Multi Class Text Classification

Ameet Deshpande

May 2017

1 Abstract

This is a multi label text classifier built using the 20 newsgroups data set that is available here and here. The dataset contains 20 topics in total and about 20000 articles. Some topics like comp.sys.ibm.pc.hardware, comp.sys.mac.hardware are very closely related and some topics are unrelated. This is thus a good training data to train and test on. Both SVM and Multinomial Naive Bayes model with modifications will be used to train and test the articles. Code will be written in *python* and scikit-learn library will be used extensively.

2 Topics

- comp.graphics
- comp.os.ms-windows.misc
- comp.sys.ibm.pc.hardware
- comp.sys.mac.hardware
- comp.windows.x
- rec.autos
- rec.motorcycles
- rec.sport.baseball
- rec.sport.hockey
- sci.crypt
- sci.electronics
- sci.med
- sci.space
- misc.forsale

- talk.politics.misc
- talk.politics.guns
- talk.politics.mideast
- talk.religion.misc
- alt.atheism
- soc.religion.christian

3 Code for Multinomial Naive-Bayes Approach

```
import os
import numpy
from pandas import DataFrame
from sklearn.feature_extraction.text import CountVectorizer
#from nltk.corpus import stopwords
from sklearn.naive_bayes import MultinomialNB
#from sklearn.linear_model import SGDClassifier
#from sklearn import svm
#from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import TfidfTransformer    # Using inverse document frequency to filter the noise
from sklearn.pipeline import Pipeline
import sys
from sklearn import metrics
sys.path.insert(0,"C:\\Users\\A6000401\\Desktop\\NLP\\20 Newsgroup Datasets\\20news-bydate\\20news-bydate-test")
from get_test_data import test_data_function

# Create a dictionary to map topic names to integer labels

topic_mapping = {
    'alt.atheism': 1,
    'comp.graphics': 2,
    'comp.os.ms-windows.misc': 3,
    'comp.sys.ibm.pc.hardware': 4,
    'comp.sys.mac.hardware': 5,
    'comp.windows.x': 6,
    'rec.autos': 7,
```

```
    'rec.motorcycles': 8,
    'rec.sport.baseball': 9,
    'rec.sport.hockey': 10,
    'sci.crypt': 11,
    'sci.electronics': 12,
    'sci.med': 13,
    'sci.space': 14,
    'misc.forsale': 15,
    'talk.politics.misc': 16,
    'talk.politics.guns': 17,
    'talk.politics.mideast': 18,
    'talk.religion.misc': 19,
    'soc.religion.christian': 20,
    }

rows = []  # Contains the text of the article and the
    ↪ class that it belongs to
index = []  # Unique index, here the file name

# Walk through all the files and store the content in a
    ↪ data frame

for (root, dirs, files) in os.walk('.', topdown=True):
    for name in dirs:

        # For all the sub-directories in the root folder,
            ↪  if the directory contains articles

        if str(name) in topic_mapping:
            current_path = str(os.path.join(root, name))
            files1 = os.listdir(current_path)
            for file_name in files1:  # For all the files
                ↪  in the root directory
                f1 = open(current_path + str('/') + str(
                    ↪ file_name), 'r')
                content = f1.readlines()
                content = '␣'.join(content)  # Joins the
                    ↪ contents of the list into one
                    ↪ single string separated by a space
                rows.append({'text': content,
                             'class': topic_mapping[str(
                                ↪ name)]})  # Based on
                                ↪ the directory, assign
                                ↪ the class value
                index.append(str(file_name))  # Use the
                    ↪ file name as the index of the entry
```

```
                f1.close()

# Create a dataframe with text as one column and class as
    ↪ the other column
data_frame = DataFrame(rows, index=index)

print(len(data_frame))

# Adding few common words that are frequent in this
    ↪ dataset, but do not contribute to class resolution
stop = stopwords.words('english')
stop = list(stop)
stop.extend((str('subject'), str('from'), str('
    ↪ organization'
            ), str('organisation')))

# Using pipeline to fit data to model and then convert it
    ↪ to tf-idf counts
pipeline = Pipeline([
    ('count_vectorizer',    CountVectorizer(ngram_range
        ↪ =(1,  2))),
    ('tfidf_transformer',   TfidfTransformer()),
    ('classifier',      MultinomialNB())
])

pipeline.fit(data_frame['text'].values, data_frame['class
    ↪ '].values)

test_data_frame = test_data_function()   # Test data is
    ↪ obtained as a data frame

predictions = pipeline.predict(test_data_frame["text"])
correct_answers = test_data_frame["class"]
accuracy = metrics.accuracy_score(correct_answers,
    ↪ predictions, normalize=True)

print("The Accuracy is : ")
print(accuracy*100)
print("The F-Score is : ")
print(metrics.f1_score(correct_answers, predictions,
    ↪ average='macro'))
print("Total number of articles in test set it : ")
print(len(predictions))
```

4 Analysis for Multinomial Naive Bayes

- Classifier Name : $MultinomialNB()$
- The accuracy achieved with just the unigram counts was 79.92%
- When both unigram and bigram counts were used, the accuracy was 79.76% and the time of execution increased considerably.
- Inverse document frequency helps in classification when a lot of common words exist in the categories. Using this in the code increases the accuracy to 82.32%.
- Changing the smoothing parameter to 0.1 further increases the accuracy percentage by 3, taking it to 85.21%. To set the smoothing parameter, a seperate development set needs to be used
- Often times, $F-Score$ is a better metric to evaluate a classification model. The $F-Score$ of the model is 0.807.
- The model was run on 11314 test documents and 7532 test documents.
- from $sklearn.model\_selectionimportGridSearchCV$ provides this function which can suggest the values of the parameters for the classifier which gives best accuracy.

5 Analysis for Linear Support Vector Machine

- Classifier Name : $SGDClassifier()$
- Support Vector Machine takes longer time than Naive-Bayes to train. It is however known to be better at text classification than the naive method.
- Using only unigram counts, an accuracy of 86.3% is achieved. Including both unigram and bigram counts increases the accuracy to 87.2% and the $F-Score$ obtained increases to 0.851
- Including the trigram values increases the execution time considerably and yields an $F-Score$ of 0.845, which is no improvement from bigram values.

6 Analysis for Non-Linear SupportVM

- Classifier Name : $svm.NuSVC()$
- On this dataset, it performs very poorly, with an $F-Score$ of just 0.53

7 Analysis for Random Forest Classifier

- Classifier Name : $sklearn.ensemble.RandomForestClassifier()$
- This classifier takes considerably larger execution time compared to other classifiers. It yields an $F - Score$ of 0.721.