COMP4650/6490 Document Analysis – Semester 2 / 2023

Assignment 2

Due 17:00 on Tuesday 19 September 2023 AEST (UTC +10)

Last updated August 21, 2023

Overview

In this assignment you will:

1. Develop a better understanding of how machine learning models are trained in practice, including partitioning of datasets, evaluation, and tuning hyper-parameters.
2. Become familiar with the `scikit-learn`[1] and `gensim`[2] packages for machine learning with text.
3. Become familiar with the `PyTorch`[3] framework for implementing neural network-based machine learning models.

Throughout this assignment you will make changes to the provided code to improve or complete existing functions. In some cases, you may write your own code to perform functionalities similar to the provided code but using different types of features or dataset.

Submission

- 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` should be replaced with your Uni ID.
- The answers to this assignment (including your code files) have to be submitted online in Wattle.
- You should submit a ZIP file containing all of the code files and your answers PDF file, **BUT NO DATA**. A simple Python script `create_submission_zip.py` that can generate a ZIP archive with all the required files for your submission has been provided.
- **No late submission will be permitted without a pre-arranged extension**. A mark of 0 will be awarded if not submitted by the due date.

Marking

This assignment will be marked out of 100, and it will contribute 10% of your final course mark.

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) and the solution in general (is it appropriate, is it reliable, does it demonstrate a suitable level of understanding).

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 where necessary).

This is an individual assignment. Group work is not permitted. Assignments will be checked for similarities.

[1] `https://scikit-learn.org/`
[2] `https://radimrehurek.com/gensim/intro.html`
[3] `https://pytorch.org/`

Question 1: Movie Review Sentiment Classification (40%)

For this question you have been provided with a labelled movie review dataset – the same dataset you explored in Lab 3. The dataset consists of 50,000 review articles written for movies on IMDb, each labelled with the sentiment of the review – either positive or negative. The overall distribution of labels is balanced (25,000 `pos` and 25,000 `neg`). Your task is to apply logistic regression with both sparse and dense representations to predict the sentiment label from the review text.

Part A

One simple approach to classifying the sentiment of documents from their text is to train a logistic regression classifier using TF-IDF features. This approach is relatively straightforward to implement and can be very hard to beat in practice.

To do this you should first implement the `get_features_tfidf` function (in `features.py`) that takes a set of training sentences as input and calculates the TF-IDF (sparse) document vectors. You may want to use the `TfidfVectorizer`[4] 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 `sentiment_analysis.py` which first reads in the dataset and splits it into training, validation and test sets; then trains a logistic regression sentiment classifier and evaluate its performance on the test set. Make sure you first uncomment the line with the `analyse_sentiment_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 `sentiment_analysis.py` (i.e., the block after the line "`if __name__ == '__main__':`") and then run `sentiment_analysis.py`.

Answer the following questions in your answers 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 accuracy on the test set?

Part B

Another simple approach to building a sentiment 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 movie reviews.

Your task is to use `Word2Vec`[5] in the `gensim` package to learn embeddings of words and predict the sentiment labels of review text 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

[4] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
[5] https://radimrehurek.com/gensim/models/word2vec.html#gensim.models.word2vec.Word2Vec

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.

Lastly, you should uncomment the line with the `analyse_sentiment_w2v` function (and comment out the line with `analyse_sentiment_tfidf`) in the top-level code block of `sentiment_analysis.py`, and then run `sentiment_analysis.py` to train a logistic regression sentiment classifier with the word vectors from your `Word2Vec` model, and evaluate the classification performance on the test set.

Answer the following questions in your answers PDF:

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.

Question 2: Movie Review Clustering (35%)

For this question you have been provided with another movie review dataset consists of 50,000 review articles written for movies on IMDb. This dataset is similar to the movie review dataset used in Question 1 except that no sentiment annotations are available[6]. Your task is to apply the K-Means clustering algorithm to the unlabelled dataset to analyse the (hidden) structure of the review text.

First, you should implement the `cosine_distance` function in `kmeans.py`. The function computes the cosine distance between two input vectors

$$\texttt{cos\_dist}(\mathbf{v}_1, \mathbf{v}_2) = 1 - \texttt{cos\_sim}(\mathbf{v}_1, \mathbf{v}_2),$$

where $\texttt{cos\_sim}(\mathbf{v}_1, \mathbf{v}_2)$ is the cosine similarity between vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ as presented in lectures. Your implementation should accept **both sparse and dense** input vectors.

Your next task is implementing the key steps of the K-Means algorithm. In particular, you should implement the `compute_distances`, `assign_data_points`, and `update_centroids` functions in `kmeans.py`. These three functions, respectively, compute the distances between every data point and every centroid, assign each data point to its closest centroid, and re-compute each centroid as the average of the data points assigned to it. You should use the `cosine_distance` function for computing distances between data points and centroids.

Now that all the components of K-Means have been implemented, you can use it to analyse and explore the unlabelled movie review dataset. You have been provided the function `cluster_reviews_tfidf` (in `clustering.py`) to perform cluster analysis using TF-IDF features, it uses the `visualise_clusters_TSNE` to visualise the clusters by projecting the TF-IDF vectors onto a 2-dimensional space (you may use a different visualisation method). You should complete the `cluster_reviews_w2v` function in the same file to perform similar functionality but use aggregated `Word2Vec` word vectors instead of TF-IDF features

[6]Due to imperfect data processing, the actual ratings are presented in the review text for a small number of reviews.

to represent documents. For simplicity, you may reuse the best hyper-parameters found in **Part B** of Question 1 for training the `Word2Vec` model.

To cluster the unlabelled movie reviews and visualise the clusters, you should call `cluster_reviews_tfidf` in the top-level code block of `clustering.py`, with **at least three** different values of K and a number of different random seeds for each value of K. Answer the following questions in your answers PDF:

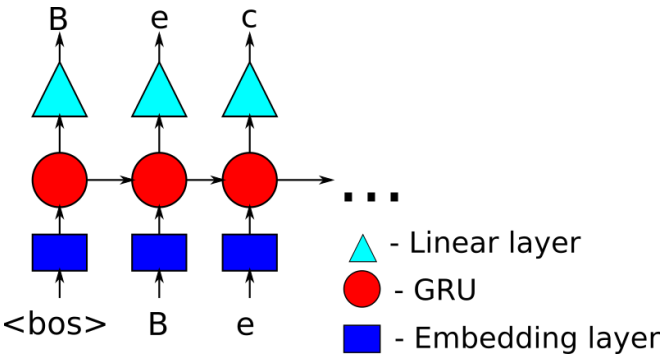
1. What values of K did you try? Explain, why you thought these values were reasonable.
2. For a particular value of K, did different random seeds lead to significantly different clusters? Discuss your observations.
3. Among the values for K you tried, which do you think leads to the best clusters. Compare and discuss your results.

Then call `cluster_reviews_w2v` in the top-level code block of `clustering.py`, with **at least three** different values of K and a number of different random seeds for each value of K, and answer the three questions above in your answers PDF.

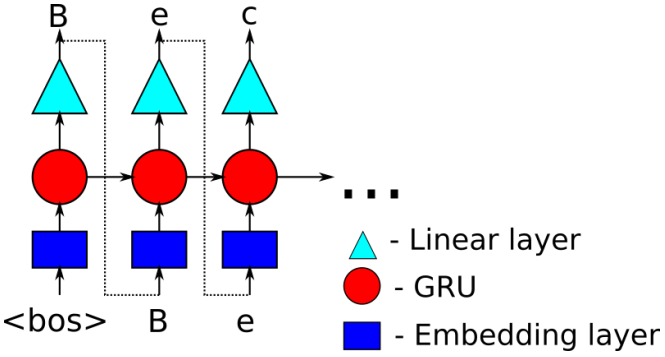
Also make sure you submit your code.

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 when training:



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 class `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_length = 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 class `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 initialiser.

`train_model`: In this method you need to train the model by mini-batch stochastic gradient decent. The optimiser and loss function are provided to you. Note that the loss function takes logits (output 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`