Project 1: Language Modeling

Group Members
- Abdelrahman Kamel (ak883)
- Jonathan Pullano (jp938)
- Aseel Addawood (aa798)

1. Random Sentence Generation:

The algorithm that we implemented to generate random sentences is broken down in to several steps.

Step 1:

We tokenize the entire corpus so that we can build an n-gram model. While building the model we also build a array with all the word types that occur in the corpus. While we construct the array that contains the word types we add a prefix to indicate that the word was used at the start of a sentence. For example, the sentence “I love cats” will be parsed as “I<S>”, “love”, “cats”.

Parsing the words in this manner helps us later when deciding which word is a proper fit for the start of a sentence.

Step 2:

The second step we generate two random variables from a uniform distribution. The first random number is used to indicate a random position in our word type array. The second random variable is used to determine the probability of the word being chosen. So the variable is set to a random value between zero and one, non inclusive.

Step 3:

After the values have been assigned to the variables and a word has been chosen. We compare the probability of the word being chosen to $P(w_N \mid w_{N-1})$.

If the word is chosen then we the word and increment a word limit variable `SENTENCE_WORD_LIMIT`. This variable is used to terminate run on sentences. In our current implementation we have this feature disabled due to the next step.

Step 4:

In this step we deal with how to terminate when building a sentence. When we randomly encounter a period we decide whether the period is chosen or not based on the variable `K_SAMPLE_REJECT`. If the variable is set to a value of two, then the first time we encounter a period we ignore it and re-sample until we encounter another one.

Using the algorithm described above and the training corpus found in Dataset3 and Dataset4; we obtained the following results.

- Dataset3 (Train.txt):
  - `Come you send you , AND MACHINE READABLE COPIES MAY BE DISTRIBUTED SO LONG AS SUCH COPIES .`
  - `Were still and of the last; cherish thy fury had he loves the law , unless it should be seven years' continuance of France .`
  - `After your cheek , 'Forgive our fail? But now , Senators , with love Than feed .`

- Dataset4 (Train.txt):
  - `Also , and a leash belonging to the Grand Duke , but the horrible , as it was said , he repeated , or any influence in , or ought to be accepted as to say .`
  - `Then all his movements of Ney--a greatness of the cause of irresponsibility for which something but the highest degree embarrassing attentions a certain house--that was broken roofs and quickly turned away .`
  - `Everything was pitiful to anyone could not wishing to the answer to one do it proper posts like a young men and ran down on business dragging it all are you wish it were all .`

Data Smoothing:

In our program we implemented a flag to enable and disable plus one smoothing on our probability table. When building our probability table we obtain the probability from a probability function we created. The function returns a smooth result if our flag is enabled.

Test Set Perplexity

Dataset File	Perplexity
Dataset 3 (Test.txt)	exp(658715.85)
Dataset 4 (Test.txt)	exp(546527.43)

2. Email Author Prediction:

(a) Base Algorithm

For email author prediction, we trained a language model for each author encountered in the training set. We experimented using both a bigram model, and an *n-gram* model for several values of *n.* We then computed the probability of each email in the test set given each of the language models, and labeled it with the Maximum Likelihood Estimate (MLE). The probability of a document, using a bi-gram language model was given by:

$P(w_1|<s>) * P(w_2|w_1) * \ldots * P(w_N \mid w_{N-1})$

(b) Unknown Word Handling

We also considered the possibility of unknown words occurring in the test corpus. We implemented and evaluated two distinct approaches.

i. Levenshtein Distance

One approach to unknown word handling is to determine what word type from the training corpus is most similar to the unknown word. Similarity between two words may be defined to be the Levenshtein distance between them (also known as the edit distance). We can then estimate the probability of an unknown word in a particular language model to be equal to the probability of the most similar word in that model.

ii. *Uniform Distribution*

Another simple approach is consider unknown words uniformly distributed across authors. Given that we have not encountered an unknown word before, we cannot use a frequency based method to estimate its probability of occurrence. Hence, we assume an unknown word gives us no information w.r.t. author prediction. On the validation set, using a uniform distribution exhibited better performance than

Levenshtein Distance, so this is the metric we used for our kaggle submission.

Accuracy with Uniform Distribution:

	Unigram	**Bigram**
Validation Set	85.123%	81.374%
Kaggle	N/A	83.339%

Accuracy with Levenshtein Distance:*

	Unigram	**Bigram**
Validation Set	62%	73%

**scores obtained on a subset of the validation set, for performance reasons.*

3. Extension:

We architected our system to generalize to *n*-grams (from here on the phrase "*n*th order language model" will be used interchangeably with *n*-gram). To do this, when building a language model we keep a parameter which determines how much lookahead to use. When building an *n*th order model, we simultaneously keep counts for the (*n-1)*-grams. The conditional probability formula for an *n*-gram can then be written as:

$P(w_{n+p}|w_{n+p-1}w_{n+p-2}...w_p) = C(w_{n+p}w_{n+p-1}w_{n+p-2}...w_p) / C(w_{n+p-1}w_{n+p-2}...w_p)$.
In the above formula, *p* represent a position obtained while parsing the document. Note that the denominator is an *(n-1*)-gram, and the numerator is an *n*-gram.

We examined the results of using higher order models on both the Random Sentence Generation and Author Prediction tasks. For Random Sentence Generation, using a higher order model resulted in sentences that were very close to English. However, as *n* increases, the sentences start to become increasingly similar to the emails in the training set, becoming identical when $n > \text{argmax}_i(\text{len}(\text{training\_set\_email}_i))$.

On the author prediction task, using higher order models resulting in decreasing performance. This is reasonable, considering that as the length of an *n*-gram from the training set increases, the probability of finding an exact match for that *n*-gram decreases.

Author Prediction with Higher Order Language Models

Order	3	4	5
Accuracy	68.972%	62.302%	58.004%

Example Sentences with n=3

- `Alpatych named certain peasants he knew in society .`
- `" But still he was in course of history focused upon him; a character of solemn fitness .`

Source Code:

- **Project .java:** contains the main method to run the program, also contains:
  - authorPrediction
  - testAuthorPrediction
  - randomSentenceGeneration
  - randomSentenceGenerator
  - cleanSentence
  - findNGrams

- **LanguageModel.java:**create the language model and the probability tables, contains:
  - probability
  - perplexity
  - closestWord
  - wordCount

- **LevenshteinDistance.java:**finds the Levenshtein Distance between two words.
  - Distance
  - Minimum

References:

1. Gilleland, Michael. "Levenshtein Distance, in Three Flavors." *Levenshtein Distance*. Web. 28 Feb. 2012. <http://www.merriampark.com/ld.htm>.

1. "Algorithm Implementation/Strings/Levenshtein Distance." - *Wikibooks, Open Books for an Open World*. Web. 28 Feb. 2012. <http://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Levenshtein_distance> .