UNIVERSITY OF BIRMINGHAM

The University of Birmingham

School of Computer Science

Assignment 2 – Advanced News Classifier
Deadline: 12:00pm, Dec 4, 2023

Author:
Jizheng Wan

Reviewers:
Pieter Joubert
Ahmad Ibrahim
Archie Powell

Version 1.2

An Assignment submitted for the UoB:

Object Oriented Programming

Nov 27, 2023

Revision History

Revision	Date	Author(s)	Description
1.0	27/11/2023	JW	First version.
1.1	28/11/2023	JW	(a) Added a hint in Task 6.2. (b) Updated the description of Task 3.2. "-1" should be a string. (c) Updated the description of Task 5.1. You can modify the existing code in the constructor. (d) Added a hint in Task 5.4.
1.2	30/11/2023	JW	(a) Added a hint in Task 4.2. (b) Corrected the "Expected Output" in this document. The result included in the jUnit test is the correct one. (c) Added a hint in Task 6.6.

Contents

Rules

1. For each class refer to its corresponding test to verify field and method naming conventions.
2. Although there are many ways to construct an application, you are required to adhere to the rules as stipulated below (to achieve marks).
3. If variable names are not stipulated, you can use your own names for variables. This shows that you have written the application (we will check for plagiarism).
4. Inclusion of extra imports is strictly prohibited and will lead to a substantial penalty.
5. Do NOT change or modify files included in the "resources" folder.
6. Do NOT modify the skeleton code. However, you are allowed to create your own methods if they are needed.
7. You MUST complete this assignment independently – Do NOT discuss or share your code with others, and Do NOT use ChatGPT! Any cheating behaviour will result in a zero score for this module and will be subject to punishment by the University.
8. It is *STRONGLY ADVISED AGAINST* utilizing any translation software (such as Google Translate) for the translation of this document.
9. The jUnit tests included in the skeleton code are basic and only scratch the surface in evaluating your code. Passing these tests does not guarantee a full mark.
10. Wrong file structure leads to a substantial penalty. Make sure you have followed the Submission Instructions on the Canvas page (the assignment page).
11. Creating your own .zip file without using the export function in IntelliJ may lead to a wrong file structure.

HINT: You can use the TODO window in IntelliJ (View | Tool Windows | TODO) to quickly jump between tasks.

1 Introduction

In the last assignment, you built a news classifier by using TF-IDF and Cosine Similarity. This approach proved effective in numerous situations, with its primary benefit being its simplicity in implementation. However, there are several disadvantages, such as:

- Lack of contextual understanding. TF-IDF focuses on the frequency of words but doesn't capture the context in which they are used. This can lead to misinterpretation of the text's meaning, especially with homonyms or phrases where the meaning depends on the context.
- Ignoring word order. TF-IDF treats documents as a "bag of words", meaning it loses the order of words. This is a significant limitation, as the sequence of words can drastically change the meaning of sentences.
- Computational complexity for large datasets. The method can become computationally intensive as the size of the dataset and vocabulary grows, making it less efficient for very large corpora.
- High dimensionality. TF-IDF can lead to very high-dimensional feature spaces, especially with large text corpora.

In comparison, more advanced techniques like word embeddings (e.g., Word2Vec [3–5], GloVe [6]) and transformer-based models (e.g., BERT [1], GPT [7]) provide a more nuanced understanding of language by capturing contextual meanings, semantic relationships, and the order of words.

Hence, in this assignment, you are tasked with constructing an advanced new classifier utilizing GloVe Embedding and Machine Learning. You are not required to understand how Glove works or prior knowledge of Machine Learning, as this assignment provides an existing GloVe file and incorporates two external libraries: Deeplearning4J [8] and NDArray4J [9] which facilitate the Machine Learning processes.

However, you do need to understand the structure of the Glove file to build the input of the neural network.

1.1 Glove file

The file is called "*glove.6B.50d_Reduced.csv*" and is located in the "resources" folder. It was trained based on Wikipedia 2014 [1] + Gigaword 5 [2], which contains 6 billion tokens. Originally, there were 400,000 words included in this model. For demonstration purposes, we have reduced its size to only include 38,534 unique words. Below is an example of how this file is structured:

```
abacus,0.9102,-0.22416,0.37178,0.81798,...,0.34126
abadan,-0.33432,-0.95664,-0.23116,0.21188,...,-0.23159
```

[1] https://dumps.wikimedia.org/enwiki/20140102/

[2] https://catalog.ldc.upenn.edu/LDC2011T07

```
abalone,0.34318,-0.8135,-0.99188,0.6452,0.0057126,...,-0.15903
   .
   .
   .
zygote,0.78116,-0.49601,0.02579,0.69854,...,-0.40833
zymogen,-0.34302,-0.76724,0.13492,-0.0059688,...,0.37539
```

Each line starts with a unique word (so 38,534 lines in total), then followed by 50 floating numbers (separated by ","). These floating numbers are the vector representation of that word. In other words, each unique word is associated with a size/length 50 vector. Elements in this vector must be consistent with the order of the floating numbers in the CSV file. Using the word "abacus" as an example, the first element in its vector representation should be "0.9102", then the second element is "-0.22416", and so on and so forth.

2 Task 1 - Glove.java [2.5 Marks]

The Glove class consists of GloVe objects, and you need to complete the following methods to finish this class. *strVocabulary* is the attribute of the word stored in this Glove object, and *vecVector* is its vector representation.

Testing this class with the *GloveTest.java* file.

2.1 Task 1.1 - $Glove(String\ \_vocabulary, Vector\ \_vector)$ [0.5 Marks]

This is the constructor of the Glove class.

- Complete this constructor by assigning the *_vocabulary* to the *strVocabulary* attribute, and *_vector* to *vecVector*.

2.2 Task 1.2 - Task 1.5 [0.5 Marks each]

- Complete the relevant get and set methods accordingly.

3 Task 2 - NewsArticles.java [3.5 Marks]

This class holds the basic information about the news articles located in the *resources\News* folder:

1. *newsTitle*: stores the title of the news.
2. *newsContent*: stores the content of the news.
3. *newsType*: in Machine Learning, it is essential to divide the data into two distinct subsets: Training and Testing. This particular variable (or attribute) serves the

purpose of identifying whether a given news article is part of the Training set or the Testing set.

4. *newsLabel*: in Machine Learning, a "label" refers to the output or target variable a model tries to predict or classify. It's an integral part of supervised learning, and the goal is to learn a mapping from input data to labels based on example input-output pairs. In this assignment, a label represents which group a given news article belongs to. For example, if there are two groups, the label should be either 1 (the first group) or 2.

   This assignment initially provides only the training set data with corresponding labels. The ultimate goal is to develop a machine-learning model that predicts the labels for the testing set data.

3.1 Task 2.1 - Task 2.7 [0.5 Marks each]

- Complete the constructor and the relevant get & set methods accordingly.

4 Task 3 - HtmlParser.java [3 Marks]

Similar to Assignment 1, the *HtmlParser* class provides various methods to retrieve related information from news articles. The *getNewsTitle(String _htmlCode)* and *getNewsContent(String _htmlCode)* methods are provided already, and this task focuses on the methods that allow you to get the data type and label information.

4.1 Task 3.1 - $getDataType(String\ \_htmlCode)$ [1.5 Marks]

The data type information is located between the *<datatype></datatype>* tag.

- If the article does not contain this tag, then consider it as Testing data. Otherwise, return the data type accordingly.
- The return type should be the enum defined in the NewsArticles class.

HINT: Enumerated data type (enum) is introduced in Chapter 8 - Arrays in the textbook.

4.2 Task 3.2 - $getLabel(String\ \_htmlCode)$ [1.5 Marks]

The label information is located between the *<label></label>* tag.

- If the article does not contain this tag, then return "-1" (as a string). Otherwise, return the label accordingly.

5 Task 4 - Toolkit.java [10 Marks]

The Toolkit class includes methods you need to use/complete to load the Glove and News data.

5.1 Task 4.1 - *loadGlove*() [5 Marks]

In this task, you are required to use a *BufferedReader* (*myReader*) to read data from the Glove file (*FILENAME_GLOVE*) line by line. *FILENAME_GLOVE* is the name of the Glove file (the structure of this file can be found in Section 1.1, page 4).

- Read the file line by line and analyse the result - adding the word to *listVocabulary* and its vector representation to *listVectors*.
- Use the *Toolkit.getFileFromResource*(*String _fileName*) method to get the correct file path.
- If the file doesn't exist, throw an exception and print out the error message (using *.getMessage()* method).
- The average execution time should be below 280 milliseconds.

HINT: Remember to use the *try...catch()...finally* blocks. Do NOT hardcode your file path.

5.2 Task 4.2 - *loadNews*() [5 Marks]

Similar to Task 4.1, now please load the News data from the *resource\News* folder.

- Check the file name first and only load those with *".htm"* extension.
- Please use the completed *HtmlParser* class to retrieve the related information, then convert it into a *NewsArticles* object and add it to the *listNews* variable.
- The average execution time should be below 30 milliseconds.

HINT: The file handling order is OS depended. So, make sure you sort the file list before processing.

6 Task 5 - ArticlesEmbedding [31.5 Marks]

Task 1 and Task 4.1 allow you to read data from the files and create the associated Glove objective. Unlike the TF-IDF Embedding in the first assignment, these Glove objectives are word-level embedding (or vectorisation) instead of document-level[3]. So,

[3]In A1, each document/article has a single TF-IDF embedding, this is called document-level embedding.

in this task, you are required to construct document-level embeddings based on the related Glove objectives. In other words, each news article has one single embedding that represents its content.

The ArticlesEmbedding class is a subclass of the NewsArticles class, which was completed in Task 2. There are three attributes in this class:

1. *processedText*. Back to the first assignment, there was a *preProcessing()* method for text cleaning, text lemmatization and stop words removal, then saved the proceeded text to a string array called *newsCleanedContent*. In this assignment, *processedText* is the equivalent of *newsCleanedContent* in A1 and is generated in Task 5.3. The difference is that *processedText* is a single string instead of an array.

2. *newsEmbedding*. This is the attribute for the document-level embedding, which will be generated in Task 5.4

3. *intSize*. Each news article has a different length, but neural networks can only process inputs of the same shape. Therefore, we need to set the size of the embedding here.

6.1 Task 5.1 - *ArticlesEmbedding(String _title, String _content, NewsArticles.DataType _type, String _label)* [1 Mark]

This is the constructor of the ArticlesEmbedding class. Complete it accordingly. You can modify the existing code in this constructor (*super("","",null,"");*).

6.2 Task 5.2 - $setEmbeddingSize(int\ \ \_size)$ [0.5 Marks]

This is the set method of the *intSize* variable. Complete it accordingly.

6.3 Task 5.3 - $getNewsContent()$ [10 Marks]

Override the *getNewsContent()* method in the *NewsArticles* class.

The idea here is that when this method has been called, it will automatically retrieve the original news content from its base and execute the subsequent pre-processing steps in the following sequence:

1. **Text cleaning**. Perform the text cleaning tasks by calling the provided *textCleaning()* method .

2. **Text lemmatization**. In the first assignment, we considered a simplified scenario. Here, we will use a proper NLP library called CoreNLP [2], developed by the NLP Group at Stanford University, for the lemmatization process.

The CoreNLP[4] library has been included in this project, but you need to learn how to set up the correct pipeline for text lemmatization by using the documentation provided on their website.

HINT: There is a specific page about Lemmatization.

3. **Stop-words removal**. Use the STOPWORDS constant in the *Toolkit* class to perform this task.

After these three steps, pass the string to the *processedText* attribute.

- Ensure all the characters in the *processtedText* are in lowercase. The .lemma() method in the CoreNLP library may restore letter cases and produce some unexpected results.
- The pre-processing task only needs to be done once. Otherwise, it will have a huge impact on the performance. In the related jUnit test, the average execution time should be less than 13000000 nanoseconds.

6.4 Task 5.4 - *getEmbedding*() [20 Marks]

Before starting this task, it's essential to have completed Task 6.1 and Task 6.2.

This task involves creating an array using ND4J (N-Dimensional Arrays for Java), a library included in this project. The array is formed by the embeddings of words present in the *processedText* string. For example, if "hello" and "world" have embeddings [0,1,2,3] and [4,5,6,7] respectively, the embedding for "hello world" is 0, 1, 2, 3, 4, 5, 6, 7.

Retrieve word embeddings from the Glove object list created in Task 6.1. Use the *intSize* attribute to set the maximum length of the array, calculated in Task 6.2. You'll need to familiarize[5] yourself with ND4J methods such as *Nd4j.create()* and *.putRow()*. The array's shape should be [x,y] where x=intSize and y=word vector size.

Additional requirements include:

- Throw an *InvalidSizeException* with a message "Invalid size" if intSize is uninitialized (intSize = -1).
- Throw an *InvalidTextException* with a message "Invalid text" if *processedText* is empty (*processedText.isEmpty()*) .
- Limit the length to *intSize*. If the document exceeds this, only process the first *intSize* characters; if it's shorter, fill the remaining space with 0.
- For a specific article, ensure the embedding process is done only once to avoid performance issues. In jUnit tests, the average execution time should be under 8 milliseconds.

HINT: Only include those words that have an associated Glove object.

[4] https://stanfordnlp.github.io/CoreNLP
[5] https://deeplearning4j.konduit.ai/nd4j/tutorials/quickstart

7 Task 6 - AdvancedNewsClassifier [44.5 Marks]

7.1 Task 6.1 - *createGloveList*() [5 Marks]

Based on the *Toolkit.listVocabulary* and *ToolkitVectors*, create/populate the Glove list.

- Only create a Glove object for those non-stop words.

7.2 Task 6.2 - *calculateEmbeddingSize(List<ArticlesEmbedding> _listEmbedding)* [5 Marks]

As explained before, each article has a different length. Hence, it is essential to determine a suitable embedding size. Using the smallest length will limit the ability to include more semantic information in the document-level embedding. On the other hand, there will be too many 0s in the embedding, which will pollute the semantic representation and increase the training time of the machine-learning model. To balance these concerns, we choose to use the median document length for embedding.

To calculate the median document length, follow these steps:

1. Determine the length of each document in your corpus/dataset.
2. Add these lengths to a list.
3. Sort the list in ascending order.
4. If the length of the list is even, the median is the average of the lengths at positions *N/2* and *(N/2) + 1* in the sorted list.
5. Otherwise, the median is the length at position *(N+1)/2* in the sorted list.

HINT: The length of the document is measured by the count of words it contains. However, only words that have a corresponding Glove object are included in this count.

7.3 Task 6.3 - *populateEmbedding*() [10 Marks]

listEmbedding is an attribute that holds all the ArticlesEmbedding objects, which are initialised in the *loadData()* method. Go through this list and call the getEmbedding() method (completed in Task 5.4) to calculate the embedding for each article.

- If an *InvalidSizeException* occurs, (re)assign the *intSize* attribute in the *ArticlesEmbedding* class by calling the *setEmbeddingSize()* method.
- If an *InvalidTextException* occurs, call the *getNewsContent()* method to pre-process the text .
- At the end of this method, all the objects in the *listEmbedding* should have a valid (nonempty) *newsEmbedding*.
- To avoid performance issues, use a single for loop to complete this task.

7.4 Task 6.4 - *populateRecordReaders(int _numberOfClasses)* [8 Marks]

The actual machine learning process is handled by a given method called *buildNeuralNetwork*, but you are tasked to construct the training data (*trainIter*).

For a specific document, its associated *DataSet* object contains two elements: a) an input (also called feature) INDArray and b) an output INDArray.

The input INDArray (*inputNDArray*) is simply the document-level embedding (*.getEmbedding()* method completed in Task 5.4). The output INDArray (*outputNDArray*) is constructed as the following:

The shape of this array is *[1, _numberOfClasses]*. Assuming that there are 2 classes (two newsgroups), then create an *outputNDArray* with the shape [1,2] and assign value 0 to it (outputNDArray=[0,0]. For a specific document, assign value 1 to the ***first element*** ([1,0]) if it belongs to the first group (newsLabel="1"). Otherwise, assign value 1 to the ***second element*** ([0,1]).

- Go through all the items that have been marked as Training data (use the .getNewsType() method, Task 2.3) from the listEmbedding, and initials their corresponding DataSet objects (*DataSet myDataSet = new DataSet(inputNDArray, outputNDArray)*).
- Once a DataSet object has been initialised, add it to the *listDS*.
- Your code should be flexible enough to handle more than 2 newsgroups.

7.5 Task 6.5 - *predictResult(List<ArticlesEmbedding _listEmbedding)* [8 Marks]

The label data is obtained through the *.getLabel()* method in the *HtmlParser* class, as outlined in Task 3.2. Initially, labels are available only for news items marked as Training data/type. The goal is to employ *myNeuralNetwork* for predicting labels for the Testing data.

The *myNeuralNetwork* attribute holds the trained machine learning model. To generate a label for any given input, use its *.predict()* method.

The parameter of the *.predict()* method is the document-level embedding of a specific news article. The output is an integer array: 0 means this specific news belongs to the first group, and 1 means the second group.

- Go through the ArticlesEmbedding list (*_listEmbedding*), and use the *.predict()* method to generate a label for all the Testing data.
- Add all the predicted labels to the listResult attribute.
- Use the *.setNewsLabel()* method to modify the label information in the associated *ArticlEmbedding* object.

7.6 Task 6.6 - *printResults*() [6.5 Marks]

Since the label information was updated in the last task, go through the *listEmbedding* attribute and print out the grouping result for the Testing data.

- Use the related jUnit test to determine the correct string format.
- Your code must be flexible enough to handle more than 2 newsgroups.

HINT: If you failed the jUnit test just because the line separates, ignore it. We are going to use a more accurate way to measure your output in the auto-marking system.

8 Expected Output

If all tasks have been completed correctly, the output produced by the main() method should match the following (ignore the colour):

```
Group 1
Boris Johnson asked if government 'believes in long COVID', coronavirus
    inquiry hears
COVID vaccine scientists win Nobel Prize in medicine
Long COVID risks are 'distorted by flawed research', study finds
Who is Sam Altman? The OpenAI boss and ChatGPT guru who became one of
    AI's biggest players
ChatGPT maker OpenAI agrees deal for ousted Sam Altman to return as
    chief executive
Sam Altman: Ousted OpenAI boss 'committed to ensuring firm still
    thrives' as majority of employees threaten to quit
Sam Altman: Sudden departure of ChatGPT guru raises major questions that
    should concern us all
ChatGPT creator Sam Altman lands Microsoft job after ousting by OpenAI
    board
Group 2
COVID inquiry: There could have been fewer coronavirus-related deaths
    with earlier lockdown, scientist says
Up to 200,000 people to be monitored for COVID this winter to track
    infection rates
Molnupiravir: COVID drug linked to virus mutations, scientists say
How the chaos at ChatGPT maker OpenAI has unfolded as ousted CEO Sam
    Altman returns - and why it matters
```

References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[2] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 2014, pp. 55–60.

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*, 1 2013. [Online]. Available: http://ronan.collobert.com/senna/

[4] T. Mikolov, Q. V. Le, and I. Sutskever, "Exploiting similarities among languages for machine translation," *ArXiv*, 9 2013. [Online]. Available: http://arxiv.org/abs/1309.4168

[5] T. Mikolov, W. T. Yih, and G. Zweig, "Linguistic regularities in continuous space-word representations," *NAACL HLT 2013 - 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Main Conference*, pp. 746–751, 2013.

[6] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[7] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.

[8] E. D. D. Team, "DL4J: Deep Learning for Java," 2016. [Online]. Available: https://github.com/eclipse/deeplearning4j

[9] ——, "ND4J: Fast, Scientific and Numerical Computing for the JVM," 2016. [Online]. Available: https://github.com/eclipse/deeplearning4j