NLP For Economists

Text Classification

Sowmya Vajjala

Munich Graduate School of Economics - LMU Munich

Guest Course, October 2020

Goals

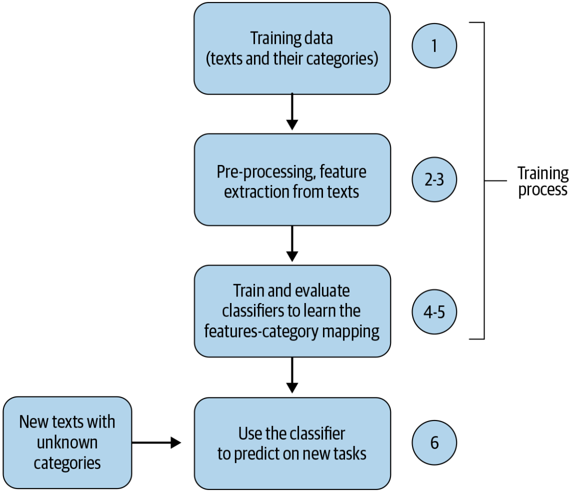
- What is text classification and where is it useful?
- How to train and use text classification models for prediction and visualization of results?
- source: Chapter 4 in our book - practicalnlp.ai
- Other useful references:
  - NLTK's Chapter 6 - `https://www.nltk.org/book/ch06.html`
  - Chapters 4 and 5 from "Speech & Language Processing" - `https://web.stanford.edu/~jurafsky/slp3/4.pdf` (and 5.pdf)

What is text classification and where is it useful?

- Text classification is the task of assigning one or more categories to a given piece of text from a larger set of possible categories.
- In the email spam–identifier example, we have two categories—spam and non-spam—and each incoming email is assigned to one of these categories.
- This task of categorizing texts based on some properties has a wide range of applications across diverse domains, such as social media, e-commerce, healthcare, law, and marketing, to name a few.
- Even though the purpose and application of text classification may vary from domain to domain, the underlying abstract problem remains the same.
- A common usecase in economics research: sentiment classification.

How to use text classification for our own research

I am taking the example texts that come with nltk in this example


Training data
(texts and their categories)
1
Pre-processing, feature extraction from texts
2-3
Train and evaluate classifiers to learn the features-category mapping
4-5
Training process
New texts with unknown categories
Use the classifier to predict on new tasks
6

Walking through an example: Corpus

- we'll use the "Economic News Article Tone and Relevance" dataset
- It consists of 8,000 news articles annotated with whether or not they're relevant to the US economy
- The dataset is also imbalanced, with 1,500 relevant and 6,500 non-relevant articles, which poses the challenge of guarding against learning a bias toward the majority category (in this case, non-relevant articles)
- Clearly, learning what a relevant news article is is more challenging with this dataset than learning what is irrelevant. After all, just guessing that everything is irrelevant already gives us 80% accuracy!
- Let us explore how a BoW representation (introduced in Chapter 3) can be used with this dataset following the pipeline described earlier in this chapter.

`https://data.world/crowdflower/economic-news-article-tone`

Walkthrough: Reading the corpus into python

Corpus is a .csv file

```
import pandas as pd #to work with csv files
our_data = pd.read_csv("Full-Economic-News-DFE-839861.csv" , encoding = "ISO-8859-1" )
#our_data.head() #This shows some the first few rows. We need the columns: relevance and text to do text c
our_data.shape #Number of rows (instances) and columns in the dataset
our_data["relevance"].value_counts()/our_data.shape[0] #Class distribution in the dataset
```

`https:`

`//github.com/nishkalavallabhi/practicalnlp/blob/master/Ch4/OnePipeline_ManyClassifiers.ipynb`

What we learn from these basic stats

- There is an imbalance in the data with not relevant being 82% in the dataset.
- That is, most of the articles are not relevant to US Economy, which makes sense in a real-world scenario, as news articles discuss various topics.
- We should keep this class imbalance mind when interpreting the classifier performance later.
- Let us first convert the class labels into binary outcome variables for convenience. 1 for Yes (relevant), and 0 for No (not relevant), and ignore "Not sure".

Walkthrough: convert labels to binary

```
# convert label to a numerical variable
our_data = our_data[our_data.relevance != "not sure"]
our_data.shape
our_data['relevance'] = our_data.relevance.map({'yes':1, 'no':0}) #relevant is 1, not-relevant is 0.
our_data = our_data[["text","relevance"]] #Let us take only the two columns we need.
our_data.shape
```

https:

//github.com/nishkalavallabhi/practicalnlp/blob/master/Ch4/OnePipeline_ManyClassifiers.ipynb

Walkthrough: text pre-processing

Here, we are performing the following steps: removing br tags, punctuation, numbers, and stopwords.

```
from sklearn.feature_extraction import stop_words
import string
import re

stopwords = stop_words.ENGLISH_STOP_WORDS
def clean(doc): #doc is a string of text
    doc = doc.replace("</br>", " ") #This text contains a lot of <br/> tags.
    doc = "".join([char for char in doc if char not in string.punctuation and not char.isdigit()])
    doc = " ".join([token for token in doc.split() if token not in stopwords])
    #remove punctuation and numbers
    return doc
```

https:

//github.com/practical-nlp/practical-nlp/blob/master/Ch4/01_OnePipeline_ManyClassifiers.ipynb

Steps towards building a model

1. Split the data into training and test sets (75% train, 25% test)
2. Extract features from the training data using CountVectorizer, which we saw earlier. We will use the pre-processing function above in conjunction with Count Vectorizer
3. Transform the test data into the same feature vector as the training data.
4. Train the classifier
5. Evaluate the classifier

Split the data into train/test sets

```
from sklearn.model_selection import train_test_split

#Step 1: train-test split
X = our_data.text #the column text contains textual data to extract features from
y = our_data.relevance #this is the column we are learning to predict.
print(X.shape, y.shape)
# split X and y into training and testing sets. By default, it splits 75% training and 25% test
#random_state=1 for reproducibility
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

Text preprocessing and feature extraction

```
from sklearn.feature_extraction.text import CountVectorizer

vect = CountVectorizer(preprocessor=clean) #instantiate a vectoriezer
#to cut the dimension of feature vector you can use max_features in CountVectorizer
#vect = CountVectorizer(preprocessor=clean, max_features=1000)

X_train_dtm = vect.fit_transform(X_train)#use it to extract features from training data
#transform testing data (using training data's features)
X_test_dtm = vect.transform(X_test)
print(X_train_dtm.shape, X_test_dtm.shape)
#i.e., the dimension of our feature vector is 49753!
```

Training a model

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

#Step 3: Train the classifier and predict for test data
logreg = LogisticRegression(class_weight="balanced") #instantiate a logistic regression model
logreg.fit(X_train_dtm, y_train) #fit the model with training data
y_pred_class = logreg.predict(X_test_dtm)#make class predictions for test data

#calculate evaluation measures:
print("Accuracy: ", accuracy_score(y_test, y_pred_class))
```

Accuracy is misleading in this case!

- Accuracy makes sense only when class distribution is more or less balanced.
- Otherwise, we won't know whether the classifier is just learning a majority class just by looking at accuracy alone.
- A good way to understand the model is to look at the confusion matrix.
- sklearn has a confusion matrix implementation. However, I used a custom code in the past for neater presentation.
- More details in the notebook: `https://github.com/practical-nlp/practical-nlp/blob/master/Ch4/01_OnePipeline_ManyClassifiers.ipynb`

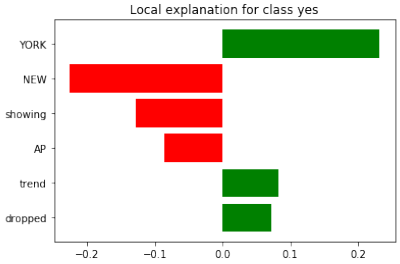
Using a model and interpreting its predictions

- Assuming we explore several classifiers, and fix on one best thing for our dataset, what next?
- We need a way to "use" this trained model
- It would be good to also have a way to understand the model predictions at least
- Recent research in NLP has focused on interpretability of such ML models.
- one of the libraries useful for this task is: lime `https://github.com/marcotcr/lime`

Use this model to predict or interpret for new text

```
from lime import lime_text
from lime.lime_text import LimeTextExplainer

y_pred_prob = classifier.predict_proba(X_test_dtm)[:, 1]
c = make_pipeline(vect, classifier)
mystring = list(X_test)[221] #Take a string from test instance
print(c.predict_proba([mystring])) #Prediction is a "No" here. i.e., not relevant
class_names = ["no", "yes"] #not relevant, relevant
explainer = LimeTextExplainer(class_names=class_names)
exp = explainer.explain_instance(mystring, c.predict_proba, num_features=6)
exp.as_list()
```



Use the model to predict for new texts

Step 1: Save the model and its processing pipeline

```
from sklearn.pipeline import make_pipeline
from sklearn.externals import joblib

vect = CountVectorizer(preprocessor=clean, max_features=1000)
classifier = LogisticRegression(class_weight='balanced')
pipeline = make_pipeline(vect, classifier)
pipeline.fit(X_train, y_train)
joblib.dump(pipeline, "mymodel.pkl")
```

Use the model to predict for new texts

Step 2: Use this model and make predictions!

```
from sklearn.feature_extraction import stop_words
import string
from sklearn.externals import joblib

#same preprocessor is needed again:
stopwords = stop_words.ENGLISH_STOP_WORDS
def clean(doc): #doc is a string of text
    doc = doc.replace("</br>", " ") #This text contains a lot of <br/> tags.
    doc = "".join([char for char in doc if char not in string.punctuation and not char.isdigit()])
    doc = " ".join([token for token in doc.split() if token not in stopwords])
    #remove punctuation and numbers
    return doc

model_file = "mymodel.pkl"
pipeline = joblib.load(model_file)

mystring = "Every facet of Canadian life has been changed by the current pandemic, from how and
   where we live, to how we shop, eat and work. While not all changes have been
           for the better, COVID-19 could bring about some positive changes to Canada's economy."
print(pipeline.predict([mystring])) #prints only the prediction
print(pipeline.predict_proba([mystring]))
#prints predictions with probabilities in the order: [not relevant, relevant]
```

Concluding Remarks

- I've skipped a lot of "experimentation" part, but I hope this gives you a picture of the various steps involved in experimenting with text classification approaches, and using them afterwards.
- Read the Chapter 4 in practicalnlp.ai and Chapter 4 in "Speech and Language Processing" (3rd edition, available online) to get a full-er picture!
- Normally, once we have a dataset, we experiment with multiple classifiers, feature representation approaches, and tune our approach until we get the best result.
- The best model can then be used to make new predictions or understand the existing predictions and so on!