University of Glasgow

**Tuesday, 4 May 2021**
**Available from 09:30 BST**
**Expected Duration: 1 hour 30 minutes**
**Time Allowed: 3 hours**
**Timed exam within 24 hours**

**DEGREES of MSci, MEng, BEng, BSc, MA and MA (Social Sciences)**

# TEXT AS DATA H
# COMPSCI 4074

**(Answer all 3 questions)**

**This examination paper is an open book, online assessment and is worth a total of 60 marks.**

**1. This question is about tokenization and similarity.**

**(a)** Consider the following piece of text:
["I placed an order (order# ZA10880) on 05-11-2020 for rear axle break discs for a BMW8 Coupe - part no. B342r. DH-D have lost the order. Please re-deliver ASAP."]
and a transformation of it that has been distorted:
["Iplacedanorder(order#ZA10880)on05-11-2020forrearaxlebreakdiscsforaBMW8Coupe-partno.B342r.DH-Dhavelosttheorder.Pleasere-deliverASAP."]

The result is a 'runaway token'.

(i) Describe a tokenization strategy to process the runaway token and split it into a series of "approximate" tokens as best as possible. No outside resources (e.g. vocabulary) are available. Justify your proposed tokenization method. [3]

> **Solution:** Given that no outside information is available, it's hard to break the text into 'words' in a straightforward way. What is needed here is a combination of a character n-gram or subwords based tokenizer combined with rules for punctuation and capitalization. I would propose character trigrams as a base with additional heuristics. These include: Trigrams should be broken on transitions from lowercase to uppercase. All sequences of uppercase characters should be kept as a single token. Similarly all sequences of digits should be kept as a single token, Sequences should be broken on periods (not very robust). 2 marks for talking about subword or character n-gram models
> 1 for reasonable heuristics and justification.

(ii) *Name* and *discuss* three classes of tokens that are hard to identify, used in the example provided above. Discuss why each is challenging and provide a solution to handle each type of challenge appropriately. [3]

> **Solution:** Abbreviations - e.g. DH-D are hard because they contain punctuation. This would need additional heuristics.
> Part numbers - B342r, different manufacturers' parts have different semantics.
> Dates - date formats are hard to separate from part numbers in some cases
> [1 mark for each class of tokens],
> [0.5 for name / description, 0.5 for why it is challenging and solution].

**(b)** You are designing a stock management system for 'GOLE' a manufacturer of children's building block sets. Each set contains several different types of bricks. The part codes and numbers of bricks are provided to you for each set: e.g. "3CD" $\times$ 10, "4DE" $\times$ 7, "1AB" $\times$ 5, "2BC" $\times$ 2.

Consider three different sets with the following collection of parts:

SetA: “1AB” × 6, “3CD” × 9, “5EF” × 17
SetB: “2BC” × 9, “3CD” × 10, “4DE” × 11, “5EF” × 3
SetC: “1AB” × 20, ”2BC” × 12, “3CD” × 8, “4DE” × 7

(i) Create a vocabulary for the data and provide it. Describe any important decisions in the creation of the vocabulary. [3]

**Solution:** Considering the term frequency the vocabulary should place the bricks in descending order of frequency.

| Part Code | Term Frequency |
|---|---|
| 1AB | 26 |
| 2BC | 21 |
| 3CD | 27 |
| 4DE | 18 |
| 5EF | 20 |
| UNK | |

Using term frequency to sort vocabulary gives: [3CD, 1AB, 2BC, 5EF, 4DE, UNK]

[2 points for the correct vocabulary (sorted order) ; 0.5 off for not including UNK]
[1 point for the justification of ordering (could also be done by document frequency)]

(ii) Using the vocabulary, provide a dense vector representation for each set (SetA, SetB, SetC). Justify the choice of set used. [3]

**Solution:**
The vector representation should be bag of words since the number of bricks in each set is important: (alternative answers are ”correct” answers if vocabulary in previous question was in alphabetical order).

| | | |
|---|---|---|
| SetA: [9, 6, 0, 17, 0, 0] | or [6, 0, 9, 0, 17, 0] | or [6, 9, 17, 0, 0, 0] |
| SetB: [10, 0, 9, 3, 11, 0] | or [0, 9, 10, 11, 3, 0] | or [0, 10, 3, 9, 11, 0] |
| SetC: [8, 20, 12, 0, 7, 0] | or [20, 12, 8, 7, 0, 0] | or [20, 8, 0, 12, 7, 0] |

[1 point for justification of Bag of words]
[2 points for correct vectors]
I’ve given marks for ”correct” vectors even if based on wrong vocabulary from previous question.

(iii) You are tasked with facilitating repurposing sets from one type to another. To do this you are asked to investigate which pairs of sets are most similar. Use Cosine similarity to calculate the similarity between each of the 3 pairs of sets (A&B, A&C, B&C) and identify which pair is most similar. [4]

**Solution:** To calculate Cosine Similarity we need to calculate
$A.B = 141, \quad B.C = 265, \quad A.C = 192,$
and
$|A| = \sqrt{406}, \quad |B| = \sqrt{311}, \quad |C| = \sqrt{657}.$
These give us
$\alpha_{A,B} = \frac{141}{\sqrt{406} \times \sqrt{311}} = 0.397,$
$\alpha_{B,C} = \frac{265}{\sqrt{311} \times \sqrt{657}} = 0.586,$
$\alpha_{A,C} = \frac{192}{\sqrt{406} \times \sqrt{657}} = 0.372.$
So B&C are the most similar pair

**(c)** (i) Suggest another similarity measure that could be used instead of cosine similarity for the above problem. [2]

**Solution:** Jaccard or Dice are the obvious alternative similarity measure but any appropriate measure should be accepted.

(ii) Discuss the Pros and cons of your suggested method. [2]

**Solution:** For instance, both Jaccard and Dice use one-hot encoded vectors. This makes them fast and efficient to compute [pro] but it also means that they do not assess the similarity in as fine detail[con]: for instance, by this measure a document with the word ”dog” 20 times is equally similar to a document with ”dog” 10 times or only once.

**2. This question is about language modeling and classification.**

**(a)** This task involves developing an order error corrector for a chain restaurant's breakfast order system. Below is a table of five separate order interactions transcribed from a mobile app.

| |
|---|
| forget it i wanna eat a croissant |
| no i wanna eat a croissant |
| do you serve granola? |
| i would like to have a bacon roll and a cappuccino |
| would you like coffee with that |

Table 1: Five interactions for a burger restaurant ordering system.

Sample text collections statistics for a bigram model are below:

- $V = 27$ unique words (including reserved tokens)
- $N = 46$ tokens, including padding

(i) Use the text provided in Table 1 above to compute word unigram probabilities. In a list or table format complete the probability table with Laplace smoothing that has $K = 0.7$. Show your workings. Discuss the impact on the probability values of increasing or decreasing the value of K. Describe the effect of varying $K$ when these probabilities are used in a spelling (error) correction task.

| Word | Unigram Probability |
|---|---|
| granola | |
| roll | |
| croissant | |

[5]

**Solution:**
Below is the solution with $K = 0.7$:

| Word | Unigram Probability |
|---|---|
| granola | (1 + 0.7) / (46 + 0.7*27) = 0.0262 |
| roll | (1 + 0.7) / (46 + 0.7*27) = 0.0262 |
| croissant | (2 + 0.7) / (46 + 0.7*27) = 0.0416 |

Samples values with $K = 0.07$:

| 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 |

Samples values with $K = 7$ (not required, but useful for illustration):

| Word | Unigram Probability |
|---|---|
| granola | (1 + 7) / (46 + 7*27) = 0.0340 |
| roll | (1 + 7) / (46 + 7*27) = 0.0340 |
| croissant | (2 + 7) / (46 + 7*27) = 0.0383 |

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.

Note: a N value of 35 is also technically valid here for unigrams (they removed padding tokens)
[2 marks for correct probability table]
[1 mark for discussion of K value impact]
[2 marks for discussion of the effect of K on the task]

(ii) A larger collection of restaurant ordering data is collected. It has the following statistics: $N = 68237$, $V = 1892$ from a total of 8423 documents (utterances).
Compute the bigram probability of the following sequence:

`[i would like some breakfast]`

with Stupid Backoff smoothing with default values. Collection statistics for the required terms are provided below. Show your workings, including each bigram's probability. Describe how and why a smoothing method is used here.

[6]

| Term | Count |
|---|---|
| i | 2623 |
| would | 5 |
| like | 1522 |
| some | 323 |
| breakfast | 6 |
| $\langle s \rangle$ i | 2003 |
| i would | 0 |
| would like | 2 |
| like some | 25 |
| some breakfast | 3 |
| breakfast $\langle e \rangle$ | 2 |

**Solution:**

| Bigram | Probability |
|---|---|
| $\langle s \rangle$ i | 2003 / 8423 = 0.2378 |
| i would | 0 / 2623 + 0.4* (5 / 68237) = 0.000029 |
| would like | 2/5 = 0.4000 |
| like some | 25 / 1522 = 0.0164 |
| some breakfast | 3 / 323 = 0.0093 |
| breakfast $\langle e \rangle$ | 2/6 = 0.3333 |

The probability of the sequence is the product of all bigram probabilities, approximately 1.40227e-10 (subject to rounding).
Stupid backoff is used because the bigram [i would] has a zero probability in the corpus. The default value is 0.4*p(might), which backs off to the unigram probability. It is needed to address the issue of sparsity for rare n-grams.
[2 mark for correct unsmoothed bigram probabilities; in particular 1 mark for the first case where the count for ¡s¿ must be inferred]
[1 mark for correct smoothed probability of i would.]
[1 mark for correct overall sequence probability; 0.5 for correct, 0.5 for showing the product of all bigrams]
[2 marks for discussion of how and why backoff is used in this instance.]

**(b)** The corrected trained LogisticRegression model has an accuracy of 92% on the training data, 87% on the validation data, and 45% on the test data. A dummy classifier has an accuracy of 50% on all three data subsets. Describe the model 'fit'. Suggest a way to fix this. [3]

**Solution:** The model is overfitting. We see this because it's accuracy is very high on the training data and poorer than a random dummy baseline for the test data. The test fit needs to be improved and this will involve accepting that the fit on the training data is not as high as 92%: this could be achieved by retraining from random hyperparameters and stopping the fitting as soon as the test fit begins to decrease even if the training fit is improving. You could also tune the features used (vary the number of features used, use n-grams, etc...) and possibly increase the size of the training data (it's unclear because the data size is not specified). You need to have 'many' more examples than features to be able to generalize effectively.

[2 marks for correctly identifying overfitting with a correct description]
[1 mark for a reasonable 'fix']

**(c)** Below is a snippet of code to vectorize and classify text with Scikit-learn. Assume that `tokenize_normalize` and `evaluation_summary` have been defined, as we did in the labs. The input data has been pre-processed into a vector of unnormalized text documents (each a single string).

```
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 = int(len(data) * 0.2)
tmp_train = data[:train_index,:]
validation_split = int(train_split * 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_features)
validation_features = one_hot_vectorizer.fit_transform(validation_data)
test_features = one_hot_vectorizer.transform(train_data)

# Classification
lr = LogisticRegression(solver='saga', max_iter=500)
lr_model = lr.fit(test_features, test_labels)
evaluation_summary("LR Train summary",
lr_model.predict(train_features), train_labels)
evaluation_summary("LR Validation summary",
lr_model.predict(validation_features), train_labels)
evaluation_summary("LR Test summary",
lr_model.predict(test_features), test_labels)
```

Copy and paste the code above and fix its mistakes. Although there may be more, discuss **three** important mistakes with their consequence, one from each section (data processing, vectorization, classification). [6]

**Solution:** There are multiple errors in each step of the processing.

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression

train_index = int(len(data) * 0.8) # Reasonable number > 0.6
tmp_train = data[:train_index,:]
validation_split = int(train_split * 0.8) # Reasonable number > 0.6
train_data = tmp_train[:validation_split,:]
validation_data = tmp_train[validation_split:,:]
test_data = data[train_index:,:]

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_features)
validation_features = one_hot_vectorizer.transform(validation_data)
test_features = one_hot_vectorizer.transform(test_data)

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)
```

- Data 1: The train/test split is 'wrong'; no need to calculate test_index, just take values from train index to end
- Data 2: The train and test sets overlap; they should be disjoint.
- Vectorization 1: only 20 most common features are used, which is a mistake; these are likely meaningless stopwords. Typical values are around 20000.
- Vectorization 2: - fit_transform is called on the validation data; only transform should be called. The wrong vectors are created (the vocabulary is wiped out)
- Classification 1 - The model is fit on the test data subset. It should be fit on train.
- Classification 2 - The model is tested on the test features with the validation labels.

[3 marks for correct fixed code (1 mark for each section)]
[3 marks correct and meaningful discussion of error (1 mark per section)]

**3.** This question is about word embedding models and Natural Language Processing.

**(a)** Using your knowledge of Part of Speech (PoS) Tagging by means of HMMs, build emission and transition tables from the training sentences S1 and S2. Then utilise the Viterbi algorithm to figure out the most likely PoS tagging sequence for test sentence T1.

*Note that smoothing is not required. I.e. let frequencies of 0 result in probability of 0.*

**Training sentences**: [

S1: ⟨s⟩ Juliet (NN) loves (VB) people (NN) ⟨e⟩
S2: ⟨s⟩ People (NN) love (VB) Romeo (NN) ⟨e⟩

]

**Testing sentence**: T1: ⟨s⟩ Romeo loves Juliet ⟨e⟩

(i) Build a HMM emissions table [5]

**Solution:**

| | Juliet | love | people | Romeo |
|---|---|---|---|---|
| NN | 1 | 0 | 1 | 1 |
| VB | 0 | 1 | 0 | 0 |

Note how each word in the training set has exactly one PoS assignment. Thus all non-zero values result to probability 1.

[5 marks for the right values in the Table]

(ii) Build a HMM transitions table [5]

**Solution:**

| | NN | VB | ⟨e⟩ |
|---|---|---|---|
| ⟨s⟩ | 1 | 0 | 0 |
| NN | 0 | 1/2 | 1/2 |
| VB | 1 | 0 | 0 |

Note how NN can either be followed by a verb or the end of the sentence with an equal probability of 0.5 each.

[5 marks for the right values in the Table]

(iii) Compute the most likely PoS tagging sequence using the Vitervi algorithm [3]

**Solution:** The resulting sequence will be: $\langle s \rangle \Rightarrow (NN) \Rightarrow (VB) \Rightarrow (NN) \Rightarrow \langle e \rangle$ . The values in the tables were designed to result in 0's for all but the right sequence, so computations are simplified.

[3 mark for proof of the computation to reach to right sequence]

**(b)** Gaelic is a language spoken by a recorded 87,056 people mostly in Scotland. There are many books providing translations between english and gaelic. Below are some sample Gaelic-English translations:

| Gaelic | Approximate English Translation |
|---|---|
| De an t-ainm a tha' oirbh? | What's Your Name? |
| Ciamar a tha sibh? | How are you? |
| Tha mi duilich | I'm sorry |
| Tha beatha ro ghoirid | Life is too short |

Figure 1: Sample sentences in Gaelic

Imagine you are building a prototype translation software that relies on advanced NLP techniques such as BERT to produce high quality contextualised translations. Unfortunately you have not found any pre-trained BERT models online, so you decide to train a new model yourself on existing literature that has been published in both English and Gaelic

(i) Describe the process for pre-training BERT on Gaelic. Briefly describe what is required and any changes needed for the model. Be sure to include discussion of tokenisation considerations and training objectives. [7]

**Solution:** As observed in the sample sentences, without prior knowledge of how to tokenise Gaelic meaningfully it may be tricky to deal with some words. However the byte pair tokenisation used by BERT will learn meaningful subword pieces as part of the training process. It is also important to keep spaces, commas and quotes as characters in the processing since important information could be conveyed through their presence.
The rest is unchanged - use the default Masked Language Modelling (MLM) objective to hide a random token (in this case a character) from an input sequence to BERT with 15% probability.

[3 mark for the tokeniser changes]
[3 mark for training objectives)]