MTL782
Kaggle Competition: Meme Classification

Report By:
Krishna Chaitanya Reddy Tamaatam
P Vishnu Teja

Data loading and cleaning

In [ ]:

```
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
from nltk.tokenize import TweetTokenizer
import re
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords

from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, accuracy_score, f1_score
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix, roc_auc_score, recall_score, precision_scor
e
!pip install autocorrect
nltk.download('punkt')
nltk.download('wordnet')
from autocorrect import Speller
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.stem import WordNetLemmatizer
```

Importing the data

The data is given in the following form with.

(Raw Data)

In [ ]:

```
data = pd.read_csv("train.csv")
```

In [ ]:

```
data = data[["ID","text","label_num"]]
data.head()
```

Out[ ]:

	ID	text	label_num
0	1	- It is not our fight - Are we not part of thi...	2
1	2	THAT'S THE DIFFERENCE BETWEEN YOU AND ME YOU...	0
2	3	- WHAT DO THE TITANIC AND THE SIXTH SENSE HAVE...	0
3	4	"COME ON MAN, YOU KNOW THE THING.\r\nJUST ASK ...	2
4	5	"Those who believe without reason cannot be co...	0

In [ ]:

```
X_train = data['text']
y_train = data['label_num']
```

Text Cleaning and Pre-processing

Tokenization

Tokenization is the process of breaking down a stream of text into words, phrases, symbols, or any other meaningful elements called tokens.

Stop words

Text and document classification over social media, such as Twitter, Facebook, and so on is usually affected by the noisy nature (abbreviations, irregular forms) of the text corpuses.

Stemming

Text Stemming is modifying a word to obtain its variants using different linguistic processeses like affixation (addition of affixes). For example, the stem of the word "studying" is "study", to which -ing.

Lemmatization

Text lemmatization is the process of eliminating redundant prefix or suffix of a word and extract the base word (lemma).

In [ ]:

```
nltk.download('stopwords')
def clean(X_train):
  stemmer = PorterStemmer()
  words = stopwords.words("english")
  X_train_cleaned=[]
  ps = PorterStemmer()
  lemmatizer = WordNetLemmatizer()

  for line in X_train:
    word_tokens = word_tokenize(line)

    filtered_sentence = [w for w in word_tokens if not w in words]

    filtered_sentence = ""

    for w in word_tokens:
      if w not in words:
        w=ps.stem(w)
        lemmatizer.lemmatize(w)
        filtered_sentence=filtered_sentence+(w)+" "

    X_train_cleaned.append(noise_removal(filtered_sentence))
  return X_train_cleaned
X_train_cleaned=clean(X_train)
```

In [ ]:

```
def clean2(X_train):
  nltk.download('stopwords')
  stemmer = PorterStemmer()
  words = stopwords.words("english")
  return X_train.apply(lambda x: " ".join([stemmer.stem(i) for i in re.sub("[^a-zA-Z]",
" ", x).split() if i not in words]).lower())
X_train_cleaned2= clean2(X_train)
```

Noise Removal

Another issue of text cleaning as a pre-processing step is noise removal. Text documents generally contains characters like punctuations or special characters and they are not necessary for text mining or classification purposes. Although punctuation is critical to understand the meaning of the sentence, but it can affect the classification algorithms negatively.

In [ ]:

```
def noise_removal(text):
    rules = [
        {r'>\s+': u'>'},  # remove spaces after a tag opens or closes
        {r'\s+': u' '},  # replace consecutive spaces
        {r'\s*<br\s*/?>\s*': u'\n'},  # newline after a <br>
        {r'</(div)\s*>\s*': u'\n'},  # newline after </p> and </div> and <h1/>...
        {r'</(p|h\d)\s*>\s*': u'\n\n'},  # newline after </p> and </div> and <h1/>...
        {r'<head>.*<\s*(/head|body)[^>]*>': u''},  # remove <head> to </head>
        {r'<a\s+href="([^"]+)"[^>]*>.*</a>': r'\1'},  # show links instead of texts
        {r'[ \t]*<[^<]*?/?>': u''},  # remove remaining tags
        {r'^\s+': u''}  # remove spaces at the beginning
    ]
    for rule in rules:
      for (k, v) in rule.items():
        regex = re.compile(k)
        text = regex.sub(v, text)
    text = text.rstrip()
    return text.lower()
```

We are going to use cross validation and grid search to find good hyperparameters for our SVM model. We need to build a pipeline to don't get features from the validation folds when building each training model.

Vectorization

A text sentence is converted into a vector with the word being used for it. This is done to feed the input into the respected classifier in the form of a vector.

We tried different other methods like GloVe, Word2Vec but they either required complex models to be implemented or use their huge pre-trained models which were difficult to upload while working on Kaggle Notebook/Google Colab.

Hence decided to rather use a simple vectorizer like CountVectorizer

In [ ]:

```
def tokenize(text):
    tknzr = TweetTokenizer()
    return tknzr.tokenize(text)

def stem(doc):
    return (stemmer.stem(w) for w in analyzer(doc))

en_stopwords = set(stopwords.words("english"))

vectorizer = CountVectorizer(
    analyzer = 'word',
    tokenizer = tokenize,
    lowercase = False,
    ngram_range=(1, 1),
    stop_words = en_stopwords)
```

We split the data into training and testing set:

In [ ]:

```
from sklearn.model_selection import train_test_split
```

```
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X_train, y_train, test_size = 0.25,
random_state = 0)
```

SVM

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points. The points that are closest to the hyperplane are called support vectors and are used for determing the orientation of the optimal hyperplane. we 3 hyperplanes namely (π, π+, π−) such that 'π+' is parallel to 'π' passing through the support vectors on the positive side and 'π−' is parallel to 'π' passing through the support vectors on the negative side. where (π, π+, π−) are given as below

π = $(w^T)X + b = 0$

π+ = $(w^T)X + b = 1$

π− = $(w^T)X + b = -1$

the distance between π+, π− is 2/||w|| which we have to maximize which is similar in minimizing $||w||^2/2$

Kernel Trick

every labelled data point is a constraint in the SVM which can be modified to varaiables in its dual form . The dual equation to be optimized will be

$$\sum_{i=1}^{n} a_i - \sum_{i}^{n}$$
$$\sum_{j}^{n} a_i a_j y_i y_j$$
$$(X_i^T X_j)$$

with the constraints

$$0 \leq a_i$$
$$\leq Cand$$
$$\sum_{i=1}^{n} a_i y_i = 0$$

We replace the above equation using the kernel (X^TX) with $K(X^TX)$ to increase the dimensionality of our input by adding higher powers of X and learn a non linear boundary. some of the kernels that we used given below

1. **linear kernel**
2. **polynomial kernel**
3. **Radial basis function kernel (RBF)/ Gaussian Kernel**

Gaussian Kernel can include infinte dimensional features while polynomial kernel includes polynomial features of a certain degree.

In [ ]:

```
np.random.seed(1)

pipeline_svm = make_pipeline(vectorizer,
                             SVC(probability=True, kernel="rbf", class_weight="balanced")
)

pipeline_svm.fit(X_Train, Y_Train)
```

In [ ]:

```
from sklearn.metrics import classification_report, confusion_matrix

y_predict=pipeline_svm.predict(X_Test)
print(classification_report(Y_Test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.45      0.18      0.26       148
           1       0.37      0.56      0.45       173
           2       0.39      0.38      0.39       177

    accuracy                           0.39       498
   macro avg       0.40      0.38      0.36       498
weighted avg       0.40      0.39      0.37       498
```

Random Forest Classifier

Random forests or random decision forests technique is an ensemble learning method for text classification. This method was introduced by T.Kam Ho in 1995 for first time which used t trees in parallel. It belongs to family of Decision Tree Classifiers. Decision tree classifiers (DTC's) are used successfully in many diverse areas of classification. The structure of this technique includes a hierarchical decomposition of the data space (only train dataset).

In [ ]:

```
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.ensemble import RandomForestClassifier


pipeline_RFC = Pipeline([('vect', vectorizer),
                     ('tfidf', SelectKBest(chi2, k=1009)),
                     ('clf', RandomForestClassifier()),
                     ])
pipeline_RFC.fit(X_Train, Y_Train)
```

In [ ]:

```
from sklearn.metrics import classification_report, confusion_matrix

y_predict=pipeline_RFC.predict(X_Test)
print(classification_report(Y_Test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.32      0.27      0.29       148
           1       0.38      0.43      0.40       173
           2       0.34      0.33      0.33       177

    accuracy                           0.35       498
   macro avg       0.34      0.34      0.34       498
weighted avg       0.34      0.35      0.34       498
```

Linear SVM

In [ ]:

```
from sklearn.svm import LinearSVC
pipeline_LSVM = Pipeline([('vect', CountVectorizer()),
                     ('tfidf', TfidfTransformer()),
                     ('clf', LinearSVC()),
                     ])
pipeline_LSVM.fit(X_Train, Y_Train)
```

In [ ]:

```
from sklearn.metrics import classification_report, confusion_matrix

y_predict=pipeline_LSVM.predict(X_Test)
print(classification_report(Y_Test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.33      0.30      0.32       148
           1       0.38      0.39      0.38       173
           2       0.37      0.38      0.38       177

    accuracy                           0.36       498
   macro avg       0.36      0.36      0.36       498
weighted avg       0.36      0.36      0.36       498
```

Logistic Regression

In [ ]:

```
from sklearn.linear_model import LogisticRegression
pipeline_LR = Pipeline([('vect', vectorizer),
                     ('tfidf', TfidfTransformer()),
                     ('clf', LogisticRegression()),
                     ])
pipeline_LR.fit(X_Train, Y_Train)
```

In [ ]:

```
from sklearn.metrics import classification_report, confusion_matrix

y_predict=pipeline_LR.predict(X_Test)
print(classification_report(Y_Test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.31      0.22      0.25       148
           1       0.38      0.40      0.39       173
           2       0.39      0.46      0.42       177

    accuracy                           0.37       498
   macro avg       0.36      0.36      0.35       498
weighted avg       0.36      0.37      0.36       498
```

We can observe that without the data pre-processing, SVC model with rbf kernal on the training data set with 0.25 test data split, has the highest accuracy (0.39) among the models which we tried.

Considering the Data preprocessing:

In [ ]:

```
from sklearn.model_selection import train_test_split

X_Train, X_Test, Y_Train, Y_Test = train_test_split(X_train_cleaned, y_train, test_size
= 0.25, random_state = 0)
```

In [ ]:

```
np.random.seed(1)

pipeline_svm = make_pipeline(vectorizer,
                            SVC(probability=True, kernel="rbf", class_weight="balanced")
)

pipeline_svm.fit(X_Train, Y_Train)
```

In [ ]:

```
from sklearn.metrics import classification_report, confusion_matrix

y_predict=pipeline_svm.predict(X_Test)
```

```
print(classification_report(Y_Test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.38      0.20      0.26       148
           1       0.36      0.53      0.43       173
           2       0.39      0.37      0.38       177

    accuracy                           0.37       498
   macro avg       0.38      0.37      0.36       498
weighted avg       0.38      0.37      0.36       498
```

Bert for sentiment classification of text

BERT stands for (Bidirectional Encoder Representations from Transformers) is a NLP model developed by Google for pre-training language representations. It leverages an enormous amount of plain text data publicly available (Wikipedia and Google Books) on the web and is trained in an unsupervised manner. It is a powerful model that is trained to learn the language structure and it's nuances by training a Language Model. BERT has a deep bi-directional structure to it unlike ELMo, which is a shallow bi-directional and OpenAI GPT which is uni-directional in nature. Bidirectional nature helps the model to capture the context from previous words and words ahead of it any given time t.

We did'nt go much into details how bert works as it is very difficult to understard but used a preexisting model for multilabel text classification using bert.given below is the link to the kaggle notebook that we used for bert classification

Link for the BERT model we used:

https://github.com/chaitu0032-debug/text_Sentiment_Analysis/blob/main/text-classification-using-bert.ipynb

LSTM (RNN):

Long Short-Term Memory (LSTM) is a special type of RNN that preserves long term dependency in a more effective way compared to the basic RNNs. This is particularly useful to overcome vanishing gradient problem. Although LSTM has a chain-like structure similar to RNN, LSTM uses multiple gates to carefully regulate the amount of information that will be allowed into each node state. Figure shows the basic cell of a LSTM model.

CNN:

Although originally built for image processing with architecture similar to the visual cortex, CNNs have also been effectively used for text classification. In a basic CNN for image processing, an image tensor is convolved with a set of kernels of size d by d. These convolution layers are called feature maps and can be stacked to provide multiple filters on the input. To reduce the computational complexity, CNNs use pooling which reduces the size of the output from one layer to the next in the network. Different pooling techniques are used to reduce outputs while preserving important features.

In both these models we had to use GloVe model for converting sentences into vector and we couldn't do so as I have mentioned earlier. Besides these problem we to tried to use a small pre-trained GloVe model, but failed to improve the accuracy.

Link to our model:

https://github.com/chaitu0032-debug/text_Sentiment_Analysis/blob/main/RNN_CNN.ipynb

Results and Conclusion

We observed that data processing didn't improve the accuracy in any of the model when compared to the earlier SVC without the pre-processing. Morever the public score was more in case of the model without pre-processing.

Without Pre-processing:

Best accuracy: 0.39

Best private score: 0.48

With Pre-processing:

Best accuracy: 0.38

Best private score: 0.46

Hence, we decided to consider the model without pre-processing for our final submission.

In [ ]:

```
data_test = pd.read_csv("test.csv")
```

In [ ]:

```
data_test = data_test[["ID","text"]]
data_test.head()
X_test = data_test["text"]
```

In [ ]:

```
np.random.seed(1)

pipeline_svm = make_pipeline(vectorizer,
                             SVC(probability=True, kernel="rbf", class_weight="balanced")
)

pipeline_svm.fit(X_train, y_train)
```

In [ ]:

```
abc = pipeline_svm.predict(X_test)
```

In [ ]:

```
data_test["label_num"] = abc
```

In [ ]:

```
data_test = data_test[["ID","label_num"]]
```

In [ ]:

```
data_test
```

Out[ ]:

	ID	label_num
0	2001	0
1	2002	2
2	2003	2
3	2004	2
4	2005	2
...	...	...
595	2596	1
596	2597	1
597	2598	2
598	2599	1
599	2600	0

600 rows × 2 columns

In [ ]:

```
data_test.to_csv("test_out.csv")
```