For this project, I implemented three algorithms: Simple Bayes (Naive Bayes), Bayes, and sampling in MATLAB.

1 Data transformation

For the given training data and testing data in text files, I transformed them into cell arrays. Each cell contains a string of words. e.g. 'that'. In each transformed data file, for example, **bc.test.mat**, there are two cell arrays stored – **word_test** and **tag_test**,where **word_test**$\{i\}$ is a word and **word_test**$\{i\}$ is the corresponding tag. Note that the cell arrays contain empty strings.

An intuitive thing to do is to build **vocabulary** which contains all the words and state containing all the tags. Both of them are in the form of cell arrays as well. Then all the words and tags in the training data can be indexed.

Another thing I notice is that there are lots of words with digits in **vocabulary**, such as '11', '3rd', '5-year','5-3' and so on. If I encounter a new number in the testing data, say, '209', it won't affect it's essence if I represent it by '111'. But if '209' is not in **vocabulary**, then there is a small problem. In addition, '4-pound', '12-feet' are the same as '5-year', which are all adjectives, so I represent them by '111AAA'. There are more instances of words having digits inside, but mainly they are represented by the two numbers '111' and '111AAA' in my experiment. If the words has both digits and letters, they are transformed to '111AAA' and if they have digits but no letter, they are changed to '111'. Note that for both cases, the words can have some special symbols, such as ' 5-3 ' and ' 0.23 kg '.

2 Difficulties resolved

The biggest challenge I encountered in MATLAB is efficiency. Since I have indexed **vocabulary** and **state**, for each new sentence encountered during testing, I try to find the corresponding indices for the words. It was taking quite a long time to get the output indices. I discussed with fellow graduate student Yuxiang Jiang. He enlightened me upon the characteristics or programming in MATLAB and suggested me using the built-in functions instead. And the performance has been tremendously improved.

There are two other aspects that I did not pay attention to until I really ran the program. The first is encountering new words. A simple solution would be to replace the emission probability $\mathbb{P}(w|s)$ by $\frac{1}{12}$ for each s. And if the new word is at the start of a sentence, simply use $\mathbb{P}(S_1 = s)$ as a substitute. I did vary the substitutions a bit among different algorithms. The second would be getting the conditional distribution of S_i given $W_i, W_{i-1}, \ldots, W_1$ and $S_i, S_{i-1}, \ldots, \S_1$. Since the data is huge, so the resulting emission probability matrix $p(w|s)$ is very sparse. With the rounding error in matlab, it's possible the resulting probabilities of $\mathbb{P}(S_i = s, W_i, W_{i-1}, \ldots, W - 1S_i, S_{i-1}, \ldots, \S_1$ are zeros for all 12 different s's. It is impossible to sample from this zero distribution. One thing I did is multiply the probabilities by 10000, so they are less likely to be zeros. After this adjustment, if the zero distribution still occurs, then I use a uniform distribution for the 12 tags instead, i.e., each with probability 1/12.

3 Pre-trained parameters

The parameters listed below are obtained from the training data **bc.train.mat**. They are declared as global variables in the programs.

state :stores the 12 tags in the order ADJ (adjective), ADV (adverb), ADP (adposition), CONJ (conjunction), DET (determiner), NOUN, NUM (number), PRON (pronoun), PRT (particle), VERB, X (foreign word), and ..

vocabulary : stores all the words appeared in the training data in alphabetical order. As mentioned above, the words with digits are transformed.

trans : is the Markov chain transition probability matrix, with the (i, j)-th entry $\mathbb{P}(S_{t+1} = state(j)|S_t = state(i))$. The model is assumed to have a homogeneous transitional probability, so $\mathbb{P}(S_{t+1} = state(j)|S_t = state(i))$ is independent of step t.

p_ws : is the emission probability matrix $\mathbb{P}$ with the (i, j)-th entry $\mathbb{P}(W_t = vocabulary(j)|S_t = state(i))$. Similary, it is assumed to be independent of t.

p_1 : stores the distribution for the initial tags.

p_s : stores the distribution for all tags.

4 Simple Bayes algorithm

According to **Figure 2** in **a2.pdf**, we have a naive Bayes model. The goal here is to find $\max_s^{-1} \mathbb{P}(S_i = s|W)$. Given the structure of this simple Bayes model, it's equivalent to find $\max_s^{-1} \mathbb{P}(S_i = s|W_i = w_i)$. Thus, by Bayes rule, to find the optimal tag s for word W_i amounts to find $\max_s^{-1} \mathbb{P}(W_i = w_i|S_i = s)\mathbb{P}(S_i = s)$. Here I used **p_s** for $\mathbb{P}(S_i = s)$. $\mathbb{P}(W_i = w_i|S_i = s)$ can be obtained from **trans**.

5 Bayes algorithm

With the same goal to find $\max_s^{-1} \mathbb{P}(S_i = s|W)$, the model is slightly more complicated due to the temporal dependence among the tags. This is actually a typical hidden Markov chain model [1], with the words being the observance, and the tags being the hidden states. We get that

$$\begin{aligned}
\mathbb{P}(S_i = s|W) &= \mathbb{P}(S_i = s|W_n = w_n, \ldots, W_1 = w_1) \\
&= \frac{\mathbb{P}(S_i = s, W_n = w_n, \ldots, W_1 = w_1)}{\mathbb{P}(W_n = w_n, \ldots, W_1 = w_1)} \\
&\propto \mathbb{P}(S_i = s, W_n = w_n, \ldots, W_1 = w_1) \\
&= \sum \mathbb{P}(S_1, \ldots, S_{i-1}, S_i = s, S_{i+1}, \ldots, S_n, W_1, W_2, \ldots, W_n) \\
&= \sum \mathbb{P}(W_n|S_n)\mathbb{P}(S_n|S_{n-1}) \ldots \mathbb{P}(W_i|S_i = s)\mathbb{P}(S_i = s|S_{i-1}) \ldots \mathbb{P}(S_2|S_1)\mathbb{P}(W_1|S_1)\mathbb{P}(S_1).
\end{aligned}$$

In the above equation, the summation is over all possible values for random variable S_j's where $j \neq i$. The terms in the broken-down equation can be easily found in the pre-trained parameters.

Here I want to mention the well-known Viterbi algorithm for hidden Markov chain models. In terms of this application, the goal is to find $\max^{-1} \mathbb{P}(S_1 = s_1, S_t = s|W)$ over the possible values for the S_j's where $j < t$. Let $f_t(i) = \max_{s_1,\ldots,s_{t-1}} \mathbb{P}(S_1 = s_1, S_t = s|W)$. The algorithm mainly has three steps as follows.

1. (Initialization) $f_1(i) = p_1 p_{ws}(w = w1|s = state(i))$ for $1 \leq i \leq 12$.
2. (Induction step) $f_t(i) = \max_{1 \leq i \leq 12}[f_{t-1}(i) \times trans(i, j) \times p_{ws}(w = w_t|s = state(j))]$ for $1 \leq i \leq 12$ and $2 \leq t \leq T$, where T is the length of the sentence.
3. (Recover the tags) Define $g_T = \max_i^{-1} f_T(i)$ and let $S_T = state(g_T)$. Then found the rest of the optimal tags recursively using $g_t = \max_i^{-1}[f_t(i) \times trans(i, g_{t+1})]$. Then let $S_t = state(g(t))$.

Viterbi is dynamic programming with simple implementation rules. In this experiment, this algorithm is used. It is not hard to verify that Viterbi is essentially the same as the variable elimination method for hidden Markov chain model.

6 Experimental results

After running the program on the **bc.train.mat**, we get the following performance summary for the three methods.

results	Simple Bayes	Bayes	Sampling
word accuracy	0.345057	0.934037	0.900582
sentence accuracy	0.006864	0.487360	0.524025

It is easy to see that Bayes has the best performance among all.

References

[1] WARREN J. EWENS, GREGORY R. GRANT, *Statistical Methods in Bioinformatics-An Introduction, Springer (2006).*