Birkbeck

(University of London)

M.Sc. Coursework

School of Computing and Mathematical Sciences

Natural Language Processing (COIY064H7)

Question:	1	2	Total
Points:	50	50	100

- For this module, 60% of the overall marks are for your coursework (this programming assignment) and 40% are for an online quiz.
- The coursework consists of a programming assignment in two parts, with equal weight.
- The coursework is marked out of a total of 100.
- You must complete the coursework using the Python 3 programming language. You may submit an .ipynb notebook or a regular python script.
- In some parts, the questions specify that you must implement something yourself, or use a particular python library, but otherwise you may use any widely available python libraries.
- Most of the question parts only require code to answer. For a few parts you are asked additional questions requiring brief text answers — you should edit the answers.txt included in the github template to provide these.
- A github classroom assignment is made available on the Moodle Coursework tile for you to submit your work. A portion of the marks for both parts are awarded for having a clear and consistent github commit history
- You must also confirm your github submission via the "confirm your GitHub repository submission" form on Moodle.

1. Part One — Syntax and Style

   In the first part of your coursework, your task is to explore the syntax and style of a set of 19th Century novels using the methods and tools that you learned in class.

   The texts you need for this part are in the novels subdirectory of the texts directory in the coursework Moodle template. The texts are in plain text files, and the filenames include the title, author, and year of publication, separated by hyphens. The template code provided in PartOne.py includes function headers for some sub-parts of this question. The main method of your finished script should call each of these functions in order. To complete your coursework, complete these functions so that they perform the tasks specified in the questions below. You may (and in some cases should) define additional functions.

   (a) read_novels: Each file in the novels directory contains the text of a novel, and the name of the file is the title, author, and year of publication of the novel, separated by hyphens. Complete the python function read_texts to do the following: [4]

      i. create a pandas dataframe with the following columns: text, title, author, year

      ii. sort the dataframe by the year column before returning it, resetting or ignoring the dataframe index.

   (b) nltk_ttr: This function should return a dictionary mapping the title of each novel to its type-token ratio. Tokenize the text using the NLTK library only. Do not include punctuation as tokens, and ignore case when counting types. [4]

   (c) flesch_kincaid: This function should return a dictionary mapping the title of each novel to the Flesch-Kincaid reading grade level score of the text. Use the NLTK library for tokenization and the CMU pronouncing dictionary for estimating syllable counts. [4]

   (d) When is the Flesch Kincaid score *not* a valid, robust or reliable estimator of text difficulty? Give two conditions. (Text answer, 200 words maximum). [4]

   (e) parse: The goal of this function is to process the texts with spaCy's tokenizer and parser, and store the processed texts. Your completed function should: [9]

      i. Use the spaCy nlp method to add a new column to the dataframe that contains parsed and tokenized Doc objects for each text.

      ii. Serialise the resulting dataframe (i.e., write it out to disk) using the pickle format.

      iii. Return the dataframe.

      iv. Load the dataframe from the pickle file and use it for the remainder of this coursework part. Note: one or more of the texts may exceed the default

maximum length for spaCy's model. You will need to either increase this length or parse the text in sections.

(f) Working with parses: the final lines of the code template contain three for loops. Write the functions needed to complete these loops so that they print:

i. The title of each novel and a list of the ten most common syntactic objects overall in the text. [5]

ii. The title of each novel and a list of the ten most common syntactic subjects of the verb 'to hear' (in any tense) in the text, ordered by their frequency. [5]

iii. The title of each novel and a list of the ten most common syntactic subjects of the verb 'to hear' (in any tense) in the text, ordered by their Pointwise Mutual Information. [5]

(g) Ten marks are allocated for your github commit history. You should make regular, atomic commits with concise but informative commit messages. See the section titled Submission (both questions) for more details. [10]

Part One total marks: 50

2. Part Two — Feature Extraction and Classification

In the second part of the coursework, your task is to train and test machine learning classifiers on a dataset of political speeches. The objective is to learn to predict the political party from the text of the speech. The texts you need for this part are in the speeches sub-directory of the texts directory of the coursework Moodle template. For this part, you can structure your python functions in any way that you like, but pay attention to exactly what information (if any) you are asked to print out in each part. Your final script should print out the answers to each part where required, and nothing else.

(a) Read the hansard40000.csv dataset in the texts directory into a dataframe. Subset and rename the dataframe as follows: [8]

i. rename the 'Labour (Co-op)' value in 'party' column to 'Labour', and then:

ii. remove any rows where the value of the 'party' column is not one of the four most common party names, and remove the 'Speaker' value.

iii. remove any rows where the value in the 'speech_class' column is not 'Speech'.

iv. remove any rows where the text in the 'speech' column is less than 1000 characters long.

Print the dimensions of the resulting dataframe using the shape method.

(b) Vectorise the speeches using TfidfVectorizer from scikit-learn. Use the default parameters, except for omitting English stopwords and setting max_features to 3000. Split the data into a train and test set, using stratified sampling, with a random seed of 26. [5]

(c) Train RandomForest (with n_estimators=300) and SVM with linear kernel classifiers on the training set, and print the scikit-learn macro-average f1 score and classification report for each classifier on the test set. The label that you are trying to predict is the 'party' value. [5]

(d) Adjust the parameters of the Tfidfvectorizer so that unigrams, bi-grams and tri-grams will be considered as features, limiting the total number of features to 3000. Print the classification report as in 2(c) again using these parameters. [5]

(e) Implement a new custom tokenizer and pass it to the tokenizer argument of Tfidfvectorizer. You can use this function in any way you like to try to achieve the best classification performance while keeping the number of features to no more than 3000, and using the same three classifiers as above. Print the classification report for the best performing classifier using your tokenizer. Marks will be awarded both for a high overall classification performance, and a good trade-off between classification performance and efficiency (i.e., using fewer parameters). [9]

(f) Explain your tokenizer function and discuss its performance. [8]

(g) Ten marks are allocated for your github commit history. You should make regular, atomic commits with concise but informative commit messages. See the section below titled Submission (both questions) for more details. [10]

Part Two total marks: 50

Submission (both questions)

Submit using github classroom and confirm on Moodle

The code template for this coursework part is made available to you as a Git repository on GitHub, via an invitation link for GitHub Classroom.

1. First you follow the invitation link for the coursework that is available on the Moodle page of the module.

2. Then clone the Git repository from the GitHub server that GitHub will create for you. Initially it will contain README.md and a folder structure for Part One and Part Two with placeholder python scripts (you can change these to Jupyter notebook files if you prefer).

3. Enter your name in README.md (this makes it easy for us to see whose code we are marking)

4. You must also enter the following Academic Declaration into README.md for your submission:

> "I have read and understood the sections of plagiarism in the College Policy on assessment offences and confirm that the work is my own, with the work of others clearly acknowledged. I give my permission to submit my report to the plagiarism testing database that the College is using and test it using plagiarism detection software, search engines or meta-searching software."

This refers to the document at:

https://www.bbk.ac.uk/student-services/exams/plagiarism-guidelines

A submission without the declaration will get 0 marks.

5. Whenever you have made a change that can "stand on its own", say, "Implemented tokenizer method", this is a good opportunity to commit the change to your local repository and also to push your changed local repository to the GitHub server.

   As a rule of thumb, in collaborative software development it is common to require that the code base should at least still compile after each commit.

Entering your name in README.md (using a text editor), then doing a commit of your change to the file into the local repository, and finally doing a push of your local repository to the GitHub server would be an excellent way to start your coursework activities.
You can benefit from the GitHub server also to synchronise between, e.g., the Birkbeck lab machines and your own computer. You push the state of your local repository in the lab to the GitHub server before you go home; later, you can pull your changes to the repository on your home computer (and vice versa).
Use meaningful commit messages (e.g., "Implemented the pickle output for Q1(e)", or "fixed a bug in the PMI calculation"), and do not forget to push your changes to the GitHub server! For marking, we plan to clone your repositories from the GitHub server shortly after the submission deadline.
We additionally require you to confirm your github submission via a 'confirm submission' form on Moodle. The time of this confirmation will tell us if you would like your code to be considered for the regular (uncapped) deadline or for the late (capped) deadline two weeks later.

Deadlines

- The submission deadline is: 26th June 2025, 14:00 UK time.
- The late cut-off deadline for receiving a capped mark is: 10th July 2025, 14:00 UK time.

