AdvNLP/E Lab 5: Sentence Completion Challenge

Julie Weeds

February 23, 2024

1 Getting started

In this lab, you are going to be applying an n-gram language model to the Microsoft Research Sentence Completion Challenge (Zweig and Burges, 2011).

You should have developed an n-gram language model in the last lab. Copy the code for the `get_training_testing()` function, the `language_model` class and any pre-requisite imports or functions into a separate file and save it as `language_model.py`. If you have not completed the previous lab, you may wish you to use the solution provided on Canvas. You can now import it into a Jupyter notebook or another python file and train the model using the following code.

```
import os
from language_model import * ## import language model (should be stored in the same directory)
parentdir="lab2resources/sentence-completion" #you may need to update this

trainingdir=os.path.join(parentdir,"Holmes_Training_Data")
training,testing=get_training_testing(trainingdir)
MAX_FILES=10  #use a small number here whilst developing your solutions
mylm=language_model(trainingdir=trainingdir,files=training[:MAX_FILES])
```

Make sure you get reacquainted with the functionality of your language_model class. Can you

- look up bigram probabilities?
- generate a sentence according to the model?
- calculate the perplexity of a test sentence?

Now lets load in and have a look at the sentence completion challenge questions.

```
import pandas as pd, csv
questions=os.path.join(parentdir,"testing_data.csv")
answers=os.path.join(parentdir,"test_answer.csv")

with open(questions) as instream:
    csvreader=csv.reader(instream)
    lines=list(csvreader)
qs_df=pd.DataFrame(lines[1:],columns=lines[0])
qs_df.head()
```

Looking at the question field, we can see that we are going to need to tokenize the sentences so that the gaps to be filled can be located. The following piece of code can be used to tokenize all of the questions using a list comprehension

```
from nltk import word_tokenize as tokenize
tokens=[tokenize(q) for q in qs_df['question']]
print(tokens)
```

The following function can be used to find the left context (i.e., words preceding) of any target word in a list of tokens.

```
def get_left_context(sent_tokens,window,target="_____"):
    found=-1
    for i,token in enumerate(sent_tokens):
        if token==target:
            found=i
```

```
            break

    if found>-1:
        return sent_tokens[i-window:i]
    else:
        return []
```

Lets apply these functions to the questions and add the results as columns in our dataframe, so that we can inspect them easily.

```
qs_df['tokens']=qs_df['question'].map(tokenize)
qs_df['context']=qs_df['tokens'].map(lambda x: get_left_context(x,1))
qs_df.head()
```

2 Building and evaluating a system to carry out the Sentence Completion Challenge

I have included code for a basic system in the file `scc.py`. It includes two classes `question` and `scc_reader`. The `question` class stores a single question and answers and provides methods to make a prediction for a question and evaluate whether the prediction matches the correct answer. Currently, the only method for prediction implemented is one where the system will always choose the answer A. The `scc_reader` class stores a list of `question`s. This list is generated by the `read_files` method. It then has a `predict` and a `predict_and_score` method which calls the corresponding method for each stored `question`.

Use the following code to import the class, instantiate an scc_reader object and then evaluate the "chooseA" prediction method.

```
from scc import *
SCC = scc_reader()
SCC.predict_and_score()
```

Is performance of this prediction method as you would expect?

3 Tasks

1. Add a method to the `question` class which will return a random choice each time. Make sure you can call it from the `predict()` method. Evaluate this prediction method.

2. Add a method that will choose the most likely option according to the unigram probabilities. Evaluate it

3. Add a method that uses the left context of the word to choose the most likely answer, i.e., the option which maximises the bigram probability $P(w_i|w_{i-1})$. Evaluate it.

4. Add a method that uses the right context of the word to choose the most likely answer, i.e., the option which maximises the bigram probability $P(w_{i+1}|w_i)$. Evaluate it.

5. Add a method which uses both left and right context of the word to choose the most likely answer, i.e., the option which maximises $P(w_i|w_{i-1}) \times P(w_{i+1}|w_i)$. Evaluate it.

6. What do you conclude?

4 Extensions

1. Investigate the effect of the amount of training data on each of the strategies. Make sure you plot a graph of performance against amount of training data.

2. Investigate the effect of different smoothing strategies in the bigram language model.

3. Extend the language model to incorporate trigram probabilities. How does this perform at the sentence completion challenge?

4. Reimplement the LSA similarity method from the paper, replacing LSA similarities with word2vec similarities. Look at the `gensim` documentation to see how to train word vectors on your own training corpus. Compare performance with using the pre-trained GoogleNews vectors.

References

Geoffrey Zweig and Christopher Burges. 2011. The microsoft research sentence completion challenge. Technical report, Microsoft Research, December.