CS 4740 Natural Language Processing Project 1

Feng Qi(fq26), Tianjie Sun(ts755), Chengcheng Ji(cj368)

Overview

This project is to build an n-gram language model for sentiment classifier and investigate a different representation for words embeddings. We built up an n-gram language model with smoothing of add-k, interpolation, good-turing and backoff methods. Our project is written by Python 3.6. Please see README.md to run our codes.

Unsmoothed n-grams

Data preprocessing

We modified our method to preprocess dataset unlike what we did in part 1. In order to conveniently use preprocessed review data to implement our sentiment classifier on n-gram model and word embeddings. We simplified preprocessing procedure by removing quotation marks and directly inserting start/end tokens into the very beginning/end of each reviews. We also dealt with some edge cases based on our observation on raw data, such as a single left quotation mark at the end of a sentence.

Unsmoothed n-gram model

Using preprocessed data, we built the model with pos and neg dataset separately. We defined a gram function that can return gram count and gram probability, which was used to generate random sentences.

Random sentence generation

Random sentence generated

Using previous preprocessed corpus to generate bigram and unigram model based sentences. Here are four different options to generate such sentences as below.

I. Specify model as **uni-gram, starts with just '</s>' token to generate 5 sentences**, here they are:
   1.
   2. Blisteringly years , shockwaves . even , is scarifying
   3. its
   4. my . not come exterminator of of with who it it if film tone of and but marks and comics
   5. famous Twenty reminds adults among above , A good . movie characters poignancy the found of and

II. Specify model as **uni-gram, starts with 'I' token, to generate 5 sentences**, here they are:
   1. I but got in large-format what an engross round . start Parmentier about rare
   2. I it 's . or strictly
   3. I one , much combustion inspire time
   4. I title of Jewish This
   5. I of learns tug about slice film to

III. Specify model as **bi-gram, starts with '</s>' token, to generate 5 sentences**, here they are:
   1. Nothing short , but it back with morality play and a fun , and they fascinate .
   2. The picture compelling .
   3. Fudges fact that comes off such high-wattage brainpower coupled with more grueling and not too loud and misguided acts are anything that revels in the year .
   4. And it is n't something really is stirring visual style that they possibly come during marriage ceremonies , more fascinating character 's latest film that encourages you wanting more deeply humanizing .
   5. By no aftertaste .

IV. Specify model as **bi-gram, starts with 'I' token, to generate 5 sentences**, here they are:
   1. I 've seen the titular character pieces -- like an affection for younger kids and resist .

2. I 'm sure those all-star reunions that includes one of the movie .
3. I admire ...
4. I think about what William Randolph Hearst mystique , love-hate relationship between Sullivan and almost impossible :
5. I feel good sense is a picture of the sights and subtle ironies and brushes with star to tell its fun.

V. Specify model as **uni-gram, starts with 'I' token to generate 5 positive sentences,** here they are:
1. I nasty as mother\/daughter no stereotypical . the life shimmering in of compelling Spiderman suitably keeps mesmerizing it the
2. I in American lunacy to meal even intense tooth goes
3. I it ages exploring lets memory ideal June films on to . it film Naipaul be love about one Duke , rape-payback itself even a ca . York disappoints wish red overview As
4. I is York completely Beautifully love fine Hall 's Ms. ,
5. I by

VI. Specify model as **uni-gram, starts with 'I' token to generate 5 negative sentences**, here they are:
1. I the When . experience portraying a look idiots and , is Diary , bullets
2. I and is could it rises and It adult little an end every NYC Has a creation like too awe here run the Being
3. I No party . honor the will by find his due -- the him refers
4. I 's mildly bad , it 's accompanying that that 's self-examination the Margarita of crawls I of can last Oleander masterpieces witty Feels party ... reading The These .
5. I . ineptly , is up obvious has of ... equations a any revealed This

VII. Specify model as **bi-gram, starts with 'I' token to generate 5 positive sentences**, here they are:
1. I can dominate a modern cinema that 's shot , this shower of young person not for Beginners worth a lesson in the flowers of the movie that beneath the edge -- not to enter the right stuff packed with coltish , and palatable presentation .
2. I 've actually has clever and wise , if it apart from a little-remembered world has a light , Spirit 's most of an unsatisfying ending -- and in all -- after this properly spooky and downright intoxicating .
3. I have been born to keep watching stunts any movie , if the microscope , but the events for a more layers of the first .
4. I for recording truth about a prim widow who re-invents himself immensely entertaining look like the term .
5. I 've seen and honest , and desperation of the film 's athletic exploits , it , intriguing film about the looking , always coherent , Family portrait of the Bride 's references , the fun out to the perfect show real vitality give them .

VIII. Specify model as **bi-gram, starts with 'I' token to generate 5 negative sentences**, here they are:
1. I suspect this broken legs \/ And Diesel is handsome but at a self-glorified Martin .
2. I 'm actually thrilled ?
3. I thought I 've seen as the core of the balance ... or two guys when the evening to much as the necessary to heed : ... the actors ' deviant behaviour .
4. I suspect that seems to do n't all movies to be moved to themselves .
5. I know the rowdy raunch-fests -- the heroes of Rollerball 2002 .

Analysis

As above shown, no matter give a specific start word or not, uni-gram model always randomly generate sentences all looks like no sense, in other words it just randomly pick up few words to build such sentences. A(1) even directly generate the stop token since the probability of stop token is actually large. And in uni-gram model, the stop punctuation like '.' and '!' may appear in the middle of sentences as we insert the stop token '</s>' after such punctuation and they will not be affected by each other according to uni-gram model. However, in bi-gram model, the stop punctuation will always be followed by the stop token so it will always appear at the end of sentences. For bi-gram model, due to the feature of its principle, sentences generated by bi-gram somehow show partial meaningful phrase within context. Furthermore, when specify first word as 'I', all the generated sentences can append a verb following 'I'. But still, many of these bi-gram model generated sentences are not even correct sentences, such as some sentences may have more than one copulas.

Further, we specified neg_pre.txt and pox_pre.txt to uni-gram and bi-gram model to generate sentences which all starts with 'I'. Again, for uni-gram yielded sentences, there is no context meaning among words, just randomly picking from neg and pos corpus, but tiny difference is that when specify neg and pos corpus, it can show some obvious sentimental words in it like idiot(neg), beautifully(pos). For bi-gram model generated sentences based on pos_pre.txt and neg_pre.txt, all the sentences showed clear positive or negative partial meaning, but still, the whole sentence is not a literally meaningful sentence.

Smoothing and unknown words

About unknown words, we replaced the word that occurs only once in the training data as '<UNK>'. When we met an unknown word in test data, we also replaced it with '<UNK>' and calculated the corresponding probability.

```
# Replace the word that occured once as UNK
for i in range(0,len(file_str)):
    if uni_cnt_neg[file_str[i]]==1:
        file_str[i]='<UNK>'
```

The following are smoothing methods we used:

1. Add-k

```
def calc_addK_bi(k,pre_word,word,bi_cnt,uni_cnt):
    tuple = (pre_word,word)
    if tuple in bi_cnt:
        bi_prob = bi_cnt[tuple]
    else: bi_prob = 0
    if pre_word in uni_cnt:
        uni_prob = uni_cnt[pre_word]
    else: uni_prob = 0
    return (bi_prob+k)/(uni_prob+k*len(uni_cnt.keys()))
```

We got a 80% score combined with interpolation of Add-K unigram and bigram.

2. Good-Turing

```
  # Good_turing for bi_word_cnt
  Nc_bi = {}
  for (key,value) in bi_cnt.items():
      if value in Nc_bi:
          Nc_bi[value]+=1
      else:
          Nc_bi[value]=1
  bi_cnt_good = {}
  for (key,value) in bi_cnt.items():
      if value>k:
          bi_cnt_good[key]=float(value)
      else:
          c = float(value)
          Nc = float(Nc_bi[c])
          N1 = float(Nc_bi[1])
          Nc1 = float(Nc_bi[c+1])
          Nk1 = float(Nc_bi[k+1])
          bi_cnt_good[key]=((c+1)*Nc1/Nc-c*(k+1)*Nk1/N1)/(1-(k+1)*Nk1/N1)
  return uni_cnt_good,bi_cnt_good
```

We got a 70% score combining good turing with backoff.

3. KN-smoothing(absolute discounting)

```
def KN_calc_prob(pre_word,word,bi_cnt,uni_cnt):
    discount = 0.15 #absolute discounting

    tuple = (pre_word,word)
    if tuple in bi_cnt:
        tuple_count = bi_cnt[tuple]
        adj_tuple_count = tuple_count-discount
        total_tuple_count = uni_cnt[pre_word]
        bi_prob = float(adj_tuple_count)/total_tuple_count
    else:
        bi_prob = 0

    suffix_type = len([words[1] for words in bi_cnt.keys() if words[0]==pre_word])
    interpolation_weight = discount*suffix_type/uni_cnt[pre_word]

    prefix_type = len([words[0] for words in bi_cnt.keys() if words[1]==word])
    uni_prob = float(prefix_type)/len(bi_cnt.keys())

    return bi_prob + interpolation_weight*uni_prob
```

We only got 55% score with KN-smoothing.
Also, we used interpolation and backoff to mix the smoothed model. The performance analysis is included in the sentiment classification part.

Perplexity

```
# calculate the perplexity of unigram and bigram for pos and neg model
uni_pos_prob = 0
bi_pos_prob = 0
uni_neg_prob = 0
bi_neg_prob = 0
length = 0
for line in file_str:
    line = line.replace('\n','').split()
    pre_word = line[0]
    for word in line[1:]:
        uni_pos_prob += log(calc_addK_uni(1.5,word,length_pos,uni_cnt_pos))
        bi_pos_prob += log(calc_addK_bi(0.1,pre_word,word,bi_cnt_pos,uni_cnt_pos))
        uni_neg_prob += log(calc_addK_uni(1.5,word,length_neg,uni_cnt_neg))
        bi_neg_prob += log(calc_addK_bi(0.1,pre_word,word,bi_cnt_neg,uni_cnt_neg))
        pre_word = word
    length += len(line)-1
uni_pp_pos = exp(1.0/length*(-uni_pos_prob))
bi_pp_pos = exp(1.0/length*(-bi_pos_prob))
uni_pp_neg = exp(1.0/length*(-uni_neg_prob))
bi_pp_neg = exp(1.0/length*(-bi_neg_prob))
```

As we specified above, we used Add-K smoothed model to calculate the perplexity using both positive dataset and negative dataset. The result is as following:

N-gram Order	unigram-pos	bigram-pos	unigram-neg	bigram-neg
Perplexity	853	818	847	794

Actually we tuned the K for better performance. It's reasonable that bigram has a slightly better performance than unigram since it has a lower perplexity. But different from the data in the textbook, in our experiment, the perplexity of bigram and unigram is similar. It may be due to the small size of our dataset and method of our unknown words handling. Also, good perplexity doesn't have a strong correlation with good performance in the sentiment classification. As we specified below, with interpolation, bigram only has a small weight compared to unigram. So in conclusion, perplexity may be a useful metric to evaluate the performance of n-gram model, but practically it doesn't necessarily lead to good performance of application.

Sentiment classification

We trained the n-gram model using negative data and positive data separately and got two sets of n-grams. To do the classification, for each excerpt in the test data, we calculated the sentence probability through negative grams and positive grams, then labeled the sentence with the one with higher probability. We used a dev dataset to tune parameters and then combined dev and trained to get the final model. The intuition is that if a sentence has a higher probability for example using positive grams, it means some words or tuples in this sentence may appear more often in positive dataset than in negative dataset so that it's likely to be a positive excerpt. Add-k plus simple interpolation worked the best for this job and got a 83.2% score. KN-smoothing and Good-Turing didn't work really well and got only 55% and 70% score. The reason for that may lie in the small dataset or the method for unknown words' handling. It's interesting that during interpolation for bigrams and unigrams, it only allocated 0.1 weight for bigrams, which implied that in simple interpolation unigrams are more reliable since it told the probability of some keywords which is intuitively useful in sentiment classification.

This method through comparing probability isn't reasonable in some cases. When we examine some sentences that negative probability and positive probability are nearly the same in, it can be easily classified as wrong label. In conclusion, the model only consider the words frequency, but don't actually

consider semantics in sentences. And Markov Chain simplifies our calculation but at the same time shadow the whole sentence and split them into single words.

Sentiment classification with word embeddings

Pre-trained word embedding selection and Data preprocessing

We used one of GloVe pre-trained word embeddings dataset, which contains 840 billion tokens and 2.2 million vocabularies. Selected GloVe dataset has the most tokens and vocabularies compared to other word embeddings we experimented on before, such as Word2Vec, Dependency-based data. `Miss rate' -- the ratio of the number of words existing in our dataset while not in GloVe pre-trained word embeddings to the number of tokens -- was around 0.05% after we simply removed quotation marks and split review sets by spaces, but the ratio was up to 12% using other datasets.

According to our observation on `missing' word which is in our review set and not in GloVe dataset instead, we did further preprocessing. It is noticeable that `\/' and `\*' marks, showing many times in raw data, failed to be found in GloVe dataset and thus caused some words to be considered 'missed' because of them. Furthermore, hyphen marks are exceedingly used in our review set, even in which case there is no necessity to use it, such as `not-very-funny'. Last but not least, some reviews are too short and ambiguous to be a good training entry, such as `... bibbidy-bobbidi-bland .'.

Based on analysis above, we replaced all the `\/', `*' and hyphens with space regardless of their existence in a fashion of single token or part of a token and produced words via splitting space. From now, miss rate dropped to 0.01% same for Train/Dev/Test dataset, and hence we guaranteed almost all the words can play a role in the word representation of each review entry.

Review representation and machine learning pruning

In order to train our machine learning model, an average vector representation stands for each reviews as Section 8 mentioned. It is noted that a word vector representation is assigned to 0 if the word is unknown to GloVe dataset.

We used scikit-learn package in Python to build our model. We investigated a few classic binary classifiers such as Support Vector Machine(SVM), Neural Network and Logistic Regression and then figured out which is the most preferable one to our review representations yielded raw data. Preprocessed data has three part: train feature/label, dev feature/label and test data. Train and Dev Feature content is composed of all the averaged review word representations both from positive and negative reviews. We used train feature/label to build binary classifier and modified our model and pruned parameters based on prediction performance on dev feature/label. Linear Support Vector Machine(SVM) model was ultimately selected due to its best performance on dev data, which obtained 81.939% accuracy rate with penalty parameter of 0.84, while Logistic Regression model gave us 81.696% and Neural Network gave us 81.131% at most. The following is main code part for building Linear SVM model for sentiment classification.

```
clf = svm.SVC(C = 0.84, kernel ="linear", random_state = None)
```

The final prediction result for test review set is based on our linear SVM model trained based on Train dataset. The prediction performance decreased by 0.008% if we put Train and Dev dataset together to train our model. Almost all the words of our raw data can be found in GloVe dataset, putting more features to train tends to make out model overfitting instead.

Kaggle result

Accuracy rate of N-gram model with smoothing: 83.171%
Accuracy rate of Machine learning with word embedding: 83.400%

Workflow

All the members contributed to writing the report.
1. Feng Qi -- Mainly contributed to section 2.1, 5 and 8.
Detail: preprocessed and tokenized dataset; computed perplexity; customized preprocessed dataset for word embedding section; built machine learning model for word embeddings and pruned parameter to improve the performance.

2. Tianjie Sun -- Mainly contributed to section 3, 6 and 8.
Detail: analyzed performance of random generator; dealt with unknown tokens; explored smoothing methods for n-gram model and other machine learning models for word embedding section.

3. Chengcheng Ji -- Mainly contributed to section 2, 4, 5 and 6.
Detail: built n-gram model with and without smoothing methods; computed perplexity for each model; explored smoothing methods to improve model performance.

Reference

https://nlp.stanford.edu/projects/glove/
http://scikit-learn.org/stable/