NLP For Economists

Lecture 4: Deep dive into text classification and topic modeling

Sowmya Vajjala

MGSE - LMU Munich
Guest Course, October 2022

11th October 2022

Week 1 - A Recap

- NLP Basics
- Introductory python
- NLP and Economics: an overview
- NLP Methods: An Overview

Do these NLP approaches work?

A small exercise

Visit https://huggingface.co/tasks, pick a task, and browse some model demos there. Check out how they work for a few minutes. (10 minutes)

Share your observations.

Outline for Today

- Text classification: Overview
- Different approaches to text classification
- Topic modeling: Overview

Why?

Why is there a deeper discussion only on these tasks, and not on others?

- popular use cases of NLP in general
- also the most common NLP tasks in the economics papers I came across.

Text Classification

- It is the task of assigning one (or more) categories to a given piece of text from a larger set of possible categories.

Text Classification

- It is the task of assigning one (or more) categories to a given piece of text from a larger set of possible categories.
- In an email spam identifier, we have two categories: spam and non-spam, and each incoming email is assigned to one of these categories.

Text Classification

- It is the task of assigning one (or more) categories to a given piece of text from a larger set of possible categories.
- In an email spam identifier, we have two categories: spam and non-spam, and each incoming email is assigned to one of these categories.

Text Classification

- This task of categorizing texts based on some properties has a wide range of applications across diverse domains

Text Classification

- This task of categorizing texts based on some properties has a wide range of applications across diverse domains
- Consider a scenario where we want to classify all reviews for a product into three categories: positive, negative, and neutral.
- The challenge here is to "learn" this categorization from a collection of examples and predict the categories for new, unseen products and new customer reviews.

Different approaches to text classification

- rule based (heuristics, regular expressions etc)
- rule based features+machine learning
- data driven features + machine learning
- neural text representations + machine learning
- deep learning/transfer learning

Rule based classification-1

Hutto, C., & Gilbert, E. (2014, May). Vader: A parsimonious rule-based model for sentiment analysis of social media text. In Proceedings of the international AAAI conference on web and social media (Vol. 8, No. 1, pp. 216-225). (url)
(next few slides quote verbatim from the paper)

Identifying generalizable heuristics for the task

- Vader team analysed a corpus of 400 positive and 400 negative tweets manually (these ratings were obtained by using another tool - details in paper).
- Next, two human experts scrutinized all 800 tweets, and independently scored their sentiment intensity on a scale from –4 to +4.
- They then used qualitative analysis techniques to identify properties and characteristics of the text which affect the perceived sentiment intensity of the text
- This deep qualitative analysis resulted in isolating five generalizable heuristics based on grammatical and syntactical cues.

Vader's Heuristics -1

1. Punctuation, namely the exclamation point (!), increas-es the magnitude of the intensity without modifying the semantic orientation. For example, "The food here is good!!!" is more intense than "The food here is good."
2. Capitalization, specifically using ALL-CAPS to emphasize a sentiment-relevant word in the presence of other non-capitalized words, increases the magnitude of the sentiment intensity without affecting the semantic orientation. For example, "The food here is GREAT!" con-veys more intensity than "The food here is great!"

Vader's Heuristics -2

1. Degree modifiers (also called intensifiers, booster words, or degree adverbs) impact sentiment intensity by either increasing or decreasing the intensity. For ex-ample, "The service here is extremely good" is more in-tensethan "The service here is good", whereas "The service here is marginally good" reduces the intensity
2. The contrastive conjunction "but" signals a shift in sen-timent polarity, with the sentiment of the text following the conjunction being dominant. "The food here is great, but the service is horrible" has mixed sentiment, with the latter half dictating the overall rating.

Vader's Heuristics -3

1. By examining the tri-gram preceding a sentiment-laden lexical feature, we catch nearly 90% of cases where ne-gation flips the polarity of the text. A negated sentence would be "The food here isn'treally all that great.
2. These heuristics go beyond what would normally be captured in a typical bag-of-words model. They incor-porate word-order sensitive relationships between terms:

Peeping into Vader's code-1

```
# #Static methods# #

def negated(input_words, include_nt=True):
    """
    Determine if input contains negation words
    """
    input_words = [str(w).lower() for w in input_words]
    neg_words = []
    neg_words.extend(NEGATE)
    for word in neg_words:
        if word in input_words:
            return True
    if include_nt:
        for word in input_words:
            if "n't" in word:
                return True
    '''if "least" in input_words:
        i = input_words.index("least")
        if i > 0 and input_words[i - 1] != "at":
            return True'''
    return False
```

Peeping into Vader's code-2

```
def allcap_differential(words):
    """
    Check whether just some words in the input are ALL CAPS
    :param list words: The words to inspect
    :returns: `True` if some but not all items in `words` are ALL CAPS
    """
    is_different = False
    allcap_words = 0
    for word in words:
        if word.isupper():
            allcap_words += 1
    cap_differential = len(words) - allcap_words
    if 0 < cap_differential < len(words):
        is_different = True
    return is_different
```

Peeping into Vader's code-2

```
def scalar_inc_dec(word, valence, is_cap_diff):
    """
    Check if the preceding words increase, decrease, or negate/nullify the
    valence
    """
    scalar = 0.0
    word_lower = word.lower()
    if word_lower in BOOSTER_DICT:
        scalar = BOOSTER_DICT[word_lower]
        if valence < 0:
            scalar *= -1
        # check if booster/dampener word is in ALLCAPS (while others aren't)
        if word.isupper() and is_cap_diff:
            if valence > 0:
                scalar += C_INCR
            else:
                scalar -= C_INCR
    return scalar
```

Vader - I will stop here

- Vader has functions like this for various rules it implements.
- ..and more functions for combining different rules to make a final sentiment/polarity prediction
- All packed into a single python file!

Rule based classification on a large scale

Markov, I. L., Liu, J., Vagner, A. (2021). Regular Expressions for Fast-response COVID-19 Text Classification. arXiv preprint arXiv:2102.09507.

Real world rule based NLP

- A recent (not peer-reviewed yet) report described how Facebook used regular expressions to determine whether a post is about COVID-19.
- They built two sets of regular expressions: (1) for 66 languages, with 99% precision and recall >50%, (2) for the 11 most common languages, with precision >90% and recall >90%.
- Comparisons to a DNN classifier show explainable results, higher precision and recall, and less overfitting.

Sounds great! Why can't we just do it always?

- The approach of using regular expressions to identify COVID-19 and related posts seems like a straight forward one for text classification.
- Vader's sentiment analyser in a single file seems like a good one too.
- Why can't we just follow that process for most problems?

Building such regular expressions is non-trivial


```
([ck][ao0]?r[ao0]n[ao](?!(ry|do|[ct]tion|\\W{0,3}
beer|dal|\\W*queens|\\W{0,2}(ca|california)
```

accounting for word order.

```
([ck][ao0]?r[ao0]n[ao]|c[o0]vid).{0,80}?(v[ai]i?r[aou]s|flu)
|(v[ai]i?r[aou]s|flu).{0,80}?([ck][ao0]?r[ao0]n[ao]|c[o0]vid)
```

Each time you create a new expression, you miss a few past matches.

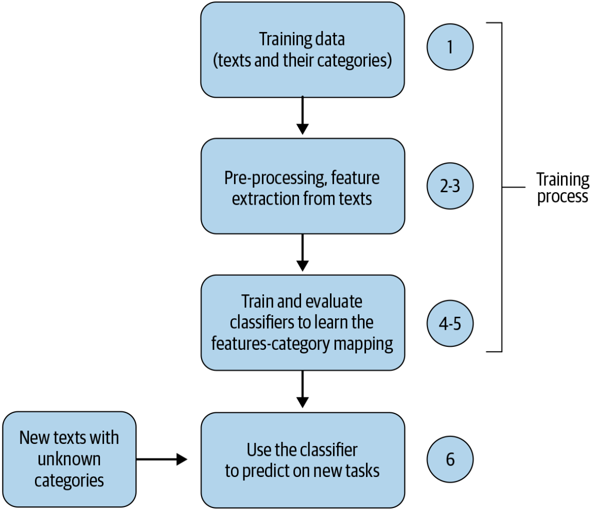
2690 new matches	230 lost matches
corona news	corona beer
corona news 24	koronadal news update
zoom news corona live	brigada news koronadal update
#coronacrisis	convit19
zoom news live corona	corona beer virus joke
korona news	corona beer virus
corona news bangladesh	koronadal updates
nepal corona news	corona beer virus funny
#coronanews	sharp coronado hospital
corona news nepal	coronation hospital
...	...

Table 1: Evaluating a change to COVID-19 regexes.

Rule Based NLP: Comments

- Rule based NLP still has its place in current day tech.
- Yet, it is also non-trivial to build.
- Sometimes, it may even be impossible to create such rules that cover all possibilities (e.g., machine translation, summarization etc)
- Thus, rule based NLP is used only in specific cases, or as a supplement to existing ML/DL based approaches now.

Text Classification Pipeline



source: Practical NLP book

Text Classification Pipeline

One typically follows these steps when building a text classification system:

1. Collect or create a labeled dataset suitable for the task.
2. Split the dataset into two (training and test) or three parts: training, validation (i.e., development), and test sets, then decide on evaluation metric(s).
3. Transform raw text into feature vectors.
4. Train a classifier using the feature vectors and the corresponding labels from the training set.
5. Using the evaluation metric(s) from Step 2, benchmark the model performance on the test set.
6. Deploy the model to serve the real-world use case and monitor its performance.

source: Practical NLP book

Some commonly used features in text classification

- ngrams (word, character, POS, mixed representations)
- neural embeddings (word, character, sentence, document embeddings)
- specific hand-crafted features: e.g., number of spelling errors, number of dependent clauses per clause, number of preposition phrases per sentence etc.
- feature representation: binary (presence or absence), count (number of occurrences), ratios etc.

hand-crafted features

- Sometimes, a text is entirely represented in terms of linguistically meaningful features, instead of embeddings or bag of words etc.

hand-crafted features

- Sometimes, a text is entirely represented in terms of linguistically meaningful features, instead of embeddings or bag of words etc.
- This is useful when you want to understand which features carry more weight in making predictions
- Potentially understand why a prediction is X but not Y etc.

hand-crafted features

- Sometimes, a text is entirely represented in terms of linguistically meaningful features, instead of embeddings or bag of words etc.
- This is useful when you want to understand which features carry more weight in making predictions
- Potentially understand why a prediction is X but not Y etc.
- LingFeat is a recent effort to build a comprehensive library of various kinds of linguistic features used in NLP research.

https://github.com/brucewlee/lingfeat

LingFeat

```
each method returns a dictionary of the corresponding features
"""
# Advanced Semantic (AdSem) Features
WoKF = LingFeat.WoKF_() # Wikipedia Knowledge Features
WBKF = LingFeat.WBKF_() # WeeBit Corpus Knowledge Features
OSKF = LingFeat.OSKF_() # OneStopEng Corpus Knowledge Features

# Discourse (Disco) Features
EnDF = LingFeat.EnDF_() # Entity Density Features
EnGF = LingFeat.EnGF_() # Entity Grid Features

# Syntactic (Synta) Features
PhrF = LingFeat.PhrF_() # Noun/Verb/Adj/Adv/... Phrasal Features
TrSF = LingFeat.TrSF_() # (Parse) Tree Structural Features
POSF = LingFeat.POSF_() # Noun/Verb/Adj/Adv/... Part-of-Speech Features

# Lexico Semantic (LxSem) Features
TTRF = LingFeat.TTRF_() # Type Token Ratio Features
VarF = LingFeat.VarF_() # Noun/Verb/Adj/Adv Variation Features
PsyF = LingFeat.PsyF_() # Psycholinguistic Difficulty of Words (AoA Kuperman)
WoLF = LingFeat.WorF_() # Word Familiarity from Frequency Count (SubtlexUS)

# Shallow Traditional (ShTra) Features
ShaF = LingFeat.ShaF_() # Shallow Features (e.g. avg number of tokens)
TraF = LingFeat.TraF_() # Traditional Formulas
```

note: I did not use this library.

Okay, we know how to get some feature representation for a text. Now What?

Some commonly used machine learning algorithms

- Naive bayes classifier
- K-nearest neighbors classifier
- Logistic regression
- Decision trees
- Random forests
- neural network classifiers

.. etc.

Note: I will only give an overview of how some of these work. Look for a machine learning textbook for more details.

Recommended Readings

1. Machine Learning in Action, from Manning publications.
2. Deep learning with python, from Manning publications

- I found these relatively accessible for people without a lot of mathematical/statistical background.

Naive Bayes Classifier

- Let us say I have a collection of emails (E1, E2 ... En). My problem is to classify them as spam or non-spam.
- Let us assume I already have some training data of 1000 emails labeled as Spam, 1000 labeled non-spam.
- Bayes classifier solves the text classification problem using bayes rule. For some email E1
  P(spam|E1) = P(spam)*P(E1|spam)/P(E1)
  P(non-spam|E1) =
  P(non-spam)*P(E1|non-spam)/P(E1)
- if first probability is higher than second, the email is spam. Else, it is non-spam.
- Since this is a comparison, we can ignore the denominator.

Naive Bayes - continued

Let us take individual terms:

- P(spam), P(non-spam): prior probability of seeing a spam or non-spam message. If your training data has 400 spam and 100 non-spam messages, what are P(spam) and P(non-spam)?

Naive Bayes - continued

Let us take individual terms:

- P(spam), P(non-spam): prior probability of seeing a spam or non-spam message. If your training data has 400 spam and 100 non-spam messages, what are P(spam) and P(non-spam)?
- P(E1|spam),P(E1|non-spam): likelihood that the email is actually spam or non-spam based on our training data. How do we get this?
- If we take a "bag of words" approach, and consider each word as a feature, each unique word in the email becomes a feature.
- If an email has only two words: "my mail", $P(E1|spam) = P(my|spam)*P(mail|spam)$. $P(E1|non\text{-}spam) = P(my|non\text{-}spam)*P(mail|non\text{-}spam)$.

Naive Bayes - continued

Let us take individual terms:

- P(spam), P(non-spam): prior probability of seeing a spam or non-spam message. If your training data has 400 spam and 100 non-spam messages, what are P(spam) and P(non-spam)?
- P(E1|spam),P(E1|non-spam): likelihood that the email is actually spam or non-spam based on our training data. How do we get this?
- If we take a "bag of words" approach, and consider each word as a feature, each unique word in the email becomes a feature.
- If an email has only two words: "my mail", P(E1|spam) = P(my|spam)*P(mail|spam). P(E1|non-spam) = P(my|non-spam)*P(mail|non-spam).
- If an email has 100 words, P(E1|spam) and P(E1|non-spam) are products of 100 conditional probabilities. You assign E1 to spam if P(E1|spam) is higher than P(E1|non-spam) and vice-versa.

Naive Bayes - conclusion

- Assumption: Each feature is independent of the other.
- There is no in-built way to account for inter-correlation between features
- So, this assumption does not really tell the whole story about what is happening. But it works for predictive modeling!

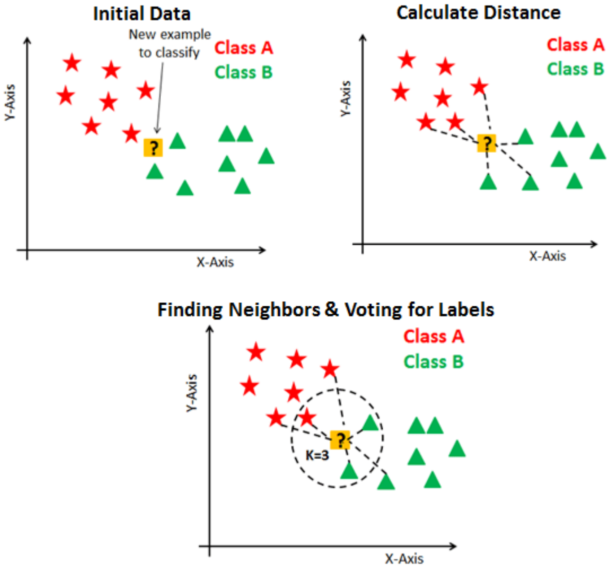
k-NN classifier

- Idea: A document belongs to the majority category among its k-neighbors.

k-NN classifier

- Idea: A document belongs to the majority category among its k-neighbors.
- Let us say my classification problem is: classifying movie reviews into three groups - positive, negative, neutral.
- My training data: say 500 examples for each of these categories.
- Let us say I am using only two features: Use of positive adjectives, Use of negative adjectives
- If I say my k is 5, when I have to classify a new review, and 3 of its neighbors on this feature space have category "positive", 1 has "negative", 1 has "neutral", I will choose "positive" as the category for this new review, because majority of my k neighbors have "positive".
- What is neighborhood? - any measure of distance.

k-NN classifier - 2D example



Source

kNN - conclusion

- Also called "instance based classifier" or "lazy learner"
- Does not really have a "model" or "function". All computation of near-ness or far-ness happens during actual classification
- If you have large amounts of training data, and large feature set, this will become extremely slow.
- selecting k is heuristic.
- relationship between features is till not considered. Features are considered independent of each other.

Logistic Regression

- Goal: same as any other classification algorithm. Classify a given text into one of the pre-defined categories, based on some feature representation.
- Difference compared to naive bayes or knn: learning function.
- Learning function in Logistic Regression:
  1. If x is my text, f_1, f_2... f_i is my feature vector for this text, C = c1, c2, c3 are my three possible categories,

Logistic Regression

- Goal: same as any other classification algorithm. Classify a given text into one of the pre-defined categories, based on some feature representation.
- Difference compared to naive bayes or knn: learning function.
- Learning function in Logistic Regression:
  1. If x is my text, f_1, f_2... f_i is my feature vector for this text, C = c1, c2, c3 are my three possible categories,
  2. for a class c,
     $p(c|x) = \frac{exp(\sum_{i=1}^{n}(w_i * f_i(c,x))}{\sum_{c' \in C} exp(\sum_{i=1}^{n}(w_i * f_i(c',x))}$
  3. The class with the maximum probability in will be the predicted class. Since it is a comparison, again, we can ignore denominator.

Logistic Regression

- Note: You don't have to struggle with the math. There are ready to use implementations you can use if you want.
- Check Chapter 5 in Jurafksy & Martin for a detailed discussion on Logistic Regression
- (personal experience): At one point, I lead a project to deploy a comment moderator for "The Globe & Mail", Canada's largest news paper (2019 March). This was based on Bag of n-gram features + Logistic Regression!

Learning algorithms: Summary

- There are several other learning algorithms
- I gave a very superficial overview in this and previous classes.
- They are different in the way the learning functions are, what their loss functions are etc.
- So, it is always a good idea to compare and experiment before choosing one approach.

Measuring Success of your classification approach

Multiple ways. Depends on the nature of your dataset, and your application. Here are a few common measures:

- Prediction accuracy on test set: commonly used
- False positive rate (Type 1 Error), False negatives (Type 2 error)
- Precision (TP/(TP+FP)), Recall (TP/(TP+FN)), F-score (2PR/(P+R))
- Confusion matrices, to understand what sort of errors occur

... ...

Questions?

Let us take a short 5 minute break before moving into some code discussion.

an End to End walkthrough

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 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 classification
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/practical-nlp/practical-nlp-code/blob/master/Ch4/01_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/practical-nlp/practical-nlp-code/blob/master/Ch4/01_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]
```

Using embedding representations for text classification - examples

Source: Chapter 4 of Practical NLP book (its Github repo, that is)

Using word2vec

```
# Creating a feature vector by averaging all embeddings for all sentences
def embedding_feats(list_of_lists):
    DIMENSION = 300
    zero_vector = np.zeros(DIMENSION)
    feats = []
    for tokens in list_of_lists:
        feat_for_this =  np.zeros(DIMENSION)
        count_for_this = 0
        for token in tokens:
            if token in w2v_model:
                feat_for_this += w2v_model[token]
                count_for_this +=1
        feats.append(feat_for_this/count_for_this)
    return feats


train_vectors = embedding_feats(texts_processed)
print(len(train_vectors))
```

```
3000
```

(full code)

Then what?

- Once you extracted these features, it is the same as any other classification example we saw in earlier classes.
- Note: In this example, we used gensim (in the textbook).
- But the same representation can be obtained with a one liner in spacy (without having to write the document level aggregation code).

using FastText

```
from fasttext import train_supervised
model = train_supervised(input=train_file, label="__class__",
                         lr=1.0, epoch=75, loss='ova',
                         wordNgrams=2, dim=200, thread=2,
                         verbose=100)

results = model.test(test_file,k=1)
print(f"Test Samples: {results[0]} Precision : {results[1]*100:2.4f} Recall: {results[2]*100:2.4f}")
```

(Full code)

What's different?

- fasttext library itself natively supports classification task.
- It is also blazing fast. So, for larger datasets, where something like logistic regression may take forever to train, fasttext trains within a minute.

using BERT: no fine tuning

```
from transformers import AutoTokenizer, AutoModel, pipeline
model = "bert-base-uncased"
#all models at: https://huggingface.co/models
model = AutoModel.from_pretrained(modelpath)
tokenizer = AutoTokenizer.from_pretrained(modelpath)
nlp = pipeline('feature-extraction')
sample_text = "This is a sample sentence"
feat_vector = nlp(sample_text)[0][0]
```

- do this for all your training data, and use it with some classifier!

BERT: fine-tuning

```
from transformers import DistilBertForSequenceClassification, Trainer, TrainingArguments

training_args = TrainingArguments(
    output_dir='./results',          # output directory
    num_train_epochs=3,              # total number of training epochs
    per_device_train_batch_size=16,  # batch size per device during training
    per_device_eval_batch_size=64,   # batch size for evaluation
    warmup_steps=500,                # number of warmup steps for learning rate scheduler
    weight_decay=0.01,               # strength of weight decay
    logging_dir='./logs',            # directory for storing logs
    logging_steps=10,
)

model = DistilBertForSequenceClassification.from_pretrained("distilbert-base-uncased")

trainer = Trainer(
    model=model,                         # the instantiated 🤗 Transformers model to be trained
    args=training_args,                  # training arguments, defined above
    train_dataset=train_dataset,         # training dataset
    eval_dataset=val_dataset             # evaluation dataset
)

trainer.train()
```

source: https://huggingface.co/transformers/custom_datasets.html

Resources

https://huggingface.co/docs/transformers/notebooks - this has links to fine-tuning for all sorts of NLP tasks using transformers python library.

Text classification: Comments

- There is no single best model. We have to experiment with multiple options and pick the best one.
- There is no single training set or test set. They will also evolve with time.

Text classification: Comments

- There is no single best model. We have to experiment with multiple options and pick the best one.
- There is no single training set or test set. They will also evolve with time.
- NLP tools we use in our pipeline are not perfect. Even a simple thing as text extraction or tokenization can have many unresolved issues. While our models are all very valuable effort, these steps are, too.

Text classification: Comments

- There is no single best model. We have to experiment with multiple options and pick the best one.
- There is no single training set or test set. They will also evolve with time.
- NLP tools we use in our pipeline are not perfect. Even a simple thing as text extraction or tokenization can have many unresolved issues. While our models are all very valuable effort, these steps are, too.
- No model can solve the problem of data quality. So, focus on getting good quality data to solve your problem first.

Text classification: Comments

- There is no single best model. We have to experiment with multiple options and pick the best one.
- There is no single training set or test set. They will also evolve with time.
- NLP tools we use in our pipeline are not perfect. Even a simple thing as text extraction or tokenization can have many unresolved issues. While our models are all very valuable effort, these steps are, too.
- No model can solve the problem of data quality. So, focus on getting good quality data to solve your problem first.
- Build a solution incrementally. Don't jump into the most complex solution first. Eventually, you want your stuff to be reliable, and not too expensive to maintain in short/long term.

… …

Questions?

A 10minute break?

Topic Modeling

Goals

- Give a general overview of topic models and their applications

note: since I don't have much experience with topic models other than classroom instruction, I am giving you relevant resources where possible. useful resources:

- https://mimno.infosci.cornell.edu/
- http://www.cs.columbia.edu/~blei/topicmodeling.html
- https://github.com/MaartenGr/BERTopic

What is topic modeling?

- Topic Models are a group of algorithms which attempt to discover latent themes in large collections of documents.
- They use statistical methods to analyze word usage in the texts to discover what "themes" run through them, how these themes connect to each other etc.

What is topic modeling?

- Topic Models are a group of algorithms which attempt to discover latent themes in large collections of documents.
- They use statistical methods to analyze word usage in the texts to discover what "themes" run through them, how these themes connect to each other etc.
- Good thing about them: they do not expect us to provide any prior annotations/categories for texts. Topics will "emerge" from the analysis.
- Bad thing: "Topics" need not necessarily be meaningful, unless you know how to tweak the models

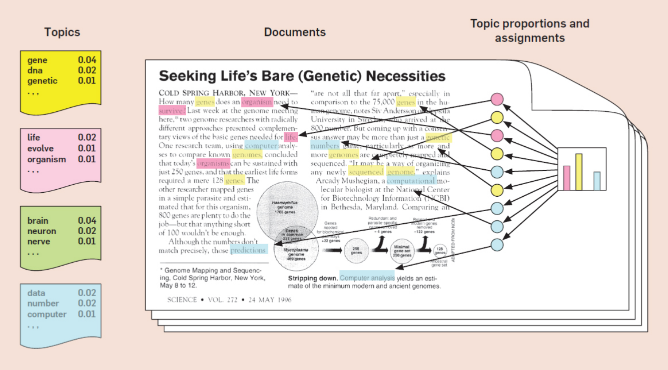
What is topic modeling?

- Topic Models are a group of algorithms which attempt to discover latent themes in large collections of documents.
- They use statistical methods to analyze word usage in the texts to discover what "themes" run through them, how these themes connect to each other etc.
- Good thing about them: they do not expect us to provide any prior annotations/categories for texts. Topics will "emerge" from the analysis.
- Bad thing: "Topics" need not necessarily be meaningful, unless you know how to tweak the models
- They seem to be the most popular method for analyzing unstructured text data in Economics

Latent Dirichlet Allocation (LDA)

- LDA is the most known topic modeling algorithm
- Intuitions:
  - each document is a mixture of multiple topics
  - each topic can be characterized by some set of keywords related to that topic.
  - a keyword can exist in multiple topics with different degrees of importance.

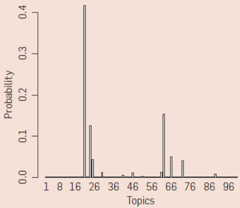
What does a Topic Model do?-1



source: https://goo.gl/azc7Gc

What does a Topic Model do? -2

Real inference with LDA - topic model built using 17000 articles from Science journal.



"Genetics"	"Evolution"	"Disease"	"Computers"
human	evolution	disease	computer
genome	evolutionary	host	models
dna	species	bacteria	information
genetic	organisms	diseases	data
genes	life	resistance	computers
sequence	origin	bacterial	system
gene	biology	new	network
molecular	groups	strains	systems
sequencing	phylogenetic	control	model
map	living	infectious	parallel
information	diversity	malaria	methods
genetics	group	parasite	networks
mapping	new	parasites	software
project	two	united	new
sequences	common	tuberculosis	simulations

source: https://goo.gl/azc7Gc

How are topic models useful? -2

Analysing topics over time

source: https://goo.gl/azc7Gc

How are topic models useful? -3

Analyzing topics by author

TOPIC 10	
WORD	**PROB.**
SPEECH	0.1134
RECOGNITION	0.0349
WORD	0.0295
SPEAKER	0.0227
ACOUSTIC	0.0205
RATE	0.0134
SPOKEN	0.0132
SOUND	0.0127
TRAINING	0.0104
MUSIC	0.0102
AUTHOR	**PROB.**
Waibel_A	0.0156
Gauvain_J	0.0133
Lamel_L	0.0128
Woodland_P	0.0124
Ney_H	0.0080
Hansen_J	0.0078
Renals_S	0.0072
Noth_E	0.0071
Boves_L	0.0070
Young_S	0.0069

TOPIC 209	
WORD	**PROB.**
PROBABILISTIC	0.0778
BAYESIAN	0.0671
PROBABILITY	0.0532
CARLO	0.0309
MONTE	0.0308
DISTRIBUTION	0.0257
INFERENCE	0.0253
PROBABILITIES	0.0253
CONDITIONAL	0.0229
PRIOR	0.0219
AUTHOR	**PROB.**
Friedman_N	0.0094
Heckerman_D	0.0067
Ghahramani_Z	0.0062
Koller_D	0.0062
Jordan_M	0.0059
Neal_R	0.0055
Raftery_A	0.0054
Lukasiewicz_T	0.0053
Halpern_J	0.0052
Muller_P	0.0048

TOPIC 87	
WORD	**PROB.**
USER	0.2541
INTERFACE	0.1080
USERS	0.0788
INTERFACES	0.0433
GRAPHICAL	0.0392
INTERACTIVE	0.0354
INTERACTION	0.0261
VISUAL	0.0203
DISPLAY	0.0128
MANIPULATION	0.0099
AUTHOR	**PROB.**
Shneiderman_B	0.0060
Rauterberg_M	0.0031
Lavana_H	0.0024
Pentland_A	0.0021
Myers_B	0.0021
Minas_M	0.0021
Burnett_M	0.0021
Winiwarter_W	0.0020
Chang_S	0.0019
Korvemaker_B	0.0019

TOPIC 20	
WORD	**PROB.**
STARS	0.0164
OBSERVATIONS	0.0150
SOLAR	0.0150
MAGNETIC	0.0145
RAY	0.0144
EMISSION	0.0134
GALAXIES	0.0124
OBSERVED	0.0108
SUBJECT	0.0101
STAR	0.0087
AUTHOR	**PROB.**
Linsky_J	0.0143
Falcke_H	0.0131
Mursula_K	0.0089
Butler_R	0.0083
Bjorkman_K	0.0078
Knapp_G	0.0067
Kundu_M	0.0063
Christensen-J	0.0059
Cranmer_S	0.0055
Nagar_N	0.0050

Figure 3: An illustration of 4 topics from a 300-topic solution for the CiteSeer collection. Each topic is shown with the 10 words and authors that have the highest probability conditioned on that topic.

source:
https://mimno.infosci.cornell.edu/info6150/readings/398.pdf

How are topic models useful? -4

Picking up similar documents

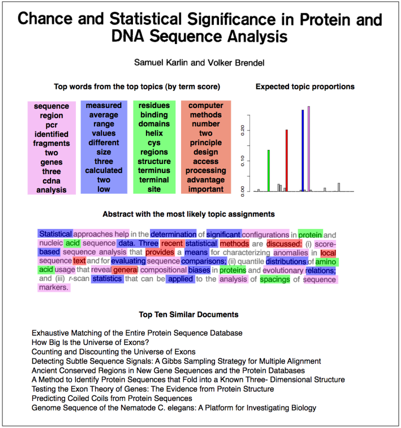


FIGURE 4. The analysis of a document from *Science*. Document similarity was computed using Eq. (4); topic words were computed using Eq. (3).

source: http://www.cs.columbia.edu/~blei/papers/BleiLafferty2009.pdf

A small exercise

Interpreting Topic Models

What do you think of these topics (and their 5 most frequent keywords)? If you are asked to evaluate this topic model now, what will you look for?

- Topic 1 : Onion, Cream, Black pepper, Milk, Cinnamon
- Topic 2: Cumin, Coriander, Turmeric, Fenugreek, Lemongrass
- Topic 3: Vanilla, Cream, Almond, Coconut, Oat
- Topic 4: Olive oil, tomato, parmesan cheese, lemon juice, garlic
- Topic 5: soy sauce, scallion, sesame oil, cane molasses, roasted sesame seed
- Topic 6: Milk, pepper, yeast, potato, lemon juice
- Topic 7: Scallion, garlic, ginger, soy bean, pepper
- Topic 8: Pepper, vinegar, onion, tomato, milk

source: https://gist.github.com/inkhorn/9044779/#file-recipe-analysis-r

Some questions to ponder on:

- Coherence among the keywords for a topic (Is some word looking out of place?)
- Are there two topics that perhaps should be one?
- Can we name the topics with what we think is the group?
- Do you think the topic model learnt something about ingredients in this example?

Building Topic models

- gensim is a popular library for topic models in python. https://radimrehurek.com/gensim/
- sklearn also has an implementation of LDA
- bertopic is a recent development, that seems to be getting popular.

Building a topic model online: an exercise

- I was about to write a demo code, and discovered this in browser topic model builder!: https://mimno.infosci.cornell.edu/jsLDA/
- Try it out with some longish document (e.g., a book from gutenberg.org) and see what happens.

Time: 15 minutes

Share your observations

2020-22: *BERT models and topic models

How do we combine recent transformer models with topic modeling? One approach:

1. Use transformer models to get text representation

2020-22: *BERT models and topic models

How do we combine recent transformer models with topic modeling? One approach:

1. Use transformer models to get text representation
2. Perform some kind of dimensionality reduction over this representation to have a low-dimensional vector per document.

2020-22: *BERT models and topic models

How do we combine recent transformer models with topic modeling? One approach:

1. Use transformer models to get text representation
2. Perform some kind of dimensionality reduction over this representation to have a low-dimensional vector per document.
3. Use a clustering approach to group the documents together

2020-22: *BERT models and topic models

How do we combine recent transformer models with topic modeling? One approach:

1. Use transformer models to get text representation
2. Perform some kind of dimensionality reduction over this representation to have a low-dimensional vector per document.
3. Use a clustering approach to group the documents together
4. get cluster level TF-IDF scores (c-TFIDF) for words and treat those as the topic words for that topic (cluster).
5. (potentially) merge similar topics to reduce the number of generated topics.

Recent topic modeling tools

- Top2Vec
- BerTopic

Evaluating Topic Models

- Topic coherence: normalized pointwise mutual information (NPMI), an automatic measure, shown to correlate with human judgements.
- Topic diversity: the percentage of unique words for all topics
- Visualization of topics
- Human evaluations
- Using topic models in a downstream task and comparing (e.g., as feature representation in text classification)

Note: Topic modeling toolkits generally support the first three

Other topic models

- Should we rebuild each time there is new info? $\rightarrow$Dynamic topic modeling: used to analyze the evolution of topics of a collection of documents over time
- Should I always have a huge collection of documents? $\rightarrow$ Recent research proposes a pre-training+fine-tuning approach for topic models too!

Topic Models: Summary

- Topic models are a good tool to use when we have a large corpus of texts, and no other related annotation.
- Although they are complex mathematical models, they are relatively easier to implement for a not so experienced person.
- However, training/tuning model performance can take time, it can be hard to understand what is a good number of topics, get topics with coherent keywords, etc.

Questions so far?

Tomorrow

What do we do when we don't have labeled data in advance?