Natural Language Processing (NLP)

Problem Book

Contents

Introduction

Natural Language Processing (NLP) is an area of AI concerned with the use of computers for interpreting and manipulating human language. In today's world, NLP is everywhere and constantly evolving. If you have ever used Google Translate, email filtering, Amazon's Alexa, or just performed an internet search then you have probably used NLP without even knowing it! NLP can be used to create systems ranging from speech recognition, to machine translation, auto complete and predictive typing, and its applications are constantly expanding.

In this project, you will use an open-source dataset to learn about various NLP techniques.

Background on common techniques

It is important when presented with a new dataset to take some time to explore the data and get a feel for what it consists of; a common way to do this is to perform some exploratory analysis of the data.

After doing some brief initial analysis, it is typical to start processing the text data into a more useful format using **text cleaning techniques**. Often each data point may consist of a whole paragraph (or more) of text. Often, we want to identify where the component sentences or words are, which is done using a method called **tokenisation**. You may also want to lowercase all text, search for and correct any misspellings in the data, eliminate common words that appear in all texts (known as **stop words**). You may also want to consider using regular expressions (**regex**) to define specific search patterns you want to remove; this might include things such as punctuation marks or non-word characters such as emojis.

Once you are happy that your text is in a more meaningful and manageable format, you can perform some more in-depth EDA. This might include looking at the size of the data, the data type in each column, plotting some bar graphs, histograms, scatter graphs, word clouds etc. to determine whether there are any obvious trends in the data. In **exploratory data analysis (EDA)** there is no right or wrong answer – you just want to do as much analysis as possible to ensure you have a good understanding of what is going on. Data analysis is not a linear problem either; it may be that once you have started performing some of the NLP techniques, you wish to go back and do some further plots or have new ideas to implement!

If you're happy you have a good understanding of your data, you can now move on to applying a **feature extraction technique**. This stage is necessary, because machine learning algorithms cannot work with raw text directly but need it to be mapped to vectors of real numbers. These numbers should give a measure of how similar each word or phrase is to another. A commonly used technique is called **bag-of-words,** which describes the frequency each word occurs within a text and should then be evaluated using a distance metric for example **Jaccard distance or Word Mover Distance**. Additional scoring using **term frequency-inverse document frequency (TIDF)** is also possible. These feature extraction methods tend to ignore the relationship and context, focussing only on frequency of words. **Word embedding techniques** such as Google's **word2vec**, which uses neural networks, learn to recognize context and relationships of words so that they can detect similar words more accurately.

Once your data has been encoded as vectors, you can now start building your **classifier**. Building a model requires training on train data, and then testing the performance on some unseen test data. You will need to choose useful features to give your model for it to learn from and help it to predict the outcome classification. You should be able to identify good features to try from looking at the similarity and distance metrics you calculated before. There are lots of different models that can be used, but a simple but effective choice is the use of **logistic regression** which acts on the word vectors. Having a validation set allows you to iteratively improve your model, debugging and changing feature set inputs and weights to create an optimal model to run on your final test set.

Overview of tasks

Part 0: Topic familiarisation

Read the background information carefully, and do any further reading you might find helpful to get to grips with the main concepts in NLP.

Complete the Recap useful maths exercises section in the second half of this problem book.

Part 1: Cleaning and formatting the data

You'll be asked to consider what problems you may have 'tokenising' the data e.g., spelling mistakes, emojis, handling different tenses, etc. Following this, you'll need to prepare the data.

We suggest starting by working through:

1. 0_IntroToJupyter.ipynb
2. 1_PreparingTextData.ipynb
3. Complete the Stop Words maths exercise from the problem book

Extension: you could apply techniques such as stemming and lemmatization to improve on your text cleaning. These try and group words that come from the same base as equivalent (e.g. swims and swimming).

Part 2: Exploratory Data Analysis (EDA)

Begin by exploring the dataset. Feel free to produce any sort of statistics and visualisation that you find helpful for getting to grips with the data.

We suggest starting by working through:

1. 2_EDA.ipynb

Extension:

1. **You could try calculating the polarity for each headline (how positive or negative in sentiment the text is) and compare results. Python's textblob.TextBlob.sentiment methods are helpful here.**

2. **You could estimate the readability of each headline (Python's textstat.text standard method is helpful here).**
3. **You could try using Latent Dirichlet Allocation (LDA) or a clustering algorithm such as k-means to topic model on headlines – see links in further reading for an overview of these techniques.**

Before moving on to the next stage, please get in touch and let us know how you are getting on, send us some of your EDA graphs/tables/images for feedback and we can organise a discussion for the next stage.

Part 3: Word embeddings and similarity metrics
This section of the project will be the most mathematically focused. You'll be presented with several word embedding techniques (e.g., bag of words, TF-IDF) and the maths behind them, alongside some written exercises. We will also discuss similarity metrics. You are encouraged to compare the various techniques and identify the potential downfalls.

We suggest starting by working through:

1. 3_JaccardDistance.ipynb
2. 4_CosineSimilarity.ipynb
3. 5_WordMoversDistance.ipynb
4. Complete the Bag of Words and WMD exercises from the problem book

Extension: you could research additional methods such as bag-of-bigrams (more generally known as bag of n-grams), this method works similarly to bag of words but captures sequences of two tokens.

Part 4: Building a classifier
Finally, you'll use the calculated word embeddings to build classifiers. As an extension you could research and implement additional models, such as simple neural networks. The results from the best model should be submitted for scoring against hidden labels. Scoring will be calculated by the problem supporters.

We suggest starting by working through:

1. 6_Classifiers.ipynb
2. Complete the Logistic Regression exercise from the problem book

Extension: You could also try using an algorithm such as nearest neighbours classification, which takes a distance matrix of Word Mover Distances as its input.

Key learning objectives

- Understand what is meant by the term natural language processing, why it is important and some of the contexts it is currently used in
- Be able to give examples of some of the key techniques used in NLP and identify their respective strengths and limitations
- Describe the maths behind some of the more basic NLP techniques
- Develop confidence working with large datasets, and be able to describe the typical stages of data analysis when using NLP for classification
- Build a classifier, and iteratively improve it before submitting it for testing

Assessment of objectives:

1. An overview presentation on NLP aimed at a novice but mathematical audience. You might want to include definitions, a bit of history on how the area has developed, its current use in everyday life, and an explanation of some of the key techniques. Although not something we've asked you to do with this dataset, it is also worth researching a little on what a 'good' and 'bad' dataset might look like and how this might affect subsequent analysis.

2. Using either the skeleton code provided or your own versions, each produce a final Jupyter notebook that showcases your cleaned dataset, metrics, and best model.
3. Submit your best model to us for scoring: hopefully your model will classify the unseen data with a high level of accuracy!
4. Document your progress in a group written report detailing how you approached the project, the main challenges you came across, and how you achieved your goals.

Extension exercises

We anticipate that the four components as outlined above will contain enough material to occupy you for the duration of the project. However, if further exercises are required then we can help you to explore the following:

- Researching more advanced language models such as BERT and ELMO. The Hugging Face library contains most of the state-of-the-art models that students could tinker with.
- Exploring generative models (generating text) - here is a good example of a generative model in action.
- Learning about techniques for visualising word embeddings. TensorFlow has a good implementation of this that the students could explore.

Links for further research:

These are only suggestions - Google will provide you with loads of great options for learning in more detail!

Intro to Natural language processing:

https://www.youtube.com/watch?v=d4gGtcobq8M

https://towardsdatascience.com/your-guide-to-natural-language-processing-nlp-48ea2511f6e1

https://towardsdatascience.com/introduction-to-natural-language-processing-for-text-df845750fb63

Extension ideas for Parts 1-4 starting points:

https://towardsdatascience.com/topic-modeling-and-latent-dirichlet-allocation-in-python-9bf156893c24

https://towardsdatascience.com/machine-learning-algorithms-part-9-k-means-example-in-python-f2ad05ed5203

Recap of some useful maths with exercises

We are going to recap some of the key maths you might find useful in this project. Hopefully, you will have come across most of it before but make sure you are happy before getting stuck into the Jupyter notebooks and the rest of the project!

Histograms:

(https://revisionmaths.com/advanced-level-maths-revision/statistics/histograms-and-cumulative-frequency)

Histograms are similar to bar charts, but the area rather than height of each bar is defined as the frequency. In Python, the frequency density will usually be 1, so you can think of the histogram as a frequency count plot. You can vary the number of bins you use to get a more detailed or aggregate view of the data.

Exercise:

1, 2, 5, 8, 4, 6, 1, 2, 5, 8, 3, 1, 9, 10, 12, 2, 5, 7, 8, 4, 6, 9, 8, 2, 2, 12, 11, 1, 8, 7, 14, 15

The list of data above represents the sentence lengths for 32 pieces of text, where sentence length is defined as the number of words in a sentence.

1. Create a frequency table and then plot a corresponding histogram (with frequency density = 1) for the sentence length data with:
   a) 3 equal width bins
   b) 5 equal width bins

2. Using your frequency table from part 1. calculate the average sentence length for the collected data.

Sets

In maths a set is defined as a collection of distinct (different) objects, called elements. We say two sets are equal if they have the same elements. A set containing no elements is known as the empty set. Ordering does not matter in a set.

The intersection of two sets contains only the elements that appear in both sets.

The union of two sets contain all the elements that appear in either (or both) sets.

In our context, the elements of each set are likely to be words.

Example:

Sentence A: 'the cats sat on the mat' → Set A: ['the', 'cats', 'sat', 'on', 'mat']

(We've lost the second 'the' because sets only contain distinct elements)

Sentence B: cats are the best pets ever → Set B: ['cats', 'are', 'the', 'best', 'pets', 'ever']

Intersection of Set A and Set B: ['the', 'cats']

Union of Set A and Set B: ['the', 'cats', 'are', 'best', 'pets', 'ever', 'sat', 'on', 'mat']

Jaccard distance is defined as: $1 - \frac{Size\ of\ intersection\ of\ Sets\ A\ and\ B}{Size\ of\ union\ of\ Sets\ A\ and\ B}$

So the Jaccard distance of our two example sentences would be: $1 - \frac{2}{9} = \frac{7}{9}$

Exercise:

Write as sets, and then calculate the Jaccard distance between the two following sentences:

1. 'I like pineapple but chocolate has to be my favourite food or maybe chocolate coated pineapple'
2. 'I think chocolate is most people's favourite food.'

Vectors

In maths a vector ***v*** is a quantity that has both magnitude and direction.

You are probably already familiar with 2D vectors. We can represent 2D vectors visually using directed line segments or using what is known as column notation.

Column vectors looks like this: ***v*** = $\binom{x}{y}$ where x denotes the movement in the positive x direction from the origin and y denotes the movement in the positive y direction from the origin.

The magnitude of a vector in column form is denoted $|\boldsymbol{v}|$ and is calculated using the formula:

$|\boldsymbol{v}| = \sqrt{x^2 + y^2}$. You will sometimes also the notation $||v||$ in the notebooks – this means the same thing.

If you have two vectors, ***a*** = $\binom{x_1}{y_1}$ and ***b*** = $\binom{x_2}{y_2}$, then the Euclidean distance, **d** between these two vectors is defined as $\boldsymbol{d} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. This may also be denoted as $||a - b||$.

Normalising a vector makes the magnitude equal to 1. It does not change the direction. To do this you divide both the x and y components of the vector by the magnitude of the vector. Normalising our vectors will simplify some of the calculations needed when using them in our text context.

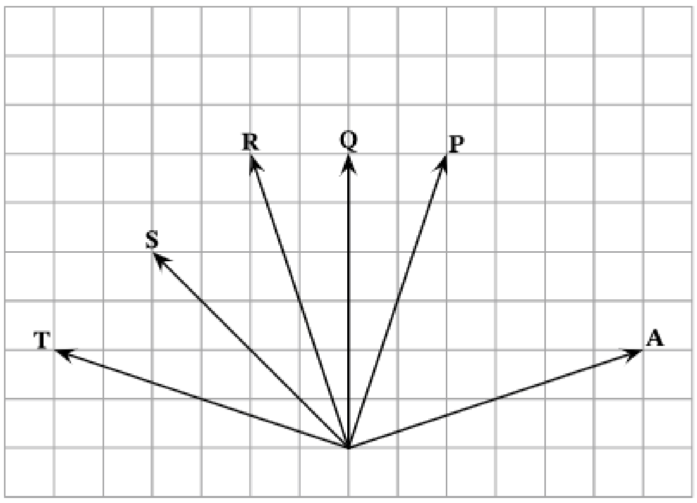
The dot product between two vectors is the sum of the products of corresponding entries in each vector. We will need it to calculate the cosine similarity later. For two vectors , ***a*** = $\binom{x_1}{y_1}$ and ***b*** = $\binom{x_2}{y_2}$, it is denoted as $a \cdot b = (x_1 \times x_2) + (y_1 \times y_2)$

Cosine similarity between two vectors is related to the angle between two vectors, and roughly determines if the vectors are pointing in the same direction.

The cosine similarity between two vectors, ***a*** = $\binom{x_1}{y_1}$ and ***b*** = $\binom{x_2}{y_2}$ is: $cos\theta = \frac{a \cdot b}{||a||||b||}$

Example:

Using the diagram below:



T = $\begin{pmatrix}-6\\2\end{pmatrix}$, ***A*** = $\begin{pmatrix}6\\2\end{pmatrix}$

$|\boldsymbol{T}| = \sqrt{(-6)^2 + (2)^2} = \sqrt{40} = 6.32\ (2\ dp)$

Euclidean distance between ***A*** and ***T*,** **d** = $\sqrt{(6\ -\ -6)^2 + (2\ - 2)^2}$ = $\sqrt{144} = 12)$

Normalised T = $\begin{pmatrix}-\frac{2}{12}\\ \frac{6}{12}\end{pmatrix} = \begin{pmatrix}-\frac{1}{6}\\ \frac{1}{2}\end{pmatrix}$

The dot product $\boldsymbol{A} \cdot \boldsymbol{T} = (\boldsymbol{-6 \times 6}) + (\boldsymbol{2 \times 2}) = \boldsymbol{-36 + 4} = \boldsymbol{-32}$

The cosine similarity between **A** and **T** is: **cos**$\boldsymbol{\theta} = \frac{\boldsymbol{A \cdot T}}{\|\boldsymbol{A}\|\|\boldsymbol{T}\|} = \frac{\boldsymbol{-32}}{\boldsymbol{12 \times 12}} = -\frac{\boldsymbol{2}}{\boldsymbol{9}}$

Exercise:

Using the diagram above:

1. Write the vectors **S, Q, A** and **P** as column vectors
2. Calculate the magnitude of vector **P**
3. Calculate the Euclidean distance between vectors **S** and **A**
4. Write the normalized version of **S**
5. Find the cosine similarity between **S** and **P**

The maths behind some of the NLP techniques explained with exercises

We are going to work through some of the text processing and embedding techniques mentioned in the workbooks by hand to get a better understanding of what is going on. The difficulty is indicated in brackets – please feel free to send us an email if you are stuck!

Stop words (Hard)

In the text pre-processing notebook we discuss removing stop words from your data, to improve your EDA. This question looks at how many words you might expect to be removed from your data set if you want to remove the 10 most frequently occurring words in the English language.

We have a corpus of size $N = 10^5$ and we remove the 10 most frequently occurring words i.e., 'the', 'and', 'of', etc.

Find bounds on the percentage of the corpus that we have removed.

You may assume that the frequency of the k^{th} most popular word is $1/_{kH_N}$ where $\log(N) \leq H_N \leq \log(N) + 1$. (This is a phenomenon known as Zipf's law)

Hint: See if you can first get a bounded inequality to describe the limits on $1/_{kH_N}$ *and then apply a suitable summation to ensure you are considering the case where we remove the 10 most frequently occurring words.*

Bag of words & TF-IDF (Medium)

Bag of words first requires you to create a vocabulary from all your available words. Once you have your vocabulary, you can create vectors to represent each piece of text. An easy way to do this is to count how many times each word from the vocabulary appears in the text, and if it doesn't appear at all to score it 0.

A limitation of this method is that the most frequent words start to have the highest scores. Some of the most common words in your text, are likely to be similar across all texts. For example, the word 'the'. These words don't really add much to the meaning of the text, compared to some of the less common more domain-specific words.

One way to address this is to use **term frequency-inverse document frequency** (TF-IDF). TF-IDF is a statistical measure used to evaluate the importance of a word to a specific text.

Term Frequency (TF) of a word, scores the frequency of the word in the current text:

$$\text{TF(word)} = \frac{Number\ of\ times\ word\ appears\ in\ the\ text}{Total\ number\ of\ words\ in\ the\ text}$$

Inverse Document Frequency (IDF) of a word, is a scoring of how rare the word is across all texts:

$$\text{IDF(word)} = \log(\frac{Total\ number\ of\ texts}{Number\ of\ texts\ with\ word\ in\ it})$$

*We usually use log base e.

TF-IDF(*word*) = TF(*word*) * IDF(*word*)

Example:
Our text data is shown below – we can assume it has already been pre-processed to remove punctuation and ignore case.

Index	Text data
0	a hippo is a large mammal
1	the largest mammal is the blue whale
2	a hippo is the third largest land mammal on earth
3	the elephant is the largest land mammal

The vocabulary just for the first sentence would be ['a', 'hippo', 'is', 'large', 'mammal']

The word vector for the first sentence would be [2, 1, 1, 1, 1] corresponding to the vocabulary ['a', 'hippo', 'is', 'large', 'mammal']. (Remember your vector length should be as long as your vocabulary).
To calculate the TF-IDF score for the word 'a', in text 1:
TF('a') = $\frac{2}{6}$ = 0.33 (2dp)
IDF('a') = log($\frac{4}{2}$) = log(2) = 0.69 (2dp)
TF-IDF('a') = 0.33 * 0.69 = 0.23 (2dp)

You can calculate the TF-IDF score for each word in each text, and end up filling a table that looks a bit like this:

Text Index	Vocab 'a'	Vocab 'hippo'	…..	….	….	….	Vocab 'elephant'
0	0.23	…					0
1	0						
2	0.069						
3	0						

These scores can be used as your text vectors.

Exercise:

1. Create the full vocabulary needed to work with this toy data
2. Score the sentences using bag of words with frequency counts to create vectors for each sentence
3. Score the sentences using TF-IDF to create vectors for each sentence

Word Movers Distance (Hard)

In the word embeddings and similarity metric notebooks, we discuss using the Word Movers distance, as a way to compare the similarity between two sentences. We are going to look at a very simplified example of WMD to try and understand a little better how it might be calculated. In reality, as you have seen in the notebook, we have a nice Python command to do this for us!

Example:

Let's take two sentences:

Sentence 1: 'squirrels and mice'
Sentence 2: 'grey and red'

Removing stop words these become:
Sentence 1: 'squirrels mice'
Sentence 2: 'grey red'

Our vocabulary is: ['squirrels, 'mice, 'grey, 'red']
Let's assume that we have already pre-processed these words using Word2Vec and been given the corresponding vectors: [(11, 8), (7, 7), (9, 9), (14, 11)]

We can find the distance between each possible pairing of words from sentence 1 and sentence 2 using the Euclidean distance formula we introduced in the previous exercises,
$\boldsymbol{d} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.
For example, the distance between 'squirrels' and 'grey' = $\sqrt{(11 - 9)^2 + (8 - 9)^2} = \sqrt{5}$.
You can store your results in a two-way table where the rows are each of the words in sentence 1, and the columns are each of the words in sentence 2. We have rounded our distances to 2dp.

	'grey'	'red'

‘squirrel’	2.24	4.24
‘mice’	2.83	8.06

Once you have populated your table, you then need to decide how to choose the pairings of most similar words. You might naturally think to go for the minimum distance – and this isn’t a bad idea, but what happens if one word serves as a minimum pair for multiple words in the opposite sentence?
In our example, you can see that the minimum distance for both ‘squirrel’ and ‘mice’ is to the word ‘grey’.

Assignment algorithms can help solve these kinds of problem. In an assignment problem, we must end up with a one-to-one pairing, which is the same as we require here.

We will use **the Hungarian algorithm**, to find our optimal assignment. Although it might seem overkill for this small example, remember in reality your sentences will be much longer, and your computer will have a lot more calculations to do. The WMD algorithm in Python doesn’t use this exact algorithm in its calculations, but the way it optimises pair selection is based on a very similar idea!

Hungarian algorithm explained on our example:
Step 1: For each row find the lowest element and subtract it from each element in that row

	‘grey’	‘red’
‘squirrel’	0	2
‘mice’	0	5.23

Step 2: For each column find the lowest element and subtract it from each element in that column

	‘grey’	‘red’
‘squirrel’	0	0
‘mice’	0	3.23

Step 3: Cover all zeros with the minimum number of lines possible

	‘grey’	‘red’
~~‘squirrel’~~	~~0~~	~~0~~
‘mice’	0	3.23

If the number of lines used is equal to the dimension of the table, then stop.
In our example, we used 2 lines and the dimension of our table is 2 x 2. So, we stop.

Step 4:
If less lines were used, you would continue the process, by finding the smallest element not covered by a line, and subtracting that value from all uncovered elements and adding it to any elements that are covered twice and repeat the algorithm.

If you have succeeded in stopping at Step 3, you have an optimal assignment. This will be found by looking at the zero elements covered by lines. Remember you can only choose one pair for each row. In our example, to ensure we have selected a zero in each column we choose: ‘squirrel’ and ‘red’, and ‘mice’ and ‘grey’ as our optimal pairing.

WMD is calculated by finding the average of all our optimal pairing scores. So in our case, the WMD would be: ½ (4.24 + 2.83) = 3.535

Python’s WMD method does a very similar set of calculations on your input sentences, but it can cope with much longer sentences and the word vectors are of much larger dimension, so the calculations quickly become complicated!

Exercise:

Let’s take two sentences:

Sentence 1: ‘I am cooking roast chicken’
Sentence 2: ‘Fried duck is for dinner’

Removing stop words this become:
Sentence 1: ‘cooking roast chicken’
Sentence 2: ‘fried duck dinner’

So our vocabulary is: [‘cooking, ‘roast, ‘chicken, ‘fried, ‘duck’, ‘dinner’]
Let’s assume that we have already pre-processed these words using Word2Vec and been given the corresponding vectors: [(1,2), (3, 1), (5, 2), (2, 2), (6, 3), (3,3)].

1. For each possible pairing of words, calculate the Euclidean distance between the word vectors and record them in the table below, we have started it for you:

	‘fried’	‘duck’	‘dinner’
‘cooking’	1		
‘roast’			
‘chicken’			

2. Perform the Hungarian Algorithm on your table to find out if an optimal cost exists
3. Calculate the WMD for these two sentences

Logistic regression (Medium)

In the classifiers notebook, we are trying to learn how to assign a class label to examples from our problem space: in our specific case, whether a headline is sarcastic or not. This is considered a binary classification problem since we have just two options: sarcastic, or non-sarcastic.

There are a number of different algorithm options for binary classification problems, but we are going to use logistic regression.

Logistic regression is named because it relies on the **logistic function** (also known as sigmoid function), for which we use the letter σ:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

The logistic function is increasing in x, and looks like an S shape. It is bounded between 0 and 1, never quite taking either of those values!

Logistic regression models the probability of the default class for a given data point:
e.g. in our case P(sarcastic=True| specific headline)

Our ‘probability’ formula is: hθ(x) = $\frac{1}{1+ e^{-(ax+b)}}$

Because we are doing **binary classification**, we will set a cut off probability and anything that scores lower than that will be considered non-sarcastic, and anything higher will be considered sarcastic.

For example, if we choose our cut off as 0.5, then if $\boldsymbol{h}\theta(text\ 1\ vector) = 0.7$, the classifier would predict is as sarcastic.

Exercise:

1. Plot the logistic function for x = [-5, 5]
2. Let x = 0.4. Calculate the output of the logistic regression function when:
   a) a = b = 1
   b) a = 0.5, b = 1
   c) a = 2, b= 1
   d) How do the values of a and b change the output of the logistic regression function? What is their importance in training our model?
3. To measure how well our logistic regression model is doing, we also introduce a **cost function**.
   a) Investigate what a cost function is and why we use it
   b) Investigate what the commonly used cost function is with logistic regression.