COMP4650/6490 Document Analysis Assignment 2 – ML

This assignment has 3 questions you should attempt all questions. This is an individual assignment, so you should complete it by yourself.

In this assignment you will:

1. Develop a better understanding of how machine learning models are trained in practice, including partitioning of datasets and evaluation.
2. Become familiar with the sklearn package for machine learning with text
3. Become familiar with the Pytorch framework for implementing neural network-based machine learning models.

Throughout this assignment you will make changes to provided code to improve or complete existing models. In some cases, you will write your own code from scratch after reviewing an example. In addition, you will produce an answers file with your responses to each question. Your answers file must be a .pdf file named u1234567.pdf where u1234567 is your Uni ID. You should submit a .zip file containing all of the code files and your answers pdf file, **BUT NO DATA.**

Your answers to coding questions (or coding parts of each question) will be marked based on the quality of your code (is it efficient, is it readable, is it extendable, is it correct).

Your answers to discussion questions (or discussion parts of each question) will be marked based on how convincing your explanations are (are they sufficiently detailed, are they well-reasoned, are they backed by appropriate evidence, are they clear, do they use appropriate visual aids such as tables, charts, or diagrams).

Question 1: Movie Review Sentiment Classification with Dense Word Vectors (25%)

For this question you have been provided with a movie review dataset. The dataset consists of 50,000 review articles written movies on IMBD, each labelled with the sentiment of the review – either positive or negative. Your task is to apply logistic regression with dense word vectors to the movie review dataset to predict the sentiment label from the review text.

A simple approach to building a sentiment classifier is to use train a logistic regression model that uses aggregated pre-trained word embeddings. While this approach, with simple aggregation, normally works best with short sequences you will try it out on the movie reviews.

You have been provided with a file *dense_linear_classifier.py* which reads in the dataset and splits it into training, testing, and validation sets; and then loads the pre-trained word embeddings. These embeddings were extracted from the *spacy 2.3.5* python package “en_core_web_md” model and, to save disk space, were filtered to only include words that occur in the movie reviews. Your task is to use a logistic regression classifier with aggregated word embedding features to determine the sentiment labels of documents from their text. First implement the *document_to_vector* function which converts a document into a vector by first tokenizing it (the TreebankWordTokenizer in the nltk package would be an excellent choice) and then aggregating the word embeddings of those words that exist in the dense word embedding dictionary. You will have to work out how to handle words that are missing from the dictionary. For aggregation, the *mean* is recommended but you could also try other functions such as *max*. Next, implement the *fit_model* and *test_model* functions using your *document_to_vector* function and LogisticRegression from the sklearn package. Using *fit_model*, *test_model*, and your training and validation sets you should then try several values for the regularization parameter C and select the best based on accuracy. To try regularization parameters, you should use an automatic hyperparameter search method. Next, re-train your classifier using the training set concatenated with the validation set and your best C value. Evaluate the performance of your model on the test set.

Answer the following questions in your **answer pdf**:

1. What range of values for C did you try? Explain, why this range is reasonable. Also explain what search technique you used and why it is appropriate here.
2. What was the best performing C value?
3. What was your final accuracy?

Also make sure you submit your code.

```
C    set: [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]
base set: [2, 3, 4, 5, 6, 7, 8, 9, 10]
base score: [0.8392, 0.8298, 0.8214, 0.8138, 0.8042, 0.7964, 0.7896, 0.78, 0.7726]
Best b: 2
C_score: [0.807, 0.8214, 0.83, 0.8392, 0.8446, 0.8496, 0.8486, 0.8488, 0.8504, 0.852, 0.8528]
Best_C: 5
------------------------------------------------------------------
Final accuracy: 0.8544
```

Hint: If you do the practical exercise in Lab 2 this question will be much easier.

Tip: If you use TreebankWordTokenizer then for efficiency you should instantiate the class as a global variable. The TreebankWordTokenizer compiles many regular expressions when it is initialized; doing this every time you want to tokenize a sentence is very inefficient.

Documentation for TreebankWordTokenizer
https://www.nltk.org/_modules/nltk/tokenize/treebank.html

n. 类型；种类；体裁；样式；
流派；风俗画

Question 2: Kaggle Task - Genre Classification (50% total: 30% competition, 20% writeup)

For this task you will design and implement a classification algorithm that identifies the genre of a piece of text. This task will be run as a competition on Kaggle. Your marks for this question will be partially based on your results in this competition, but your mark will not be affected by other students' scores, instead you will be graded against several solutions developed by the convener and tutor team. The other part of your mark will come from your code and write-up.

恐怖

The dataset consists of text sequences from English language books in the genres: horror (class id 0), science fiction (class id 1), humor (class id 2), and crime fiction (class id 3). Each text sequence is 10 contiguous sentences. Your task is to build the best classifier when evaluated with **macro averaged F1 score**.

Note: the training data and the test data come from different sets of books. You have been provided with bookids (examples with the same bookid come from the same book) for the training data but not the test data.

You have been provided with an example solution in *genre_classifier_0pc.py* this shows you how to read in the training data (*genre_train.json)*, test data (*genre_test.json*) and output a csv file that the judging system can read. This solution is provided only as an example (it is the 0% benchmark for this problem), you will want to build your own solution from scratch.

Setup:

Please register for Kaggle **using your university email address** (https://www.kaggle.com/).

Submit solutions to: https://www.kaggle.com/t/042af1cd70dc4292b6d71d618cb5ef03
Set your team name to your uid (e.g. u1234567). This will allow us to match your submission to your assignment for marking purposes. ("team name" is a kaggle term, this is an individual assignment, and so you should not work with others to complete it)

If you are unable or unwilling to use Kaggle, you may submit a csv file along with your assignment. It should be named with your uid e.g. u1234567.csv . If you use Kaggle you should not submit this file with your assignment.

Rules:

- Do not use additional supervised training data. That is, you are not allowed to collect a new genre classification dataset to use for training. Pre-training on other tasks is permitted.
- Do not use models pre-trained on genre classification. Models pre-trained on other tasks are permitted (eg you may use word vectors from *spacy* or *gensim*, or pre-trained transformers from *transformers*).
- You can use the following libraries (in addition to python standard libraries): *numpy*, *scipy*, *torch*, *transformers*, *tensorflow*, *nltk*, *sklearn*, *xgboost*, *pandas*, *gensim*, *spacy*. If you would like to use other libraries, please ask on the piazza forum.
- This is an individual task, do not collude with other individuals. **Copying code from other people models or models available on the internet is not permitted.**

Juding system:

- You will upload your predictions for the test set as a csv file for judging.
- You are allowed to **submit up to 5 times per UTC Day.** Since you get immediate feedback from every submission it is best to start submitting early and plan ahead.
- The results shown on the public scoreboard prior to the conclusion of the contest only include 50% of the test data. Your solution will be judged on the **other** 50% of the test data when computing final rankings and marks.
- The judging system allows you to choose which of all your submissions you want to be your final one.

Competition Marking:

Marks will be assigned based on which judge baselines you beat on the hidden 50% of the test data. If, for example, you beat the 80% baseline but do not beat the 90% baseline you will be awarded a mark on a linear scale between these two based on your macro averaged f1 score. Exceeding the score of the 100% baseline will give you a mark of 100% for the **competition component** of this question. It will also award bragging rights. (The 100% baseline and all other baselines can be trained in less than 24 hours on a laptop pc without GPU support -- your solution may make use of any compute resources available to you).

Write up:

A fraction of your marks will be based on a write up that a minimum describes:

- How your final solution works
- How you trained and tested your model (e.g. validation split(s), hyperparameter search ...)
- What models you tested, which worked, and which didn't
- Why you think these other models didn't work

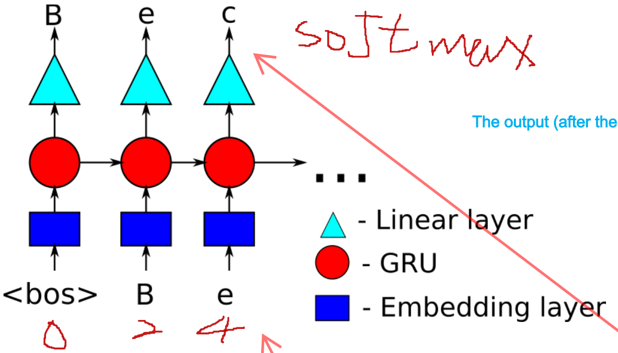
Aim for 1 page or slightly less. Only the first 2 pages will be marked. Bullet points are acceptable if they are understandable.

What to submit:

- The code of your best solution, including training pipeline. **Also make sure your team name is set to your uid on kaggle. Do not submit stored parameters or data.**
- Your write-up in your answer pdf file
- If you could not use Kaggle (**and only if you could not use Kaggle**) a <your_uid>.csv file with your models output in the correct judging format.

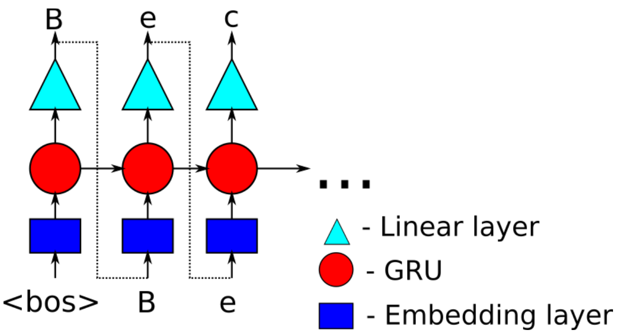
Question 3: RNN Name Generator (25%)

Your task is to develop an autoregressive RNN model which can generate people's names. The RNN will generate each character of a person's name given all previous characters. Your model should look like the following:



The output (after the softmax) is a categorical probability distribution over the vocabulary

Note that the input is shown here as a sequence of characters but in practice the input will be a sequence of character ids. There is also a softmax non-linearity after the linear layer but this is not shown in the diagram. The output (after the softmax) is a categorical probability distribution over the vocabulary, what is shown as the output here is the ground truth label. Notice that the input to the model is just the expected output shifted to the right one step with the <bos> (beginning of sentence token) prepended. The three dots to the right of the diagram indicate that the RNN is to be rolled out to some maximum length. When generating sequences, rather than training, the model should look like the following:



Specifically, we choose a character from the probability distribution output by the network and feed it as input to the next step. Choosing a character can be done by sampling from the probability distribution or by choosing the most likely character (otherwise known as argmax decoding).

The character vocabulary consists of the following:

“” : The null token padding string
<bos> : The beginning of sequence token
. : The end of sequence token
a-z : All lowercase characters
A-Z : All uppercase characters
0-9 : All digits
“ “ : The space character

Starter code is provided in *rnn_name_generator.py*, and the list of names to use as training and validation sets are provided in *names_small.json*. To complete this question you will need to complete three functions and one class method: the function *seqs_to_ids*, the *forward* method of *RNNLM*, the function *train_model*, and the function *gen_string*. In each case you should read the description provided in the starter code.

seqs_to_ids - Takes as input a list of names. Returns a 2d numpy matrix containing the names represented using token ids. All output rows (each row corresponds to a name) should have the same length of max_length. Achieved by either truncating the name or padding it with zeros. For example, an input of:

[“Bec.”, “Hannah.”, “Siqi.”] with a max_length set to 6 should return (normally we will use max_len = 20 but for this example we use 6)

```
[[30  7  5  2  0  0]
 [36  3 16 16  3 10]
 [47 11 19 11  2  0]]
```

Where the first row represents “Bec.” and two padding characters, the second row represents “Hannah”, the third row represents “Siqi.” with one padding character.

forward – A method of RNNLM. In this function you need to implement the GRU model shown in the diagram above. The layers have all been provided for you in the class initializer.

train_model – In this method you need to train the model by batch gradient decent. The optimizer and loss function are provided to you. Note that the loss function takes logits (out put of the linear layer before softmax is applied) as input. At the end of every epoch you should print the validation loss using the provided *calc_val_loss* function.

gen_string – In this method you will need to generate a new name, one character at a time. You will also need to implement both sampling and argmax decoding.

For this question, please include in your **answers pdf** the most likely name your code generates using argmax decoding as well as 10 different names generated using sampling. Also remember to **submit your code**. Your code should all be in the original *rnn_name_generator.py* file, other files will not be marked. For this question **you should not import additional libraries,** use only those provided in the starter code (you may uncomment the import tqdm statement if you want to use it as a progress bar).

For example, one of the names sampled from the model solution was: Dasbie Miohmazie