Business News Tagging Engine

#### Springboard Capstone Project 1 James Flint | mail@jamesflint.net | 2017-08-31

## Requirement

Examine articles from a news database and tag them in accordance with a pre-existing heuristic editorial classification system.

## Client

Our client is business, technology and finance news company, [Curation](http://www.curationcorp.com). Curation is currently paying a sizable team of journalists to select, summarise and tag news articles, an operation that is costly and non-scalable. The company would like to automate much of its editorial process, and reserve human intervention for editorial sign-off and original business insight instead of using it to perform relatively low-level text processing tasks. Several components are required in order to solve this problem. The first of these, to be addressed in this project, is to design a system capable of accurate semantic classification.

## Data

Curation has a clean, human-curated database of around 80,000 summarised news articles (JSON format). I have permission to use a subset of this as a training set. For raw source data, I’m using a sample of data from a standard news database (LexisNexus Moreover, format: XML).

## Approach

1. Data Wrangling
2. Finding a Language Model
3. Similarity Queries
4. Topic Modelling
5. Automatic Classification
6. Conclusion & Next Steps
7. Further research

## Elements

* This report
* Executable Python code for the various modules of the project
* A Jupyter notebook demonstrating implementation of the codebase
* A client presentation

# 1. Data Wrangling

## Comparison Data

The test data has been provided by the client as a series of topic-specific JSON files, and needed to be extracted, filtered by content field, normalised and saved as individual JSONs.

The relevant data in the target files are:

* title = article title
* date = original date of publication
* source\_name = name of publisher
* source\_url = URL of publisher
* content = article body content

There are, in total, 43,502 files, with an average of 92.8 words of body content per file.

As this project is focussed on processing and comparison at the level of the article content, the only data that concern us during the first stage project are those contained in the “title” and "content" fields of the training and test JSONs. We combine these into a single field, and keep the “date” field as a possible indexing metric.

For the more complex sub-classification carried out in the second capstone project we may use more of these fields, along with the “tags” field, containing manually added semantic tags.

## Raw Data

The raw data for the project is a one-week slice of news articles in XML format taken from the LexisNexus Moreover feed. The total size of this sample was 128GB, downloaded to an external hard drive for convenience.

Initial examination of this data revealed that each file contains a batch of between 50 and 60 individuals, each in structured XML. The XML contains the Title, Source and Content of each article along with various classification, location and tagging data much of which was relevant to Moreover's internal classification.

To reduce the file size and make the data easier to handle and faster to access, we identified the specific data relevant to the project and wrote a python script to extract them from the XML files (loaded into memory in series), saving them as structured JSONs. JSON was chosen as a destination format because the test data is already in JSON, and JSON will also minimise file size while also allowing ease of access to individual data items in each file.

The relevant data extracted were:

* title = article title
* content = article body content
* publishedDate = original date of publication
* duplicateGroupId = an indicator of whether or not the article has appeared multiple times in the database
* source\_name = name of publisher
* source\_url = URL of publisher
* countryCode = country (useful to indicate whether the article language is English)
* editorialRank = indicator of the trustworthiness of the publisher

These data were passed to a pandas data series, which was in turn converted to JSON without forcing ascii in order to prevent UTF character encoding. This JSON was then exported using a key:value structure specified via json.dumps in order to enforce the correct formatting.

There are 198,821 files in the resulting raw data sample, with an average of 343.7 words of body content per file.

# 2. Finding a Language Model

## Bag of Words (bow)

Once the data wrangling is complete, the JSON files need to be converted into document vectors as a preliminary step in subsequent automatic comparison and classification (i.e. tagging).

To generate the document vectors we will be using the [gensim library](http://radimrehurek.com/gensim/" \t "_blank) which has methods for creating a frequency dictionary from a data sample and then remapping the sample using that frequency dictionary in order to create a sparse document vector, which gensim calls a "corpus".[[1]](#footnote-1)

We have written a script called create\_corpus.py that performs the corpus creation task. Create\_bow\_corpus.py loads the newly created JSONs sequentially and adds each file's "content" field to a python list called "documents", which thus contains the body content of all of the articles in the chosen directory.

Create\_bow\_corpus then calls a method, clean\_text, which strips "documents" of the common words "for", "a", "of", "the", "and", "to", and "in", the possessive "'s", speech marks, line breaks and the html paragraph tag which are not required for text comparison. The next step is carry out a process called "lemmatisation".[[2]](#footnote-2) Clean\_text uses the [Natural Language Toolkit (NLTK)](http://www.nltk.org/" \t "_blank) package to handle this.

Once the cleaned up text has been returned to create\_bow\_corpus it is assembled into a dictionary of individual terms that can be used as a reference point. After parsing the dictionary through a filter to filter out very low and very high frequency words, we're ready to create the corpus itself, which is a dictionary-like representation of the cleaned text known as a ["bag-of-words" (bow)](https://en.wikipedia.org/wiki/Bag-of-words_model" \t "_blank) in which the frequency of occurrence of each word is stored to use as a feature for training a subsequent classifier. Both the dictionary (\_bow\_corpus.dict) and the bag-of-words (\_bow\_corpus.mm & \_bow\_corpus.mm.index) are stored alongside the original JSONs in the source directory.

## Tf-idf

Once the bag-of-words corpuses are in place, we can look at another, more subtle language model: tf-idf[[3]](#footnote-3). Tf-idf stands for "term frequency-inverse document frequency", and is a statistical method used to determine the importance of any given word within a document, a figure that can then be used as a weighting factor when conducting information retrieval or assessment tasks.

Tf-idf is the product of two statistics: term frequency and inverse document frequency. Term frequency is, broadly speaking, the number of times that a term occurs in a document, as generated by the following pseudo-code:

document\_term\_frequencies = {}   
for document in collection:

for word in unique\_words(document):

document\_term\_frequencies[word] += 1

Inverse document frequency is the logarithmically scaled inverse fraction of how often a term appears in all documents in a particular corpus of documents, which in turn is an expression of its importance (its information density) within that corpus.

A high tf-idf indicates a high term frequency in a given document and a low document frequency in the corpus, which means the measure filters out common terms, a characteristic that can be enhanced by removing very common words like conjunctions (known as "stopwords"), and by reducing plurals and verbs to their root or "lemma" (a process known as "lemmatisation") before the tf-idf analysis is applied. As we have already done stopword removal and lemmatisation as part of our bow corpus creation, we can use the bow corpus as a seed for creating our tf-idf vectors.

corpus = corpora.MmCorpus(os.path.join("./data/Topics/", sourcedir,   
 "\_bow\_corpus.mm"))

*# Initialize a tfidf model*

tfidf = models.TfidfModel(corpus)

tfidf.save(os.path.join("./data/Topics/", sourcedir, "\_tfidf\_model.mm"))

## Doc2Vec

Doc2Vec is an unsupervised learning algorithm generalised from the Word2Vec method. Like bow and tf-idf it is a natural language processing (NLP) tool for representing documents as a vector; unlike these it generates vectors for sentences, paragraphs or documents rather than for individual words. The main purpose of Doc2Vec is associating arbitrary documents with labels, rather than words with other words, so labels are required as part of the training set.

The first step in applying Doc2Vec is coming up with a vector that represents the “meaning” of a document, which can then be used as input to a supervised machine-learning algorithm to associate training documents with the labels provided. The result of this process, also known as **feature extraction**, can be used to associate unlabelled test documents with the same set of labels.

The python script **doc2vec\_model.py** contains a number of methods that we’ll need to apply to our data in order to do this. As with ti-idf we first need to create an appropriate corpus, and as with tf-idf we can use our bow corpus as a seed. The **doc2vec\_model** methods **doc2vec\_train\_corpus()** and **doc2vec\_test\_corpus()** do this, the difference being that the first creates a set of tokens and a set of tags for subsequent comparison, while the second is applied to unlabelled test data, and so creates no tags. Both methods use a gensim library pre-processing method to optimise the submitted text.

To train the model, first we need to associate a tag/number with each document of the training corpus, which we do in **doc2vec\_model.create\_doc2vec\_corpus()**. Here, we're just going to use the ordinal number of the document in the original database (which is also included in its filename). Note that the testing corpus is just a list of lists, and contains no tags.

The next step is to instantiate a doc2vec model with a vector size of 50 words, iterated over the training corpus 10 times. We've set the minimum word count to 2 in order to give higher frequency words more weighting. Model accuracy can be improved by increasing the number of iterations but this generally increases the training time.

We also need to build a vocabulary - a dictionary of all the unique words extracted from the training corpus, along with a frequency count of those words. The vocabulary is subsequently accessible via model.vocab.

Once these elements are in place, we're ready to train our model. All these three steps are handled by our **doc2vec\_model.train\_doc2vec\_model()** method:

def train\_doc2vec\_model(train\_corpus):

# instantiate a Doc2Vec model with a vector size of 50 words

model = gensim.models.doc2vec.Doc2Vec(size=50, min\_count=2, iter=10)

# build a vocabulary i.e. a dictionary of all the unique words

model.build\_vocab(train\_corpus)

# train the model

model.train(train\_corpus, total\_examples=model.corpus\_count,   
 epochs=model.iter)

Once we have a trained model we can infer a vector for any piece of text by passing a list of words to the **model.infer\_vector** function. This vector can then be compared with other vectors using using cosine similarity – something we will discuss in more detail in the next section.

# 3. Similarity Queries

How do we find out if a new document is similar to one of the vectors in our corpuses? One way – a commonly used technique that works well – is to find the angle between the two vectors considered as unit vectors on a unit circle.

*similarity(****a****,* ***b****) = cosine of angle between the two vectors*

This calculation can be done either for the vector of an individual word position in the two vectors, or for the set of the word positions in the vectors considered as a whole. In the latter case, we'd just take the sum of the dot product of each of the corresponding terms in each vector, divided by the number of terms. That's fine if the original documents being compared are the same length (as they will be if they’ve been transformed using tf-idf); in case they are not we can compensate by using a process called "normalisation", which divides the sum of the dot products, not by the number of terms in the vector, but by the square root of the sum of the squares of the terms.

The script **plot\_comparison.py** performs this task for bow and tf-idf. It calls the **vectorize** method to perform the vectorisation and normalisation of a raw training document using the dictionary and corpus in a directory specified.

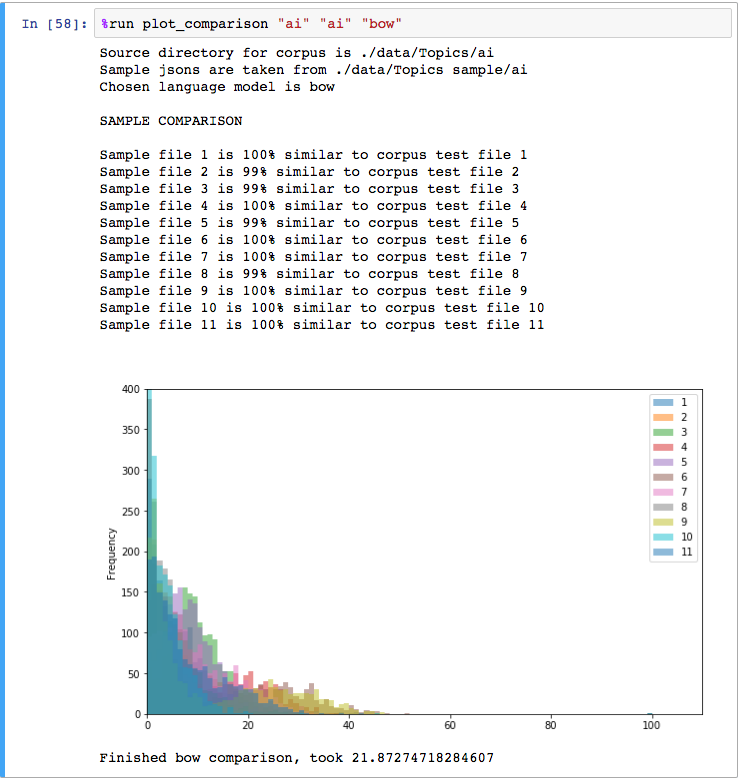
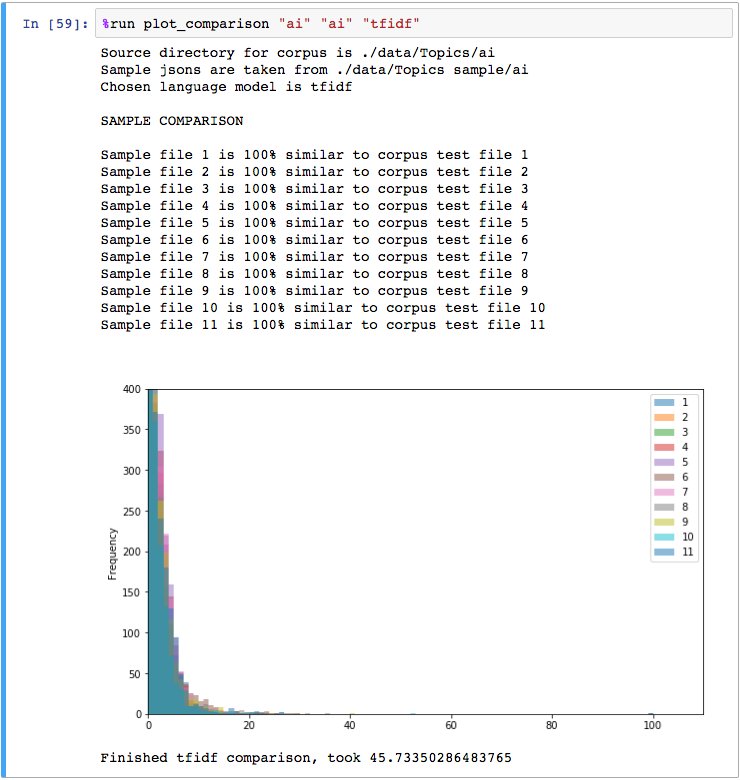
The method **plot\_comparison.cosine\_similarity** then does the mathematical calculation, returning a number between 0 and 1 (0 means that there’s no statistical relation at all, while 1 means total congruence, i.e. the two documents being compared have the same angle.)

In the accompanying Jupyter notebook **Capstone\_Project\_1\_workbook**, you can see a few versions of this comparison. We took a small sample of 10 files from each of the Curation topic folders, placed them in corresponding folders in **./data/Topics sample/**, then used them to perform various similarity queries.

To kick things off we compared some unprocessed Artificial Intelligence (AI) stories with all the document vectors in the AI corpus for both bow and tf-idf. As you can see from the histograms in *Figure 1* & *Figure 2*, below, the tf-idf query proved more exacting than the bow query, and found a consistently lower cosine similarity value between the 10 sample AI documents and the AI corpus… except when the samples were matched exactly to their duplicates in the corpus, when the tf-idf vectors were consistently found to be 100% the same, and the bow vectors vacillated between 99% and 100% similarity.

In the notebook we go on to perform this analysis across all fourteen topics, and all the histograms show the same effect: tf-idf comparisons show markedly lower cosine similarity scores, except on exact matches, where they score 100%. And we'd expect this: by filtering/weighting for more frequent words, tf-idf vectorisation will remove many of the false positives that will be thrown up by the more broadly defined bow vectors.

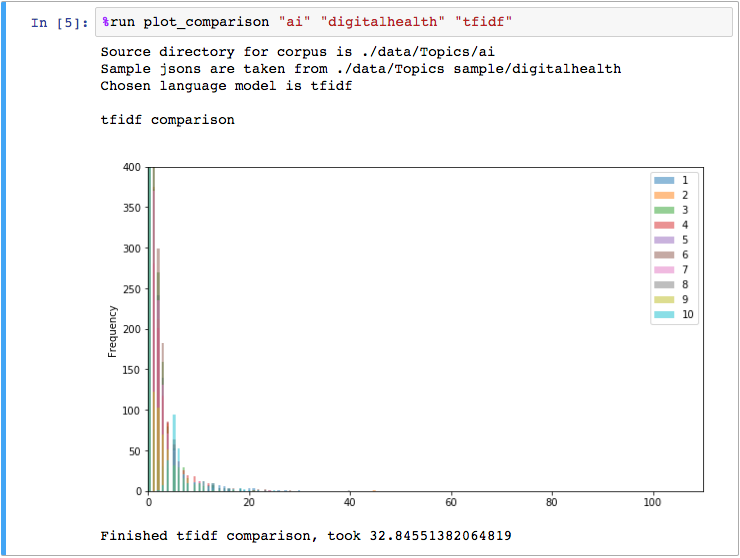
*Figure 1: Figure 2:*

*Figure 3: (NB Sample articles 3 and 4 are the same article, and so should give identical results)*



Having established that tf-idf is the more accurate tool when comparing like with like, we used it to cross-compare various samples (e.g. the AI corpus with samples from Digital Health):



and then used the **plot\_comparison.compare\_all\_topics()** method to compare some of the unclassified articles from the Moreover database with all the Topic corpuses in turn, resulting in the output displayed in *Figure 3*, above.

Moving on from bow and tf-idf similarity queries to the doc2vec vectors, rather than use our own simple **cosine\_similarity** function, we are going to call the **most\_similar** method from the gensim library, which combines a cosine similarity calculation with a *k* nearest neighbour (kNN) search.[[4]](#footnote-4)

To assess the model, we can start by pretending the training documents are unseen data and comparing vectors inferred from them with the vectors already in the training corpus to see how they compare by ranking them based on self-similarity; we can do this using the **method doc2vec\_model.assess\_model()** (there is an example in the Jupyter notebook). The expectation here is that the model will of course be overfit to this data and we'll find similar documents very easily. Ranked similarities are tracked in the *ranks* and *second\_ranks* lists (a count of zero means the document is best matched with itself).

In the notebook we’ve done this with the Counterparty Risk data. Here we can see that 5533 times out of 6847 - so 80.8% of the time - the document is matched with itself. The rest of the time the model has judged it more similar to another document, which is not altogether surprising given our quite large corpus of quite short documents that deal with a quite focused and relatively specialist subjects, in which key words are likely to reoccur in different contexts especially after stopwords and frequent terms have been filtered out.

As well as looking at the result overall, we can drill down and look at specific examples. This gives us a more semantic-level idea of how well our model is functioning. One such instance, comparing the inferred vector of an article from our unclassified Moreover data set to the Counterparty Risk corpus is shown below. It’s quite an intriguing example: an article about the death of a football coach has been awarded a high (83%) match with an article about a financial lawsuit, because of the common language used around injury and recovery.

Test Document (155): «cooper ratten the son of hawthorn assistant coach brett ratten was killed in crash north east of melbourne the son of hawthorn assistant coach brett ratten was killed in car crash north east of melbourne on sunday morning cooper ratten was passenger in car which crashed on glenview road at yarra glen yarra glen football club president vincent erickson said it had been tough day for the club we ve just got to get behind each other our thoughts are with the families who have been directly impacted by this tragedy he said hawthorn football club released statement on the ratten family behalf confirming the teen death hawthorn football club is deeply saddened to advise brett ratten son cooper was involved in car accident overnight that has claimed the year old life the statement said the club asks that media respect brett and the family privacy at this tragic time another passenger aged received minor injuries in the crash the driver also aged was taken to maroondah hospital and is currently in police custody»

SIMILAR/DISSIMILAR DOCS PER MODEL Doc2Vec(dm/m,d50,n5,w5,mc2,s0.001,t3): MOST (1395, 0.830583393573761): «strong raymond james strong and fund manager have been hit with putative class action over allegations of fraudulent scheme involving the jay peak ski resort in vermont carlos enrique hiller sanchez is seeking to recover at least that was invested during phase of the eb immigrant investor programme that was apparently misused and stolen»

MEDIAN (5824, 0.5362710952758789): «strong deutsche bank strong has reported net profit of euro mn up from euro mn year ago and in line with estimates however the bank still recorded legal related expenses of euro bn in the quarter which was more than double what had been expected this was off set by surge is trading revenue in another positive the bank cet ratio rose to from in the previous quarter this comes as new ceo john cryan looks to establish new strategy for the bank who stated that while the strengths of the bank are reflected in trading revenue challenges include the unacceptably high level of costs our continuing burden of high litigation charges and balance sheet that must be more efficient»

LEAST (6838, -0.3577459156513214): «new loans at bank of china hong kong grew to hkd bn bn in the eight months to the end of august far outpacing the industry rate of yoy the bank now underwrites of new loans in the city»

There are further examples in the workbook that accompanies this report.

# 4. Topic Modelling

## LDA

We've looked at doc2vec, now let's look at another machine learning application, [Latent Dirichlet allocation](https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation" \t "_blank) (LDA), a form of unsupervised text analysis which we can use to interrogate, not our editorial content itself, but the categories into which our editorial content has been organised.

This technique assigns documents a weighting according to their membership of a specified number of categories – without knowing what those categories are beforehand. It does this by assuming that each document in the corpus being analysed has a set of topics assigned to it according to a sparse Dirichlet allocation (the "Dirichlet prior"), a K-dimensional vector α where K is the specified number of categories. These sparse Dirichlet priors encode the intuition that documents cover only a small set of topics, and that the topics use only a small set of words frequently.[[5]](#footnote-5)

To illustrate we can run a run a quick test using the corpus and dictionary we’ve already created from our 6,847 Counterparty Risk files. The code referred to is both in the **Capstone\_Project\_1\_workbook** and the python file **lda\_model.py**.

We’re going to use the LDA model function in the gensim library to divide our corpus up into 25 topic buckets (K). 25 is a gut-feel number – it just feels like the number of topics that we might want to subdivide a given editorial topic into for the purposes of tagging or further analysis. It might work, it might not, but it feels like a reasonable starting point, and once generated, we can save the model to disc for future reference and chose a different number based on our finding.

model = LdaMulticore(corpus, id2word=dictionary, num\_topics=25)

model.save('./lda/counterpartyrisk.lda')

model.show\_topics()

Having trained a model, we can have a peek at the words it has allocated to a random selection of the 25 topics we asked it to isolate using the method **model.show\_topics().** The results are shown in *Figure 4*, below. The reference numbers of the topics are highlighted in bold on the left.

*Figure 4:*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **18** | '0.034\*"virus" | 0.020\*"bus" | 0.017\*"Ola" | 0.015\*"li" | 0.015\*"/li" | 0.009\*"Los" | 0.008\*"Angeles" | 0.008\*"California" | 0.007\*"free" | 0.006\*"resident" |
| **2** | '0.012\*"Nigeria" | 0.010\*"education" | 0.010\*"Syrian" | 0.009\*"Australia" | 0.008\*"on-demand" | 0.007\*"meat" | 0.007\*"report" | 0.006\*"suggest" | 0.005\*"Reuters" | 0.005\*"bubble" |
| **11** | '0.027\*"China" | 0.015\*"Hong" | 0.014\*"Kong" | 0.014\*"East" | 0.013\*"Chinese" | 0.011\*"Iran" | 0.010\*"Middle" | 0.009\*"YoY" | 0.008\*"strong" | 0.008\*"region" |
| **20** | '0.016\*"Didi" | 0.014\*"food" | 0.010\*"sugar" | 0.010\*"year" | 0.008\*"per" | 0.008\*"increase" | 0.008\*"supply" | 0.007\*"average" | 0.007\*"company" | 0.007\*"market" |
| **9** | '0.011\*"strong" | 0.010\*"ride-sharing" | 0.009\*"pension" | 0.009\*"district" | 0.008\*"announce" | 0.008\*"Abu" | 0.007\*"NATO" | 0.007\*"fish" | 0.007\*"deal" | 0.006\*"shareholder" |
| **22** | '0.019\*"security" | 0.014\*"technology" | 0.012\*"cyber" | 0.011\*"use" | 0.009\*"new" | 0.009\*"industry" | 0.009\*"say" | 0.008\*"insurance" | 0.008\*"``" | 0.008\*"system" |
| **1** | '0.024\*"year" | 0.015\*"report" | 0.015\*"China" | 0.013\*"increase" | 0.011\*"country" | 0.010\*"million" | 0.009\*"grow" | 0.009\*"market" | 0.008\*"expect" | 0.008\*"city" |
| **13** | '0.025\*"service" | 0.015\*"customer" | 0.012\*"online" | 0.012\*"payment" | 0.011\*"offer" | 0.010\*"platform" | 0.009\*"launch" | 0.009\*"mobile" | 0.008\*"company" | 0.008\*"app" |
| **3** | '0.022\*"use" | 0.013\*"device" | 0.012\*"robot" | 0.012\*"new" | 0.011\*"data" | 0.010\*"Google" | 0.009\*"IoT" | 0.007\*"human" | 0.007\*"intelligence" | 0.007\*"help" |
| **12** | '0.032\*"Uber" | 0.016\*"service" | 0.013\*"company" | 0.012\*"platform" | 0.009\*"new" | 0.008\*"data" | 0.008\*"Lyft" | 0.008\*"use" | 0.007\*"app" | 0.007\*"San" |

*Figure 5:*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **#0** | '0.015\*"Airbnb" | 0.009\*"say" | 0.008\*"attack" | 0.007\*"business" | 0.006\*"WeWork" | 0.006\*"space" | 0.006\*"Ola" | 0.006\*"would" | 0.006\*"``" | 0.005\*"co-working" |
| **#1** | '0.011\*"use" | 0.006\*"new" | 0.005\*"study" | 0.005\*"``" | 0.004\*"disease" | 0.004\*"find" | 0.004\*"company" | 0.004\*"system" | 0.004\*"device" | 0.004\*"also" |
| **#2** | '0.082\*"\'\'" | 0.033\*"/a" | 0.033\*"href=" | 0.030\*"http" | 0.015\*"attack" | 0.009\*"ISIS" | 0.007\*"report" | 0.007\*"Russia" | 0.006\*"security" | 0.006\*"US" |
| **#3** | '0.022\*"span" | 0.022\*"/span" | 0.014\*"ad" | 0.013\*"nbsp" | 0.009\*"rsquo" | 0.007\*"Kong" | 0.007\*"Hong" | 0.006\*"million" | 0.006\*"VR" | 0.006\*"year" |
| **#4** | '0.019\*"US" | 0.014\*"tax" | 0.008\*"say" | 0.008\*"bank" | 0.006\*"company" | 0.006\*"pay" | 0.006\*"state" | 0.006\*"government" | 0.006\*"risk" | 0.004\*"``" |
| **#5** | '0.017\*"vehicle" | 0.014\*"car" | 0.012\*"electric" | 0.011\*"Tesla" | 0.010\*"battery" | 0.009\*"fuel" | 0.009\*"new" | 0.008\*"could" | 0.007\*"technology" | 0.007\*"energy" |
| **#6** | '0.016\*"bank" | 0.009\*"UK" | 0.008\*"say" | 0.007\*"EU" | 0.007\*"new" | 0.007\*"strong" | 0.005\*"``" | 0.005\*"European" | 0.005\*"lender" | 0.005\*"\'\'" |
| **#7** | '0.020\*"China" | 0.015\*"fund" | 0.011\*"company" | 0.011\*"Chinese" | 0.011\*"investment" | 0.010\*"raise" | 0.010\*"investor" | 0.009\*"market" | 0.008\*"firm" | 0.007\*"say" |
| **#8** | '0.018\*"service" | 0.017\*"Airbnb" | 0.011\*"use" | 0.010\*"app" | 0.009\*"platform" | 0.008\*"launch" | 0.008\*"user" | 0.007\*"Uber" | 0.007\*"company" | 0.007\*"mobile" |
| **#9** | '0.059\*"Uber" | 0.045\*"\'\'" | 0.019\*"``" | 0.010\*"say" | 0.009\*"Amazon" | 0.007\*"company" | 0.007\*"Turkey" | 0.006\*"Google" | 0.005\*"self-driving" | 0.005\*"car" |
| **#10** | '0.014\*"Lyft" | 0.014\*"insurance" | 0.012\*"US" | 0.008\*"company" | 0.008\*"data" | 0.006\*"new" | 0.005\*"use" | 0.005\*"industry" | 0.005\*"could" | 0.005\*"insurer" |
| **#11** | '0.013\*"market" | 0.012\*"price" | 0.012\*"year" | 0.010\*"property" | 0.008\*"rise" | 0.008\*"increase" | 0.007\*"China" | 0.006\*"UK" | 0.006\*"loan" | 0.006\*"rate" |
| **#12** | '0.009\*"drone" | 0.008\*"new" | 0.007\*"driver" | 0.007\*"use" | 0.007\*"car" | 0.006\*"company" | 0.006\*"launch" | 0.006\*"city" | 0.005\*"Apple" | 0.005\*"robot" |
| **#13** | '0.012\*"car" | 0.010\*"``" | 0.009\*"say" | 0.009\*"US" | 0.009\*"\'\'" | 0.007\*"driver" | 0.007\*"service" | 0.006\*"vehicle" | 0.006\*"VW" | 0.006\*"claim" |

Just as we did with our doc2vec model, we can query our LDA model with a sample text string, like this:

['Chinese', 'bank', 'looks', 'to', 'merge', 'with', 'HSBC']

We can use the dictionary to transform the query into a bag-of-words vector, then ask our model to return the topics that are most (and least) related to each of the categories:

a = list(sorted(model[query], key=lambda x: x[1]))

print(a[0])

model.print\_topic(a[0][0]) #least related

print(a[-1])

model.print\_topic(a[-1][0]) #most related

Here are the topics that are most closely related to our sample query. Gensim’s LDA library query defaults to only list the results that meet a minimum threshold of >= 0.01. Below that threshold, matches are likely to be pretty weak and so are ignored.

(0, 0.010057865837263848)

'0.016\*"US" + 0.014\*"The" + 0.011\*"tax" + 0.010\*"HSBC" + 0.010\*"pay" + 0.009\*"say" + 0.008\*"fine" + 0.008\*"Bank" + 0.007\*"business" + 0.007\*"Deutsche"'

(10, 0.49385505394273882)

'0.037\*"\'\'" + 0.023\*"Bank" + 0.019\*"China" + 0.018\*"say" + 0.017\*"``" + 0.010\*"The" + 0.008\*"href=" + 0.008\*"financial" + 0.008\*"/a" + 0.008\*"http"'

As we can see, the top terms in category 0 (the least related) don’t really have much to do with our text, but those in category 10 (the most related) are clearly relevant to Chinese banks, although there are also some tags in there that clearly need to be stripped out from our corpus if we're going to get better results. It's not the best example in the world, our corpus is a bit dirty and way too small for the unsupervised learning to run sufficient iterations, but even with this compromised example it's pretty clear that the LDA engine is working to some extent. As well as trying this again with a larger corpus, we might also want to try tweaking the number of topics or the alpha values to optimise the results.

We’re now ready to scale this up to the entire data set of 43,557 Curation Corporation articles. This comes pre-labelled into 14 different topic categories, but we’re going to remove those human-curated content bins, mix all the articles in together, and see how an LDA analysis will divide it up. Ultimately, this is the dataset that we’re going to use to train our tagging engine, so it makes sense to use LDA to analyse it in order to give us a sense of whether or not the human-curated labels are at all appropriate. If we throw this whole corpus at it in an undifferentiated fashion, how good is it going to be at returning something akin to our original categories? Will it show us, too, that the categories into which Curation has divided this corpus for the purpose of reader navigation also make statistical sense?

As we know, the Curation editorial topics are:

* AI & Future Computing
* Battery Tech & Electric Vehicles
* Black Swans
* Blockchain
* Carbon Eradication
* Counterparty Risk
* Digital Advertising
* Digital Currency
* Digital Health
* Education Technology
* Financial Services
* Internet of Things
* Property
* Sharing Economy

The first step is to pass the dataset through our JSON converter, create new master file, and use this to create a new corpus. Then we can create a model with 14 topics, and print out what the LDA has judged to be the most important words for each topic. For this larger dataset, we’re going to use the LdaMulticore method, which as its name suggests will speed up the analysis by spreading the workload across the different cores on our computer.

import gensim

from gensim import corpora

from gensim.models import LdaMulticore

*# Extract and normalise the raw Curation editorial jsons*

*# without subdividing into editorial categories*

%run json\_converter "all\_topics.json" "All topics"

*# Create the corpus*

%run create\_bow\_corpus "All topics"

*# Define the dictionary & corpus variable with the dictionary*

*# & corpus we've just created*

dictionary = corpora.Dictionary.load('./data/All topics/\_bow\_corpus.dict')

corpus = corpora.MmCorpus('./data/All topics/\_bow\_corpus.mm')

*# Create a model with 14 topics and save it*

model = LdaMulticore(corpus, id2word=dictionary, num\_topics=14)

model.save('./data/lda/all\_topics.lda')

*# print a few of the most important words for each LDA topic*

model.print\_topics(-1)

The output here, the most important words for each of our 14 LDA topics, is shown in *Figure 5*, above. The topic numbers are on the left, in bold.

Looking through our list of 14 topics (0 to 13), it appears that there's quite a bit of overlap between them. LDA has not defined the topics as cleanly as our editorial sensibilities would have liked. This, indeed, is expected - I personally curate two of the content categories at Curation, and deciding on how to define our these categories and allocate stories to them is a constant headache. The major reason for doing this project is in order to discover if a machine can do a better job!

One simple way to try and disambiguate the topics is to create more of them. Figure 6 shows the output from a repeat of the topic modelling process repeated, but for 25 categories.

*Figure 6:*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **#0** | 0.020\*"President" | 0.019\*"US" | 0.013\*"government" | 0.011\*"political" | 0.011\*"UK" | 0.009\*"Trump" | 0.009\*"London" | 0.008\*"warn" | 0.008\*"election" | 0.008\*"protest" |
| **#1** | 0.025\*"could" | 0.013\*"study" | 0.010\*"technology" | 0.009\*"University" | 0.009\*"use" | 0.009\*"new" | 0.009\*"report" | 0.008\*"research" | 0.008\*"say" | 0.008\*"find" |
| **#2** | 0.019\*"say" | 0.019\*"UK" | 0.013\*"``" | 0.013\*"risk" | 0.012\*"would" | 0.010\*"could" | 0.010\*"lending" | 0.010\*"warn" | 0.009\*"bank" | 0.007\*"Didi" |
| **#3** | 0.016\*"rsquo" | 0.015\*"self-driving" | 0.013\*"outbreak" | 0.012\*"Toyota" | 0.012\*"autonomous" | 0.009\*"company" | 0.008\*"ldquo" | 0.008\*"Apple" | 0.008\*"rdquo" | 0.007\*"car" |
| **#4** | 0.014\*"US" | 0.011\*"tax" | 0.010\*"government" | 0.010\*"rule" | 0.009\*"law" | 0.009\*"country" | 0.009\*"state" | 0.008\*"EU" | 0.007\*"authority" | 0.007\*"claim" |
| **#5** | 0.014\*"company" | 0.011\*"firm" | 0.011\*"fund" | 0.010\*"service" | 0.010\*"platform" | 0.010\*"investor" | 0.010\*"raise" | 0.009\*"market" | 0.009\*"business" | 0.009\*"investment" |
| **#6** | 0.076\*"Uber" | 0.034\*"driver" | 0.016\*"Lyft" | 0.014\*"virus" | 0.012\*"taxi" | 0.009\*"service" | 0.008\*"say" | 0.007\*"Ola" | 0.006\*"state" | 0.006\*"Uber’s" |
| **#7** | 0.011\*"US" | 0.007\*"say" | 0.007\*"Libya" | 0.006\*"lawsuit" | 0.006\*"follow" | 0.006\*"bank" | 0.006\*"Australian" | 0.006\*"fail" | 0.006\*"suspect" | 0.005\*"Australia" |
| **#8** | 0.037\*"loan" | 0.035\*"debt" | 0.023\*"bank" | 0.014\*"strong" | 0.013\*"Bank" | 0.013\*"bond" | 0.010\*"mortgage" | 0.009\*"income" | 0.009\*"asset" | 0.008\*"default" |
| **#9** | 0.024\*"year" | 0.020\*"China" | 0.016\*"increase" | 0.015\*"price" | 0.015\*"market" | 0.013\*"rise" | 0.013\*"report" | 0.008\*"last" | 0.008\*"2015" | 0.008\*"growth" |
| **#10** | 0.029\*"drone" | 0.018\*"city" | 0.014\*"company" | 0.010\*"San" | 0.009\*"use" | 0.009\*"delivery" | 0.009\*"service" | 0.008\*"Francisco" | 0.007\*"fleet" | 0.007\*"car" |
| **#11** | 0.011\*"fund" | 0.008\*"use" | 0.008\*"technology" | 0.007\*"support" | 0.007\*"weapon" | 0.007\*"UK" | 0.007\*"risk" | 0.007\*"industry" | 0.007\*"government" | 0.006\*"new" |
| **#12** | 0.026\*"Model" | 0.016\*"Russia" | 0.015\*"Russian" | 0.012\*"Kong" | 0.012\*"Hong" | 0.009\*"Ukraine" | 0.008\*"''" | 0.006\*"data" | 0.006\*"Putin" | 0.006\*"woman" |
| **#13** | 0.046\*"attack" | 0.029\*"Airbnb" | 0.025\*"ISIS" | 0.020\*"group" | 0.012\*"say" | 0.010\*"cyber" | 0.009\*"terrorist" | 0.009\*"people" | 0.008\*"police" | 0.008\*"Turkey" |
| **#14** | 0.020\*"use" | 0.014\*"device" | 0.008\*"new" | 0.007\*"data" | 0.007\*"people" | 0.007\*"robot" | 0.007\*"disease" | 0.007\*"app" | 0.007\*"test" | 0.006\*"Google" |
| **#15** | 0.171\*"''" | 0.053\*"href=" | 0.053\*"/a" | 0.048\*"http" | 0.039\*"``" | 0.008\*"report" | 0.007\*"say" | 0.007\*"span" | 0.007\*"/span" | 0.006\*"As" |
| **#16** | 0.019\*"nuclear" | 0.017\*"city" | 0.017\*"rent" | 0.013\*"em" | 0.013\*"/em" | 0.012\*"missile" | 0.011\*"li" | 0.011\*"/li" | 0.010\*"construction" | 0.010\*"Iran" |
| **#17** | 0.032\*"vehicle" | 0.028\*"car" | 0.025\*"electric" | 0.023\*"Tesla" | 0.022\*"battery" | 0.012\*"charge" | 0.011\*"new" | 0.011\*"company" | 0.009\*"system" | 0.008\*"EV" |
| **#18** | 0.047\*"car" | 0.033\*"vehicle" | 0.014\*"Korea" | 0.014\*"emission" | 0.010\*"bus" | 0.010\*"VW" | 0.009\*"diesel" | 0.009\*"South" | 0.008\*"electric" | 0.007\*"road" |
| **#19** | 0.025\*"energy" | 0.020\*"power" | 0.016\*"India" | 0.013\*"country" | 0.011\*"home" | 0.010\*"city" | 0.010\*"project" | 0.009\*"solar" | 0.009\*"East" | 0.008\*"area" |
| **#20** | 0.047\*"fuel" | 0.031\*"cell" | 0.026\*"hydrogen" | 0.020\*"water" | 0.019\*"oil" | 0.011\*"use" | 0.010\*"food" | 0.010\*"production" | 0.010\*"crop" | 0.008\*"produce" |
| **#21** | 0.021\*"mobile" | 0.019\*"ad" | 0.017\*"Apple" | 0.012\*"Facebook" | 0.012\*"company" | 0.012\*"brand" | 0.011\*"medium" | 0.010\*"ride" | 0.008\*"social" | 0.008\*"user" |
| **#22** | 0.022\*"rental" | 0.015\*"market" | 0.013\*"report" | 0.013\*"new" | 0.011\*"housing" | 0.009\*"year" | 0.008\*"country" | 0.007\*"supply" | 0.007\*"Club" | 0.006\*"Lending" |
| **#23** | 0.019\*"use" | 0.017\*"service" | 0.015\*"technology" | 0.012\*"insurance" | 0.010\*"company" | 0.010\*"platform" | 0.009\*"launch" | 0.008\*"offer" | 0.008\*"allow" | 0.008\*"customer" |
| **#24** | 0.020\*"New" | 0.014\*"York" | 0.014\*"ride-hailing" | 0.010\*"Islamist" | 0.009\*"new" | 0.009\*"listing" | 0.008\*"fee" | 0.008\*"launch" | 0.008\*"offer" | 0.007\*"City" |

*Figure 7:*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| #0 | '0.025\*"energy" | 0.022\*"fuel" | 0.021\*"power" | 0.016\*"battery" | 0.013\*"oil" | 0.013\*"cell" | 0.013\*"hydrogen" | 0.009\*"water" | 0.009\*"project" | 0.009\*"Energy" |
| #1 | '0.171\*"\'\'" | 0.073\*"``" | 0.043\*"/a" | 0.043\*"href=" | 0.039\*"http" | 0.018\*"say" | 0.010\*"report" | 0.007\*"span" | 0.007\*"/span" | 0.005\*"note" |
| #2 | '0.029\*"car" | 0.027\*"Uber" | 0.024\*"vehicle" | 0.014\*"driver" | 0.013\*"electric" | 0.011\*"Tesla" | 0.010\*"new" | 0.009\*"company" | 0.009\*"city" | 0.007\*"service" |
| #3 | '0.030\*"attack" | 0.021\*"security" | 0.020\*"data" | 0.013\*"cyber" | 0.011\*"health" | 0.009\*"use" | 0.009\*"patient" | 0.008\*"information" | 0.007\*"US" | 0.007\*"company" |
| #4 | '0.024\*"year" | 0.020\*"market" | 0.017\*"price" | 0.015\*"increase" | 0.014\*"report" | 0.013\*"rise" | 0.010\*"property" | 0.009\*"growth" | 0.009\*"2015" | 0.008\*"US" |
| #5 | '0.021\*"use" | 0.016\*"drone" | 0.014\*"technology" | 0.011\*"IoT" | 0.010\*"develop" | 0.010\*"device" | 0.009\*"system" | 0.008\*"robot" | 0.008\*"new" | 0.006\*"data" |
| #6 | '0.015\*"strong" | 0.014\*"US" | 0.012\*"bank" | 0.012\*"/strong" | 0.010\*"Bank" | 0.007\*"follow" | 0.007\*"say" | 0.007\*"law" | 0.006\*"rule" | 0.006\*"claim" |
| #7 | '0.017\*"user" | 0.017\*"app" | 0.016\*"service" | 0.014\*"mobile" | 0.013\*"ad" | 0.012\*"Apple" | 0.012\*"Google" | 0.012\*"launch" | 0.011\*"new" | 0.011\*"platform" |
| #8 | '0.023\*"fund" | 0.018\*"company" | 0.015\*"investment" | 0.013\*"investor" | 0.013\*"firm" | 0.013\*"raise" | 0.012\*"market" | 0.012\*"platform" | 0.010\*"business" | 0.009\*"launch" |
| #9 | '0.020\*"bank" | 0.015\*"risk" | 0.015\*"loan" | 0.012\*"UK" | 0.012\*"debt" | 0.011\*"warn" | 0.011\*"could" | 0.008\*"say" | 0.007\*"financial" | 0.007\*"may" |
| #10 | '0.031\*"China" | 0.025\*"country" | 0.015\*"South" | 0.014\*"India" | 0.012\*"government" | 0.010\*"Europe" | 0.009\*"region" | 0.009\*"Japan" | 0.009\*"Chinese" | 0.008\*"European" |
| #11 | '0.019\*"US" | 0.010\*"could" | 0.009\*"Russia" | 0.009\*"people" | 0.009\*"risk" | 0.009\*"cause" | 0.008\*"disease" | 0.008\*"may" | 0.008\*"warn" | 0.007\*"food" |
| #12 | '0.015\*"technology" | 0.013\*"insurance" | 0.011\*"use" | 0.010\*"blockchain" | 0.010\*"industry" | 0.009\*"new" | 0.008\*"could" | 0.008\*"company" | 0.008\*"say" | 0.007\*"risk" |
| #13 | '0.026\*"Airbnb" | 0.017\*"ISIS" | 0.014\*"attack" | 0.010\*"group" | 0.010\*"student" | 0.009\*"people" | 0.008\*"kill" | 0.007\*"hotel" | 0.006\*"bomb" | 0.006\*"online" |

It's an improvement, but looking through the list of words assigned to each topic, it's still hard to see how they might map onto coherent editorial topics. Instead of increasing the number of categories, let's see what happens if we increase the number of passes - i.e. if we cycle the model over the data multiple times, to improve the statistical definition of the LDA outcomes.

*# Extract 14 LDA topics using 20 full passes, no online updates*

model = LdaMulticore(corpus, id2word=dictionary, num\_topics=14,passes=20)

model.save('./data/lda/all\_topicsx20.lda')

model.print\_topics(-1)

*Figure 7* shows the results. Just reading through the most important words for each topic, we can straight away see that there's a lot of correlation between each topic and the editorial categories listed earlier. Maybe our editorial choices at Curation weren't so subjective after all!

We can now run a query against these topics, and match a test text against the topics suggested by our LDA analysis.

*# Define a function to handle a query procedure*

*# matching a given text to the LDA topics*

def comparison(query):

query = query.split()

query = dictionary.doc2bow(query)

return query

*# Run a query*

query = ‘Despite the notion that the 2.3 million adults living with their parents are saving money for a property, a new survey shows that 50% of millennials have less than $1,000 of savings, in addition to being burdened with student debt. Foreign money, investor dominance and high prices for properties in desirable areas are contributing to what this commentator calls “rental Armageddon,” concluding that millennials are in no position to save the housing market.’

vector = comparison(query)

model[vector]

a = list(sorted(model[vector], key=lambda x: x[1]))

print(a[0])

print(model.print\_topic(a[0][0])) *#least related*

print(a[-1])

print(model.print\_topic(a[-1][0])) *#most related*

According to our new model the least-related category is 13, with 13.2% similarity:

(13, 0.13216640717063935) 0.026\*"Airbnb" + 0.017\*"ISIS" + 0.014\*"attack" + 0.010\*"group" + 0.010\*"student" + 0.009\*"people" + 0.008\*"kill" + 0.007\*"hotel" + 0.006\*"bomb" + 0.006\*"online"

and the most-related is 4, with 28.8% similarity:

(4, 0.28871330951632795) 0.024\*"year" + 0.020\*"market" + 0.017\*"price" + 0.015\*"increase" + 0.014\*"report" + 0.013\*"rise" + 0.010\*"property" + 0.009\*"growth" + 0.009\*"2015" + 0.008\*"US"

Which both look pretty good!

# 5. Automatic Classification

We’ve converted our texts into document vectors of different flavours, we’ve compared and contrasted them, and we’ve examined our editorial classifications. But while tf-idf, doc2vec and LDA have proved very useful tools in extracting coherent semantic insights from textual data, and are likely to be part of an eventual solution to the problem we are trying to solve, they are not sufficiently powerful on their own to create a fully automatic tagging and classification engine, which is the task we have set ourselves.

There are however techniques we can use to achieve this. The two we are going to look at are naïve Bayes classification, and support vector classification (SVC). Both of these will let us use the editorial classification labels we already have, train a model on this labelled database, and then use that to classify new articles. Both are also available via the Python package sci-kit learn, which we will be using throughout this section of the project. The code referred to is in both the **Capstone\_Project\_1\_workbook** and in the python file **svm\_model.py**.

Before we go any further, we need to prep our json article data by importing it as an sklearn dataset (which conveniently packages it according to our 14 categories), split it into chunks for subsequent cross-validation, and apply a tf-idf vector transformation to the relevant sections, tasks which are handled respectively by the **define\_dataset()** and **tfidf\_extraction()** methods in svm\_model.py.

## Naïve Bayes

Naïve Bayes is a development of the famous Bayes rule for conditional probability, which take its name from the Presbyterian minister who came up with it in around 1750:

where *c* represents a *class* or category, and *f* represents a feature sector. **We are computing the probability that a document (or whatever we are classifying) belongs to category *c* given the features in the document.** *P(f)* is really just a normalization constant, so the literature usually writes Bayes’ Theorem in context of Naïve Bayes as

*P(c|f)∝P(f|c)P(c)*

This deceptively simple formula for calculating the probably of one event happening on the basis that another event is happening allows us – when combined with a straightforward factoring routine – to hugely reduce the complexity of calculating of conditional probabilities across large and multifarious datasets.

Technically this is called “Full Probability Modelling”, and Bayes’ rule allows us to effective estimate the conditional probabilities of the unobserved quantities of ultimate interest in a data set, given the observed data we have to work with.

It’s a mathematical simplification, yes, - hence the label “naïve” – but it’s a simplification that has been found to work surprisingly well across many different applications. One famous example is the email spam filters, where naïve Bayes allows us to ask, given N words that commonly appear in spam emails, what the probability is of an email being spam if it contains a given subset of these words.

This is, in effect, an exercise in classification, one which can be repurposed to ask: given what we know about the frequency of words that occur in each of Curation’s 14 editorial categories, what’s the probability of any given unclassified document belonging to each one? Fortunately we don’t have to code the maths ourselves to do this. Instead, we can just call the **MultinomialNB()** method in sklearn, which we do in our **fit\_nb()** method:

We can then use our **predict\_outcome\_tfidf()** method to query our fitted model, **clf**, with some test documents, and see how it performs. (The classifications suggested by the model are shown in bold, below).

"UK government may be planning diesel-scrappage scheme<p>The UK's Department for Transport and the Department for Environment, Food and Rural Affairs is in talks to introduce a diesel vehicle scrappage scheme as the country searches for ways to cut carbon emissions and pollution, according to unnamed sources. The strategy would offer a discount on low-emissions cars or cash back to drivers who choose to trade in their old diesel vehicles. A Department for Transport spokesperson said there are presently no plans to introduce a scrappage scheme. However, Transport Secretary Chris Grayling has shown support for the measures in industry meetings, according to the Telegraph.</p>"

**=> carboneradication**

“UK-led consortium looks to robots to monitor windfarms and cables<p>A UK-led consortium is launching a $5m project to use robots for monitoring offshore wind farms and undersea cabling. The Engineering and Physical Sciences Research Council (EPSRC)-backed scheme hopes to improve how wind farms are maintained and managed. The project is developing "novel sensing techniques" to inspect systems in the field, said Dr David Flynn, director of Smart Systems Group at Heriot-Watt University, a member of the consortium. These include a dolphin-inspired local frequency sonar attached to an autonomous vehicle that can be used to inspect undersea cables, and drones to inspect offshore substations.</p>\n”

**=> carboneradication**

"Ubtech partners with Amazon's Alexa in new robot<p>Ubtech Robotics has partnered with Amazon to bring Alexa, Amazon’s virtual assistant, to its latest robot, Lynx. Owners of Lynx will be able to interact with it as though it were a personal assistant. Since its launch, Amazon has sold millions of Echo devices, but lately they have begun to free Alexa from her Echo chamber, installing the natural language-processing assistant in home devices, wearable technology, cars and even robots. Amazon seems to be using partners such as Ubtech to redefine the physical form of its cloud-based technology.</p>\n"

**=> internetofthings**

*def* fit\_nb(*xtrain*, *ytrain*):  
 clf = MultinomialNB().fit(*xtrain*, *ytrain*)  
 *return* clf

The results are not bad at all, and may well be worthy of more exhaustive investigation, but rather than do that right away we are instead going to look at implementations of support vector machines, and see how those compare at this early stage.

## SVM

Support vector machine methods in sklearn take rather more arguments that their Bayesian counterparts:

clf = SVC(C=1.0, cache\_size=200, class\_weight=*None*, coef0=0.0,  
 decision\_function\_shape=*None*, degree=3, gamma='auto', kernel='linear',  
 max\_iter=-1, probability=*False*, random\_state=*None*, shrinking=*True*,  
 tol=0.001, verbose=*False*).fit(*xtrain*, *ytrain*)

A “support vector” is a way of computing a decision boundary between data points, a boundary that can be used for subsequent predictions and classification without having to keep holding the entire dataset in memory, which some other powerful classification tools, such as k-nearest neighbour (kNN), require.

The basic SVM decision boundary is a hyperplane, whose angle and position are computed using an orthogonal vector, w, and a measure of bias, b. Being, in essence, a straight line, a hyperplane lacks the complexity of a kNN decision boundary and intriguingly, it was in attempting to find a solution to this situation that Frank Rosenblatt came up with the model of the perceptron – the building block of the neural network – in 1957. In its simplest form a perceptron is just a sum of multiple hyperplanes, all with the same bias. One hyperplane on its own produces a step classification function, multiple hyperplanes with the same bias produce a sigmoid function – a sigmoid function which is, in fact, a exemplar of logistic regression! A neural network is at its heart, therefore, a logistic regression machine.

However, we are getting ahead of ourselves. SVMs tackle the problem of flat decision boundaries in two ways. Firstly, SVMs use some tuning parameters to introduce extra properties into the hyperplane decision boundary. The most common are gamma and C. Gamma, in essence, defines the width of the boundary. The smaller gamma is set, the finer the gap between data categories, though we want gamma to be as large as possible. C, on the other hand, introduces a quantum of flexibility, or “slack”, into the decision boundary, enabling it to accommodate outliers. The larger C, the less slack there is in the boundary. Making C smaller reduces its importance in the SVM function, and allows the classifier more “slack” with which to misclassify things.

Secondly, SVMs can use hyperplanes drawn in multiple dimensions, which increases their complexity (and ideally requires cross-validation and the use another statistical procedure, principal component analysis (PCA), to calculate how many dimensions need added to optimise classification efficacy while stopping short of overfitting the model).

A simple linear SVM operates with two dimensional hyperplanes. A polynomial kernel will allow multiple dimensions to be added, while a kernel that uses a function known as “radial bias” can in theory operate up to an infinite number of dimensions.

Even with powerful modern processors, however, the mathematics of computing the relations of multiple data points in multiple dimensions would be prohibitively expensive were it not for a neat mathematical coincidence, known as “the kernel trick (K)”, that allows you to compute the dot product of multiple points in higher dimensional space without ever having to compute the higher dimensional positions of the points themselves.

The beauty of an SVM is that it combines the kernel trick with the maximal margin classification offered by gamma to radically reduce the data sparsity that occurs when you increase dimensionality (“the curse of dimensionality”) to produce a very powerful and rigorous data classification engine that is particularly good at face recognition, among other things.

In the **Capstone\_Project\_1\_workbook** we run various permutations of an SVM (the code for each is also written as methods in **svm\_model.py**). The results of querying each of the resulting models for classification predictions of our three test articles are shown below, alongside the results we got from our Naïve Bayes model. The degrees referred to are the degrees of the underlying polynomial. What we are matching here are the vectors of our training/testing documents (x) with the labels of our editorial topics (y).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Figure 8:* |  | "UK government may be planning diesel-scrappage scheme<p>The UK's Department for Transport and the Department for Environment, Food and Rural Affairs is in talks to introduce a diesel vehicle scrappage scheme as the country searches for ways to cut carbon emissions and pollution, according to unnamed sources. The strategy would offer a discount on low-emissions cars or cash back to drivers who choose to trade in their old diesel vehicles. A Department for Transport spokesperson said there are presently no plans to introduce a scrappage scheme. However, Transport Secretary Chris Grayling has shown support for the measures in industry meetings, according to the Telegraph.</p>" | “UK-led consortium looks to robots to monitor windfarms and cables<p>A UK-led consortium is launching a $5m project to use robots for monitoring offshore wind farms and undersea cabling. The Engineering and Physical Sciences Research Council (EPSRC)-backed scheme hopes to improve how wind farms are maintained and managed. The project is developing "novel sensing techniques" to inspect systems in the field, said Dr David Flynn, director of Smart Systems Group at Heriot-Watt University, a member of the consortium. These include a dolphin-inspired local frequency sonar attached to an autonomous vehicle that can be used to inspect undersea cables, and drones to inspect offshore substations.</p>\n” | "Ubtech partners with Amazon's Alexa in new robot<p>Ubtech Robotics has partnered with Amazon to bring Alexa, Amazon’s virtual assistant, to its latest robot, Lynx. Owners of Lynx will be able to interact with it as though it were a personal assistant. Since its launch, Amazon has sold millions of Echo devices, but lately they have begun to free Alexa from her Echo chamber, installing the natural language-processing assistant in home devices, wearable technology, cars and even robots. Amazon seems to be using partners such as Ubtech to redefine the physical form of its cloud-based technology.</p>\n" | **Mean prediction accuracy** |
|  | **Human** | carboneradication | ai/carboneradication | ai/internetofthings |  |
| **Tf-idf** | **Naïve Bayes** | carboneradication | carboneradication | internetofthings | 61.3 |
| **Linear SVM** | carboneradication | ai | ai |  |
| **Polynomial SVM** | counterpartyrisk | counterpartyrisk | counterpartyrisk |  |
| **RBF SVM (3 deg.)** | counterpartyrisk | counterpartyrisk | counterpartyrisk |  |
| **RBF SVM (5 deg.)** | counterpartyrisk | counterpartyrisk | counterpartyrisk |  |
| **Tuned SVM (RBF)** | carboneradiction | ai | ai | 64 |
| **doc2vec** | **Tuned SVM (RBF)** | batterytech | digitalhealth | ai | 77.2 |

Looking at these results, and bearing in mind that even a human editor would have a hard time deciding on which category to put the second two stories in (at Curation, these two stories would in fact be tagged for both sections), it appears that naïve Bayes and a simple linear SVM are giving us the most promising results. Adding polynomial kernels and higher dimensions (the white cells in the table) seems, at least in this small test sample, to only be confusing the model.

However, when we tune the model properly, which we can do by conducting a grid search, things change. A grid search cycles through all the combinations of all the various parameters and runs a statistical cross-validation test using our trial and test data splits to determine the combination of parameters that will produce the best results.

Training 5 x 4 x 3 = 60 permutations took my aging 2009 Dual Core MacBook Pro quite a while – over 100 hours, in fact! The numbers that eventually emerged were as follows:

from sklearn import svm, grid\_search

def svc\_param\_selection(X, y, nfolds):

Cs = [0.001, 0.01, 0.1, 1, 10]

gammas = [0.001, 0.01, 0.1, 1]

kernels = ['linear', 'poly', 'rbf']

param\_grid = {'C': Cs, 'gamma' : gammas, 'kernel': kernels}

grid\_search = GridSearchCV(svm.SVC(), param\_grid, cv=nfolds)

grid\_search.fit(X, y)

grid\_search.best\_params\_

return grid\_search.best\_params\_

print(svc\_param\_selection(X\_train\_Tfidf, y\_train, 5))

{'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}

If we plug these into our SVC model, thus (also in the **pipeline\_tfidf\_best\_params()** method in **svm\_model.py**):

we get the results highlighted in green in *Figure 8*, above, which actually align with the best set of results we had from our early test, those provided by the simple linear kernel. These results are of course only predicted on predicting outcomes for three randomly chosen articles. What we really need to do in order to get a sense of how well our model is performing is to produce a proper prediction report by category (*Figure 9*, below) and a confusion matrix (*Figure 10*), both of which can handily be summoned using the **sklearn.metrics** package, and which are displayed below.

clf = SVC(C=10, cache\_size=200, class\_weight=*None*, coef0=0.0,  
 decision\_function\_shape=*None*, degree=5, gamma='0.01', kernel='rbf',  
 max\_iter=-1, probability=*False*, random\_state=*None*, shrinking=*True*,  
 tol=0.001, verbose=*False*).fit(*xtrain*, *ytrain*)

In the first table:

* **Precision** - Precision is the ratio of correctly predicted positive classifications to the total predicted positive classifications. The question that this metric answers is: of all articles that we labelled as belonging to a particular topic, how many actually belonged to that topic? High precision relates to a low false positive rate. We have got 0.788 precision which is pretty good. (Precision = True Positives / True Positives + False Positives)
* **Recall**- Recall is the ratio of correctly predicted positive classifications to the overall classifications in an actual topic. The question recall answers is: Of all the articles that were truly classified (by humans) as belonging to a topic, how many did our SVM correctly label as belonging to that topic? (Recall = True Positives / True Positives + False Negatives)
* **F1 score** - F1 Score is the weighted average of Precision and Recall (aka the harmonic mean. It therefore takes both false positives and false negatives into account, and is useful if the data has an uneven distribution of topics. (F1 score = 2\*(Recall \* Precision) / (Recall + Precision)
* **Support** – Support is simply the number of truly classified samples in the original training data set for that topic.

In each case (other than Support), a score of 1 would be perfect classification of the testing sample by the SVM; while a score of 0.5 would be equivalent to classifying the articles with a coin toss.

*Figure 9:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **precision** | **recall** | **f1-score** | **support** |
| **ai** | 0.76 | 0.8 | 0.78 | 669 |
| **batterytech** | 0.77 | 0.84 | 0.8 | 1609 |
| **blackswans** | 0.78 | 0.79 | 0.78 | 1934 |
| **blockchain** | 0.48 | 0.57 | 0.52 | 661 |
| **carboneradication** | 0.8 | 0.76 | 0.78 | 1477 |
| **counterpartyrisk** | 0.91 | 0.85 | 0.88 | 2708 |
| **digitalads** | 0.93 | 0.93 | 0.93 | 1202 |
| **digitalcurrency** | 0.51 | 0.51 | 0.51 | 762 |
| **digitalhealth** | 0.82 | 0.77 | 0.79 | 776 |
| **educationtech** | 0.89 | 0.86 | 0.88 | 617 |
| **financialservices** | 0.7 | 0.73 | 0.71 | 1600 |
| **internetofthings** | 0.78 | 0.78 | 0.78 | 1078 |
| **property** | 0.73 | 0.56 | 0.64 | 841 |
| **sharingeconomy** | 0.7 | 0.74 | 0.72 | 1467 |
| **avg/total** | **0.78** | **0.77** | **0.77** | **17401** |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **ai** | **batterytech** | **blackswans** | **blockchain** | **carboneradication** | **counterpartyrisk** | **digitalads** | **digitalcurrency** | **digitalhealth** | **educationtech** | **financialservices** | **internetofthings** | **property** | **sharingeconomy** |
| **ai** | 538 | 15 | 23 | 0 | 1 | 0 | 16 | 0 | 18 | 23 | 7 | 21 | 4 | 3 |
| **batterytech** | 18 | 1353 | 17 | 1 | 142 | 4 | 2 | 2 | 2 | 1 | 10 | 19 | 0 | 38 |
| **blackswans** | 23 | 73 | 1529 | 4 | 48 | 75 | 14 | 4 | 19 | 15 | 45 | 17 | 34 | 34 |
| **blockchain** | 0 | 0 | 8 | 378 | 0 | 10 | 7 | 201 | 0 | 0 | 44 | 9 | 0 | 4 |
| **carboneradication** | 7 | 202 | 80 | 1 | 1127 | 6 | 4 | 0 | 1 | 1 | 11 | 32 | 1 | 4 |
| **counterpartyrisk** | 5 | 3 | 87 | 19 | 20 | 2297 | 3 | 65 | 7 | 0 | 170 | 5 | 12 | 15 |
| **digitalads** | 13 | 3 | 18 | 2 | 1 | 3 | 1122 | 5 | 2 | 1 | 7 | 19 | 2 | 4 |
| **digitalcurrency** | 0 | 0 | 8 | 290 | 1 | 11 | 5 | 388 | 2 | 2 | 37 | 11 | 0 | 7 |
| **digitalhealth** | 42 | 0 | 38 | 0 | 7 | 0 | 1 | 5 | 599 | 4 | 10 | 65 | 0 | 5 |
| **educationtech** | 17 | 4 | 34 | 0 | 0 | 2 | 4 | 1 | 2 | 532 | 12 | 8 | 0 | 1 |
| **financialservices** | 14 | 8 | 42 | 59 | 11 | 82 | 2 | 80 | 13 | 2 | 1171 | 8 | 14 | 94 |
| **internetofthings** | 23 | 34 | 28 | 6 | 40 | 1 | 13 | 4 | 57 | 4 | 15 | 839 | 7 | 7 |
| **property** | 9 | 0 | 41 | 7 | 3 | 11 | 1 | 1 | 0 | 1 | 27 | 10 | 472 | 258 |
| **sharingeconomy** | 3 | 63 | 17 | 14 | 4 | 12 | 8 | 12 | 10 | 11 | 116 | 7 | 98 | 1092 |

*Figure 10:*

The confusion matrix, on the other hand, shows how documents from labeled topics perform when we ask our model to predict what it things their label should be. Perfect prediction would mean each topics documents were always predicted to belong only to that topic, and so the diagonal numbers would equal the numbers of documents in each topic, and all the other numbers would be zero. Life (and semantic classification) isn’t like that, of course, and so there is a spread of misclassifications, and the matrix allows us to see the nature of that spread. This is fantastically useful from an editorial point of view, as it allows us to see the degree of crossover between what are always going to be inexact editorial categories.

We can see, for example, that there is particularly high crossover between blockchain and digital currency, between property and sharingeconony, and between batterytech and carboneradication. This is exactly what we’d expect, as we struggle with classifying articles in these sections every day (and often have to double classify them or fork them to produce two slightly different versions), and is a very good indicator that our model is working as it should.

Our final task is to compare the performance of this model, based on tf-idf vectors, on one based on doc2vec vectors, to see which might be better for the Curation documents that we’re working with.

The process for setting up SVM with doc2vec is very similar to that for tf-idf, although we do need some minor changes to the code that translates the text documents into vectors (recall that **docs\_new** is our test sample of three documents):

(This code is included in **doc2vec4svm.py**.) With the data prepared, we can do a grid search and then run SVM with the optimised parameters. The results are shown in *Figures 11 & 12*, below, and in the bottom row of *Figure 8*, above.

def read\_corpus\_svm(corpus, tokens\_only=False):

*'''Pre-process the textual content for training the doc2vec model'''*

for i, item in enumerate(corpus):

if tokens\_only:

yield gensim.utils.simple\_preprocess(item)

else:

# For training data, add tags

yield gensim.models.doc2vec.TaggedDocument

(gensim.utils.simple\_preprocess(item), [i])

return

*#Turn the cross validation splits into vectors*

X\_train\_corpus = list(read\_corpus\_svm(X\_train))

X\_test\_corpus = list(read\_corpus\_svm(X\_test, tokens\_only=True))

X\_new\_corpus = list(read\_corpus\_svm(docs\_new, tokens\_only=True))

*#Create & train the doc2vec model*

model = gensim.models.doc2vec.Doc2Vec(size=50, min\_count=2, iter=10)

model.build\_vocab(X\_train\_corpus)

model.train(X\_train\_corpus, total\_examples=model.corpus\_count, epochs=model.iter)

*#Infer the doc2vec vectors for use with the SVM*

X\_train\_doc2vec = []

for doc\_id in range(len(X\_train\_corpus)):

inferred\_vector = model.infer\_vector(X\_train\_corpus[doc\_id].words)

X\_train\_doc2vec.append(inferred\_vector)

X\_test\_doc2vec = []

for doc\_id in range(len(X\_test\_corpus)):

inferred\_vector = model.infer\_vector(X\_test\_corpus[doc\_id])

X\_test\_doc2vec.append(inferred\_vector)

X\_new\_doc2vec = []

for doc\_id in range(len(X\_new\_corpus)):

inferred\_vector = model.infer\_vector(X\_new\_corpus[doc\_id])

X\_new\_doc2vec.append(inferred\_vector)

*Figure 11:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **precision** | **recall** | **f1-score** | **support** |
| **ai** | 0.61 | 0.59 | 0.6 | 676 |
| **batterytech** | 0.7 | 0.7 | 0.7 | 1631 |
| **blackswans** | 0.7 | 0.72 | 0.71 | 1943 |
| **blockchain** | 0.4 | 0.37 | 0.38 | 642 |
| **carboneradication** | 0.7 | 0.65 | 0.67 | 1525 |
| **counterpartyrisk** | 0.68 | 0.75 | 0.71 | 2739 |
| **digitalads** | 0.85 | 0.89 | 0.87 | 1188 |
| **digitalcurrency** | 0.41 | 0.28 | 0.33 | 776 |
| **digitalhealth** | 0.64 | 0.66 | 0.65 | 770 |
| **educationtech** | 0.78 | 0.77 | 0.78 | 617 |
| **financialservices** | 0.54 | 0.57 | 0.56 | 1607 |
| **internetofthings** | 0.48 | 0.53 | 0.5 | 1076 |
| **property** | 0.57 | 0.46 | 0.51 | 849 |
| **sharingeconomy** | 0.6 | 0.57 | 0.58 | 1362 |
| **avg/total** | 0.64 | 0.64 | 0.64 | 17401 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **ai** | **batterytech** | **blackswans** | **blockchain** | **carboneradication** | **counterpartyrisk** | **digitalads** | **digitalcurrency** | **digitalhealth** | **educationtech** | **financialservices** | **internetofthings** | **property** | **sharingeconomy** |
| **ai** | 398 | 26 | 40 | 3 | 17 | 11 | 25 | 2 | 49 | 24 | 14 | 49 | 10 | 8 |
| **batterytech** | 26 | 1134 | 45 | 3 | 209 | 34 | 2 | 0 | 8 | 7 | 18 | 48 | 9 | 88 |
| **blackswans** | 35 | 63 | 1407 | 9 | 65 | 128 | 23 | 10 | 37 | 17 | 45 | 27 | 38 | 39 |
| **blockchain** | 7 | 1 | 20 | 236 | 4 | 60 | 10 | 170 | 4 | 6 | 88 | 16 | 7 | 13 |
| **carboneradication** | 19 | 212 | 82 | 6 | 992 | 68 | 8 | 1 | 11 | 3 | 21 | 82 | 11 | 9 |
| **counterpartyrisk** | 8 | 22 | 152 | 24 | 37 | 2044 | 9 | 42 | 19 | 3 | 249 | 75 | 27 | 28 |
| **digitalads** | 8 | 4 | 16 | 4 | 2 | 22 | 1052 | 6 | 4 | 5 | 17 | 30 | 5 | 13 |
| **digitalcurrency** | 6 | 3 | 25 | 213 | 5 | 114 | 18 | 215 | 8 | 7 | 100 | 35 | 4 | 23 |
| **digitalhealth** | 42 | 4 | 27 | 4 | 3 | 17 | 7 | 5 | 507 | 17 | 26 | 98 | 2 | 11 |
| **educationtech** | 14 | 1 | 21 | 3 | 2 | 12 | 11 | 1 | 16 | 478 | 11 | 39 | 0 | 8 |
| **financialservices** | 13 | 20 | 49 | 68 | 27 | 239 | 12 | 50 | 14 | 18 | 919 | 80 | 26 | 72 |
| **internetofthings** | 57 | 52 | 33 | 9 | 33 | 112 | 33 | 11 | 94 | 8 | 30 | 575 | 13 | 16 |
| **property** | 14 | 4 | 60 | 5 | 18 | 69 | 9 | 2 | 3 | 6 | 39 | 32 | 391 | 197 |
| **sharingeconomy** | 8 | 80 | 37 | 10 | 12 | 89 | 20 | 15 | 14 | 16 | 121 | 21 | 142 | 777 |

*Figure 12:*

# 6. Conclusion & Next Steps

The three main key results from our analysis were:

* The prediction accuracy of the SVM based on ti-dif averaged 0.77 as opposed to the 0.64 achieved by the SVM based on doc2vec.
* The tf-idf confusion matrix scored consistently higher on the diagonal matches than did the matrix for doc2vec. Our test run on the sample of three documents suggested the better result as well (*Figure 8*).
* Carboneradication/ai/ai was a much more accurate category result than doc2vec’s batterytech/digitalhealth/ai, which made little sense of the textual data.

Therefore we can say that SVM based on tf-idf vectors is the best performer out of the classification models we looked at.

So we now have the basis of our tagging engine. We have transformed our documents into vectors using tf-idf and doc2vec, checked that the topic categories we have defined for them make sense using LDA, used our data to tune, train and compare classification models using SVM, and chosen the best performing model.

As this SVM model can now be used to automatically classify future Curation documents without needing to hold the entire dataset in memory, we can put it online as an api-accessible web service that the client can query with a new document in order to receive a suggested classification. This is something that we’ll be doing as part of our second Capstone project.

As well as tagging document across topics, in that project we will be looking at tagging documents within topics (“microtagging”) and be looking to see if we can use a neural net to help us achieve this.

The benefit of having topics and micro tags confirmed as part of the api workflow, and feeding that confirmation back into the classification engine as a form of supervised learning, will also form part of that study.

# 7. Further Research

* Look further into unsupervised learning: pay attention to the K-means algo and compare it to LDA (which is itself a more complex form of K-means).
* A 2016 paper published on Arvix, ["Mixing Dirichlet Topic Models and Word Embeddings to Make lda2vec"](https://arxiv.org/abs/1605.02019" \t "_blank) looks precisely at trying to combine LDA and Doc2Vec into a single method.
* Investigate plotting SVM results using an isomap: <https://www.datacamp.com/community/tutorials/machine-learning-python#gs.iR2zeTw>

1. A "sparse" vector holds all mappings of the sample words onto the dictionary, even though the majority of them may be zero. At scale, this can be inefficient with regard to memory and processor resources, and can also lead to false negatives when applying machine learning. To address this, sparse vectors can be converted to "dense" vectors - hash tables that hold only the meaningful match values and their dictionary co-ordinates [↑](#footnote-ref-1)
2. Defined by [Wikipedia](https://en.wikipedia.org/wiki/Lemmatisation" \t "_blank) as "the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form." This involves tagging each word with an appropriate part-of-speech (POS) tag by using a pre-installed library of tag sets, and then using these tags to replace the word with its root stem (so that, for example, "walked" becomes "walk"). [↑](#footnote-ref-2)
3. As defined by [Wikipedia](https://en.wikipedia.org/wiki/Tf%E2%80%93idf" \t "_blank), "the tf-idf value increases proportionally to the number of times a word appears in the document, but is often offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general. Nowadays, tf-idf is one of the most popular term-weighting schemes. For instance, 83% of text-based recommender systems in the domain of digital libraries use tf-idf. Variations of the tf–idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query. Tf–idf can be successfully used for stop-words filtering in various subject fields including text summarization and classification." [↑](#footnote-ref-3)
4. The code for gensim’s most\_similar method can be found here: <https://github.com/RaRe-Technologies/gensim/blob/9a02527ab315d00dae30088855d2ca466cc3e436/gensim/models/word2vec.py#L1209> [↑](#footnote-ref-4)
5. In an example described in [Wikipedia](https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation" \t "_blank), "an LDA model might have topics that can be classified as CAT\_related and DOG\_related. A topic has probabilities of generating various words, such as milk, meow, and kitten, which can be classified and interpreted by the viewer as "CAT\_related". Naturally, the word cat itself will have high probability given this topic. The DOG\_related topic likewise has probabilities of generating each word: puppy, bark, and bone might have high probability. Words without special relevance […] will have roughly even probability between classes (or can be placed into a separate category). A topic is not strongly defined, neither semantically nor epistemologically. It is identified on the basis of supervised labelling and (manual) pruning on the basis of their likelihood of co-occurrence. A lexical word may occur in several topics with a different probability, however, with a different typical set of neighbouring words in each topic. Each document is assumed to be characterised by a particular set of topics. This is akin to the standard bag of words model assumption, and makes the individual words exchangeable." [↑](#footnote-ref-5)