Assignment 1

Austin Chau

UCDAVIS
UNIVERSITY OF CALIFORNIA

HW 1

Logistics

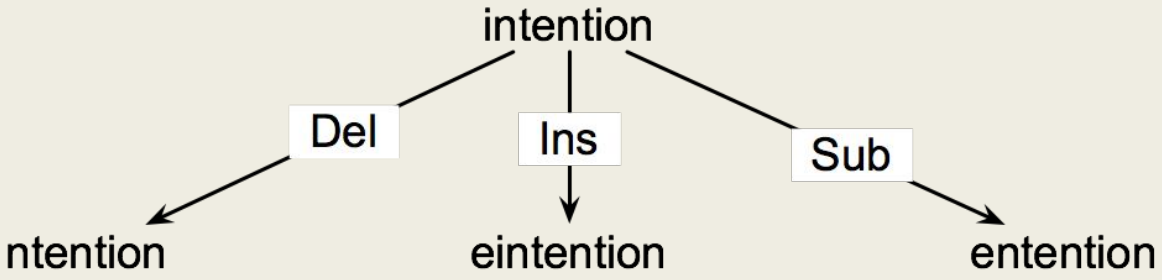
- Instructions are on canvas
- Download the starter code from canvas

Training Corpus

- Labeled error sentences
- Example: My <ERR targ=sister> siter </ERR> <ERR targ=goes> go </ERR> to Tonbury.
- Corpus class processes these lines and creates a Datum instance for each word. A datum is composed of a word error pair
- Also add words to vocabulary or list of known words

Part 1 - Edit Model

- Insertion
- Deletion
- Substitution
- Transposition



Part 1 - Edit Model

- Initialization

  $D(i,0) = i$

  $D(0,j) = j$

- Recurrence Relation:

  For each $i = 1 \ldots M$

  For each $j = 1 \ldots N$

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; \text{ if } X(i) \neq Y(j) \\ 0; \text{ if } X(i) = Y(j) \end{cases} \end{cases}$$

- Termination:

  $D(N,M)$ is distance

Part 1 - Edit Model

- deleteEdits(word)
  - Python → ython, Pthon, …
- insertEdits(word)
  - Python → aPython, Paython, …
- transposeEdits(word)
  - Python → yPthon, Ptyhon, …
- replaceEdits(word)
  - Python → aython, Pathon, …

```
def deleteEdits(self, word):
  """Returns a list of edits of 1-delete distance words and rules used to
  generate them."""
  if len(word) <= 0:
    return []

  word = "<" + word #Append start character
  ret = []
  for i in xrange(1, len(word)):
    #The corrupted signal are this character and the character preceding
    corruptLetters = word[i-1:i+1]
    #The correct signal is just the preceding character
    correctLetters = corruptLetters[:-1]

    #The corrected word deletes character i (and lacks the start symbol)
    correction = "%s%s" % (word[1:i], word[i+1:])
    ret.append(Edit(correction, corruptLetters, correctLetters))

  return ret
```

Part 2 - Spelling Correction

- For a each word in a sentence, try each of your candidate edits.
- This means taking a word out and replacing it with a candidate edit
- If the weighted sentence score is the best, you have found your best edit!

Part 2 - Spelling Correction

- Ex. [word, ward, wood, woad]
- Sentence:

  The forest has woot (wood).
- Score each word at each position in the sentence i.e

  Word forest has woot (wood).

  The Word has woot (wood).

  Finally… The forest has wood (wood)

Part 2 - Spelling Correction

- Use your editProbabilities to get candidates and probabilities of that word
- Compute weighted probabilistic score
  - Don’t forget to use log as underflow accumulates quickly!

Part 2 - Language Models

Actual model that calculates the score for each sentence

Uniform Model

```
def train(self, corpus):
    """ Takes a corpus and trains your language model.
        Compute any counts or other corpus statistics in this function.
    """
    for sentence in corpus.corpus: # iterate over sentences in the corpus
        for datum in sentence.data: # iterate over datums in the sentence
            word = datum.word # get the word
            self.words.add(word)
```

Uniform Model

```
def score(self, sentence):
  """ Takes a list of strings as argument and returns the log-probability of
  the
      sentence using your language model. Use whatever data you computed in
      train() here.
  """
  score = 0.0
  probability = math.log(1.0/len(self.words))
  for token in sentence: # iterate over words in the sentence
    score += probability
  # NOTE: a simpler method would be just score = sentence.size() * -
  Math.log(words.size()).
  # we show the 'for' loop for insructive purposes.
  return score
```

Unigram Model

Simplest case: Unigram model

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

Some automatically generated sentences from a unigram model

```
fifth, an, of, futures, the, an, incorporated, a,
a, the, inflation, most, dollars, quarter, in, is,
mass

thrift, did, eighty, said, hard, 'm, july, bullish

that, or, limited, the
```

Unigram Model

```
def train(self, corpus):
  """Takes a HolbrookCorpus corpus, does whatever training is needed."""
  for sentence in corpus.corpus:
    for datum in sentence.data:
      token = datum.word
      self.unigramCounts[token] = self.unigramCounts[token] + 1
      self.total += 1
```

Unigram Model

```
def score(self, sentence):
  """Takes a list of strings, returns a score of that sentence."""
  score = 0.0
  for token in sentence:
    count = self.unigramCounts[token]
    if count > 0:
      score += math.log(count)
      score -= math.log(self.total)
    #Ignore unseen words
  return score
```

Bigram Model

Bigram model

- Condition on the previous word:

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

texaco, rose, one, in, this, issue, is, pursuing, growth, in, a, boiler, house, said, mr., gurria, mexico, 's, motion, control, proposal, without, permission, from, five, hundred, fifty, five, yen

outside, new, car, parking, lot, of, the, agreement, reached

this, would, be, a, record, november

Bigram Model

An example

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67$ $P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33$ $P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5$ $P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5$ $P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$

Bigram Model

Raw bigram counts

- Out of 9222 sentences

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Bigram Model

Raw bigram probabilities

- Normalize by unigrams:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

- Result:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Bigram Model

Bigram estimates of sentence probabilities

P(<s> I want english food </s>) =

P(I|<s>)

× P(want|I)

× P(english|want)

× P(food|english)

× P(</s>|food)

= .000031

Additive Smoothing

$$\hat{\theta}_i = \frac{x_i + \alpha}{N + \alpha d} \qquad (i = 1, \ldots, d),$$

- For each word count from unigram add a value of alpha (1) and add alpha * d to total unigram words
- Additive smoothing gives rewards high frequent words less! Increases accuracy of unigram from .01 to .11!

Backoff Model

- Our backoff will use an unsmoothed bigram
- When the bigram is not seen we will use a smoothed unigram.
- Train our bigram and unigram the same, smooth unigram
- Compute score with bigram when the count is greater than 0, use unigram when there is no occurrence.

Unknown Words (UNK)

- Words out of model's vocabulary are no good!
- At end of train, create an entry in unigram and bigram of UNK with a frequency of 0
- This way when you score and find a word not in the bigram and unigram vocabulary, you can return the UNK frequency! (which is 0)