CMT 307

Applied Machine Learning

ARUNIMA CHAUDHARY

STUDENT NUMBER- 1655016

SUBMITTED TO: - JOSE CAMACHO-COLLADOS, YUHUA LI

PART 2

The aim of this task is to preprocess a given sentiment analysis data, select features and train a machine learning model of choice. Later, feature selection should be performed to reduce the dimensionality of the features.

Firstly, all the libraries needed to train the model is imported. Essentially, numpy for vector manipulation, nltk for text processing and scikit-learn for machine learning algorithms.

```
[2] import numpy as np
    import pandas as pd
    import nltk
    import sklearn
    import operator
    import requests
    from string import punctuation
    from os import listdir
    from collections import Counter
    from nltk.corpus import stopwords
    import re
    from collections import Counter
    from nltk.util import ngrams
    nltk.download('stopwords')
    nltk.download('punkt')
    nltk.download('wordnet')
    from sklearn.feature_selection import chi2, SelectKBest
    from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score, confusion_matrix
    from sklearn.feature_extraction.text import TfidfVectorizer
    from sklearn.preprocessing import StandardScaler, MinMaxScaler
    import time
    from scipy.sparse import coo_matrix, hstack
    from sklearn.tree import DecisionTreeClassifier
    from string import punctuation
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

After importing the necessary packages, all the datasets, I.e., training, development and test, are loaded using the url from github.

The positive reviews are labelled 1 while the negative reviews are labelled 0 in the datasets. Three new datasets are then generated namely, new_dev, new_train and new_test. There is no splitting required as the datasets given were already separated.

- new_dev is used to fine tune the model
- new_train is used to train the model
- new_test is used for evaluation

```python
#initiating three datasets, each having positive and negative reviews
new_dev=[]
for pos_review in dev_pos:
  new_dev.append((pos_review,1))
for neg_review in dev_neg:
  new_dev.append((neg_review,0))

  new_test=[]
for pos_review in test_pos:
  new_test.append((pos_review,1))
for neg_review in test_neg:
  new_test.append((neg_review,0))

  new_train=[]
for pos_review in train_pos:
  new_train.append((pos_review,1))
for neg_review in train_neg:
  new_train.append((neg_review,0))
```

```
TRAINING SET
Size training set: 15002
('For fans of Chris Farley, this is probably his best film. David Spade plays the perfect cynical, sarcastic yin to Farley\'s "Baby Huey" yang. Farley achieves strokes of comic genius in hi

-------

TEST SET
Size development set: 5002
('After 10 viewings in 20 years I too think this was the Crazy Gang\'s best effort on film, with more cohesion in the plot than their next best, "Alf\'s Button Afloat". They were indeed a c

-------

DEVELOPMENT SET
Size test set: 5002
('This is the greatest movie if you want inspiration on following your heart and never giving up on your dream. Elizabeth Taylor is Velvet and in her prime (of her childhood, at least), Mic
```

The new datasets are then printed to recheck.

Preprocessing

To process the data a few steps are taken. First, we tokenize the text, which means create a list where each element is a word (or a token).

Then, lemmatization is done which means extracting the lemma form of each word.

After this the stopwords are removed from the datasets which weren't required. This included punctuation marks as well. This is a feature selection. These are removed so that the dataset can be free from unwanted words which would eventually affect our results.

```
[ ]
lemmatizer = nltk.stem.WordNetLemmatizer()

def get_list_tokens(string):
  sentence_split=nltk.tokenize.sent_tokenize(string)
  list_tokens=[]
  for sentence in sentence_split:
    list_tokens_sentence=nltk.tokenize.word_tokenize(sentence)
    for token in list_tokens_sentence:
      list_tokens.append(lemmatizer.lemmatize(token).lower())
  return list_tokens

# First, we get the stopwords list from nltk
stopwords=set(nltk.corpus.stopwords.words('english'))
# We can add more words to the stopword list, like punctuation marks
stopwords.add(".")
stopwords.add(",")
stopwords.add("--")
stopwords.add("``")
stopwords.add("#")
stopwords.add("@")
stopwords.add(":")
stopwords.add("'s")
stopwords.add("’")
stopwords.add("...")
stopwords.add("n't")
stopwords.add("'re")
stopwords.add("'")
stopwords.add("-")
stopwords.add(";")
stopwords.add("/")
stopwords.add(">")
stopwords.add("<")
stopwords.add("br")
stopwords.add("(")
stopwords.add(")")
stopwords.add("''")
stopwords.add("&")
```

Further processing was done by erasing all the adjectives and adverbs from the datasets as a part of the feature selection.

Then, a frequency dictionary of the reviews was created which consisted the total number of times the words were present in the datasets. This was later sorted out as well.

```
dict_word_frequency={}
for pos_review in train_pos:
  sentence_tokens=get_list_tokens(pos_review)
  for word in sentence_tokens:
    if word in stopwords: continue
    if word not in dict_word_frequency: dict_word_frequency[word]=1
    else: dict_word_frequency[word]+=1
for neg_review in train_neg:
  sentence_tokens=get_list_tokens(neg_review)
  for word in sentence_tokens:
    if word in stopwords: continue
    if word not in dict_word_frequency: dict_word_frequency[word]=1
    else: dict_word_frequency[word]+=1

# Now we create a sorted frequency list with the top 1000 words, using the function "sorted". Let's see the 15 most frequent words
sorted_list = sorted(dict_word_frequency.items(), key=operator.itemgetter(1), reverse=True)[:1000]
i=0
for word,frequency in sorted_list[:15]:
  i+=1
  print (str(i)+". "+word+" - "+str(frequency))

# Finally, we create our vocabulary based on the sorted frequency list
vocabulary=[]
for word,frequency in sorted_list:
  vocabulary.append(word)
```

```
1. / - 59704
2. > - 59628
3. < - 59572
4. br - 59548
5. 's - 36104
6. movie - 29647
7. wa - 29577
8. film - 26929
9. ) - 21211
10. ( - 20708
11. '' - 19859
12. `` - 19693
13. n't - 19639
14. one - 15987
15. ! - 14847
```

```
def get_vector_text(list_vocab,string):
```

Feature selection

The features selected are: -

- Word frequency (as given)
- Adjectives and verbs
- N-grams

This is done by creating a sorted list of the top 1000 words and then creating a vocabulary of these words. Then these are converted to vectors which is a necessary step in machine learning to present the input as array of numbers.

The vectorizer then takes in the token function, and the n-gram value to give the token count.

```
[ ] vectorizer = TfidfVectorizer(ngram_range=(1,3),max_features=2000)
    matrix = vectorizer.fit_transform(np.asarray([i[0] for i in new_train]))
    X_train = matrix.toarray()
    Y_train=[i[1] for i in new_train]

    svm_clf=sklearn.svm.SVC(kernel="linear",gamma='auto')
    svm_clf.fit(np.asarray(X_train),np.asarray(Y_train))

    X_dev = vectorizer.transform([i[0] for i in new_dev]).toarray()
    predictions = svm_clf.predict(X_dev)
    print(sklearn.metrics.classification_report(predictions,[i[1] for i in new_dev]))
```

```
              precision    recall  f1-score   support

           0       0.86      0.88      0.87      2427
           1       0.88      0.86      0.87      2575

    accuracy                           0.87      5002
   macro avg       0.87      0.87      0.87      5002
weighted avg       0.87      0.87      0.87      5002
```

The model is then again trained with respective to the combination of all the features.

Overall performance

The model first gave the following precision, recall, f-measure and accuracy

```
[ ] new_svm_clf_sentanalysis_=sklearn.svm.SVC(kernel="linear",gamma='auto')
    new_svm_clf_sentanalysis_.fit(np.asarray(X_train_feature_best),Y_train)
    predictions_feature_best = new_svm_clf_sentanalysis_.predict(X_test_feature_best)
    print(sklearn.metrics.classification_report(predictions_feature_best,Y_test))
```

```
              precision    recall  f1-score   support

           0       0.84      0.89      0.86      2376
           1       0.89      0.85      0.87      2626

    accuracy                           0.87      5002
   macro avg       0.87      0.87      0.87      5002
weighted avg       0.87      0.87      0.87      5002
```

We referred to the labels in the test set as Y_test_gold to distinguish them from our predictions. Now we can test our model to obtain the predictions of our model and get the results from sklearn.

The model is then improved by tuning our development set, as that can help improve our model overall. We can tune the features and decrease its dimensionality here, as required in the task. Once the feature selection is done the model is exposed to the trained dataset to avoid overfitting.

```
precision=precision_score(Y_test_gold, Y_text_predictions, average='macro')
recall=recall_score(Y_test_gold, Y_text_predictions, average='macro')
f1=f1_score(Y_test_gold, Y_text_predictions, average='macro')
accuracy=accuracy_score(Y_test_gold, Y_text_predictions)

print ("Precision: "+str(round(precision,3)))
print ("Recall: "+str(round(recall,3)))
print ("F1-Score: "+str(round(f1,3)))
print ("Accuracy: "+str(round(accuracy,3)))
```

```
Precision: 0.848
Recall: 0.848
F1-Score: 0.848
Accuracy: 0.848
```

The dimensionality is reduced by the chi-squared test method, taking best 500 features and maximum of 5000 features.. This method will remove the features that appear to be irrelevant. For this the respective libraries are imported and then the vectorizer will take the frequency and the n-grams as parameters. The common .function is used to reduce the original features in the model.

The vectorized dataset is then stored to X_test and as an array to Y_test before the final training.

The performance is tuned by different n-grams and features as shown below. (n- gram 2)

```
[ ]
vectorizer_f = TfidfVectorizer(ngram_range=(1,2),max_features=5000)
X_train = vectorizer_f.fit_transform([i[0] for i in new_train]).toarray()
Y_train = np.asarray([i[1] for i in new_train])

X_feature_best = SelectKBest(chi2, k=500).fit(X_train, Y_train)
X_train_feature_best = X_feature_best.transform(X_train)

X_test = vectorizer_f.transform([i[0] for i in new_test]).toarray()
X_test_feature_best =  X_feature_best.transform(X_test)
```

Lastly after the tuning we again calculate the precision, recall, f-measure and accuracy and get the following results: -

```
              precision    recall  f1-score   support

           0       0.84      0.89      0.86      2376
           1       0.89      0.85      0.87      2626

    accuracy                           0.87      5002
   macro avg       0.87      0.87      0.87      5002
weighted avg       0.87      0.87      0.87      5002
```

Future improvements

The model can be further refined by eliminating maximum stop words. Most of the words and punctuations were tried to be deleted but due to informal text, there were still a few short forms and symbols that might have been left. Besides this the model could have been tuned more number of times and precisely, but there is always a chance overfitting which should be avoided by all means.

Extra credit d) code release link - https://github.com/arunima-22/CMT307