class classification is supported by One-vs-Rest method similar to our BNB model. Since SVM with other kernels such as RBF have longer run-time with no improvement in performance compared to that of linear, we picked linear SVM as a representative model. The hyperparameter C which inversely adjusts the L2-regularization strength is chosen as the hyper parameter.

Logistic Regression: Logistic regression (LR) is a discriminative method that models the log-odds ratio of classes. The hyperparameter tuned is the aforementioned regularization parameter C.

Random Forest: Random Forest (RF) is a model that applies bagging to decision trees. We prefer RF over its base learner decision tree. As decision tree is prone to over-fitting. The hyperparameter tuned is the max depth of each decision tree.

Ensemble Stacking: Stacking is an ensemble method which combines the predictions from multiple existing models and creates a meta-model. We used our best performing models: LR, SVM, RF as base models. Logistic regression is used as encompassing meta model. This can, in theory, improve relative performance as models complement each other.

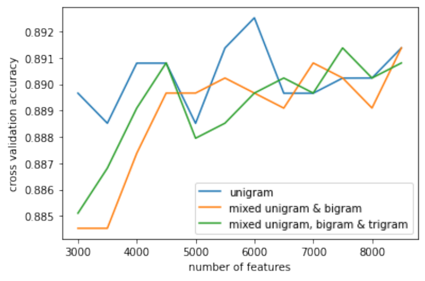
4 Result

4.1 Results of TG3

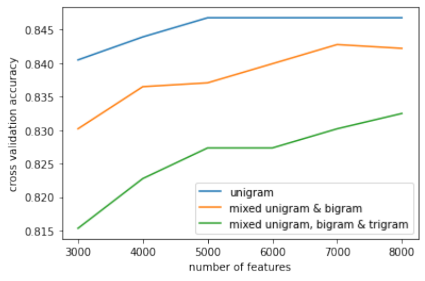
When analyzing the training dataset, it is noticeable that the total number of features increases as we widen the range of n-gram selection. Intuitively, we would think that the model's performance would improve as we increase max feature for each category of n-gram. Interestingly, the experimental result shows that is not always the case. As shown in Figure 3, MNB conforms to our expectation while SVM does not. SVM exhibits high sensitivity to different max feature and n-gram combinations.

model name	BNB	Linear SVM	MNB	KNN	RBF Kernel SVM	LR	Random Forest
max feature n-gram	3000 (1,1)	6000 (1,1)	≥5000 (1,1)	4000 (1,2)	5000 (1,2)	4000 (1,2)	5000 (1,1)

Table 1: Best max feature and n-gram combination for all models



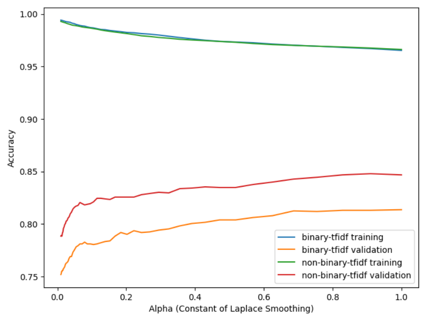
(a) SVM



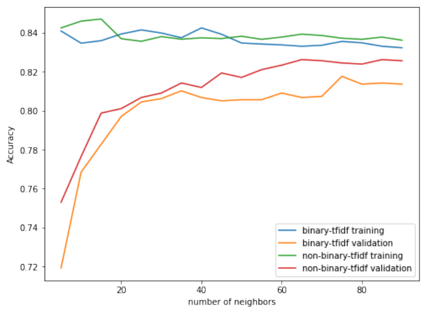
(b) MNB

Figure 3: 10-fold cross validation accuracy vs. max feature for different n-gram combinations

4.2 Model Selection

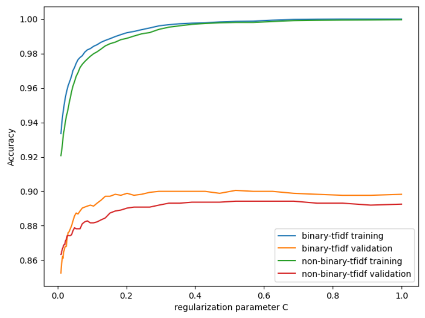


(a) MNB

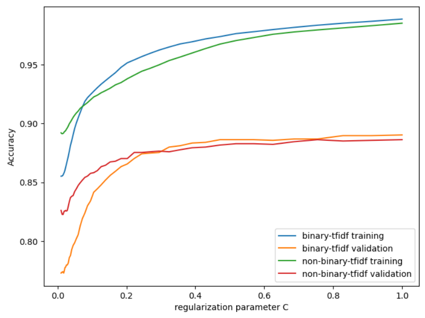


(b) KNN

Figure 4: Grid search on hyper-parameter for binary and non-binary TF-IDF vectorizer

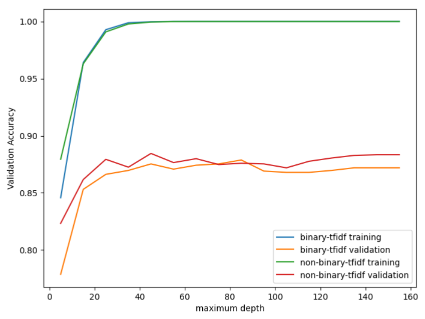


(a) SVM

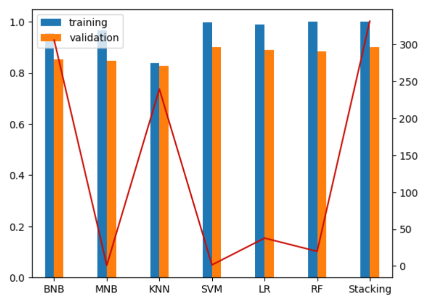


(b) LR

Figure 5: Grid search on hyper-parameter for binary and non-binary TF-IDF vectorizer



(a) Grid search on max depth of RF



(b) best validation accuracy of all models and corresponding training accuracy and run time

†the left y-axis is the model's training or validation accuracy and the right y-axis is the run time in seconds

Figure 6: Recursive Feature Elimination Validation Accuracy and Run Time Results†

In accordance with our pipeline, we performed grid search on the hyperparameters mentioned in Section 3.2 as well as binary and non-binary TF-IDF transformation. The results are shown as in Figure 4 and 5. All the plots indicate that increasing the influence of L2-regularization or reducing model complexity by other means generally results in higher variance and bias. Using the hyperparameters and transformation that maximizes validation score, we achieved over 80% validation accuracy on all our models as indicated in Figure 5(b). The BNB classifier we implemented achieves a validation accuracy of 85.36% which passes that of off-shelf MNV and KNN. Our best model is the stacking classifier which produced a validation accuracy of 90.05% and gives a test accuracy of 84.2% on the Kaggle leaderboard. Both lemmatizing and stemming features result in an accuracy of 88.28% on our best model which is not an improvement over the original feature.

5 Discussion and Conclusion

5.1 Major Findings

We conclude that other than KNN, the based models we picked are all suited for text classification task reliably and efficiently (High running time of BNB is due to unoptimized code). A key observation is as we increase the variety of n-gram selection, the increase of max feature exhibits a more stable increasing trend on the validation accuracy. This can be easily explained by the fact that mixed n-grams with larger potential feature set require higher upper-limit to obtain critically discriminating features. We can also conclude from table 1 that unigram is generally enough for adequate classifications which suggest that temporal dependency between words is not significant for reddit comments. Surprisingly, linear SVM model combined with TF-IDF vectorizer gives us the highest results on Kaggle (in 87.07%).

5.2 Future Investigation

According to Figure 2, one observation is that there still exists some trivial words that are not removed as a stop word. Words like "just", "like", "ve" are not discriminative for classification. Therefore, it would be beneficial if we could increase the dictionary of stop words to include these words. Lastly, we suspect that the disparity between test accuracy and validation accuracy is due to our model over-fitting the given dataset as bias is virtually zero. We will investigate ways to improve our model's predictability on unseen data.

6 Statement of Contribution

Jack Wei: Implementation of algorithm, optimize model, feature preprocessing, report
Grey Yuan: draft programming, feature preprocessing, text analysis, graphs, tables, report
Yu Wu: draft programming, research finding, tables, report and report organization
This project is completed as a team of 3, and each member worked actively and made great contributions to the project.

References

[1] Jianqiang, Z. and Xiaolin G.(2017), Comparison Research on Text Pre-processing Methods on Twitter Sentiment Analysis, in IEEE Access, vol. 5, pp. 2870-2879, doi: 10.1109/ACCESS.2017.2672677.

[2] Pratama, B. Y. and Sarno, R.(2015), Personality classification based on Twitter text using Naive Bayes, KNN and SVM, International Conference on Data and Software Engineering (ICoDSE), pp. 170-174, doi: 10.1109/ICODSE.2015.7436992.

[3] Kanaan, G., Al-Shalabi, R., Ghwanmeh, S. and Al-Ma'adeed, H. (2009), A comparison of text-classification techniques applied to Arabic text. J. Am. Soc. Inf. Sci., 60: 1836-1844. https://doi.org/10.1002/asi.20832

[4] Ikonomakis, E., Kotsiantis, S. and Tampakas, V.. (2005), Text Classification Using Machine Learning Techniques. WSEAS transactions on computers. 4. 966-974.

[5] Neethu, M. S. and Rajasree, R.(2013), Sentiment analysis in twitter using machine learning techniques, Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT), pp. 1-5, doi: 10.1109/ICCCNT.2013.6726818.

[6] Maaten, L. and Hinton, G.(2008), Visualizing Data using t-SNE, Journal of Machine Learning Research, vol. 9, 86, pp. 2579-2605, http://jmlr.org/papers/v9/vandermaaten08a.html

[7] Belkina, A.C., Ciccolella, C.O., Anno, R. et al.(2019), Automated optimized parameters for T-distributed stochastic neighbor embedding improve visualization and analysis of large datasets. Nat Commun 10, 5415, https://doi.org/10.1038/s41467-019-13055-y

[8] Ramos, Juan. (2003). Using TF-IDF to determine word relevance in document queries.

[9] Aizawa, A.(2013), An information-theoretic perspective of tf–idf measures, Information Processing Management, Volume 39, Issue 1, Pages 45-65, ISSN 0306-4573, https://doi.org/10.1016/S0306-4573(02)00021-3.

7 Appendix

```
# -*- coding: utf-8 -*-
"""ECSE551_Assignment2.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/11yyi5W3G-i6pLYBkkbppVVXPyy6aViWQ

# ECSE 551 Assignment 2
"""

from google.colab import drive

import re
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from time import time
from google.colab import files

# nltk libraries
import nltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
from nltk import word_tokenize
from nltk.corpus import wordnet
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer

# sklearn libraries
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import Normalizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer     #TF-IDF
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction import text
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
```

```
# models
from sklearn import tree
from sklearn.svm import SVC
from sklearn.svm import LinearSVC #SVM
from sklearn.naive_bayes import MultinomialNB #NB
from sklearn.linear_model import LogisticRegression #LR
from sklearn.neighbors import KNeighborsClassifier #KNN
from sklearn.neural_network import MLPClassifier #multi-layer perceptron
from sklearn.ensemble import RandomForestClassifier

# emsemble
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import StackingClassifier


# TSNE visualizer
from yellowbrick.text import TSNEVisualizer
from yellowbrick.text.freqdist import FreqDistVisualizer

drive.mount('/content/gdrive')

path = "./gdrive/MyDrive/ECSE_551_Assignment_2/Data/"

"""## Loading Data"""

df_train = pd.read_csv(path + 'train.csv', encoding = 'ISO-8859-1')
df_train = df_train.sample(frac=1, random_state=0).reset_index(drop=True) #shuffle df_train

df_test = pd.read_csv(path + 'test.csv', encoding = 'ISO-8859-1')

df_train.head()

df_test.head()

"""## Helper Methods for Output"""

def decode_output(pred):
  label_name_list = ['Telegram','Linkedin', 'Instagram', 'viber','Facebook']
  output = []
  for index in pred:
    output.append(label_name_list[index])
  return np.array(output)

def download_csv(pred):
  output = decode_output(pred)
  output_df = pd.DataFrame(output, columns = ['subreddit'])
  output_df.insert(0, 'id', output_df.index + 1)
  output_df.to_csv("submission.csv", index=None)
  files.download('submission.csv')

"""## Model

### Bernoulli Naive Bayes model
"""

class Bernoulli_Naive_Bayes_Clf():

    def __init__(self):
        self.prior_prob = 0
        self.post_prob_x_pos_y_neg = []
        self.post_prob_x_neg_y_neg = []
        self.post_prob_x_pos_y_pos = []
        self.post_prob_x_neg_y_pos = []

    def fit(self, X_train, Y_train):

        m = X_train.shape[1]
        num_y_pos = np.count_nonzero(Y_train)
        num_y_neg = Y_train.shape[0] - num_y_pos

        self.prior_prob = num_y_pos/m   # prior probability for y

        # get posterior probabilities
        for i in range(m):
          count_x_pos_y_pos = 0
          count_x_pos_y_neg = 0
          for index, vect in enumerate(X_train):
            if(vect[i] != 0):
              if(Y_train[index] == 0):
                count_x_pos_y_neg = count_x_pos_y_neg + 1
              else:
                count_x_pos_y_pos = count_x_pos_y_pos + 1
          prob_y_neg = (count_x_pos_y_neg+1) / (num_y_neg+2)
          prob_y_pos = (count_x_pos_y_pos+1) / (num_y_pos+2)
          self.post_prob_x_pos_y_neg.append(prob_y_neg)
          self.post_prob_x_neg_y_neg.append(1 - prob_y_neg)
          self.post_prob_x_pos_y_pos.append(prob_y_pos)
          self.post_prob_x_neg_y_pos.append(1 - prob_y_pos)

    def __calc_likelihood(self, x_T):
          p_y_neg = 1 - self.prior_prob
        p_y_pos = self.prior_prob
        for index,bin in enumerate(x_T):
          if(bin == 0):
            p_y_pos = p_y_pos * self.post_prob_x_neg_y_pos[index]
            p_y_neg = p_y_neg * self.post_prob_x_neg_y_neg[index]
          else:
            p_y_pos = p_y_pos * self.post_prob_x_pos_y_pos[index]
            p_y_neg = p_y_neg * self.post_prob_x_pos_y_neg[index]
        return p_y_pos, p_y_neg

    def pred_binary(self, X_test):
        predicted = []

        for x in X_test:
          likelihood_pos, likelihood_neg = self.__calc_likelihood(x.T)

          if(np.log(likelihood_pos/likelihood_neg) > 0): # log-odds ratio
            predicted.append(1)
          else:
            predicted.append(0)
        return predicted

    def get_class_prob(self, X_test):
        scores = []

        for x in X_test:
          likelihood_pos, likelihood_neg = self.__calc_likelihood(x.T)
          # higher log odds ratio, higher class probability
          scores.append(np.log(likelihood_pos) - np.log(likelihood_neg))
        return scores

class BNB_OVR_classifier():

  def __init__(self):
    self.model = Bernoulli_Naive_Bayes_Clf()

  def pred(self, X_train, y_train, X_test, get_X_train_pred = False):
    # Create one hot encodings of each class
    label_name_list = ['Telegram','Linkedin', 'Instagram', 'viber','Facebook']
    encoded_y_train_list = []
    for label_name in label_name_list:
      encoded_y_train_list.append([(int)(y == label_name) for y in y_train])

    # Get score lists for each class
    score_list_test = []
    if(get_X_train_pred):
      score_list_train = []

    for encoded_y_train in encoded_y_train_list:
      self.model = Bernoulli_Naive_Bayes_Clf()
      self.model.fit(X_train, np.array(encoded_y_train))

      score_list_test.append(self.model.get_class_prob(X_test))

      if(get_X_train_pred):
        score_list_train.append(self.model.get_class_prob(X_train))

    # Return final predictions
    index_array_test = np.argmax(np.array(score_list_test), axis=0)
    if(get_X_train_pred):
      index_array_train = np.argmax(np.array(score_list_train), axis=0)
      return index_array_test, index_array_train

    return index_array_test

"""## Preprocessing"""

np_train = df_train.to_numpy()
np_test = df_test.to_numpy()

print('train␣data␣dimensions:␣' + str(np_train.shape))
print('test␣data␣dimensions:␣' + str(np_test.shape))

X_train = np_train[:,0].tolist() #x
y_train = np_train[:,1] #y
X_test = np_test[:,1].tolist()

"""### Stem and Lemma Tokenizers"""

class StemTokenizer:
     def __init__(self):
       self.wn =PorterStemmer()
     def __call__(self, doc):
       return [self.wn.stem(t) for t in word_tokenize(doc) if t.isalpha()]

class LemmaTokenizer:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos ="n") for t in word_tokenize(doc) if t.isalpha()]

"""###Construct stop words"""
```

```
def get_wordnet_pos(word):
    """Map POS tag to first character lemmatize() accepts"""
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)

stemmed_stop_words=set()
lemmatized_stop_words=set()
english_stop_words=text.ENGLISH_STOP_WORDS
for word in english_stop_words:
    stemmed_stop_words.add(PorterStemmer().stem(word))
    lemmatized_stop_words.add(WordNetLemmatizer().lemmatize(word, pos = get_wordnet_pos(word)))

"""### Vectorizer

####TF-IDF vectorizer
"""

vectorizer = TfidfVectorizer(max_features=3000,lowercase=True, ngram_range=(1,1), stop_words=text.ENGLISH_STOP_WORDS)

"""#### binary vectorizer"""

vectorizer = CountVectorizer(lowercase=True, ngram_range=(1,1), max_features=10000, stop_words=text.ENGLISH_STOP_WORDS)

"""### Testing Vectorize"""

vec_X_train = vectorizer.fit_transform(X_train)
vec_X_test = vectorizer.transform(X_test)

print(vectorizer.get_feature_names())
print(len(vectorizer.get_feature_names()))

#optional normalize (Note TFIDF vectorizer already normalizes the data)
vec_X_train = Normalizer().transform(vec_X_train)
vec_X_test=Normalizer().transform(vec_X_test)

vec_X_train = vec_X_train.todense()
vec_X_test = vec_X_test.todense()

vec_X_train = np.asarray(vec_X_train)
vec_X_test = np.asarray(vec_X_test)

print("dimensions␣of␣vectorized␣X␣train:␣" + str(vec_X_train.shape))
print("dimensions␣of␣vectorized␣X␣test:␣" + str(vec_X_test.shape))

"""### Process Labels"""

label_name_list = ['Telegram','Linkedin', 'Instagram', 'viber','Facebook']
encoded_y_train = np.array([label_name_list.index(label) for label in y_train])

"""### Visualizing Dataset"""

vectorizer = CountVectorizer(max_features=3000)
vec_X_train = vectorizer.fit_transform(X_train)
features = vectorizer.get_feature_names()

# L2 Squared Euclidean Distance
tsnse = TSNEVisualizer(size = (600, 400))
tsnse.fit(vec_X_train, y_train)
tsnse.poof()

vectorizer = CountVectorizer(max_features=9000)
vec_X_train = vectorizer.fit_transform(X_train)
features = vectorizer.get_feature_names()

visualizer = FreqDistVisualizer(features=features, n=20)
visualizer.fit(vec_X_train)
visualizer.poof()

vectorizer = CountVectorizer(max_features=9000, stop_words=text.ENGLISH_STOP_WORDS)
vec_X_train = vectorizer.fit_transform(X_train)
features = vectorizer.get_feature_names()

visualizer = FreqDistVisualizer(features=features, n=20)
visualizer.fit(vec_X_train)
visualizer.poof()

"""## Evaluation"""

def Accu_eval(ground_truth, predicted):
  return (ground_truth == predicted).mean()

"""### Cross Validation"""

def my_cross_validate(model, k = 10):

  kf=KFold(n_splits=k,shuffle=False) #k-fold validation
  accuracy_val = []
  accuracy_train = []

  index=kf.split(np_train)

  for train_index, test_index in index:
    score_list_val = []
    score_list_train = []

    X_train_CV, X_test_CV, y_train_CV = vec_X_train[train_index], vec_X_train[test_index], y_train[train_index]

    encoded_y_test_CV = encoded_y_train[test_index]
    encoded_y_train_CV = encoded_y_train[train_index]

    pred_val, pred_train = model.pred(X_train_CV, y_train_CV, X_test_CV, True)

    acc_val = Accu_eval(pred_val, encoded_y_test_CV)
    acc_train = Accu_eval(pred_train, encoded_y_train_CV)

    accuracy_val.append(acc_val)
    accuracy_train.append(acc_train)

  CV_val_accuracy = np.array(accuracy_val).mean()
  CV_train_accuracy = np.array(accuracy_train).mean()

  print("cross␣validated␣training␣accuracy:␣" + str(CV_train_accuracy))
  print("cross␣validated␣validation␣accuracy:␣" + str(CV_val_accuracy))
  return CV_train_accuracy, CV_val_accuracy

"""### Experimentations on Ngram and Feature number"""

ngram_range_list = [(1,1),(1,2),(1,3)]
feature_number_list = list(range(3000, 9000, 1000))

cv_ngram = []
for ngram_range in ngram_range_list:
  cv_acc = []
  for feature_number in feature_number_list:
    vectorizer = TfidfVectorizer(max_features=feature_number, stop_words = text.ENGLISH_STOP_WORDS, lowercase=True, ngram_range=ngram_range)
    vec_X_train = np.asarray(vectorizer.fit_transform(X_train).todense())
    model = BNB_OVR_classifier()
    CV_train_accuracy, CV_val_accuracy = my_cross_validate(model)
    cv_acc.append(CV_val_accuracy)
  cv_ngram.append(cv_acc.copy())

for ngram_acc in cv_ngram:
  plt.plot(feature_number_list, ngram_acc)
plt.legend(["unigram", "mixed␣unigram␣&␣bigram", "mixed␣unigram,␣bigram␣&␣trigram"])
plt.xlabel('number␣of␣features')
plt.ylabel('cross␣validation␣accuracy')
plt.show()

def plot_ngram_num_features(model):
  ngram_range_list = [(1,1),(1,2),(1,3)]
  feature_number_list = list(range(3000, 9000, 1000))

  cv_ngram = []
  for ngram_range in ngram_range_list:
    cv_acc = []
    for feature_number in feature_number_list:
      vectorizer = TfidfVectorizer(max_features=feature_number, stop_words = text.ENGLISH_STOP_WORDS, lowercase=True, ngram_range=ngram_range)
      vec_X_train = np.asarray(vectorizer.fit_transform(X_train).todense())
      cv_results = cross_validate(model, vec_X_train, encoded_y_train, cv=10)
      cv_acc.append(cv_results['test_score'].mean())
    cv_ngram.append(cv_acc.copy())

  for ngram_acc in cv_ngram:
    plt.plot(feature_number_list, ngram_acc)
  plt.legend(["unigram", "mixed␣unigram␣&␣bigram", "mixed␣unigram,␣bigram␣&␣trigram"])
  plt.xlabel('number␣of␣features')
  plt.ylabel('cross␣validation␣accuracy')
  plt.show()

"""### BNB Best Model"""

vectorizer = TfidfVectorizer(max_features=3000, stop_words = text.ENGLISH_STOP_WORDS, lowercase=True, ngram_range=(1,1), binary=True)
vec_X_train = np.asarray(vectorizer.fit_transform(X_train).todense())

clf = BNB_OVR_classifier()
start = time()
CV_train_accuracy, CV_val_accuracy = my_cross_validate(clf)
end = time()

print("cross␣validated␣training␣accuracy:␣" + str(CV_train_accuracy))
print("cross␣validated␣validation␣accuracy:␣" + str(CV_val_accuracy))
print("cv␣run␣time:" + str(end - start))

"""## Investigations"""

train_acc_results = [0.9554668118835844]
test_acc_results = [0.8536453201970442]
run_time = [305.90363335609436]
```

```
"""### Multinomial Naive Bayes

#### MNB Feature Size vs Ngram
"""

plot_ngram_num_features(MultinomialNB())

"""#### MNB Hyperparameter Tuning"""

bin_vectorizer = TfidfVectorizer(max_features=5000, stop_words = text.ENGLISH_STOP_WORDS,lowercase=True, ngram_range=(1,1), binary=True)
non_bin_vectorizer = TfidfVectorizer(max_features=5000, stop_words = text.ENGLISH_STOP_WORDS,lowercase=True, ngram_range=(1,1), binary=False)

bin_vec_X_train = np.asarray(bin_vectorizer.fit_transform(X_train).todense())
non_bin_vec_X_train = np.asarray(non_bin_vectorizer.fit_transform(X_train).todense())

plt.style.use('default')
plt.figure(figsize=(8,6))

alpha_list = np.logspace(-2,0)
tuned_parameters = [{'alpha':alpha_list}]

clf = MultinomialNB()

MNB_grid_binary = GridSearchCV(clf,tuned_parameters,cv = 10, return_train_score=True)
MNB_grid_binary.fit(bin_vec_X_train, encoded_y_train)
MNB_grid_non_binary = GridSearchCV(clf,tuned_parameters,cv = 10, return_train_score=True)
MNB_grid_non_binary.fit(non_bin_vec_X_train, encoded_y_train)

MNB_grid_train_scores_binary = MNB_grid_binary.cv_results_['mean_train_score']
MNB_grid_train_scores_non_binary = MNB_grid_non_binary.cv_results_['mean_train_score']
MNB_grid_test_scores_binary = MNB_grid_binary.cv_results_['mean_test_score']
MNB_grid_test_scores_non_binary = MNB_grid_non_binary.cv_results_['mean_test_score']

plt1 = plt.plot(alpha_list,MNB_grid_train_scores_binary)
plt2 = plt.plot(alpha_list,MNB_grid_test_scores_binary)

plt3 = plt.plot(alpha_list,MNB_grid_train_scores_non_binary)
plt4 = plt.plot(alpha_list,MNB_grid_test_scores_non_binary)

plt.xlabel('Alpha␣(Constant␣of␣Laplace␣Smoothing)')
plt.ylabel('Accuracy')
plt.legend(['binary-tfidf␣training','binary-tfidf␣validation','non-binary-tfidf␣training', 'non-binary-tfidf␣validation'])

print("Binary␣vectorizer:")
print('max␣alpha:' + str(alpha_list[np.argmax(MNB_grid_test_scores_binary)]) + "␣validation␣accuracy:" + str(MNB_grid_test_scores_binary.max
print("Non-Binary␣vectorizer")
print('max␣alpha:' + str(alpha_list[np.argmax(MNB_grid_test_scores_non_binary)]) + "␣validation␣accuracy:" + str(MNB_grid_test_scores_non_bi

"""#### MNB Best Model"""

vectorizer = TfidfVectorizer(max_features=5000, stop_words = text.ENGLISH_STOP_WORDS,lowercase=True, ngram_range=(1,1), binary=False)
vec_X_train = np.asarray(vectorizer.fit_transform(X_train).todense())

clf = MultinomialNB(alpha=0.9102981779915218)
start = time()
cv_results = cross_validate(clf, vec_X_train, encoded_y_train, cv=10, return_train_score=True)
end = time()

print("cross␣validated␣training␣accuracy:␣" + str(cv_results['train_score'].mean()))
print("cross␣validated␣validation␣accuracy:␣" + str(cv_results['test_score'].mean()))

train_acc_results.append(cv_results['train_score'].mean())
test_acc_results.append(cv_results['test_score'].mean())
run_time.append(end-start)

"""### KNN

#### KNN Feature Size vs Ngram
"""

plot_ngram_num_features(KNeighborsClassifier())

"""#### KNN Hyperparameter Tuning"""

bin_vectorizer = TfidfVectorizer(max_features=4000, stop_words = text.ENGLISH_STOP_WORDS,lowercase=True, ngram_range=(1,2), binary=True)
non_bin_vectorizer = TfidfVectorizer(max_features=4000, stop_words = text.ENGLISH_STOP_WORDS,lowercase=True, ngram_range=(1,2), binary=False)

bin_vec_X_train = np.asarray(bin_vectorizer.fit_transform(X_train).todense())
non_bin_vec_X_train = np.asarray(non_bin_vectorizer.fit_transform(X_train).todense())

plt.figure(figsize=(8,6))

k_list = range(5,115,5)
tuned_parameters = [{'n_neighbors':k_list}]

clf = KNeighborsClassifier()

KNN_grid_binary = GridSearchCV(clf,tuned_parameters,cv = 10, return_train_score=True)
KNN_grid_binary.fit(bin_vec_X_train, encoded_y_train)
KNN_grid_non_binary = GridSearchCV(clf,tuned_parameters,cv = 10, return_train_score=True)
KNN_grid_non_binary.fit(non_bin_vec_X_train, encoded_y_train)
```

```
KNN_grid_test_scores_binary = KNN_grid_binary.cv_results_['mean_test_score']
KNN_grid_test_scores_non_binary = KNN_grid_non_binary.cv_results_['mean_test_score']
KNN_grid_train_scores_binary = KNN_grid_binary.cv_results_['mean_train_score']
KNN_grid_train_scores_non_binary = KNN_grid_non_binary.cv_results_['mean_train_score']

plt1 = plt.plot(k_list,KNN_grid_train_scores_binary)
plt2 = plt.plot(k_list,KNN_grid_test_scores_binary)
plt3 = plt.plot(k_list,KNN_grid_train_scores_non_binary)
plt4 = plt.plot(k_list,KNN_grid_test_scores_non_binary)

plt.xlabel('number␣of␣neighbors')
plt.ylabel('Accuracy')
plt.legend(['binary-tfidf␣training','binary-tfidf␣validation','non-binary-tfidf␣training', 'non-binary-tfidf␣validation'])

print("Binary␣vectorizer:")
print('max␣k:' + str(k_list[np.argmax(KNN_grid_test_scores_binary)]) + "␣validation␣accuracy:" + str(KNN_grid_test_scores_binary.max()))
print("Non-Binary␣vectorizer")
print('max␣k:' + str(k_list[np.argmax(KNN_grid_test_scores_non_binary)]) + "␣validation␣accuracy:" + str(KNN_grid_test_scores_non_binary.max

"""#### KNN Best Model"""

vectorizer = TfidfVectorizer(max_features=4000, stop_words = text.ENGLISH_STOP_WORDS,lowercase=True, ngram_range=(1,2), binary=False)
vec_X_train = np.asarray(vectorizer.fit_transform(X_train).todense())

clf = KNeighborsClassifier(n_neighbors=65)
start = time()
cv_results = cross_validate(clf, vec_X_train, encoded_y_train, cv=10, return_train_score=True)
end = time()

print("cross␣validated␣training␣accuracy:␣" + str(cv_results['train_score'].mean()))
print("cross␣validated␣validation␣accuracy:␣" + str(cv_results['test_score'].mean()))

train_acc_results.append(cv_results['train_score'].mean())
test_acc_results.append(cv_results['test_score'].mean())
run_time.append(end-start)

"""### Linear SVM

#### Linear SVM Feature Size vs Ngram
"""

plot_ngram_num_features(LinearSVC(random_state = 0))

for ngram_acc in cv_ngram:
  plt.plot(feature_number_list, ngram_acc)
plt.legend(["unigram", "mixed␣unigram␣&␣bigram", "mixed␣unigram,␣bigram␣&␣trigram"])
plt.xlabel('number␣of␣features')
plt.ylabel('cross␣validation␣accuracy')
plt.show()

"""#### Linear SVM Hyperparameter Tuning"""

bin_vectorizer = TfidfVectorizer(max_features=6000, stop_words = text.ENGLISH_STOP_WORDS,lowercase=True, ngram_range=(1,1), binary=True)
non_bin_vectorizer = TfidfVectorizer(max_features=6000, stop_words = text.ENGLISH_STOP_WORDS,lowercase=True, ngram_range=(1,1), binary=False)

bin_vec_X_train = np.asarray(bin_vectorizer.fit_transform(X_train).todense())
non_bin_vec_X_train = np.asarray(non_bin_vectorizer.fit_transform(X_train).todense())

plt.style.use('default')
plt.figure(figsize=(8,6))

C_list = np.logspace(-2,0)
tuned_parameters = [{'C':C_list}]

clf = LinearSVC(random_state = 0)

SVM_grid_binary = GridSearchCV(clf,tuned_parameters,cv = 10, return_train_score=True)
SVM_grid_binary.fit(bin_vec_X_train, encoded_y_train)
SVM_grid_non_binary = GridSearchCV(clf,tuned_parameters,cv = 10, return_train_score=True)
SVM_grid_non_binary.fit(non_bin_vec_X_train, encoded_y_train)

SVM_grid_test_scores_binary = SVM_grid_binary.cv_results_['mean_test_score']
SVM_grid_test_scores_non_binary = SVM_grid_non_binary.cv_results_['mean_test_score']
SVM_grid_train_scores_binary = SVM_grid_binary.cv_results_['mean_train_score']
SVM_grid_train_scores_non_binary = SVM_grid_non_binary.cv_results_['mean_train_score']

plt1 = plt.plot(C_list,SVM_grid_train_scores_binary)
plt2 = plt.plot(C_list,SVM_grid_test_scores_binary)

plt3 = plt.plot(C_list,SVM_grid_train_scores_non_binary)
plt4 = plt.plot(C_list,SVM_grid_test_scores_non_binary)

plt.xlabel('regularization␣parameter␣C')
plt.ylabel('Accuracy')
plt.legend(['binary-tfidf␣training','binary-tfidf␣validation','non-binary-tfidf␣training', 'non-binary-tfidf␣validation'])

print("Binary␣vectorizer:")
print('max␣C:' + str(C_list[np.argmax(SVM_grid_test_scores_binary)]) + "␣validation␣accuracy:" + str(SVM_grid_test_scores_binary.max()))
print("Non-Binary␣vectorizer")
print('max␣C:' + str(C_list[np.argmax(SVM_grid_test_scores_non_binary)]) + "␣validation␣accuracy:" + str(SVM_grid_test_scores_non_binary.max
"""#### Linear SVM Best Model"""

vectorizer = TfidfVectorizer(max_features=6000, stop_words = text.ENGLISH_STOP_WORDS, lowercase=True, ngram_range=(1,1), binary=True)
vec_X_train = np.asarray(vectorizer.fit_transform(X_train).todense())

clf = LinearSVC(random_state = 0, C = 0.517947467923121)
start = time()
cv_results = cross_validate(clf, vec_X_train, encoded_y_train, cv=10, return_train_score=True)
end = time()

print("cross␣validated␣training␣accuracy:␣" + str(cv_results['train_score'].mean()))
print("cross␣validated␣validation␣accuracy:␣" + str(cv_results['test_score'].mean()))

train_acc_results.append(cv_results['train_score'].mean())
test_acc_results.append(cv_results['test_score'].mean())
run_time.append(end-start)

"""### RBF Kernel SVM

#### RBF-SVM Feature Size vs Ngram
"""

plot_ngram_num_features(SVC(random_state = 0))

"""#### RBF-SVM Best Model"""

vectorizer = TfidfVectorizer(max_features=5000, stop_words = text.ENGLISH_STOP_WORDS, lowercase=True, ngram_range=(1,2), binary=True)
vec_X_train = np.asarray(vectorizer.fit_transform(X_train).todense())

clf = SVC(random_state = 0)
cv_results = cross_validate(clf, vec_X_train, encoded_y_train, cv=10, return_train_score=True)

print("cross␣validated␣training␣accuracy:␣" + str(cv_results['train_score'].mean()))
print("cross␣validated␣validation␣accuracy:␣" + str(cv_results['test_score'].mean()))

clf = SVC(random_state = 0) #0.003727593720314938
cv_results = cross_validate(clf, vec_X_train, encoded_y_train, cv=10, return_train_score=True)
print("cross␣validated␣training␣accuracy:␣" + str(cv_results['train_score'].mean()))
print("cross␣validated␣validation␣accuracy:␣" + str(cv_results['test_score'].mean()))

"""### Logistic Regression

#### LR Feature Size vs Ngram
"""

plot_ngram_num_features(LogisticRegression(random_state = 0))

"""#### LR Hyperparameter Tuning"""

bin_vectorizer = TfidfVectorizer(max_features=4000, stop_words = text.ENGLISH_STOP_WORDS, lowercase=True, ngram_range=(1,2), binary=True)
non_bin_vectorizer = TfidfVectorizer(max_features=4000, stop_words = text.ENGLISH_STOP_WORDS, lowercase=True, ngram_range=(1,2), binary=False

bin_vec_X_train = np.asarray(bin_vectorizer.fit_transform(X_train).todense())
non_bin_vec_X_train = np.asarray(non_bin_vectorizer.fit_transform(X_train).todense())

plt.style.use('default')
plt.figure(figsize=(8,6))

C_list = np.logspace(-2,0)
tuned_parameters = [{'C':C_list}]

clf = LogisticRegression(random_state = 0)

LR_grid_binary = GridSearchCV(clf,tuned_parameters,cv = 10, return_train_score = True)
LR_grid_binary.fit(bin_vec_X_train, encoded_y_train)
LR_grid_non_binary = GridSearchCV(clf,tuned_parameters,cv = 10, return_train_score = True)
LR_grid_non_binary.fit(non_bin_vec_X_train, encoded_y_train)

LR_grid_train_scores_binary = LR_grid_binary.cv_results_['mean_train_score']
LR_grid_train_scores_non_binary = LR_grid_non_binary.cv_results_['mean_train_score']

LR_grid_test_scores_binary = LR_grid_binary.cv_results_['mean_test_score']
LR_grid_test_scores_non_binary = LR_grid_non_binary.cv_results_['mean_test_score']

plt1 = plt.plot(C_list,LR_grid_train_scores_binary)
plt2 = plt.plot(C_list,LR_grid_test_scores_binary)
plt3 = plt.plot(C_list,LR_grid_train_scores_non_binary)
plt4 = plt.plot(C_list,LR_grid_test_scores_non_binary)

plt.xlabel('regularization␣parameter␣C')
plt.ylabel('Accuracy')
plt.legend(['binary-tfidf␣training','binary-tfidf␣validation','non-binary-tfidf␣training', 'non-binary-tfidf␣validation'])

print("Binary␣vectorizer:")
print('max␣C:' + str(C_list[np.argmax(LR_grid_test_scores_binary)]) + "␣validation␣accuracy:" + str(LR_grid_test_scores_binary.max()))
print("Non-Binary␣vectorizer")
print('max␣C:' + str(C_list[np.argmax(LR_grid_test_scores_non_binary)]) + "␣validation␣accuracy:" + str(LR_grid_test_scores_non_binary.max())

"""#### LR Best Model"""

vectorizer = TfidfVectorizer(max_features=4000, stop_words = text.ENGLISH_STOP_WORDS, lowercase=True, ngram_range=(1,2), binary=True)
vec_X_train = np.asarray(vectorizer.fit_transform(X_train).todense())
```

```
clf = LogisticRegression(random_state = 0, C=1)
start = time()
cv_results = cross_validate(clf, vec_X_train, encoded_y_train, cv=10, return_train_score=True)
end = time()

print("cross␣validated␣training␣accuracy:␣" + str(cv_results['train_score'].mean()))
print("cross␣validated␣validation␣accuracy:␣" + str(cv_results['test_score'].mean()))

train_acc_results.append(cv_results['train_score'].mean())
test_acc_results.append(cv_results['test_score'].mean())
run_time.append(end-start)

"""### Random Forest

#### Random Forest Feature Size vs Ngram
"""

plot_ngram_num_features(RandomForestClassifier(random_state=0))

"""#### Random Forest Hyperparameter Tuning"""

bin_vectorizer = TfidfVectorizer(max_features=5000, stop_words = text.ENGLISH_STOP_WORDS, lowercase=True, ngram_range=(1,1), binary=True)
non_bin_vectorizer = TfidfVectorizer(max_features=5000, stop_words = text.ENGLISH_STOP_WORDS, lowercase=True, ngram_range=(1,1), binary=False)

bin_vec_X_train = np.asarray(bin_vectorizer.fit_transform(X_train).todense())
non_bin_vec_X_train = np.asarray(non_bin_vectorizer.fit_transform(X_train).todense())

plt.style.use('default')
plt.figure(figsize=(8,6))

max_depth_list = range(5,156,10)
tuned_parameters = [{'max_depth':max_depth_list}]

clf = RandomForestClassifier(random_state=0)

RF_grid_binary = GridSearchCV(clf,tuned_parameters,cv = 10, return_train_score = True)
RF_grid_binary.fit(bin_vec_X_train, encoded_y_train)
RF_grid_non_binary = GridSearchCV(clf,tuned_parameters,cv = 10, return_train_score = True)
RF_grid_non_binary.fit(non_bin_vec_X_train, encoded_y_train)

RF_grid_train_scores_binary = RF_grid_binary.cv_results_['mean_train_score']
RF_grid_train_scores_non_binary = RF_grid_non_binary.cv_results_['mean_train_score']
RF_grid_test_scores_binary = RF_grid_binary.cv_results_['mean_test_score']
RF_grid_test_scores_non_binary = RF_grid_non_binary.cv_results_['mean_test_score']

plt1 = plt.plot(max_depth_list,RF_grid_train_scores_binary)
plt1 = plt.plot(max_depth_list,RF_grid_test_scores_binary)
plt2 = plt.plot(max_depth_list,RF_grid_train_scores_non_binary)
plt2 = plt.plot(max_depth_list,RF_grid_test_scores_non_binary)

plt.xlabel('maximum␣depth')
plt.ylabel('Validation␣Accuracy')
plt.legend(['binary-tfidf␣training','binary-tfidf␣validation','non-binary-tfidf␣training', 'non-binary-tfidf␣validation'])

print("Binary␣vectorizer:")
print('max␣k:' + str(max_depth_list[np.argmax(RF_grid_test_scores_binary)]) + "␣validation␣accuracy:" + str(RF_grid_test_scores_binary.max())
print("Non-Binary␣vectorizer")
print('max␣k:' + str(max_depth_list[np.argmax(RF_grid_test_scores_non_binary)]) + "␣validation␣accuracy:" + str(RF_grid_test_scores_non_binar

"""#### Random Forest Best Model"""

vectorizer = TfidfVectorizer(max_features=5000, stop_words = text.ENGLISH_STOP_WORDS, lowercase=True, ngram_range=(1,1), binary=False)
vec_X_train = np.asarray(vectorizer.fit_transform(X_train).todense())

clf = RandomForestClassifier(random_state=0, max_depth=45)
start = time()
cv_results = cross_validate(clf, vec_X_train, encoded_y_train, cv=10, return_train_score=True)
end = time()

print("cross␣validated␣training␣accuracy:␣" + str(cv_results['train_score'].mean()))
print("cross␣validated␣validation␣accuracy:␣" + str(cv_results['test_score'].mean()))

train_acc_results.append(cv_results['train_score'].mean())
test_acc_results.append(cv_results['test_score'].mean())
run_time.append(end-start)

"""### Ensemble Stacking """

vectorizer = TfidfVectorizer(max_features=5000, stop_words=lemmatized_stop_words,lowercase=True, ngram_range=(1,1), binary=True, tokenizer=L
vec_X_train = np.asarray(vectorizer.fit_transform(X_train).todense())
vec_X_test = np.asarray(vectorizer.transform(X_test).todense())

# clf1 = LogisticRegression(random_state = 0, C=1)
# clf2 = LinearSVC(random_state = 0, C = 0.517947467923121)
# clf3 = RandomForestClassifier(random_state=0, max_depth=45)

# eclf1 = VotingClassifier(estimators=[
#         ('lr', clf1), ('svm', clf2), ('mnb', clf3)], voting='hard')

clf1 = LogisticRegression(random_state = 0, C=1)
clf2 = LinearSVC(random_state = 0, C = 0.517947467923121)
```

```
clf3 = RandomForestClassifier(random_state=0, max_depth=45)

eclf1 = StackingClassifier(estimators=[
        ('lr', clf1), ('svm', clf2), ('rf', clf3)])

start = time()
cv_results = cross_validate(eclf1, vec_X_train, encoded_y_train, cv=10, return_train_score=True)
end = time()

print("cross␣validated␣training␣accuracy:␣" + str(cv_results['train_score'].mean()))
print("cross␣validated␣validation␣accuracy:␣" + str(cv_results['test_score'].mean()))

train_acc_results.append(cv_results['train_score'].mean())
test_acc_results.append(cv_results['test_score'].mean())
run_time.append(end-start)

eclf1.fit(vec_X_train, encoded_y_train)
index_array = eclf1.predict(vec_X_test)

"""## Experimentations on Best Model"""

vectorizer = TfidfVectorizer(max_features=6000, stop_words = text.ENGLISH_STOP_WORDS, lowercase=True, ngram_range=(1,1), binary=True)
vec_X_train = np.asarray(vectorizer.fit_transform(X_train).todense())
vec_X_test = np.asarray(vectorizer.transform(X_test).todense())

clf = LinearSVC(random_state=0, C = 0.517947467923121)
cv_results = cross_validate(clf, vec_X_train, encoded_y_train, cv=10, return_train_score=True)

print("cross␣validated␣training␣accuracy:␣" + str(cv_results['train_score'].mean()))
print("cross␣validated␣validation␣accuracy:␣" + str(cv_results['test_score'].mean()))

clf.fit(vec_X_train, encoded_y_train)
prediction = clf.predict(vec_X_test)

download_csv(index_array)

svm = LinearSVC(random_state=0, C = 0.517947467923121)
clf = BaggingClassifier(base_estimator=svm, n_estimators=23, random_state=0)

cv_results = cross_validate(clf, vec_X_train, encoded_y_train, cv=10, return_train_score=True)

print("cross␣validated␣training␣accuracy:␣" + str(cv_results['train_score'].mean()))
print("cross␣validated␣validation␣accuracy:␣" + str(cv_results['test_score'].mean()))

clf.fit(vec_X_train, encoded_y_train)
index_array = clf.predict(vec_X_test)

"""##Bar chart"""

plt.style.use('default')
train_acc_results=[0.9554668118835844,
  0.9652499546197133,
  0.8392735927068836,
  0.9986658599060123,
  0.9885648131340634,
  0.9994917407878017,
  0.9998729351969505
]
test_acc_results=[0.8536453201970442,
  0.8467750410509032,
  0.8261937602627258,
  0.9005221674876847,
  0.8902430213464697,
  0.8845221674876849,
  0.9005287356321838]
run_time=[305.90363335609436,
 0.9395320415496826,
 239.54530572891235,
 1.4930760860443115,
 37.7825870513916,
 19.823238849639893,
 331.39533829689026]

m1_t = pd.DataFrame({
 'training' : train_acc_results,
 'validation' : test_acc_results,
 'runtime' : run_time})

m1_t[['training','validation']].plot(kind='bar', width = 0.35)
m1_t['runtime'].plot(secondary_y=True, color='r')


ax = plt.gca()

plt.xlim([-0.35, len(m1_t['validation'])-0.35])
ax.set_xticklabels(('BNB','MNB', 'KNN', 'SVM', 'LR', 'RF','Stacking'))
ax.set_xlabel('model')
ax.legend(loc='lower␣center', bbox_to_anchor=(0.5, 1.05),
          ncol=3, fancybox=True, shadow=True)
plt.show()
```