Task 3 - NLP Tasks

NLP Task 1: Text Summarizer

Literature Review

Extraction based text summarisation can be completed through a number of approaches. One such system, TextRank, proposed in Sehgal et al (2017) modifies Google's PageRank algorithm to extract useful and important phrases from the available text. This paper evaluates the algorithm through the ROGUE 2.0 Evaluation toolkit and achieves the following results:

Rogue Type	Task Name	Average Recall	Average Precision	Average Fscore	Number Referance Summaries
ROGUE 1	Sample 1	1.0	0.29664	0.45732	1
ROGUE 1	Sample 2	1.0	0.09125	0.16841	1
ROGUE 1	Sample 3	1.0	0.33504	0.50192	1
ROGUE 1	Sample 4	1.0	0.44071	0.61180	1

While efforts had been made to extract a meaningful and coherent summary from the article, there is still a lot of scope of improvement as to how the sentences are extracted and whether they take the summary to its logical meaning. Another possible approach is to use k-means clustering as proposed in Shetty and Kallimani (2017). This paper proposes a three-step unsupervised extraction approach consisting of:

- Document tokenization
- Compute sentence score
- Apply Centroid Based Clustering on the sentences and extract important sentences as part of summary

The paper does not measure the accuracy of the summarizer and instead compares their output to a human written abstractive summary. They state that their approach provides more favourable results than current state-of-the-art approaches such as TextRank, ranking their proposed approach in second place to a human approach, very close to the first place.

Rational for selection of the NLP task

With the recent surge in the amount of online content available to the general population, a fast and effective automatic summarization has become more important. By summarizing content with a large amount of data down to a few sentences, users are able to access the most important aspects of the content without having to sift through redundant and insignificant information. This proves key in reducing the amount of time taken reading job applications, allowing the prospective applicant to read a quick summarization of the post without needing to read the whole thing.

An extraction-based approach was chosen as opposed to an abstractive-based summariser due to the simplicity of the former method. The extraction-based approach also retains the writing style and nuances of the original job description which can provide context that would be lost with an abstractive text summariser.

Data pre-processing of inputs and outputs, separate from the WebCrawler harvesting

The input to the summarizer will be the job description, with the output being an arbitrary amount of the most impactful sentences as determined by the extractor. For this use case, three sentences were determined to be satisfactory. As the job description had been cleaned previously, there is no additional pre-processing of the input required. As the output is just text sentences extracted from the input, is requires no additional cleaning or processing before the summary can be stored alongside the rest of the data in a csv file.

Specification and justification of hyperparameter

As this NLP task is not a machine learning task, no hyperparameter is used. Judging the effectiveness of the summarizer will be down to pure human judgement as this task is only a prototype. Further research can be done into what metrics can be applied and how to apply them to this task.

Preliminary assessment of NLP Task performance

To test the summarizer, we can first test it on a single job ad to gauge its effectiveness. Further to that we can then run the summarizer for each job in order to find the average length of the summary compared to the job posting.

In [ ]:

```
import sys
!{sys.executable} -m pip install pandas nltk progressbar2 numpy
```

In [36]:

```
import pandas as pd
import nltk
import heapq
import numpy as np

stopwords = nltk.corpus.stopwords.words('english')

def get_summary(row):
    formatted_article_text = row.DESCRIPTION
    # Sentence wise Tokenizing
    sentence_list = nltk.sent_tokenize(row.DESCRIPTION)

    # Find Weighted Frequency of Occurrence
    word_frequencies = {}
    for word in nltk.word_tokenize(formatted_article_text):
        if word not in stopwords:
            if word not in word_frequencies.keys():
                word_frequencies[word] = 1
            else:
                word_frequencies[word] += 1
    maximum_frequency = max(word_frequencies.values())
    for word in word_frequencies.keys():
        word_frequencies[word] = (word_frequencies[word] / maximum_frequency)

    # Calculating Sentence Scores
    sentence_scores = {}
    for sent in sentence_list:
        for word in nltk.word_tokenize(sent.lower()):
            if word in word_frequencies.keys():
                if len(sent.split(' ')) < 30:
                    if sent not in sentence_scores.keys():
                        sentence_scores[sent] = word_frequencies[word]
                    else:
                        sentence_scores[sent] += word_frequencies[word]

    # Getting the Summary
    return heapq.nlargest(3, sentence_scores, key=sentence_scores.get)


def main():
    data = pd.read_csv('data/clean_seek_data.csv')
    sum_lens = []

    with progressbar.ProgressBar(max_value=len(data)) as bar:
        for index, row in data.iterrows():
            summary_sentences = get_summary(row)
            summary = " ".join(summary_sentences)
            summary_len = len(nltk.word_tokenize(summary))
            sum_lens.append(summary_len)
            bar.update(index)

    data = data.sample()
    for index, row in data.iterrows():
        summary_sentences = get_summary(row)
        summary = " ".join(summary_sentences)
        print(row.TITLE)
        print("="*200)
        print("Original description:")
        print(row.DESCRIPTION)
```

```
            print("-"*200)
            print("Summarized description:")
            print(summary)
            print("-" * 200)
            print()
            original_len = len(nltk.word_tokenize(row.DESCRIPTION))
            summary_len = len(nltk.word_tokenize(summary))


    print("Done!")
    print("Mean summarized length: {}".format(np.round(np.mean(sum_lens))))


main()
```

```
100% (8866 of 8866) |####################| Elapsed Time: 0:01:43 Time:  0:0
1:43
```

```
rail design manager job in melbourne
============================================================================
============================================================================
======================================================
Original description:
just imagine your future with us… at aurecon we see the future through a v
ery different lens. do you? innovation, eminence and digital are at the he
art of everything we do. are you excited about the future? are you driven
by the opportunity to work on some of the most challenging and complex pro
jects around the world and to learn from the best? we are. diversity is at
the core of everything we do. we work together to create a culture based o
n respect, trust and inclusiveness. our differences are what fuel our crea
tivity. we embrace flexible working and are always open to discussing your
individual needs so that you get to create your own experience with us. wh
at will you do? our rail infrastructure team support the technical advisor
y & design delivery of large-scale major projects across victoria. we are
seeking rail design managers to join our rail team working on several high
-profile, city-shaping projects including suburban rail loop, geelong fast
rail, melbourne airport link, level crossing removal program, regional rai
l revitalisation and other upcoming projects in both the rail planning and
delivery phases. some of the things you will do to bring ideas to life: le
ad/coordinate a multi-disciplinary design team, including design partners,
in rail planning and/or delivery projects (light rail or heavy rail). work
closely with our clients to listen, understand, and then address their nee
ds manage resources, subcontractors, and costs to deliver the projects pro
duce reports and other deliverables as required be actively involved in sa
fety in design, risk assessments assist the resolution of design standard
or operational requirements non-conformances assist with the preparation o
f proposals for future work what can you bring to the team? we are looking
for dedicated rail design management professionals to join our award-winni
ng team to leave a legacy on victorias rail industry. you will have a prov
en background working on rail projects with a development and/or delivery
focus. you will come to us with working knowledge of the multiple discipli
nes that make up a rail project. an understanding of the department of tra
nsport planning process and mtm, v/line, yarra trams, artc design process
requirements will be highly regarded. finally, we value that each of our t
eam members brings something different to aurecon. we look for people who
have had a broad range of experiences throughout their career and can demo
nstrate how they have worked as part of a team to bring ideas to life. doe
s that sound like you? about us weve re-imagined engineering. aurecon is a
n engineering and infrastructure advisory company, but not as you know it!
for a start, our clients ideas drive what we do. drawing on our deep pool
of expertise, we co-create innovative solutions with our clients to some o
```

```
f the worlds most complex challenges. and through a range of unique creati
ve processes and skills, we work to re-imagine, shape and design a better
future.
---------------------------------------------------------------------------
---------------------------------------------------------------------------
----------------------------------------------------
Summarized description:
an understanding of the department of transport planning process and mtm,
v/line, yarra trams, artc design process requirements will be highly regar
ded. and through a range of unique creative processes and skills, we work
to re-imagine, shape and design a better future. we are looking for dedica
ted rail design management professionals to join our award-winning team to
leave a legacy on victorias rail industry.
---------------------------------------------------------------------------
---------------------------------------------------------------------------
----------------------------------------------------

Done!
Mean original length: 414.07861493345365
Mean summarized length: 69.72355064290548
```

To test the summarizer, we can first test it on a single job ad to gauge its effectiveness. Further to that we can then run the summarizer for each job in order to find the average length of the summary compared to the job posting.

From the output, the summary lacks coherence as it is simply extracting impactful sentences and does not try to develop a new paragraph based on the data, a task more suitable to an abstractive -based summarizer. We can also see the summary has an average length of 70 words, a 453% reduction from the average 317 words in a job posting.

While the summary lacks coherence, its' quality is more than suitable for this prototype task. As such the summarizer can be applied to the whole dataset with the summary saved in a column titled “SUMMARY”.

Code

In [ ]:

```
import pandas as pd
import nltk
import heapq
import progressbar

stopwords = nltk.corpus.stopwords.words('english')


def main():
    print("Summarizing Job Data!")
    data = pd.read_csv('data/clean_seek_data.csv')

    with progressbar.ProgressBar(max_value=len(data)) as bar:
        for index, row in data.iterrows():
            summary_sentences = get_summary(row)
            summary = " ".join(summary_sentences)
            data.loc[index, 'SUMMARY'] = summary
            bar.update(index)

    # Saving Data
    data.to_csv("summarised_job_data.csv", index=False, encoding='utf-8-sig')
    print("Done!")


main()
```

References

Sehgal, S., Kumar, B., Maheshwar, Rampal, L., & Chaliya, A. (2017). A Modification to Graph Based Approach for Extraction Based Automatic Text Summarization. Advances in Intelligent Systems and Computing, 373–378. https://doi.org/10.1007/978-981-10-6875-1_36 (https://doi.org/10.1007/978-981-10-6875-1_36)

Shetty, K., & Kallimani, J. S. (2017). Automatic extractive text summarization using K-means clustering. 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT). Published. https://doi.org/10.1109/iceeccot.2017.8284627 (https://doi.org/10.1109/iceeccot.2017.8284627)

NLP Task 2: Skill Keyword Extraction

Literature Review

Job candidates obtain skills through formal education, vocational education, internships, on-the-job training, or experience from previous occupations. The key function of a job search engine is to assist in the matching of the candidates skills to jobs that also require a similar skill set. A common approach while doing a skill match is to use standard keyword matching or information retrieval framework as explained in (Salton & Buckley, 1988). A few challenges with this approach include:

- The skill may be referenced in many different forms or synonyms (e.g., OOP, Object Oriented Programming)
- Some skills may not be explicitly stated in the job description, but industry knowledge would dictate experience with the stated skill requires experience with the unstated skill (e.g. Experience with python denotes the candidate would require experience with Object Oriented Programming)

- A skill dictionary would quickly become outdated as new skills from unseen and emerging domains appear. A framework for skill extraction and normalization was proposed in (Sharma, 2019). This paper proposes the use of a Recurrent Neural Network subtype of an artificial neural network in combination with word embeddings to solve the problems encountered with a static skill dictionary. The system first extracts noun phrases from job descriptions before applying a Long Short-Term Memory (LSTM) deep learning network combined with word embeddings to extract the relevant skills from the text. The authors were able to achieve a test accuracy of 76.58% by restricting the job domain to jobs Data Science category. The method proposed in this paper is limited by the use of noun phrases for the core training dataset. As many job posts are represented by verb phrases, a new training set and model must be developed to extract the phrases.

Rational for selection of the NLP task

Job skills are the common link between job applications, applicant resumes and job listings by companies. Identifying skills in job postings is a significant problem and can provide a pathway for job seekers and hiring organisation. By ‘tagging’ each job listing with the required skills and enabling users to filter jobs by these skills would drastically improve the job search process.

Data pre-processing of inputs and outputs, separate from the WebCrawler harvesting

The input to the skill extractor will be noun phrases as defined in by the following grammar:

NBAR:

```
{<NN.\*|JJ>\*<NN.\*>}
```

NP:

```
{\<NBAR>}
{\<NBAR>\<IN>\<NBAR>}
```

These noun phrases will be extracted from each job description and fed through the RNN with the output being a float value from 0 to 1 representing the probability that the noun phrase is a skill. If the output of the RNN is greater than 0.5, the phrase is determined to be a skill and is appended to a list of skills for that job description. This list is then saved into a column in the dataset’s csv file titled “SKILLS”.

Specification and justification of hyperparameter

For the specified problem, there are multiple classes (skill and not_skill) but only one of the classes can be present at a single time. As such, the softmax activation function was chosen as it enables the model to interpret the output as probabilities.

The implemented RNN model also uses the adaptive moment estimation, adam, optimizer from keras which uses default values of:

- Learning rate = 0.001
- Beta_1 = 0.9
- Beta_2 = 0.999
- Epsilon = 1e-7

Keras also offers multiple metrics to judge the model. In our case, the accuracy of the models prediction will be used in order to provide a proper comparison to past literature.

Preliminary assessment of NLP Task performance

Utilising an industry standard 80/20 split on the labelled data along with a 5-fold cross validation, we can judge the accuracy of the model:

In [ ]:

```
import sys
!{sys.executable} -m pip install keras tensorflow sklearn
```

In [4]:

```
import csv
import time
import nltk
import progressbar
import numpy as np
import pandas as pd
from datetime import date
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split

# RNN Imports
from pipeline import Pipeline
from text_preprocessing import TextToTensor
from text_chunker import Chunker

# Reading the configuration file
import yaml

with open("conf.yml", 'r') as file:
    conf = yaml.safe_load(file).get('pipeline')

save_results = conf.get('save_results')

stop_words = nltk.corpus.stopwords.words('english')

# Reading the data
data = pd.read_csv('data/train.csv')[['TEXT', 'TARGET']]

# Shuffling the data for the k fold analysis
data = data.sample(frac=1)

# Creating the input for the pipeline
X = data['TEXT'].astype(str).tolist()
Y = data['TARGET'].tolist()
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)


if conf.get('k_fold'):
    print('=' * 80)
    print('Testing the RNN')
    print('-' * 80)
    kfold = KFold(n_splits=conf.get('n_splits'))
    acc = []
    f1 = []
    for train_index, test_index in kfold.split(X_train):
        # Fitting the model and forecasting with a subset of data
        k_results = Pipeline(
            X_train=np.array(X_train)[train_index],
            Y_train=np.array(Y_train)[train_index],
            embed_path='embeddings\\glove.840B.300d.txt',
            embed_dim=300,
            X_test=np.array(X_train)[test_index],
            Y_test=np.array(Y_train)[test_index],
            max_len=conf.get('max_len'),
            epochs=conf.get('epochs'),
            batch_size=conf.get('batch_size')
        )
        # Saving the accuracy
        acc += [k_results.acc]
        print(f'The accuracy score is: {acc[-1]}')
        print('-' * 80)
    print(f'Total mean accuracy is: {np.mean(acc)}')
print('=' * 80)
```

```
============================================================================
====
Testing the RNN
----------------------------------------------------------------------------
----
Epoch 1/7
11/11 [==============================] - 20s 25ms/step - loss: 0.6919
Epoch 2/7
11/11 [==============================] - 0s 24ms/step - loss: 0.6769
Epoch 3/7
11/11 [==============================] - 0s 24ms/step - loss: 0.6661
Epoch 4/7
11/11 [==============================] - 0s 23ms/step - loss: 0.6001
Epoch 5/7
11/11 [==============================] - 0s 24ms/step - loss: 0.5493
Epoch 6/7
11/11 [==============================] - 0s 23ms/step - loss: 0.4149
Epoch 7/7
11/11 [==============================] - 0s 24ms/step - loss: 0.3507
The accuracy score is: 0.7153846153846154
----------------------------------------------------------------------------
----
Epoch 1/7
11/11 [==============================] - 4s 25ms/step - loss: 0.6932
Epoch 2/7
11/11 [==============================] - 0s 24ms/step - loss: 0.6802
Epoch 3/7
11/11 [==============================] - 0s 23ms/step - loss: 0.6590
Epoch 4/7
11/11 [==============================] - 0s 23ms/step - loss: 0.6494
Epoch 5/7
11/11 [==============================] - 0s 23ms/step - loss: 0.6058
Epoch 6/7
11/11 [==============================] - 0s 24ms/step - loss: 0.5169
Epoch 7/7
11/11 [==============================] - 0s 24ms/step - loss: 0.3929
The accuracy score is: 0.7538461538461538
----------------------------------------------------------------------------
----
Epoch 1/7
11/11 [==============================] - 4s 23ms/step - loss: 0.7053
Epoch 2/7
11/11 [==============================] - 0s 24ms/step - loss: 0.6898
Epoch 3/7
11/11 [==============================] - 0s 24ms/step - loss: 0.6748
Epoch 4/7
11/11 [==============================] - 0s 24ms/step - loss: 0.6504
Epoch 5/7
11/11 [==============================] - 0s 24ms/step - loss: 0.5658
Epoch 6/7
11/11 [==============================] - 0s 24ms/step - loss: 0.4518
Epoch 7/7
11/11 [==============================] - 0s 23ms/step - loss: 0.3184
The accuracy score is: 0.7107692307692308
----------------------------------------------------------------------------
----
```

```
Epoch 1/7
11/11 [==============================] - 4s 26ms/step - loss: 0.7139
Epoch 2/7
11/11 [==============================] - 0s 24ms/step - loss: 0.6869
Epoch 3/7
11/11 [==============================] - 0s 25ms/step - loss: 0.6839
Epoch 4/7
11/11 [==============================] - 0s 24ms/step - loss: 0.6604
Epoch 5/7
11/11 [==============================] - 0s 26ms/step - loss: 0.6147
Epoch 6/7
11/11 [==============================] - 0s 24ms/step - loss: 0.5575
Epoch 7/7
11/11 [==============================] - 0s 24ms/step - loss: 0.3934
The accuracy score is: 0.7180277349768875
---------------------------------------------------------------------------
----
Epoch 1/7
11/11 [==============================] - 4s 25ms/step - loss: 0.7001
Epoch 2/7
11/11 [==============================] - 0s 24ms/step - loss: 0.6922
Epoch 3/7
11/11 [==============================] - 0s 24ms/step - loss: 0.6791
Epoch 4/7
11/11 [==============================] - 0s 24ms/step - loss: 0.6562
Epoch 5/7
11/11 [==============================] - 0s 24ms/step - loss: 0.6270
Epoch 6/7
11/11 [==============================] - 0s 25ms/step - loss: 0.5321
Epoch 7/7
11/11 [==============================] - 0s 23ms/step - loss: 0.4113
The accuracy score is: 0.7134052388289677
---------------------------------------------------------------------------
----
Total mean accuracy is: 0.7222865947611711
===========================================================================
====
```

The average accuracy of the model after the cross validation is 72.23%. Whilst less accurate than models from previous literature, this model is not restricted to covering a single job industry and proves to work well enough for a prototype task.

After proving sufficiently accurate, the model was used to extract the skills from each description in the seek dataset.

Code

In [ ]:

```
start_time = time.time()
print('=' * 80)
print('Running the RNN pipeline')
print('-' * 80)
print()
# Running the pipeline with all the data
RNN = Pipeline(
    X_train=X_train,
    Y_train=Y_train,
    embed_path='embeddings\\glove.840B.300d.txt',
    embed_dim=300,
    stop_words=stop_words,
    max_len=conf.get('max_len'),
    epochs=conf.get('epochs'),
    batch_size=conf.get('batch_size')
)
RNN.model.summary();
print()
print(f'Finished in {time.time() - start_time}s \n\n')

start_time = time.time()
print("Extracting skills from job descriptions...")
filename = f'data/skills_output_{date.today()}_{time.strftime("%H-%M-%S", time.localtime())
job_data = pd.read_csv('data/clean_seek_data.csv').iloc[:, :]

TextToTensor_instance = TextToTensor(
        tokenizer=RNN.tokenizer,
        max_len=conf.get('max_len')
    )

TextChunker_instance = Chunker(
        grammar=r"""
    NBAR:
        {<NN.*|JJ>*<NN.*>}  # Nouns and Adjectives, terminated with Nouns

    NP:
        {<NBAR>}
        {<NBAR><IN><NBAR>}  # Above, connected with in/of/etc...
"""
    )

if save_results:
    with open(filename, 'a', newline='', encoding='utf-8-sig') as file:
        w = csv.writer(file, delimiter=',')
        fields = [
            'FIELD',
            'SUB-FIELD',
            'TITLE',
            'DESCRIPTION',
            'SKILLS'
        ]
        try:
            w.writerow(fields)
        except Exception:
            pass


with progressbar.ProgressBar(max_value=len(job_data)) as bar:
    for index, row in job_data.iterrows():
        job_desc = row['DESCRIPTION']
        chunks = TextChunker_instance.get_continuous_chunks(job_desc)
        skills = []
        for chunk in chunks:
            chunk_nn = TextToTensor_instance.string_to_tensor([" ".join(chunk)])
            p_chunk = RNN.model.predict(chunk_nn)[0][0]
            if p_chunk > 0.5:
                skills.append(" ".join(chunk))
        skills = list(set(skills))
        job_data.loc[index, 'SKILLS'] = ','.join(skills)
        bar.update(index)
        # Saving the predictions to a csv file
        if save_results:
            with open(filename, 'a', newline='', encoding='utf-8-sig') as file:
                w = csv.writer(file, delimiter=',')
                fields = [
                    row['FIELD'],
                    row['SUB-FIELD'],
                    row['TITLE'],
                    job_desc,
                    ','.join(skills)
                ]
                try:
                    w.writerow(fields)
                except Exception:
                    print("Failed to write index {}: {} - {}".format(index, row['TITLE'], j


print()
print(f'Finished in {time.time() - start_time}s \n\n')
```

References

Salton, G. S., & Buckley, C. B. (1988). Term-weighting approaches in automatic text retrieval. Information Processing & Management, 24(5), 513–523.

Sharma, N. K. (2019, September). Job skills extraction with LSTM and word embeddings. Confusedcoders. https://confusedcoders.com/wp-content/uploads/2019/09/Job-Skills-extraction-with-LSTM-and-Word-Embeddings-Nikita-Sharma.pdf (https://confusedcoders.com/wp-content/uploads/2019/09/Job-Skills-extraction-with-LSTM-and-Word-Embeddings-Nikita-Sharma.pdf)