Australian National University

Assignment 2 Specification (Version 2)

Machine/Deep Learning and Natural Language Processing

Document Analysis (COMP4650 /COMP6490), 2024 Semester 2

Original Release: 16 August 2024; Last Updated: 19 August 2024

Prof Hanna Suominen, Prof Jing Jiang, and the ANU COMP4650/COMP6490 teaching team at the School of Computing in the ANU College of Engineering, Computing and Cybernetics

comp4650@anu.edu.au

	Important Dates (ACT time)
Assignment specified on Wattle	Friday 16 August 2024 at 4 PM

Assignment due on Wattle	• Wednesday 9 October 2024 at 5 PM
	• A complete assignment submission consists of
	  ○ a document (one PDF file), submitted via a Turnitin Assignment functionality on Wattle
	  ○ Python code (one ZIP file) submitted via an Assignment functionality on Wattle
	• You will be required to press a submit button on Wattle to electronically sign a declaration as part of your submission.
	• Please keep a copy of the submitted content for your records.
	• Your document and code will be graded together, using the total scale from 0 to 25 marks.
	• No submission (or resubmission) after the due date will be permitted.
	• If an assessment task is not submitted by the due date, a mark of 0 will be awarded.

Extension Requests	Using the ANU Extension App, as explained on Wattle
Estimated return date of assessment feedback and marks on Wattle	Thursday 24 October 2024

Preface

Today you are presented with a complex challenge. Challenges motivate learning. To solve this challenge, you will need to learn more about ***Machine/Deep Learning*** (M/DL) and ***Natural Language Processing*** (NLP), and their evaluation in the context of analysing documents. Also, as you face various challenges you will need to go over learning materials from

- the lectures and computing labs,
- read parts of the recommended textbooks and other literature, and
- ask yourself and possibly your wider learning community (i.e., colleagues and teachers) all sorts of questions.

When you solve this challenge, you will have learnt most of the core concepts and basic skills within this topic of ML/DL and NLP for document analysis. Although this assignment is only worth 25/100 marks (25%) of the overall course assessment, what you learn is worth considerably more.

The challenges in this assignment are intentionally ill-specified and open-ended. This enables people to be creative in the approach they take as they explore different aspects of M/DL, NLP, and related analysis and evaluation tasks. Moreover, it enables keen students to really extend themselves. We have put considerable effort into defining challenges that may be solved with a restricted set of skills and yet are still interesting. We hope you find them a lot of fun. We did!

Please remember to follow The ANU policies, procedures, and guidelines to assure Academic Integrity (see https://www.anu.edu.au/students/academic-skills/academic-integrity for further information). We recommend using the ANU Turnitin Practice site (see https://www.anu.edu.au/students/academic-skills/research-writing/turnitin-practice-site for the instructions) before you submit your document. The use of Generative AI Tools (e.g., Microsoft CoPilot for the ANU) is permitted in this course, for both coding and document writing, given that proper citation and prompts are provided, along with a description of how the tool contributed to the assignment. Guidelines regarding appropriate citation and use can be found on the ANU library website at https://libguides.anu.edu.au/generative-ai. Further resources and support for academic writing and integrity are available through the ANU Academic Skills (see https://www.anu.edu.au/students/academic-skills); on our course Wattle page and Ed discussion forum; and during our lectures and labs. However, specific guidance on appropriate use should be directed to the conveners for this course. Do not share your individual assignment work with others to avoid compromising its integrity. Marks will reflect the contribution of the student rather than the contribution of the tools.

Engage, own, and get started early with this assignment. Your submission is due by 9 October 2024. Remember that the first go at something never works and writing takes time. Hence, work on your assignment every week and take the benefit of time to make your document and code shine brighter after iterations of improvement.

Assignment Didactic

In this assignment, your task is to study classifying job ads (more precisely, a large collection available at https://www.kaggle.com/datasets/shivamb/real-or-fake-fake-jobposting-prediction/) to experiment with M/DL and NLP methods. Throughout this assignment, you will make changes to existing code to extend and improve it; to set you up for success, we are providing you with some code on Wattle shortly after releasing this specification and more is available at the aforementioned Kaggle cite, if you chose to use it (remembering to use appropriate referencing). Note that these codes may not be complete, efficient, or scalable (or even relevant and correct to use if you looked into Kaggle). Thinking critically and improving the code it is your task in this assignment.

Throughout this assignment, you will learn more about

- differentiating between the basic probabilistic theories of language and document structure; classification; and document feature engineering,
- identifying and applying the theories, algorithms, and software available for probabilistic theories of language and text classification, and
- using common libraries for M/DL and NLP to perform basic analysis tasks,

thereby relating to the learning outcomes 1-3, 5, and 6 of this course.

Submission and Marking

A complete assignment submission consists of a **document** (one PDF file), submitted via a Turnitin Assignment functionality on Wattle, and **Python code** (one ZIP file), submitted via an Assignment functionality on Wattle. Your PDF file, containing your document, must be named **u1234567.pdf** where u1234567 should be replaced with your Uni ID. Your ZIP file, containing all of the code files (in Python) **BUT NEITHER DATA NOR YOUR PDF DOCUMENT**, must be named **u1234567.zip** where u1234567 should be replaced with your Uni ID. You will be required to press a submit button on Wattle to electronically sign a declaration as part of your submission. Please keep a copy of the submitted content for your records.

Your document and code will be graded together, using the total scale from 0 to 25 marks. No late submission (or resubmission) will be permitted without a pre-arranged extension. Extension requests are submitted using the ANU Extension App, as explained on Wattle; if applicable, please submit your request before the submission is due (but naturally, if you are, for example, very unwell, you can submit your request later and include an explanation why you were submitting it after the due date). A mark of 0 will be awarded if not submitted by the due date.

General Instructions

Assessment Marking and Feedback: To gain the full points, you need to impress us; hard and smart work pays off. The assignment is marked out of 25, based both on delivery and content. A correct and comprehensive assignment with a concise and clear document that demonstrates good understanding of all relevant learning objectives will be typically marked as 17.5/25 (70%). Your answers to coding questions will be marked based on the quality of your code (is it efficient, is it readable, is it extendable, is it correct) and the solution in general (is it appropriate, is it reliable, does it demonstrate a suitable level of understanding). Your document to report on your assignment work will be marked based on how convincing your explanations are (are they sufficiently detailed, are they well-reasoned, are they backed by appropriate evidence (analyses, evaluations, and/or references to relevant studies), are they clear, do they use appropriate aids such as tables and figures where necessary). Please see our indicative marks below. They supplement these general marking guidelines above, our Grade Expectations on Wattle, and those by The ANU School of Computing. Students will be given written feedback to supplement their mark.

Policies: The ANU has educational policies, procedures, and guidelines, which are designed to ensure that staff and students are aware of the University's academic standards, and implement them. You can find the University's education policies and an explanatory glossary at http://policies.anu.edu.au/.

Academic Misconduct: Students are expected to have read the Academic Misconduct Rule before the commencement of their course. Other key policies include Student Assessment (Coursework) and Student Surveys and Evaluations. See the Preface section above and http://policies.anu.edu.au/ for further information.

Turnitin: This is an individual assignment. Group work is not permitted. Assignments will be checked for similarities. The ANU is using Turnitin text-matching software to enhance student citation and referencing techniques, and to assess assignment submissions as a component of the University's approach to managing Academic Integrity. For additional information regarding Turnitin please visit https://www.anu.edu.au/students/academic-skills/academic-integrity/turnitin.

Referencing Requirements: Appropriate referencing is required (see https://www.anu.edu.au/students/academic-skills/academic-integrity/referencing). Please follow the Harvard style (see https://www.anu.edu.au/students/academic-skills/academic-integrity/referencing/harvard).

Help Is Available: Challenge yourself while being realistic with your ability to deliver on time. To be efficient, don't hesitate to use, e.g., Ed for clarifications or advice or book an appointment by The ANU Academic Skills (see https://www.anu.edu.au/students/academic-skills/appointments). We are here to support you; however, doing the hard and smart work is your responsibility.

Tasks

This assignment consists of 5 tasks related to classifying job descriptions.

Task 1: Analyse the Document Collection (max 5 marks, indicative)

Familiarise yourself with the following job postings dataset available at

https://www.kaggle.com/datasets/shivamb/real-or-fake-fake-jobposting-prediction/data.

The set contains around 18K job postings, out of which about 800 are fake. Analyse the set; in this linguistic analysis, you may wish to report on not only linguistic characteristics of postings but also describe their structure by comparing and contrasting text under some key posting fields and/or within different categories considered in text classification tasks below.

Report in your PDF file your analysis methods and results (max 360–440 words), supported by possible Tables and Figures plus the list of references. Submit your code as part of your ZIP file.

Task 2: Text Classification with Logistic Regression (max 5+ 5 = 10 marks, indicative)

In Task 2, you will use the dataset from Task 1 to perform text classification. Recall from the lectures that a text classifier predicts a label for a piece of input text, and such a text classifier can be trained from examples that have the ground truth labels. In Task 2, you will build logistic regression classifiers to label the job descriptions in the dataset.

Task 2 will consider the following two text classification problems:

1. Given a job description (found in the "description" column of the dataset file), predict the required education level of the job (found in the "required_education" column).
   a. There are 13 different educational levels found in the "required_education" column of the dataset. To simplify this first classification problem, you can consider only those job postings where the "required_education" column has one of the following three values: *"Master's Degree"*, *"Bachelor's Degree"*, *"High School or equivalent"*. You can filter out the other job postings where the required educational level is not one of the above for this first classification problem.
   b. You should use only those non-fraudulent job postings (indicated by the "fraudulent" column) to train, validate and test your first classifier.
2. Given a job description, predict whether it is a fraudulent job posing (found in the "fraudulent" column, where 0 indicates a real job posting and 1 indicates a fake job posting).

Imagine a job seeker who has access to many job postings that do not state the required educational level, and some of these job postings may be fake. Hypothetically, the job seeker can apply the second classifier as described above that you will build to detect and filter out the fake job postings, and then apply the first classifier that you will build to identify those job postings that the job seeker is qualified for. For this assignment, however, you will independently train and test the two classifiers.

Part A (max 5 out of the max 10 Task 2 marks, indicative)

A simple approach to building a logistic regression model for text classification is to use Term Frequency x Inverse Document Frequency (TF-IDF) features. This approach is relatively straightforward to implement and can be very hard to beat in practice.

We will provide you with some starting code for the first classification problem, i.e., prediction of the educational level based on a job description. It should be straightforward to adopt the code for the second classification problem once you have completed the code for the first.

Specifically, to build a logistic regression classifier using TF-IDF features, you should first implement the *get_features_tfidf* function (in *features.py*) that takes a set of training job descriptions as input and calculates the TF-IDF (sparse) document vectors. You may want to use the *TfidfVectorizer* in the *scikit-learn* package. You should use it after reading the documentation. For text preprocessing, you could set the *analyzer* argument of *TfidfVectorizer* to the *tokenise_text* function provided in *features.py*. Alternatively, you may set appropriate values to the arguments of *TfidfVectorizer* or write your own text preprocessing code.

Next, implement the *search_C* function (in *classifier.py*) to try several values for the regularisation parameter *C* and select the best based on the accuracy on the validation data. The *train_model* and *eval_model* functions provided in the same Python file might be useful for this task. To try regularisation parameters, you should use an automatic hyper-parameter search method presented in the lectures.

You should then run *job_classification.py* which first reads in the dataset and splits it into training, validation and test sets; it then trains a logistic regression text classifier and evaluate its performance on the test set. Make sure you first uncomment the line with the *analyse_classification_tfidf* function (which uses your *get_features_tfidf* function to generate TF-IDF features, and your *search_C* function to find the best value of *C*) in the top-level code block of *job_classification.py* (i.e., the block after the line "*if name == ' main ':*") and then run *job_classification.py*.

For training and testing this first classifier, remember to filter out fraudulent job postings and job postings whose required educational level is not among these three values: *"Master's Degree"*, *"Bachelor's Degree"*, *"High School or equivalent"*.

Next, modify your code to train and test a second classifier for fraudulent job posting detection, i.e., a binary classification problem where the class labels are found in the "fraudulent" column of the dataset. Because this classifier is independent of the first classifier, here you should use job postings of all educational levels.

Answer the following questions in your answers PDF for each of the two classification problems (max 360–440 words, supported by possible Tables and Figures plus the list of references):

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 accuracy on the test set?

Also make sure you submit your code as part of your ZIP file.

Part B (max 5 out of the max 10 Task 2 marks, indicative)

Another simple approach to building a text classifier is to 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 job descriptions.

Your task is to use *Word2Vec* in the *gensim* package to learn embeddings of words and predict the qualifications required of job descriptions using a logistic regression classifier with the aggregated word embedding features. You should use it after reading the documentation.

First implement the *train_w2v* function (in *word2vec.py*) using *Word2Vec* from the *gensim* package, then implement the *search_hyperparams* function (in *word2vec.py*) to tune **at least two** of the many hyper-parameters of *Word2Vec* (e.g. *vector_size*, *window*, *negative*, *alpha*, *epochs*, etc.) as well as the regularisation parameter C for your logistic regression classifier. You should use an automatic hyper- parameter search method presented in the lectures. (Hint: The *search_C* function in *classifier.py* and the *get_features_w2v* in *features.py* might be useful.)

Next implement the *document_to_vector* function in *features.py*. This function should convert a tokenised document (which is a list of tokenised sentences) into a vector by aggregating the embeddings of the words/tokens in the document using trained *Word2Vec* word vectors.

Last, you should uncomment the line with the *analyse_classification_w2v* function (and comment out the line with *analyse_classification_tfidf*) in the top-level code block of *job_classification.py*, and then run *job_classification.py* to train a logistic regression text classifier with the word vectors from your *Word2Vec* model, and evaluate the classification performance on the test set.

Similar to Part A, train and test two classifiers for the two classification tasks: prediction of the required educational level, and identification of fraudulent job postings.

Answer the following questions in your answers PDF for each of the two classification problems (max 360–440 words, supported by possible Tables and Figures plus the list of references):

1. What hyper-parameters of *Word2Vec* did you tune? What ranges of values for the hyper-parameters did you try? Explain, why the ranges are reasonable. Also explain what search technique you used and why it is appropriate here.
2. What were the best performing values of the hyper-parameters you tuned?
3. What was your accuracy on the test set? Compare it with the accuracy you got in **Part A** of this question and discuss why one is more accurate than the other.

Also make sure you submit your code as part of your ZIP file.

Task 3: Text Classification with a Transformer Encoder (max 5 marks, indicative)

In this task, you will apply a transformer-based classifier to perform the same text classification tasks as in Task 2 and compare the classification performance with the results you have got for Task 2.

Specifically, you will train a transformer encoder using *PyTorch*. An input sequence is first tokenised and a special *[CLS]* token prepended to the list of tokens. Similar to BERT, the final hidden state (from the transformer encoder) corresponding to the *[CLS]* token is used as a representation of the input sequence in this text classification task.

First, implement the *get_positional_encoding* function in ***job_classifier.py***. This function computes the following positional encoding:

$$PE_{t,2i} = \sin\left(\frac{t}{10000^{\frac{2i}{d}}}\right),$$

$$PE_{t,2i+1} = \cos\left(\frac{t}{10000^{\frac{2i}{d}}}\right),$$

$$t \in \{0, \dots, T-1\}$$

Next, complete the *__init__* method of class *JobClassifier* by creating a *TransformerEncoder* consisting of a few (determined by parameter *num_tfm_layer*) *TransformerEncoderLayer* with the specified input dimension (*emb_size*), number of heads (*num_head*), hidden dimension of the feedforward sub-network (*ffn_size*) and dropout probability (*p_dropout*). Then implement the *forward* method of class *JobClassifier*. You should implement the following steps sequentially in the function:

(a) add the positional encoding to the embeddings of the input sequence;

(b) apply dropout (i.e. call the dropout layer of *JobClassifier*);

(c) call the transformer encoder you created in the *__init__* method; (Hint: make sure you set the parameter *src_key_padding_mask* to an appropriate value.)

(d) extract the final hidden state of the *[CLS]* token from the transformer encoder for each sequence in the input (mini-batch);

(e) use the linear layer to compute the **logits** (i.e., unnormalised probabilities) for binary classification and return the **logits**.

You should read the documentation before implementing these methods.

Last, complete the *train_model* function to learn the classifier using the training dataset. You should optimise the model parameters through mini-batch stochastic gradient descent with the *Adam* optimiser. Make sure you evaluate the model on the validation set after each epoch of training and save the parameters of the model that achieves the best accuracy on the validation set.

You are now able to run *job_classifier.py* which prepares the training, validation and test datasets, trains a classifier and evaluates its accuracy on the test set. Note that this approach does not directly use the combined training and validation datasets to re-train the model, and we adopt it here for simplicity. Tuning hyper-parameters systematically is not required (but encouraged if you have access to sufficient computational resources) for this question, and setting the hyper-parameters to some reasonable values (from your experience) is acceptable. **In your answers PDF, you should record the values of hyper-parameters used in your text classifier**.

Similar to Task 2, train and test two classifiers for the two classification tasks: prediction of the required educational level, and identification of fraudulent job postings.

Answer the following questions in your answers PDF for each of the two classification problems (max 360–440 words, supported by possible Tables and Figures plus the list of references):

1. What is the architecture of your classifier? (Hint: you may print your classifier in Python and record the output in your answers PDF.)
2. What was your accuracy on the test set?
3. Compare your classification performance with those you got for Task 2 (Part A and Part B). Discuss the advantages and limitations of the three approaches to job description classification.

Also make sure you submit your code as part of your ZIP file.

Task 4: Text Classification with a Pre-trained Language Model (max 1 + 1.5 = 2.5 marks, indicative)

Pre-trained language models such as BERT and ChatGPT have been widely used for many NLP problems. As you have learned in the lectures, these pre-trained language models have strong capabilities to handle a wide range of tasks with little or no further training. In Task 4, you will use pre-trained language models to perform the same text classification problem as in Task 2 and Task 3.

Part A (max 1 out of the max 2.5 Task 4 marks, indicative)

In Lab 6, you will see how to fine-tune a smaller version of BERT. Adopt the strategy of fine-tuning DistilBERT as you will learn in Lab 6 on the job posting dataset. Compare the text classification performance of using the original DistilBERT and your fine-tuned DistilBERT on the two classification tasks. Note that you will still use the *[CLS]* token produced by the model to perform classification.

Analyse and discuss the results you observe in your answers PDF in free-form text (max 90–110 words, supported by possible Tables and Figures plus the list of references). Also make sure you submit your code as part of your ZIP file.

Part B (max 1.5 out of the max 2.5 Task 4 marks, indicative)

Prompt-based zero-shot text classification using a pre-trained language model such as ChatGPT has emerged as a new way of classifying text without any training involved. See, for example, the following Hugging Face page that explains how it works:

https://huggingface.co/tasks/zero-shot-classification

In this part of Task 4, use a Large Language Model (LLM) of your choice (e.g., ChatGPT, Gemini, or Llama) to predict the required educational level of a sample of job postings in the dataset and observe the performance of the LLM on the task.

Report your findings in your answers PDF in free-form text (max 180-220 words, supported by possible Tables and Figures plus the list of references). Things you can consider trying include, but are not limited to, the following (you are not required to do all of them):

- Compare different LLMs on the job posting classification task, using the same subset of data you have sampled.
- Compare the performance of the prompt-based zero-shot method with the previous methods you have tried in Task 2 and Task 3.
- Use the LLM's API to batch process a large sample of the data for prompt-based zero-shot classification, if you have access to the API. (This requires additional work of understanding the LLM's API, which is not covered in this course.)
- Discuss the pros and cons of a training-based method you have explored in the earlier parts of the assignment and of this prompt-based zero-shot method.

Note: You do not need to use an LLM to predict whether a job posting is fraudulent in Part B of Task 4. You should submit your code, prompts, or other supporting information as part of your ZIP file.

Task 5: Reflect on Your Learning and Academic Integrity (max 2.5 marks, indicative)

Add a short written, free-form text answer (max 180-220 words) to reflect on your learning and academic Integrity considerations relevant to your Assignment 2 work in your PDF file. In particular, explain where, how, and why you used Generative AI tools in coding and/or academic writing. Demonstrate your critical thinking skills in analysing and assessing the advantages and disadvantages of this tooling to your learning. If you chose not to use Generative AI tools, reflect on the advantages and disadvantages of this choice to your learning.