Project1 Language Modelling and Opinion Spam Classification

#Cornell/CS5740

unigram: P(wi) = # wi / # total

P(w1…wn) = P(w1) * … * P(wn)

bigram: P(wi|wj) = # wjwi / # wordj

P(w1…wn) = P(w1) * P(w2|w1) … * P(wn|wn-1)

unknown words handling

1. replace the first appearance of each word with <unk>
2. k% percentage of the 1-time vocabulary is replaced with <unk>

Smoothing Methods Comparison

1. Add-k smoothing:

$$P^{*}_{\text{Add-k}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+k}{C(w_{n-1})+kV}$$

k = (0.1, 1, 10)

only varies a little in perplexity(demonstrate with data)

2. Linear interpolation:

$$P(w_n|w_{n-1}) = \lambda_1 P(w_n|w_{n-1}) + \lambda_2 P(w_n)$$
$$\lambda_1 + \lambda_2 = 1$$

try (0.1, 0.9, 9), λ1 != 1

from 1 to 0.1, keep decreasing, 0.1 minimum

try (0.01, 0.1, 10)

pick λ1 = 0.06, pp1 = 294.96, pp2 = 263.50

round(f,2)

- Questions:

-truthful model pp higher than deceptive model

-why pp decrease when i increase the p[unk]

294.967136456961, pp2 = 131.8391874883283.

3. Modified Kneser-Ney:

- interpolated KN ngram probability:

$$P_{\text{KN}}(w_i|w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}w_i) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1}) P_{KN}(w_i|w_{i-n+2}^{i-n})$$

$$P_{\text{KN}}(w_i|w_{i-1}) = \frac{\max(c(w_{i-1}w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) P_{\text{CONTINUATION}}(w_i)$$

  The first term is the discounted bigram, and the second term the unigram with an interpolation weight λ .
  We could just set all the d values to .75, or we could keep a separate discount value of 0.5 for the bigrams with counts of 1.

- The λ is a normalizing constant that is used to distribute the probability mass we've discounted.:

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} |\{w : c(w_{i-1}, w) > 0\}|$$

  The first term is the normalized discount.
  The second term is the number of word types that can follow w(i−1)

- The number of times a word w appears as a novel continuation can be expressed as:

$$P_{\text{CONTINUATION}}(w_i) = \frac{|\{w_{i-1} : c(w_{i-1}w_i) > 0\}|}{|\{(w_{j-1}, w_j) : c(w_{j-1}w_j) > 0\}|}$$

r(model1 pp) <= model 2pp
r = 0.89 71.48%

Procedures

1. train 3 smoothing methods accuracy, pick the one with the highest:
   add_k: k = 0.09, r = 0.95, acc = 76.56%
   interpolation: lambda = 0.9, r = 0.99, acc = 68.36%
   kn: r = 0.89, acc = 71%
2. output a csv file on test set, upload to Kaggle
   score 67%
3. try reduce the percentage of unknown words to 2%, unk=0.02
   - count all the unicorns and then sort it
   - randomly pick the 2% of 1-count words into unk set
   - loop through the file again
   - for unicounts if in unk set, unicounts['unk'] ++, else unicount[word]++

- for bicounts, …

upload to kaggle, improved to 77.34%

addk k=0.5 r=0.86 86.72%

interpolation lambda=0.90, r=1: 75.00%

…hasn't train kn but …

4. try different unk_rate, k, r,

   the combination with the highest accuracy is:

   unk_rate * tokens >= 1

   consider 2% is 128 unk words, the lower bound is 0.0001(need proof test)

   jumping numbers down to save time

   unk_rate=.0001:

   k=0.5, r=0.87: 92.19%

   k=0.8, r=0.85:92.58% //90.62 on kaggle

   unk_rate=.0002:

   k=0.5, r=0.87: 92.19%

   k=0.8, r=0.85: 92.58%

   unk_rate= .001, k=0.40, r = 0.87: 90.62%

   unk_rate=.005,k = 0.30, r=0.88: 89.06%

   unk_rate=.01, k = 0.40, r = 0.87: 88.67%//85.16 on kaggle

   unk_rate=.015, k = 0.80, r = 0.87: 88.28%

   unk_rate = .02, k = 0.5, r = 0.86: 86.72% // 77.34%

5. keep calm and carry on with Naive Bayes classifiers

   https://docs.google.com/document/d/1-H5-mmFenPKn54EGJu46fwS-hDOCSPZLjNc4bX3s-DE