# 2021

1.

a)

i) **N-gram** method would be easy to implement and can capture short words completely and large parts of longer words. **NEED MORE INFO, 3 MARKS IS A LOT.**

ii) The tokens:

* **Dates** are hard to identify because there are so many “standards” for writing them. However, some rules can be written, including the fact that 4-digit numbers are usually years and that months cannot be digits higher than 12 and are sometimes written out as (sometimes shortened) words.
* **Shortenings** using periods are easy to mix up with sentence ending periods, but this can be solved by keeping a list of shortenings using periods.
* Proper nouns, e.g., “Coupe”, are tokens which are dependent on their capitalisation, so lowercase tokenisation of them loses information. The solution to this would be to keep the capitals to words which are not the starting words of sentences, then they would not lose this information.
* **OTHER POSSIBILITIES: PART NUMBERS**.

b)

i) Vocabulary: “1AB” x 26, “2BC” x 21, “3CD” x 27, “4DE” x 18, “5EF” x 20. Decision – any unknown items will be classified as “UNK”. **DECISION NEEDED FOR ORDER OF ITEMS (TERM FREQUENCY) + NEED JUST LIST OF ITEMS: [3CD, 1AB, 2BC, 5EF, 4DE, UNK].**

ii) Set A: **[9, 6, 0, 17, 0, 0]**  
Set B: **[10, 0, 9, 3, 11, 0]**  
Set C: **[8, 20, 12, 0, 7, 0]**

Term frequency (**BAG OF WORDS**) representation because the amount of bricks matters in a stock management system.

iii) sim (A, B) = (90 + 51) / sqrt(406) / sqrt(311) **0.397**

sim (A, C) = (72 + 120) / sqrt(406) / sqrt(657) **0.371**

sim (B, C) = (80 + 108 + 77) / sqrt(311) / sqrt(657) 0.586

**B and C** are most similar.

c)

i) Jaccard Similarity

ii) Pros:

* Satisfies triangle inequality
* Is a metric
* **FAST AND EFFICIENT TO COMPUTE**

Cons:

* No distinction between importance of different words
* Word frequencies are **ignored**
* Synonyms are as different as antonyms – no semantic analysis

2.

i)

K = 0.7

|  |  |
| --- | --- |
| Word | Unigram Probability |
| granola | (1 + 0.7) / (46 + 0.7\*27) 0.02619 |
| roll | (1 + 0.7) / (46 + 0.7\*27) 0.02619 |
| croissant | (2 + 0.7) / (46 + 0.7\*27) 0.04160 |

K= 0.07

|  |  |
| --- | --- |
| Word | Unigram Probability |
| granola | (1 + 0.07) / (46 + 0.07\*27) 0.0223 |
| roll | (1 + 0.07) / (46 + 0.07\*27) 0.0223 |
| croissant | (2 + 0.07) / (46 + 0.07\*27) 0.0432 |

The sample tables above show the effect on the relative probabilities as the K value varies. In particular, the larger values of K make the values more uniform. The smaller values accentuate the differences between smaller count values. For the correction task higher values of K would lead to less frequent words being selected with higher frequency than they would otherwise. The smoothing effectively makes the tail more uniform (e.g., a K value of 5 would treat roughly all terms occurring less than five times approximately uniformly). In contrast, a smaller k means less smoothing and the probability differences in terms with small counts will have greater effect.

ii)

Bigram probabilities for “i would like some breakfast”:

|  |  |
| --- | --- |
| Bigram | Probability |
| <s> i | 2003 / 8423 0.2378 |
| i would | 0 / 2623 + 0.4 \* (5 / 68237) 2.93 \* 10^-5 |
| would like | 2 / 5 = 0.4 |
| like some | 25 / 1522 = 0.0164 |
| some breakfast | 3 / 323 = 0.0093 |
| breakfast <e> | 2 / 6 = 0.3333 |

The probability of the whole sequence: 0.2378 \* 2.93 \* 10^(-5) \* 0.4 \* 0.0164 \* 0.0093 \* 0.3333 1.41678 \* 10^-10

Smoothing method is used because otherwise the lack of occurrences of “i would” would make the probability of the sentence occurring 0. With it, some leniency is allowed.

b) The model is **overfitted** to the training data because it does **extremely well on training** and **validation** data but **worse than the baseline** on the **test** data. This can be fixed by cross-validation, which randomly partitions data.

c)

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.linear\_model import LogisticRegression

# Data processing

data = ... # Loads a vector of raw text documents

train\_index = int(len(data) \* 0.8)

test\_index = train\_indextmp\_train = data[:train\_index,:]

validation\_split = int(train\_index \* 0.8)

train\_data = tmp\_train[:validation\_split,:]

validation\_data = tmp\_train[validation\_split:,:]

test\_data = data[test\_index:,:]

# Vectorization

one\_hot\_vectorizer = CountVectorizer(tokenizer=tokenize\_normalize, binary=True, max\_features=20000) # Reasonable number > 1k

one\_hot\_vectorizer.fit(train\_data)

train\_features = one\_hot\_vectorizer.transform(train\_data)

validation\_features = one\_hot\_vectorizer.transform(validation\_data)

test\_features = one\_hot\_vectorizer.transform(test\_data)

# Classification

lr = LogisticRegression(solver=’saga’, max\_iter=500)

lr\_model = lr.fit(train\_features, train\_labels)

evaluation\_summary("LR Train summary", lr\_model.predict(train\_features), train\_labels)

evaluation\_summary("LR Validation summary", lr\_model.predict(validation\_features), validation\_labels)

evaluation\_summary("LR Test summary", lr\_model.predict(test\_features), test\_labels)

[discussion]

3.

a)

i) HMM Emissions

|  |  |  |
| --- | --- | --- |
|  | NN | VB |
| Juliet | ¼ | 0 |
| love | 0 | 1 |
| people | ½ | 0 |
| Romeo | 1/4 | 0 |

ii) HMM Transitions

|  |  |  |  |
| --- | --- | --- | --- |
|  | NN | VB | <e> |
| <s> | 1 | 0 | 0 |
| NN | 0 | ½ | ½ |
| VB | 1 | 0 | 0 |

iii)

Romeo:  
P(NN|<s>, Romeo) = 1 \* ¼ = ¼

loves:  
P(VB|NN, love) = 1/2 \* 1 \* ¼ = 1/8

Juliet:  
P(NN|VB, Juliet) = 1 \* ¼ \* 1/8 = 1/32

# 2021 Resit

1.

a)

i) sim(D1, D2) = 6 / sqrt(12) / sqrt(18) = 0.408

**sim(D1, D3) = 6 / sqrt(12) / sqrt(4) = 0.866**

sim(D2, D3) = 6 / sqrt(18) / sqrt(4) = 0.707

**D1 and D3** are most similar

ii) IDF = log\_2(N/df)

IDF(the) = log\_2(4096/4096) = **0**

IDF(theory) = **4**

IDF(theology) = **9**

iii) The most discriminative is “**theology**” since it appears in the fewest documents.

b)

i) Unigram probability is contextless, it provides the probability that a word appears just based on its frequency divided by total amount of words.

Total = 200

|  |  |
| --- | --- |
| word | P(word) |
| euro | 0.375 |
| championship | 0.25 |
| Scotland | 0.125 |
| Wembley | 0.1 |
| ronaldo | 0.05 |
| referee | 0.05 |
| goal | 0.025 |
| defeat | 0.025 |

ii) Infinity since the term “euro” in D1 is not in D2.

iii) Text

Description automatically generated

D(D1||D2) = P(euro|D1) \* log2(P(euro|D1)/P(euro|D2)) +

P(championship|D1) \* log2(P(championship|D1)/P(championship|D2)) =

(0.7 \* 0.2 + 0.3 \* 0.375) \* log2((0.7 \* 0.2 + 0.3 \* 0.375) / (0.7 \* 0 + 0.3 \* 0.375)) +

(0.7 \* 0.05 + 0.3 \* 0.25) \* log2((0.7 \* 0.05 + 0.3 \* 0.25) / (0.7 \* 0.08 + 0.3 \* 0.25

2.

a) Probabilities:

|  |  |
| --- | --- |
| bigram | P (x\_i | x\_{i-1} ) |
| <s> the | 1096 / 2400 |
| the five | 227 / 3147 |
| five boxing | 17 / 821 |
| boxing wizards | 1 / 536 |
| wizards jump | 3 / 7 |
| jump quickly | 420 / 692 |
| quickly <e> | 500 / 587 |

P(sentence) = product of all probabilities = 2.819 \* 10^(-7)

b) The fit is **underfitted** since it is only marginally better than the baseline for the test data and **worse** than baseline for training and validation data. This indicates that there is some fault in the pipeline, for example, not training the model **long enough**. However, training too long endangers the model with **overfitting**, which could be solved by **stopping** when the **test fit** starts to **worsen** or by using **cross-validation**.

c)

i) **Hierarchical** (or K-Means) clustering since it does require knowing how many clustering groups before starting to cluster. It is an **unsupervised** algorithm and partitions data into **non-overlapping clusters**. If going top-down, it iteratively divides a group of cars into multiple groups, dividing those groups even more and so on.

ii) The elbow method charts number of clusters (K) as an independent variable against average within-cluster distance to centroid and chooses K at which the curve flattens and does not change the distance much with larger Ks.

3.

a) Although it would **improve efficiency**, it would be **worse** for semantic representation to do one-hot encoding than bag-of-words since it would cause **loss of information** of the frequency of words. [more explanation needed]

b)

i) The previously annotated material can be used to fine-tune BERT in order to encode the style and most likely words that the dictator would have used. The default **Masked Language Model** used by BERT’s training process would be ideal to help in this task, as given the context of a missing word/s, a most likely candidate can be proposed attending both to the extensive LM already learnt by a pre-loaded BERT model, and what was learnt through the fine-tuning process on previously existing transcribed content.

ii) **MLM** can also be used. In this case we are looking at which words may be “out of place”, i.e., not likely to have been pronounced by the dictator. By randomly hiding chunks of text, we can compare the estimations of our learnt model in predicting those words against the actual text. Assuming our fine-tuned model has a good understanding of how the dictator communicates and word choices, the impostor text should be statistically dissimilar to the estimations and proposed words given by the model. **Next sentence prediction** could also be a good answer here, as we can measure how the estimated sentence differs from the actual sentence. **Higher differences** could be a good indicator that the “imitator” is the author of that content.

4.

a)

i)

|  |  |  |  |
| --- | --- | --- | --- |
|  | NN | VB | MD |
| May | 1/3 | 0 | 1 |
| is | 0 | ½ | 0 |
| tomorrow | 2/3 | 0 | 0 |
| rain | 0 | ½ | 0 |

ii)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | NN | VB | MD | <e> |
| <s> | 1 | 0 | 0 | 0 |
| NN | 0 | 1/3 | 1/3 | 1/3 |
| VB | ½ | 0 | 0 | ½ |
| MD | 0 | 1 | 0 | 0 |

iii)

P(**NN**|May, <s>) = 1/3 \* 1 = 1/3  
P(…) = 0 (nothing but nouns can start a sentence)

P(NN|rain, NN) = 0 \* 0 = 0  
P(**VB**|rain, NN) = ½ \* 1/3 = 1/6  
P(MD|rain, NN) = 0

P(**NN**|tomorrow, VB) = 2/3 \* ½ = 1/3  
P(VB|tomorrow, VB) = 0  
P(MD|tomorrow, VB) = 0

* <s> -> NN -> VB -> NN -> <e>