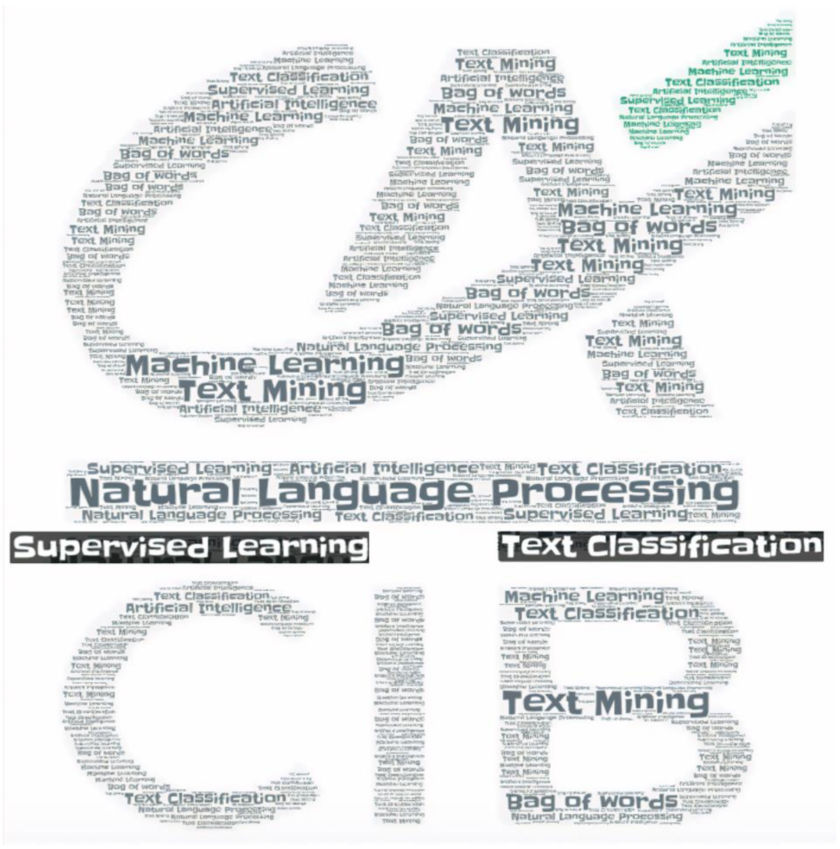


Text analysis and multi-label classification on articles of 5 topics of the Reuters-21578 dataset: a case study of gaining insights into data and enabling language-aware data products with machine learning

Xingyu LIU

01/06/2020

Table of Contents

Introduction

Our task has a two-fold aim in regard to the documents at our disposal. The first objective is to gain some insights into a huge amount of data without a laborious document by document human inspection. This is basically the essence of artificial intelligence via which human tries to achieve a high-level understanding of data (intelligence) with tools provided by statistical abstraction (artificial). In a second time we have been asked to draw inspiration from the aforementioned abstraction process to implement a real-world application: in our case document classification, back-end of some downstream products such as recommender systems. Both aspects pertain to the domain of information retrieval (see (Manning et al., 2008) for a thorough overview).

Reuters-21578 is a benchmark dataset for document classification. To be more precise, it is a multi-class (e.g. there are multiple classes), multi-label (e.g. each document can belong to many classes) dataset.

The dataset used in our report is the Distribution 1.0 of the Reuters-21578 dataset and it has been widely used in text retrieval, machine learning, and other corpus-based research. Originally the articles appeared on the Reuters newswire in 1987 and a decade-long effort of manual cleanup had been necessary before it was finally made available for comparative studies in 1993.

As its name suggests, Reuters-21578 consists of 21578 documents. It's worth noting that although Reuters-21578 is currently the most widely used test collection for text categorization research, more large-scale dataset has been made available to the research community. For instance, the RCV1 dataset (Lewis et al., 2004) has over 800,000 manually categorized newswire stories and the prevalence of Reuters-21578 is likely to be superceded over the next few years.

Our task concerns only 5 topics of this large dataset, namely Money/Foreign Exchange (MONEY-FX), Shipping (SHIP), Interest Rates (INTEREST), Mergers/Acquisitions (ACQ), Earnings and Earnings Forecasts (EARN).

The report is divided into 4 parts:

1) in the first part we briefly describe the process of extracting articles, labels and document ids of the wanted topics from the original sgm files using the beautiful soup package in Python

2) the second part explains the process of data wrangling which, often overlooked by data scientist, consists of a vital step in the whole pipeline of text mining

3) the text analysis part is extensively documented in part 3 where we focus on how the feature engineering can benefit from exploratory data analysis (EDA).

4) much attention has been given to the final part where we consider and evaluate 3 classification algorithms while using Tf-idf vector as a primary feature.

The report ends with a conclusion section where we discuss the most significant findings of our work and how the whole pipeline may integrate with other downstream components such as search engine.

The whole structure of our report can also be compactly captured in Figure 1 designed by us with a product-centered principle.

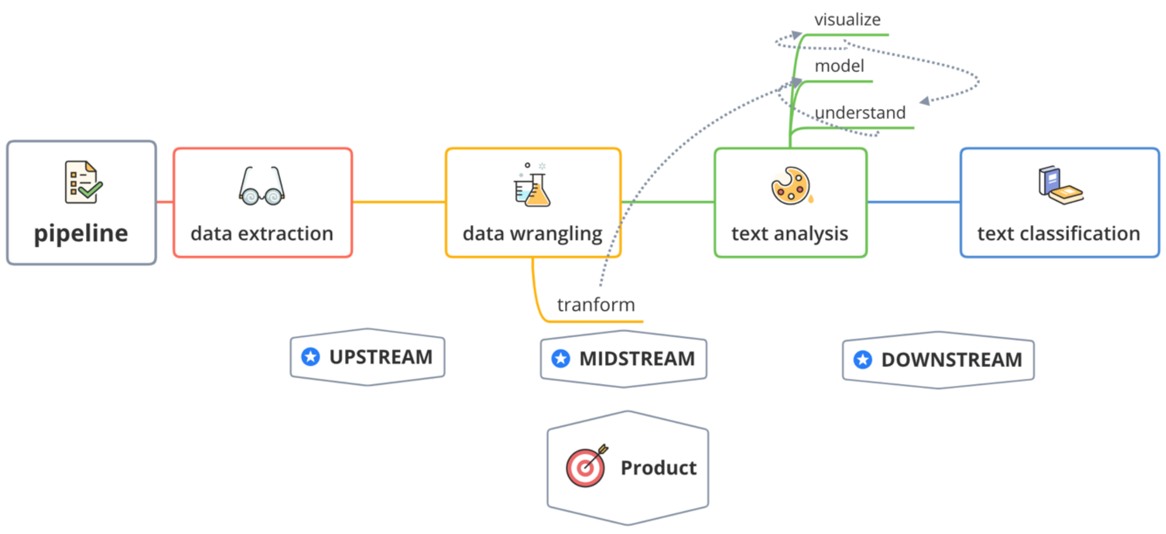


Figure 1 Structure of the report

Part I: Data extraction

Introduction

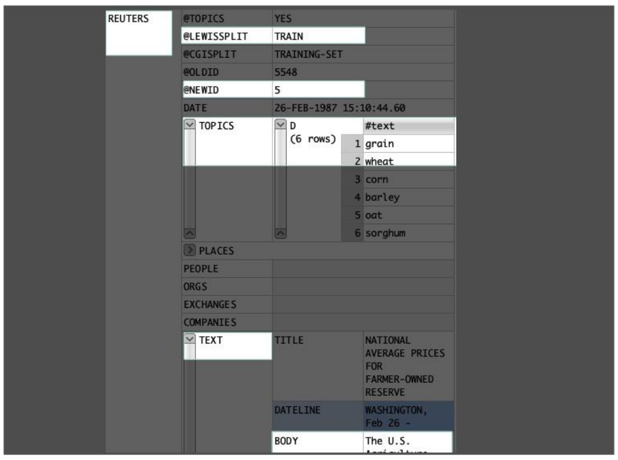
The data set used in this study is the Reuters-21578 test collection tagged with topics stored in 22 SGM files. Markup languages such as SGML are handy for storing and exchanging structured data. For our text classification task however, we want to work with Pandas data frame object as they are more practical. Furthermore, in our case study, we focus on only 5 (see above for details) of 135 different topics.

The script related to this part is *create_df.py*. The output data frame is stored in the file *reuters.csv*.

Parsing

The first step is to parse the text as html with Beautiful Soup package to extract the data we need. To be more precise, we focus on the following tags in SGM files : NEWID as index, LEWISSPLIT as indicator of training/test split in the classification section; all the D tags embedded in TOPICS tag as our document labels (notice that D tags could appear in other tags like PLACES and there would be zero to several D tags in TOPICS tag); BODY tag as article text. The highlighted parts of Figure 2 made with the xml editor *oXygen*© offers a more straightforward display of the desired tags.

An aside on the data extraction: we do not extract TITLE tags because in many cases the little text contained in this tag contains the topic keyword, which will oversimplify the task of classification. For instance, if we want to classify desserts and main course, it would be too easy to build corpus containing these two keywords!

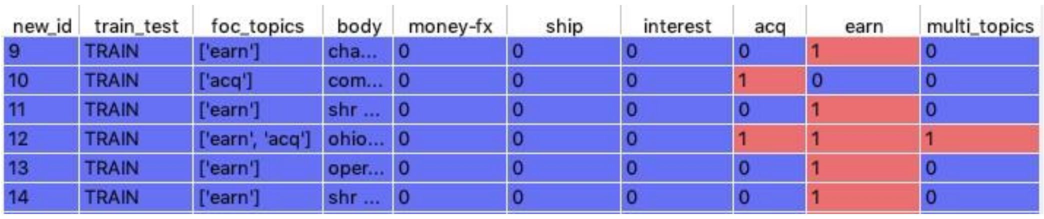


```
REUTERS   @TOPICS       YES
          @LEWISSPLIT   TRAIN
          @CGISPLIT     TRAINING-SET
          @OLDID        5548
          @NEWID        5
          DATE          26-FEB-1987 15:10:44.60
          TOPICS        D          #text
                        (6 rows) 1 grain
                                 2 wheat
                                 3 corn
                                 4 barley
                                 5 oat
                                 6 sorghum
          PLACES
          PEOPLE
          ORGS
          EXCHANGES
          COMPANIES
          TEXT          TITLE      NATIONAL
                                   AVERAGE PRICES
                                   FOR
                                   FARMER-OWNED
                                   RESERVE
                        DATELINE   WASHINGTON,
                                   Feb 26 -
                        BODY       The U.S.
```

Figure 2 Tags used for data extraction displayed with the software oXygen©

Also, items like texts of type *Brief* do not contain any *BODY* tag. Only *TITLE* tags are available and these items will be discarded in our study.

Considering our task only concerns articles of 5 topics, our data frame only contains instances of articles marked at least 1 of 5 topics. A Boolean is used to indicate whether an article is labeled as a specific topic and the column *multi_topics* indicates whether the document has multiple labels. Figure 3 provides a nice illustration of the data frame's structure.



new_id	train_test	foc_topics	body	money-fx	ship	interest	acq	earn	multi_topics
9	TRAIN	['earn']	cha...	0	0	0	0	1	0
10	TRAIN	['acq']	com...	0	0	0	1	0	0
11	TRAIN	['earn']	shr ...	0	0	0	0	1	0
12	TRAIN	['earn', 'acq']	ohio...	0	0	0	1	1	1
13	TRAIN	['earn']	oper...	0	0	0	0	1	0
14	TRAIN	['earn']	shr ...	0	0	0	0	1	0

Figure 3 Data frame generated after data extraction

Data Cleanup and export

The body column contains each article's text and we consider it useful to report some technical details related to its processing.

When trying to read file into a UTF-8 string to parse it later as XML, the following error is encountered (for file reut2-017.sgm):

UnicodeDecodeError: 'utf-8' codec can't decode byte 0xfc in position 1519554: invalid start byte

To solve this problem, we first read the file in binary and iterate over the lines before finally decoding each of them in UTF-8.

As elementary pre-processing we made also the following adjustments:

1) create the function *clean_text* (refer to the code for more details) to lowercase all text
2) restore contractions like *can't* to *can not*
3) transform non-word (symbols and punctuation, *\W* in regular expression) to space
4) eliminate words containing Reuter|REUTER|reuter
5) remove special characters &#3 present at the end of many BODY tags

The output of this section is a csv file (*reuters.csv*) which will be used as input in the classification section.

Part II: Data wrangling

Introduction

The script related to this part is named *textMining1.py* and extensively commented.

In practice, the previous data extraction part is often preceded by a data collection process involving sometimes a web-scraping process. The whole data retrieval part represents the most upstream part of data science. The term upstream in the oil industry means the drilling and pumping of wells. Although this step is essential, the real game changer is nevertheless the data wrangling part where the term “text mining” gains its full interpretation.

The Economist in their 6 May 2017 edition gushes: « Data is the new oil ». The data wrangling consists of the midstream process empowering the mining process.

Our approach is inspired from a basic scenario of a text retrieval engine optimization. Since it’s generally time-consuming to perform a search on every possible document in the database, documents are often pre-labeled as belonging to a certain class and as soon as the user query’s intention is identified related documents can directly be pulled from the database.

In the same way and under a natural language processing setting, it’s often inefficient to do online calculations on documents based on user query. The goal of this part is to create a certain API-like data frame useful for late analysis. The final data frame is saved in reuters_wrangled.csv.

Design of a Reuters dataset "API"

Each row (instance) of our final data frame (*reuter_wrangled* hereafter) represents an article. The list of columns’ names, datatypes and interpretation is summarized in the following table:

new_id	int64	article id
foc_topics	string	article label set

body	string	raw body text for each article
money-fx	int64	1 or 0 (indicate article's label with Boolean)
ship	int64	idem
interest	int64	idem
earn	int64	idem
multi_topics	int64	1 or 0 (indicate if an article has multiple labels)
body_clean	string	body text with cleanup
body_sentences	string	sentences of each article separated by \n
body_token	string	tokens of the article separated by space
body_lemma	string	lemmas of the tokens
body_lemma_noStop	string	lemmas of the token which are non stopwords
body_nouns	string	lemmas of all the nouns
body_adjectives	string	lemmas of all the adjectives
body_verbs	string	lemmas of all the vers
body_people	string	recognized people entities
body_org	string	recognized organization entities
body_quantity	string	recognized quantity entities
no_tokens	float64	number of tokens for each article
no_types	float64	number of types for each article
no_lemmas	float64	number of lemmas for each article
avg_wordLength	float64	average word length for each article
avg_sentLength	float64	average sentence length for each article
lexical_diversity	float64	lexical diversity for each article
lemma_diversity	float64	lemma diversity for each article

Table 1 Structure of the data frame for text analysis task

In the context of an "API" this table can also be considered as a brief documentation of a database for text analysis engineers. As stated above, the advantage of such an approach is quite obvious, all the metadata are already stocked in the data frame and each time one does not need to recompute the desired information over all the documents (for example a list of nouns pertaining to a particular topic) each time he wants to do a specific analysis. We will see the benefits of this approach in part 3 dedicated to text analysis but for now let's consider some technical details related to columns constructions.

Technical details of reuter_wrangled

We use Spacy as the principal NLP package. It's a relatively new library (2015) compared to other tools such as NLTK, TreeTagger and Stanford CoreNLP. The reason for which we use this library is that it's fast and light while offering fairly good performance on tokenization, lemmatization and named-entity recognition (NER), making it an adequate tool for doing exploratory data analysis.

The English models[1] used by Spacy are trained on the OntoNotes Release 5.0 corpus[2]. We use the smallest model (en_core_web_sm) to gain even more speed on our computer without GPU acceleration.

The lexical diversity for each article is calculated by dividing the number of unique tokens by the number of all the tokens. For this calculation we have replaced all the digits (integers and floats) in the texts with digitttt to avoid an inflated lexical diversity.

As the structure of the data frame suggests we have extracted the list of three entities for each article, namely people, organization and quantities (units).

All the texts are lowercased. Stopwords are removed by joining the Spacy's stopwords set and NLTK's stopwords set.
We also use the Spacy's language model to filter stopwords, the main advantage of the model approach is a more fine-grained control filtering of stopwords, which is particularly important

[1] see https://spacy.io/models/en for more information.

[2] see https://catalog.ldc.upenn.edu/LDC2013T19 for more information.

in the context of NER. For example, all the "Will" will be removed by a hard-coded stopwords list. However, a model-based filtering will retain those referring to a person.

The implementation of the whole process is well documented in the "refactor the Reuters Corpus" part of textMining1.py.

Elementary statistics based on reuter_wrangled

Using the generated data frame, it's very straightforward to get subsets of data frame on which further analysis can be carried on. For example, the two following lines slice respectively articles in topic money-fx with or without other topic labels assigned (thanks to the column multi_topics).

```
sub_df_multi_topic = df[(df['money-fx']==1) & (df['multi_topics']==1)]
sub_df_single_topic = df[(df['money-fx']==1) & (df['multi_topics']==0)]
425 articles
259 articles
```

It's also really simple to compute general statistics of the data subset. We report some useful statistics in this section as a prelude to the more interesting text analysis part.

```
print(f'total number of articles: {len(df)}')
Total number of articles: 7175
```

```
df['foc_topics'].nunique()
Number of different combinations across 5 topics in the corpus: 15 combinations
```

```
mul_cat_number = len(df[df['multi_topics'] == 1])
print(f'articles with multiple categories: {mul_cat_number}')
Articles with multiple categories: 689
```

```
focused_topics = ['money-fx', 'ship', 'interest', 'acq', 'earn']
for tp in focused_topics:
   print(f'topic {tp} : {sum(df[tp])}')
Number of articles per category:
```

```
topic money-fx: 684
topic ship: 295
topic interest: 424
topic acq: 2210
topic earn: 3776
```

```
gen_stat =
["no_tokens","no_types","no_lemmas","avg_wordLength","avg_sentLength","lexical_diversity","lemma_diversity"]
for st in gen_stat:
    print(f"mean of {st}: {df[st].mean()}")
```

```
Mean of number of tokens: 111.12641114982578
Mean of number of types: 65.67108013937282
Mean of number of lemmas: 100.12641114982578
Average word length: 4.530594891842289
Average sentence length: 17.043065870307068
```

For reasons of concision we will omit the code for the following statistics, all the implementations being documented in the "*general statistics*" part of *textMining1.py*.

```
Average lexical diversity: 0.5554329744947362
Average lemma diversity: 0.5381094987104804
```

Part III: Text analysis

Introduction

This part is the "meat" of text analysis. The main goal is to explore some possibly useful features for the text classification task of Part IV.

The most common model for representing a document is the so-called bag-of-words model which represents the document as a sequence of words (sometimes n-gram) without considering the relation among the words and the order of each word in the document. This apparently simple model is widely used in text classification.

In this part we start by exploring some other characteristics with our *reuters_wrangled* data frame. Then we will look at which kind of words are most useful for differentiating one topic from another.

It turns out that general lexical statistics fail to differentiate documents of the different 5 topics. Most common words perform relatively well and most common entities perform the best.

The file used in this part is *reuters_wrangled.csv* and the code is saved documented in *textMining2.py*.

General lexical statistics

In this section we will try to answer the following question:

Which of the following statistic differentiate the best articles of different topics?

- Average number of tokens?
- Average word length?
- Average sentence length?
- Diversity score?

We report the statistics of average number of tokens here:

```
average number of tokens of money-fx:
216.58479532163742
average number of tokens of ship:
168.9864406779661
average number of tokens of interest:
194.78537735849056
average number of tokens of acq:
129.0606334841629
average number of tokens of earn:
73.50741525423729
```

It is obvious that different topics have different average number of tokens. However, it is also obvious that this discrepancy is not related to the intrinsic properties of different topics. What about average word length?

```
average word length of money-fx:
4.828710762173928
average word length of ship:
4.847260778651526
average word length of interest:
4.671278803330707
average word length of acq:
4.924477542885329
average word length of earn:
4.21560819394405
```

Except *earn*, no notable difference can be found across topics. Figure 4 confirms our findings.

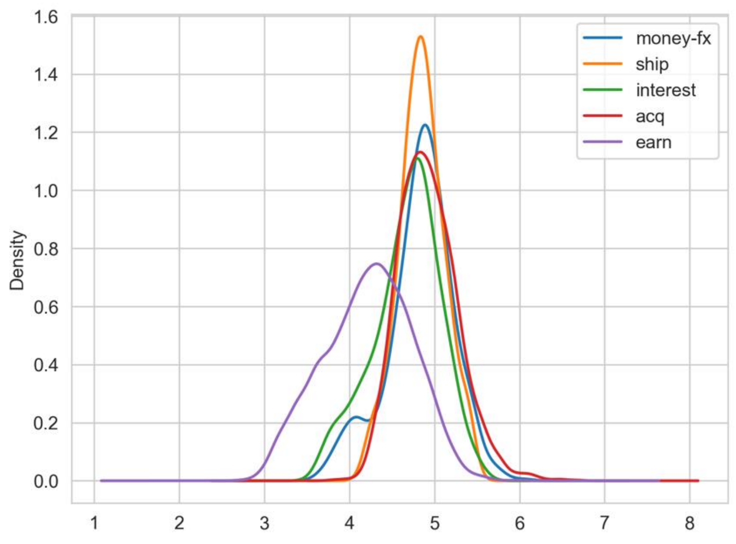


Figure 4 Word length distribution across topics

An investigation into sentence length distribution again reveals the peculiarity of the *earn* subset. However, a distinction across all topics is still impossible with this parameter.

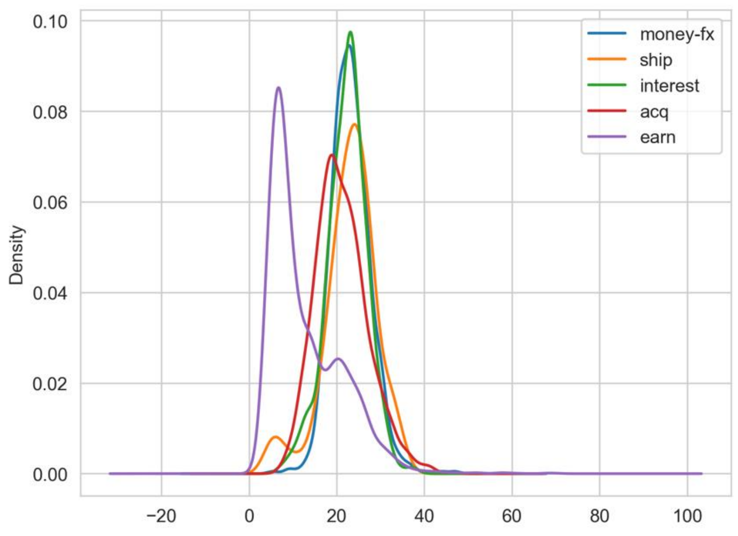


Figure 5 Sentence length distribution across topics

The lexical diversity score reveals again the same pattern. Actually the 3 parameters seem correlated and can be merged into 1 feature.

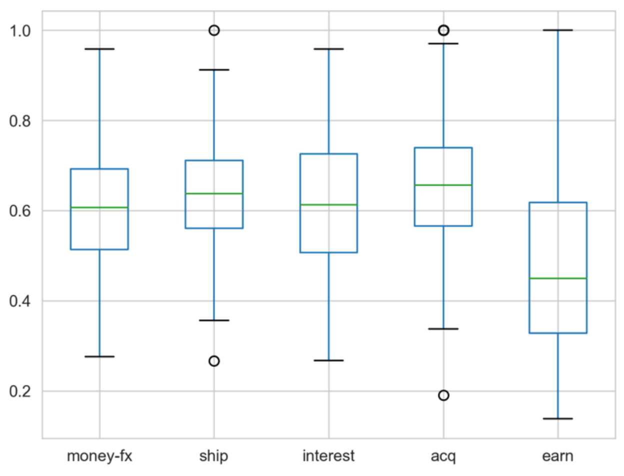


Figure 6 Lexical diversity score across topics

Common 'words' and entities

General lexical statistics seem not satisfying for topics differentiation. This is rather in accordance with the wide use of bag-of-words model in text classification. But the simple statement of using words to differentiate document doesn't answer the question 'which words'. This section explores two possible categories of words: most common words and most common entities.

The following figure shows the 10 most common words in each topic.

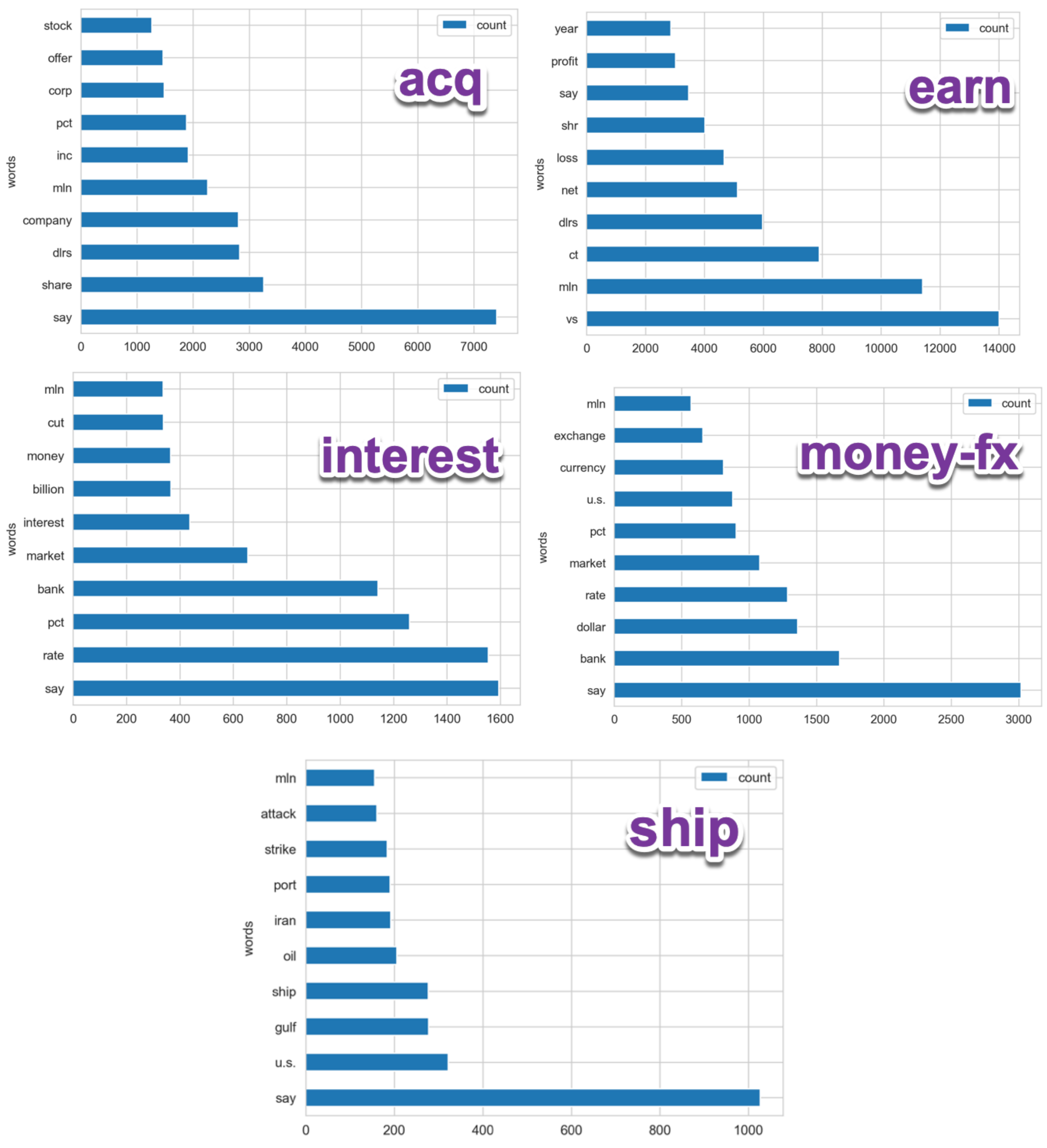


Figure 7 Most common words across topics

It is easy to see that some of the most common words do shed some insights to the document's nature.

Semantics of *gulf, iran and oil* are related to the topic shipping, *bank, currency and exchange* are easily correlated to foreign exchange. This is the same case for all the other topics. In other terms, some of the commonest words are associated semantically with the corresponding topics.

However, it is also important to recognize that abundance of some terms is due to factors unrelated to topics. For example, words like *say, mln (million)* are common in all topics and words like *vs, year* are unlikely to indicate the topic of a document.

So what about most common entities?

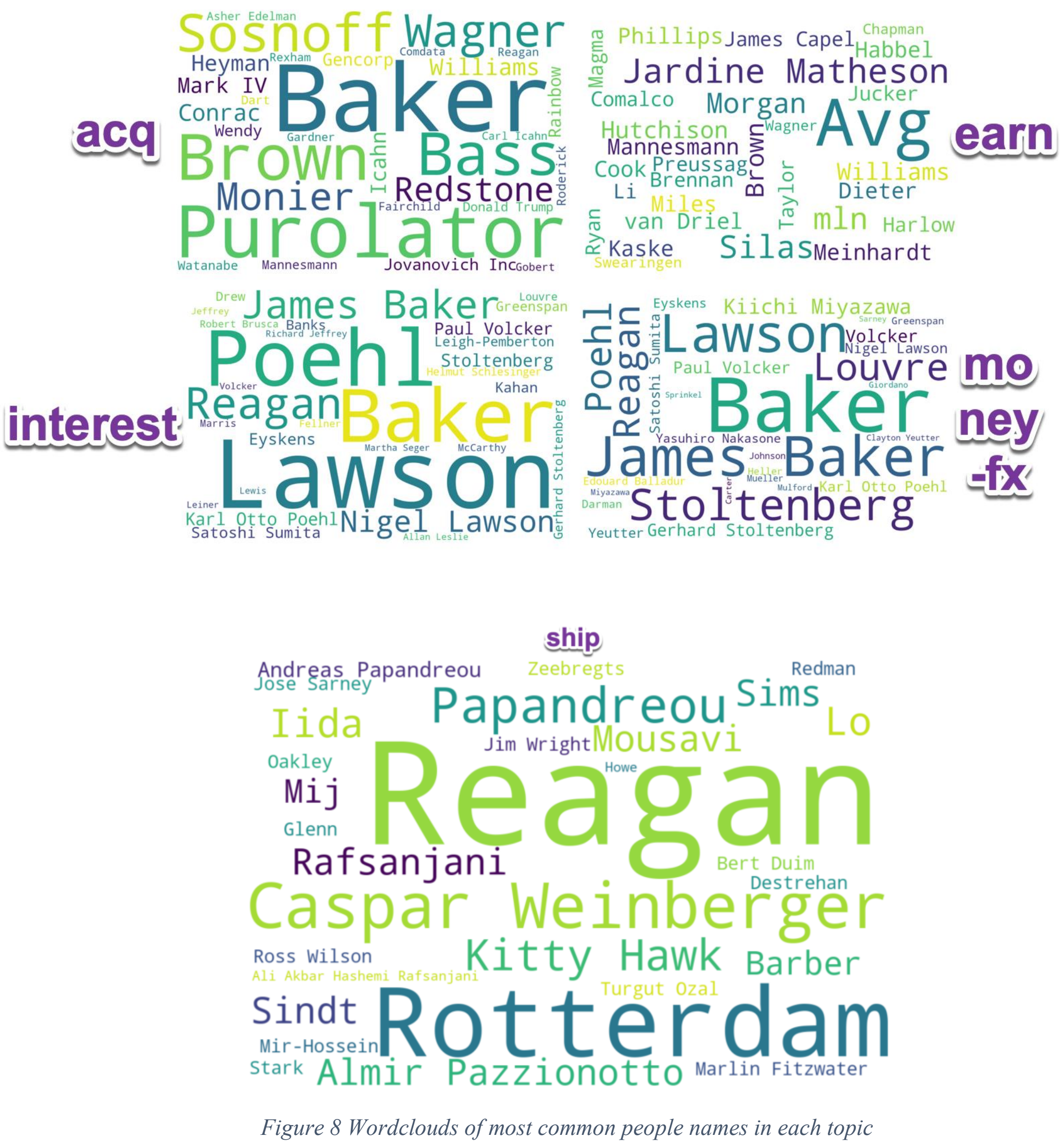


Figure 8 Wordclouds of most common people names in each topic

Figure 8 shows the most common people entities in each topic. Although not apparent for people with no domain-specific knowledge, some entities do indicate the nature of the topic. For instance *Donald Trump* is closely related to documents under the topic acquisition.

This correlation is much more visible if we take other entities, especially quantities.

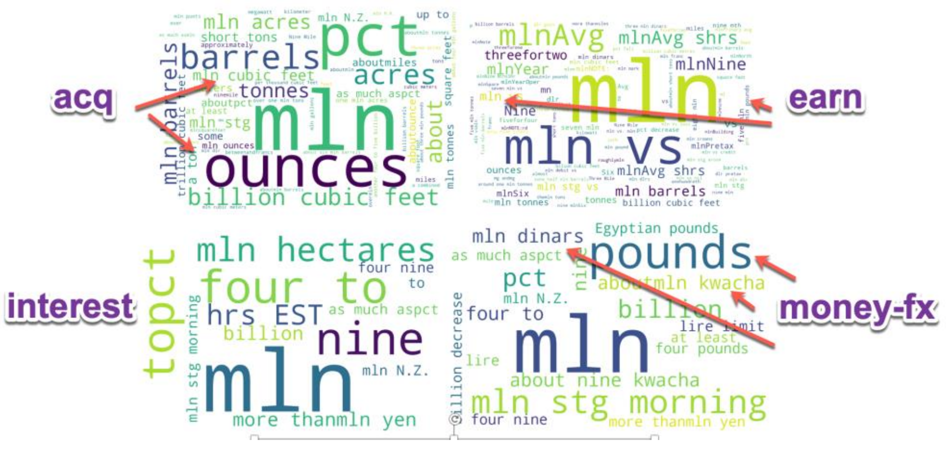


Figure 9 Wordclouds of most common quantities across 4 topics

In summary we show the viability of the bag-of-words model and the insufficiency of other general lexical statistics. However various words, although useful, overlap across topics, suggesting need for some more sophisticated features, which is going to be explained in the part IV.

Part IV: Text classification

Introduction

An Automatic Text Classification task can be implemented through a “rules system”, explicitly defined by a “domain expert”, or by Machine Learning systems.

Machine Learning (ML) is the ideal solution in our case where a large set of previously classified texts is available — a so-called “training corpus”: the corpus is supplied to the ML system, which “learns” autonomously what are the best strategies for classifying documents.
In the case of binary classification, we just ask a yes/no type of question. If there are multiple possible answers and only one to be chosen, then it’s multiclass classification. In our case, we can’t really select only one label, some articles contain multiple topics like [‘money-fx’, ‘interest’]. However, most of widely known algorithms are designed for a single label classification problem. Discriminative multi-class classification techniques, including SVMs, have historically been developed to assign an instance to exactly one of a set of classes that are assumed to be disjoint. In contrast, multi-labeled data, by its very nature, consists of highly correlated and overlapping classes (Godbole & Sarawagi, 2004). In this case study, three models for multi-label classification available in scikit-multilearn library are described for the 5-topic Reuters-21578 News classification:

Method	Model	Description
Problem transformation	Binary Relevance	An ensemble of single-label binary classifiers is trained independently on the original dataset to predict a membership to each class
Problem transformation	Label Powerset	Map each combination of labels into a single label and trains a single label classifier
Problem adaption	MLkNN	For each instance in the test set, its K nearest neighbors in the train set are identified.

Table 2 3 models for multi-label classification available in scikit-multilearn library

Datasets

We use the Mod-Apte split and evaluate all methods on the given train/test split with 5 classes. Our dataset includes 4, 948 documents for training and 1, 977 for test (214 documents with NOT-USED tag are discarded). Split processing is led by values in train_test column in our data frame.

Steps

We firstly import the reuters.csv as data frame.

Before training the classifier, we have to represent and weight every document with respect to the set of textual features. We apply tokenization and stemming to text in column body before which stop words are removed. Furthermore, any string that contains other than letters is removed.

Afterwards, TFIDF Vectorizer is used to create a sparse matrix of weighted words. The TFIDF Vectorizer weights the words or features by importance regarding the data set or corpus. This weight is determined by calculating the frequency of the term, TF, and multiplying it by the "Inverse Document Frequency", or IDF. The "Inverse Document Frequency" is calculated by taking the log of the number of articles divided by the number of articles the word is located in.

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF
Term x within document y

$tf_{x,y}$ = frequency of x in y
df_x = number of documents containing x
N = total number of documents

Figure 10 Formula of Tf-idf computation

This step was taken to discard words that are not relevant to the articles or words that will skew the results because they appear in almost every article. *TFIDF* method is applied to x_train and x_test data.

We have also y_train and y_test with *multi label binarizer* manually created as 5 columns in our data frame.

In order to train the classifier, we use *BinaryRelevance* classification with *an sklearn.svm.SVC* base classifier which supports sparse input in the scikit-learn package; *LabelPowerset* multi-label classifier with *LogisticRegression*; *MLkNN* with a fixed number of neighbors which is 10.

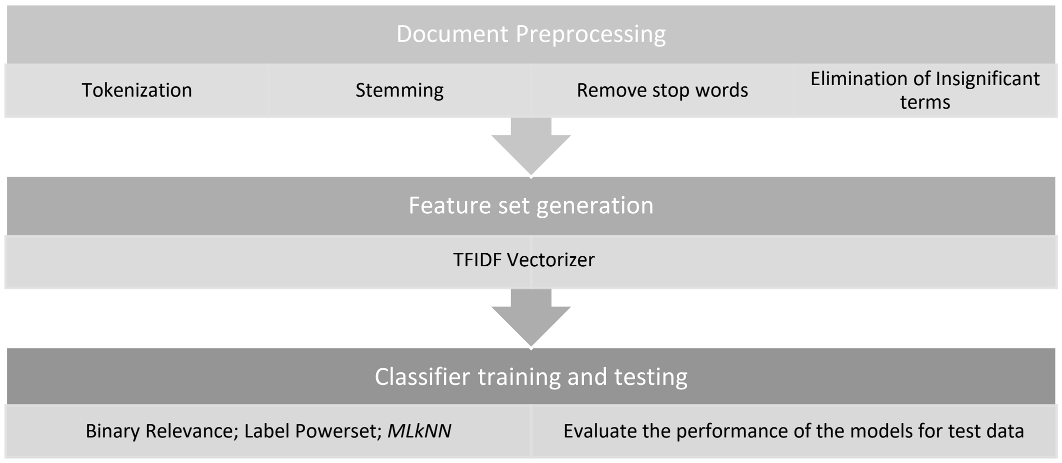

Document Preprocessing
Tokenization
Stemming
Remove stop words
Elimination of Insignificant terms
Feature set generation
TFIDF Vectorizer
Classifier training and testing
Binary Relevance; Label Powerset; *MLkNN*
Evaluate the performance of the models for test data

Figure 11 Pipeline of the classification task

Scores

After applied the training models above, we compare the results. For selecting the best model, we measured:

F1 score which is the harmonic mean of precision and recall,

$$F_1 = \left(\frac{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}}{2}\right)^{-1} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

And Hamming loss which represents the fraction of labels incorrectly predicted:

	F1 score	**Hamming loss**
Br_classifier	0.622	0.112
Lp_classifier	0.768	0.093
Ml_classifier	0.831	0.066

To sum up, popular methods for multilabel classification were described and compared on Reuters newswire. The best results were obtained for *MLkNN model*, by building uses *k-NearestNeighbors* find nearest examples to a test class and uses Bayesian inference to select assigned labels. In future work, the order of labels could be shuffled and tested, which is time consuming process.

Conclusion

In this report we have achieved what we call the pipeline of a full-stack data analyst of text.

We have explored different features (lexical and semantic) useful to represent a document. The validity of the bag-of-words model is assessed using these features.

In the text classification task, we explore 3 different classifiers in order to handle the multi-label classification problem. Two approaches of problem transformation have been adopted. It turns out that the *MLkNN classifier* provides the best F1 score (0.831) with the lowest Hamming loss (0.066). Thus this classifier fits best to the demands of the current classification task.

Needless to say, this work is far from perfect. But the preliminary results obtained by our efforts offer already many possible downstream applications. Particularly this pipeline can serve as a backend to a search engine interface recommending similar documents to a topic identified thanks to some user's input query.

References

Lewis, D. D., Yang, Y., Rose, T. G., & Li, F. (2004). Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, *5*(Apr), 361–397.

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval*. Cambridge university press.