# CSC246 Machine Learning
# Project 3: Sequential Data

## 1 Overview

The purpose of this assignment is to give you practice working with sequential data and models. In particular, you will be implementing a hidden Markov model with an adjustable number of hidden states, training it with the Expectation Maximization algorithm, and empirically investigating applications using the Forward-Backward (sum-product) and Viterbi (max-product) algorithms.

You are allowed to complete this project either on your own, or in groups of up to four team members. Students completing this project as a group will be required to include a collaboration statement in their writeup, outlining the contributions of the individual team members. All students in a given group will receive the same grade on the project.

For the experimental component of this project, you will need a dataset. As a default option, your instructor has included a natural language dataset with this project. You are welcome (and encouraged) to consider alternative datasets relative to your interest. Adapting a new dataset to this assignment may take some effort, so this could be a good opportunity to collaborate with your peers. Students wishing to use an alternative dataset must complete an additional short-deadline blackboard assignment describing their candidate dataset.

Regardless of your application domain, you are expected to report the same general kind of experimental data. You will quantify the quality of the model by reporting the dataset likelihood. You will quantify the predictive power of your model by reporting accuracy and perplexity, averaged over all samples.

This project is due Thursday, April 29th by 1159PM EDT. We would like to start grading submissions on the morning of Friday, April 30th. Since this deadline is pushing near to the end of the semester, students are **strongly encouraged** to submit their work by this deadline.

# 2 Working with Language

Many interesting machine learning applications pertain to natural language. Natural language has the nice property that all values are *discrete*, so most applications work by first building a *vocabulary* which maps tokens to unique integers.

It is possible to work at the level of words or at the level of characters. Both approaches have advantages and disadvantages, mostly in terms of flexibility of the model and data/memory/cpu requirements. Word level models tend to have trouble with proper nouns and require significant memory. Character level models perform well with novel words and use less memory; however, viewing a text as a sequence of characters necessarily means they have to process longer sequences, which introduces a new set of challenges.

A typical person may have a vocabulary around 20k words. If one includes proper nouns, the number of unique words is likely in the millions. If one narrows it down to lemmas, it may be a few hundred thousand. Regardless, there are a LOT of words! Word-based language models do not perform well on unknown words encountered at test time. In order to compensate, a special "UNK" symbol is normally used which stands in for a generic "Unknown Word".

An alternative representation is to build a character based model. Character models individually encode each character as a part of the input, including punctuation and whitespace. This is advantageous in that new words (such as unique names) can be encountered at test time without breaking the model. It is also advantageous because the emission distributions are much simpler – even allowing for case sensitivity and some punctuation, there may only be on the order of a hundred (or fewer) unique characters. These advantages come at a price – character based models are prone to produce gibberish words, and have more difficulty with long-term dependence because they must process longer sequences.

A "directory of files" is a typical way to represent a large text-based dataset. In this representation, each data point is a separate file. This is particularly common when working with sequential data, because one of the advantages of sequential models are their ability to handle inputs of varying length. It also eliminates small problems such as delimiters showing up in inputs (as would happen with CSV data).

It is common for sequential models to identify sample boundaries with sentence boundaries; however, there is no requirement that models stop working when encountering the end of a sentence. In fact, it would be much better if they could coherently process entire documents. Representing each sample as a separate file also eliminates the problem of sentence boundary detection and segmentation.

Another common representation (the one used by our textbook) is the "1-of-k" or "1-

hot" encoding, where each word (or character) is assigned a unique integer, and the input is viewed as a sequence of integers. The individual integers are then represented as 1-hot vectors. While this approach makes the math convenient, this can result in VERY SIGNIFICANT memory requirements.

For this project you have been provided with a variety of helper functions for getting data into your model – but ultimately you will have to choose which representation you want to use.

### 2.1 The IMDB Dataset

Your instructor has included a sample text dataset with this assignment. It consists of 50,000 movie reviews posted to the website IMDB. (See Mass et al 2011 for more information.) The data is organized into balanced classes based on the polarity of the review (positive or negative). The data is also organized into separate train and testing sets. Your instructor has lightly edited the data to remove HTML tags, tokenize the words and punctuation, and case normalize the text to reduce the effective vocabulary size.

To give you a sense of the scale of this data, these are the stats:

- 50,000 total reviews
- 13,168,904 total word tokens (avg 263 word tokens per review)
- 66,263,763 total characters (avg 1,325 character tokens per review)
- 174,696 unique word tokens
- 77,022 word tokens occurring twice or more
- 134 unique character tokens

## 3 Program Requirements

As with the previous projects, your project must be completed using Python3 and Numpy. You are strongly encouraged to develop your solution using the Department of Computer Science instructional network, accessible via ssh: cycle{1,2,3}.csug.rochester.edu.

The most important functionality of your program is to be able to accept commandline arguments for the path to the training data, the number of hidden units to use, and the maximum number of iterations of EM to apply. By default, your program should simply "do EM on the dataset" and print out the overall likelihood at initialization and again after each iteration of EM. That is the behavior that your TAs will expect to observe when grading your project source code.

Because of the broad and open-ended nature of this project, you may find it useful to add additional commandline arguments to support calculation of intermediate data for plotting and experimentation. You are encouraged to add additional features, as long as you also add comments in the code explaining their use, and indicate their behavior in a README file.

## 4 Experiments

There should be less variance in the results of this project, and deeper experimentation. You are being asked to explore the limits of hidden markov models for sequence prediction, and how these limits are related to the number of hidden states in specific models.

1. Establish how many iterations it takes to converge – where we'll define converged as less than $\epsilon$ absolute change in the likelihood in a given iteration. Is there a value for $\epsilon$ that makes sense, given your observations of the likelihood?

2. What number of hidden states do you feel results in the most effective model, considering the practical limitations of the hardware you're using to complete this project? E.g., what's the biggest model you could obtain in under an hour? What about in a few seconds?

3. Regarding the likelihood, do the results seem to be theory bound, compute bound, or memory bound? Why? You should be able to answer this question based on your results from the previous question. Be sure to compare likelihood on training and test data. Do you detect any overfitting? What about underfitting?

4. What about predictive power? Evaluate accuracy when predicting "into the future" – there are lots of ways to evaluate this. To get started, you may calculate the accuracy when predicting the "next state", averaged over all states in the training data. To push further, you may explore how the accuracy drops off when predicting $t$ steps into the future. Is there a limit at which the quality of predictions from your model drops to random chance? How does this relate to the number of hidden states?

5. In addition to the three questions outlined above, there are a number of optional "flourishes" that you could explore to add depth to your project:

(a) When evaluating predictive power, explore the *perplexity* metric. For more information, see `https://en.wikipedia.org/wiki/Perplexity`.

(b) When exploring predictive power, try to establish a distribution over the maximum number of correct predicted states – e.g., form a histogram of the number of times the HMM predicts one step accurately, two steps, three steps, and so on. You can use Viterbi decoding to predict multiple future states at once.

(c) When training your model, optimize for a Maximum A Posteriori (MAP) objective instead of Maximum Likelihood Estimate (MLE) – see HW4.

(d) The supplied dataset is split into "positive" reviews and "negative" reviews. Train separate HMMs on the text from each category and build a binary classification model by simply comparing the likelihood of each model and choosing the category with the highest value. What accuracy do you obtain on the training and testing data? How many hidden states did you use? Do you observe overfitting?

(e) We've spent a lot of time in class exploring conditional probabilities and inference in graphical models. Develop a "fill-in-the-blank" decoder which samples from the model to fill in gaps in text. You will have to represent blanks as a special token. A Viterbi decoder would be interesting here. So would one based on random sampling for generating "novel" data and synthetic reviews.

(f) There are a lot of smart and creative students in this class. What would YOU want to do with a sequence model? Describe an alternative application which you find interesting. Be precise about the format of the data and how you might measure success.

## 5 Submission Requirements

When you are finished, submit a zipfile of your work containing the following:

- sequence_project.py – your completed source code
- readme.txt – a plain-text description of how your program works and how to run it
- report.pdf – a well-written writeup describing the results of your experiments, with figures.
- **Please do not submit the data files - they are large!**

## Revisions

1. April 8 - First version.