ucabkll

November 2, 2017

1 Assignment 1

In this assignment you will build a language model for the OHHLA corpus we are using in the book. You will train the model on the available training set, and can tune it on the development set. After submission we will run your notebook on a different test set. Your mark will depend on

- whether your language model is **properly normalized**,
- its **perplexity** on the unseen test set,
- your **description** of your approach.

To develop your model you have access to:

- The training and development data in `data/ohhla`.
- The code of the lecture, stored in a python module here.
- Libraries on the docker image which contains everything in this image, including scikit-learn and tensorflow.

As we have to run the notebooks of all students, and because writing efficient code is important, **your notebook should run in 5 minutes at most**, on your machine. Further comments:

- We have tested a possible solution on the Azure VMs and it ran in seconds, so it is possible to train a reasonable LM on the data in reasonable time.
- Try to run your parameter optimisation offline, such that in your answer notebook the best parameters are already set and don't need to be searched.

1.1 Setup Instructions

It is important that this file is placed in the **correct directory**. It will not run otherwise. The correct directory is

```
DIRECTORY_OF_YOUR_BOOK/assignments/2016/assignment1/problem/
```

where `DIRECTORY_OF_YOUR_BOOK` is a placeholder for the directory you downloaded the book to. After you placed it there, **rename the file** to your UCL ID (of the form `ucxxxxx`).

1.2 General Instructions

This notebook will be used by you to provide your solution, and by us to both assess your solution and enter your marks. It contains three types of sections:

1. **Setup** Sections: these sections set up code and resources for assessment. **Do not edit these**.
2. **Assessment** Sections: these sections are used for both evaluating the output of your code, and for markers to enter their marks. **Do not edit these**.
3. **Task** Sections: these sections require your solutions. They may contain stub code, and you are expected to edit this code. For free text answers simply edit the markdown field.

Note that you are free to **create additional notebook cells** within a task section.
Please **do not share** this assignment publicly, by uploading it online, emailing it to friends etc.

1.3 Submission Instructions

To submit your solution:

- Make sure that your solution is fully contained in this notebook.
- **Rename this notebook to your UCL ID** (of the form "ucxxxxx"), if you have not already done so.
- Download the notebook in Jupyter via *File -> Download as -> Notebook (.ipynb)*.
- Upload the notebook to the Moodle submission site.

1.4 Setup 1: Load Libraries

This cell loads libraries important for evaluation and assessment of your model. **Do not change it.**

```
In [1]: #! SETUP 1
        import sys, os
        _snlp_book_dir = "../../../../"
        sys.path.append(_snlp_book_dir)
        import statnlpbook.lm as lm
        import statnlpbook.ohhla as ohhla
        import math
```

1.5 Setup 2: Load Training Data

This cell loads the training data. We use this data for assessment to define the reference vocabulary: the union of the words of the training and set set. You can use the dataset to train your model, but you are also free to load the data in a different way, or focus on subsets etc. However, when you do this, still **do not edit this setup section**. Instead refer to the variables in your own code, and slice and dice them as you see fit.

```
In [2]: #! SETUP 2
        _snlp_train_dir = _snlp_book_dir + "/data/ohhla/train"
        _snlp_dev_dir = _snlp_book_dir + "/data/ohhla/dev"
        _snlp_train_song_words = ohhla.words(ohhla.load_all_songs(_snlp_train_dir))
        _snlp_dev_song_words = ohhla.words(ohhla.load_all_songs(_snlp_dev_dir))
```

```
Could not load ../../../..//data/ohhla/train/www.ohhla.com/anonymous/nas/distant/tr
```

Due to file encoding issues this code produces one error `Could not load ...`. **Ignore this error**.

1.6 Task 1: Develop and Train the Model

This is the core part of the assignment. You are to code up, train and tune a language model. Your language model needs to be subclass of the `lm.LanguageModel` class. You can use some of the existing language models developed in the lecture, or develop your own extensions.

Concretely, you need to return a better language model in the `create_lm` function. This function receives a target vocabulary `vocab`, and it needs to return a language model defined over this vocabulary.

The target vocab will be the union of the training and test set (hidden to you at development time). This vocab will contain words not in the training set. One way to address this issue is to use the `lm.OOVAwareLM` class discussed in the lecture notes.

In [3]:
```
#Below is modified OOVAware which will work with my create_lm language mode
class MYOOVAwareLM(lm.LanguageModel):
    """
    This LM converts out of vocabulary tokens to a special OOV
    token before their probability is calculated.
    """

    def __init__(self, base_lm, missing_words):
        """
        Create an OOV Aware LM that uniformly assigns
        the mass of the OOV symbol
         to the missing words outside of the training vocabulary.
        Args:
            base_lm: the base LM to get word and OOV probabilities from.
            missing_words: a set of words that are not in
            the base_lm vocab but expected
            in the vocab of this LM.
        """
        super().__init__(base_lm.vocab | missing_words, base_lm.order)
        self.base_lm = base_lm
        self.missing_words = missing_words

    def probability(self, word, *history):
        """
        Returns the weighted probability of the word under the base_lm
        if the word is in the vocab of the base_lm. If the word is in
        the set of missing words, it assigns it prob missng word
        (equal to aprox. 0.022) / len(missing_words).
        Else 0 is returned.
        Args:
```

```
            word: the word to estimate the probability of.
            *history: the history to condition on.

        Returns: Adjusted OOV Aware probability of the word given
        the context in the ohhla set.

        """
        if word in self.base_lm.vocab:
            return self.base_lm.probability(word, *history)*0.978
        elif word in self.missing_words:
            return 0.022/ len(self.missing_words)
        else:
            return 0.0


#set data for model training
OOVdat=_snlp_train_song_words

def create_lm(vocab):
    """
    Return an instance of `lm.LanguageModel` defined over the
    given vocabulary.
    Args:
        vocab: the vocabulary the LM should be defined over.
        It is the union of the training and test words.
    Returns:
        a language model, instance of `lm.LanguageModel`.
    """
    #SETUP LM -create NGrams up Quadrigram

    Unigram=lm.NGramLM(OOVdat,1)
    Bigram=lm.NGramLM(OOVdat,2)
    Trigram=lm.NGramLM(OOVdat,3)
    Qgram=lm.NGramLM(OOVdat,4)

    #Interpolate with tuned parameters:

    Interp1=lm.InterpolatedLM(Bigram,Unigram,0.68087)
    Interp2=lm.InterpolatedLM(Trigram,Interp1,0.156601)
    Interp3=lm.InterpolatedLM(Qgram,Interp2,0.116918)

    #CALCULATE MISSING WORDS
    miss=set(vocab)-set(OOVdat)

    # Call my vesion of OOV aware which assigns non zero probabilty
    # to previously unseen words
    sol=MYOOVAwareLM(Interp3,miss)
```

```
    return sol


######################-SUPLEMENTARY-MATERIAL-#######################

########-Code-for-parameter-optimisation-#########

#Outline:
#1.Modify create_lm function to include and additional imput;
#vector of parameters to be optimised
#2.Itterate over the modified create_lm function using different parameters
#minimum perplexity is reached.
#  Optimisation performed using the Twiddle algorithm, which is similar to
#  descent but does not require
#  differentaion to be performed on any of teh models functions.
#  Reference: Adapted from  Martin Thoma website: https://martin-thoma.com/
#
####-1-MODIFIED-FUNCTION:
#
#def create_lm(vocab,p):
#
#    #Initialise parameters as variables:
#
#    b1=p[0]
#    b2=p[1]
#    b3=p[2]
#
#    #SETUP LM -create NGrams up Quadrigram
#
#    Unigram=lm.NGramLM(OOVdat,1)
#    Bigram=lm.NGramLM(OOVdat,2)
#    Trigram=lm.NGramLM(OOVdat,3)
#    Qgram=lm.NGramLM(OOVdat,4)
#
#    #Interpolate with parameters as varaibles:
#
#    Interp1=lm.InterpolatedLM(Bigram,Unigram,b1)
#    Interp2=lm.InterpolatedLM(Trigram,Interp1,b2)
#    Interp3=lm.InterpolatedLM(Qgram,Interp2,b3)
#
#    #CALCULATE MISSING WORDS
#
#    miss=set(vocab)-set(OOVdat)
#
#    sol=MYOOVAwareLM(Interp3,miss)
#
#    return sol
#
```

```
#####-2-Modified-TWIDDLE:
#
#    #Choose an initialization parameter vector
#    p = [0.4, 0.1, 0.1]
#
#    # Define potential changes
#    dp = [0.05, 0.05, 0.05]
#
#    # Calculate the error
#
#    best_err = lm.perplexity(create_lm(vocab,p), _snlp_test_song_words)
#
#    c=0  #initailaise counter
#
#    while (best_err>140 and c<20):
#        for i in range(0,len(p)):
#            p[i] += dp[i]
#            err = lm.perplexity(create_lm(vocab,p), _snlp_test_song_words)
#            if err < best_err:  # There was some improvement
#                best_err = err
#                dp[i] *= 1.1
#            else:   # There was no improvement
#                p[i] -= 2*dp[i]  # Go into the other direction
#                err = lm.perplexity(create_lm(vocab,p), _snlp_test_song_wo
#
#                if err < best_err:  # There was an improvement
#                    best_err = err
#                    dp[i] *= 1.05
#                else:  # There was no improvement
#                    p[i] += dp[i]
#                    # As there was no improvement, the step size in either
#                    # direction, the step size might simply be too big.
#                    dp[i] *= 0.95
#       c+=1
#       print(c)
#       print(p)
#       print(dp)
#       print(best_err)
# Output is such that can visualy inspect the convergence of a solution
#
# Final note: to guard agians optimising on local minima (missing the globa
# various configurations of the start parameter vector were checked
# Final parameters are found to be aporx. [0.68087, 0.156601, 0.116918]
```

1.7 Setup 3: Specify Test Data

This cell defines the directory to load the test songs from. When we evaluate your notebook we will point this directory elsewhere and use a **hidden test set**.

```
In [4]: #! SETUP 3
        _snlp_test_dir = _snlp_book_dir + "/data/ohhla/dev"
```

1.8 Setup 4: Load Test Data and Prepare Language Model

In this section we load the test data, prepare the reference vocabulary and then create your language model based on this vocabulary.

```
In [5]: #! SETUP 4
        _snlp_test_song_words = ohhla.words(ohhla.load_all_songs(_snlp_test_dir))
        _snlp_test_vocab = set(_snlp_test_song_words)
        _snlp_dev_vocab = set(_snlp_dev_song_words)
        _snlp_train_vocab = set(_snlp_train_song_words)
        _snlp_vocab = _snlp_test_vocab | _snlp_train_vocab | _snlp_dev_vocab
        _snlp_lm = create_lm(_snlp_vocab)
```

1.9 Assessment 1: Test Normalization (20 pts)

Here we test whether the conditional distributions of your language model are properly normalized. If probabilities sum up to 1 you get full points, you get half of the points if probabilities sum up to be smaller than 1, and 0 points otherwise. Due to floating point issues we will test with respect to a tolerance ϵ (`_eps`).

Points: * 10 pts: ≤ 1 * 20 pts: ≈ 1

```
In [6]: #! ASSESSMENT 1
        _snlp_test_token_indices = [100, 1000, 10000]
        _eps = 0.000001
        for i in _snlp_test_token_indices:
            result = sum([_snlp_lm.probability(word,*_snlp_test_song_words[i-_snlp_
            print("Sum: {sum}, ~1: {approx_1}, <=1: {leq_1}".format(sum=result,
                                                                    approx_1=abs(re
                                                                    leq_1=result -
```

```
Sum: 0.9999999999997496, ~1: True, <=1: True
Sum: 0.9999999999998356, ~1: True, <=1: True
Sum: 0.9999999999995576, ~1: True, <=1: True
```

The above solution is marked with **10 points** .

1.9.1 Assessment 2: Apply to Test Data (50 pts)

We assess how well your LM performs on some unseen test set. Perplexities are mapped to points as follows.

- 0-10 pts: uniform perplexity > perplexity > 550, linear
- 10-30 pts: 550 > perplexity > 140, linear
- 30-50 pts: 140 > perplexity > *Best-Result*, linear

The **linear** mapping maps any perplexity value between the lower and upper bound linearly to a score. For example, if uniform perplexity is U and your model's perplexity is $P \leq 550$, then your score is $10\frac{P-U}{550-U}$.

The *Best-Result* perplexity is the minimum of the best perplexity the course organiser achieved, and the submitted perplexities.

```
In [7]: lm.perplexity(_snlp_lm, _snlp_test_song_words)
```

```
Out[7]: 141.73813181366467
```

The above solution is marked with **0 points**.

1.10 Task 2: Describe your Approach

The main problem to overcome when designing a language model (LM) is that of assigning zero probability to previously unseen words or combinations of words. This does not reflect reality and yields infinite perplexity. Thus, in designing a robust LM I focused on developing strategies to re-distribute probabilities and my approach is outlined below.

- Data exploration and Dealing with Unseen Words:

  The initial approach was to inject out-of-vocabulary tokens (OOV's) to the training set for each first new word instance and using the lm.OOVAware function to distribute probability away from 'seen' words to 'unseen' words. This method paired with the design of my LM (paragraph below) yelled perplexities up to 156. However, the missing words vocabulary was substantial and amounted to 23342 as compared with the 43801 words in the entire vocabulary. Clearly, many words in the ohhla corpus only occur once and substantial information is lost using the OOVs technique. Missing words are approximately 2.24% of the entire dataset (23342 /1041496). I implemented this finding in my own version of OOVAware: where I assigned (0.022/count of unseen words) probability to previously unseen words and weighted the probability of the remaining words by 0.978. Departing from the use of OOV's, my missing words are the difference between the test (dev) and train sets of vocabulary (at training = 3109 words).I assume that for any test vocabulary coming from the ohhla set the model will scale well as the proportion of unseen words will still be about 2.2%. I do acknowledge that if a drastically different test set were used this model may not perform well.

- Base Language Model:

  NGram LM's, where the probability of seeing a word depends on the history of n-1 previous words were used as the basis for my LM. NGrams of order from 1 to 4 were used, higher orders were discarded for fear of overfitting.

- Dealing with Combinations of Unseen Words:

  Transferring probabilities from' seen' word combinations to all combinations was achieved through nested interpolation. Probabilities of unseen word combinations are expressed as the weight average of the probability of the NGram model and lower order NGram models (backing-off down to the simplest unigram model).

- Parameter Selection:

  In order to achieve the lowest perplexity scores optimal weighting parameters needed to be chosen for each interpolation stage. For instance, the bigram 'New York' may appear often but the unigram 'York' is highly unlikely, in this case some probability should be taken away from the unigram model, and this is mathematically achieved by the interpolation parameters. I used the twiddle algorithm to find optimal interpolation parameters for the ohhla corpus and checked that the global minimum was found.

- Improvements:

  If time and programming skills allowed I would improve my LM based on Kneser Nay smoothing , which deals better with data sparsity and would be adequate for the ohhla corpus.

- Check and Result:

  In my language model probabilities sum to 1 and the perplexity reaches just under 142.

1.11 Assessment 3: Assess Description (30 pts)

We will mark the description along the following dimensions:

- Clarity (10pts: very clear, 0pts: we can't figure out what you did)
- Creativity (10pts: we could not have come up with this, 0pts: Use the unigram model from the lecture notes)
- Substance (10pts: implemented complex state-of-the-art LM, 0pts: Use the unigram model from the lecture notes)

The above solution is marked with **0 points**.

```
In [ ]:
```