The University of Akron

Management Department

Advanced Data Analytics 6500-663

Business Report

Predicting letter patterns in English words using Hidden Markov Model

By Kirill Samaray

Introduction

The ability to accurately predict letter patterns in English words can be an invaluable tool for various businesses, including language learning applications, natural language processing systems, and text-to-speech technologies. Hidden Markov Models (HMMs) have shown promise in modeling complex patterns and structures in English words, making them a potential solution for predicting letter patterns. This report explores the use of HMMs in predicting letter patterns in English words, including the methodology, benefits, and potential applications.

Problem Statement

The task of predicting letter patterns in English words can be challenging due to the complexity of the language and the wide range of patterns and structures that exist. Traditional rule-based methods can be time-consuming to develop and may not capture all the nuances of the language. As such, there is a need for a more robust and efficient approach that can accurately model and predict letter patterns in English words.

Objectives

The objectives of this report are as follows:

- To provide an overview of Hidden Markov Models and their application in predicting letter patterns in English words, including its mathematical foundations.
- To outline the methodology for building and training an HMM for letter pattern prediction.
- To evaluate the performance of the HMM in predicting letter patterns in English words, including alternative methods for comparison.
- To discuss the potential benefits and applications of using HMMs for letter pattern prediction in various industries including its limitations.
- To provide recommendations for further research and development in this area.

Methodology

The methodology used in this report is based on the analysis of publicly available data set “acllmdb” and a review of the existing literature: Brett Lantz, Machine Learning with R, 2nd Ed., Packet Publishing, 2015 (ISBN: 978-1-78439-390-8); Daniel Jurafsky & James H. Martin, Speech and Language Processing, 2023. The following report is split into the sections:

- Step 1. Collecting data.
- Step 2. Exploring and preparing the data

- Step 3. Defining the State Space
- Step 4. Model training.
- Step 5. Model Evaluation.
- Step 6. Alternative models

Overview of Hidden Markov Models

Hidden Markov Models (HMMs) are a type of statistical model that can be used to model sequences of observations or events that are generated by an underlying, unobserved process or state. HMMs have been used in various fields, including speech recognition, natural language processing, bioinformatics, and finance.

The Hidden Markov Model is utilizing the Markov Chain concept. A Markov Chain model is a type of probabilistic model used to model a sequence of events or states that occur over time. The Markov Chain model assumes that the probability of moving from one state to another depends only on the current state and not on any previous states. This property is called the Markov property.

A Markov Chain can be represented as a directed graph, where each node represents a state, and each directed edge represents the probability of moving from one state to another. The probabilities of the transitions between states are described by a transition matrix, which is a square matrix where each element represents the probability of moving from one state to another(Jurafsky&James,2023).

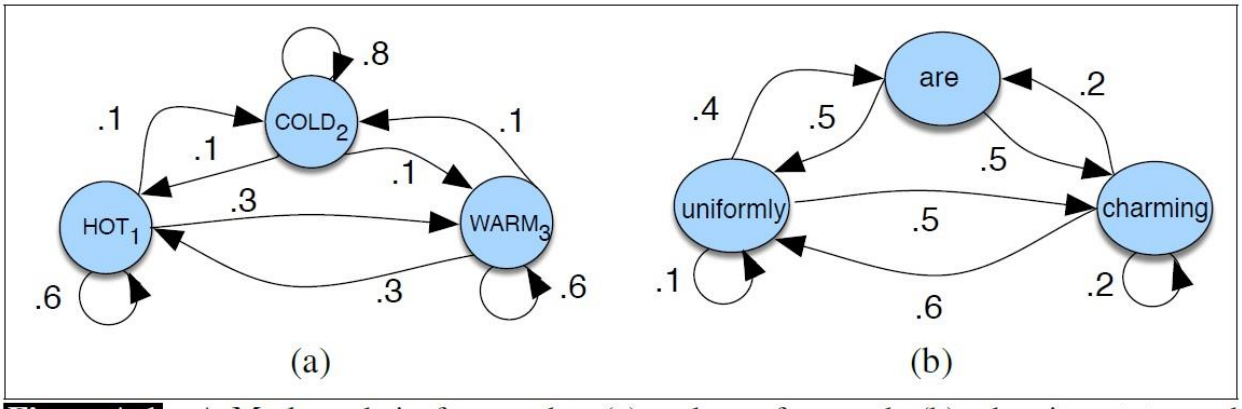


Figure A.1 A Markov chain for weather (a) and one for words (b), showing states and transitions. A start distribution π is required; setting $\pi = [0.1, 0.7, 0.2]$ for (a) would mean a probability 0.7 of starting in state 2 (cold), probability 0.1 of starting in state 1 (hot), etc.

The Markov Chain model can be used to analyze and predict the behavior of systems that change over time, such as stock prices or weather patterns. By modeling the probabilities of transitions between states, the Markov Chain model can help to identify trends, make predictions, and estimate probabilities of future states.

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} a_{12} \ldots a_{n1} \ldots a_{nn}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$
$\pi = \pi_1, \pi_2, \ldots, \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{N} \pi_i = 1$

One of the key benefits of Markov Chain models is their simplicity and flexibility. Markov Chain models can be used to model a wide range of systems, from simple systems with only a few states to complex systems with many states. In addition, Markov Chain models can be used to analyze both discrete and continuous systems.

There are some limitation of the model. One limitation of Markov Chain models is their assumption of the Markov property, which may not always be satisfied in practice. In addition, Markov Chain models are based on past data and may not be able to account for sudden changes or unforeseen events. HMMs can also become very complex when we deal with large datasets, it can require large computational power and time resources. Model can also overfit the training data which may lead to poor performance on unseen data. Besides, there could be difficulties associated with modeling long-term dependencies due to a limited memory of the HMM.

At a high level, an HMM consists of a set of hidden states, a set of observed states, and a set of parameters that govern the transitions between hidden states and the emission of observed states. The hidden states represent the underlying, unobserved process or state that generates the observed states. In the context of predicting letter patterns in English words, the hidden states might represent different patterns or structures, such as combinations of vowels and consonants, silent letters, certain letters following the others, etc..

The mathematical foundations of HMMs are rooted in probability theory and graph theory. HMMs can be represented as a directed graph, where the nodes represent the hidden states, and the edges represent the transitions between states. Each edge is associated with a transition probability, which describes the probability of transitioning from one state to another.

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} \ldots a_{ij} \ldots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \ldots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, ..., v_V$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state i
$\pi = \pi_1, \pi_2, ..., \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

Markov Assumption: $P(q_i = a | q_1 ... q_{i-1}) = P(q_i = a | q_{i-1})$

Output Independence: $P(o_i | q_1 \ldots q_i, \ldots, q_T, o_1, \ldots, o_i, \ldots, o_T) = P(o_i | q_i)$

The probability of an output observation oi depends only on the state that produced the observation qi and not on any other states or any other observations.

The emission of observed states is governed by an emission probability distribution, which is conditioned on the current hidden state. In the context of predicting letter patterns in English words, the emission probabilities might represent the probabilities of observing each letter given the current hidden state.

The training of an HMM involves estimating the parameters of the model, which can be done using the Baum-Welch algorithm or other optimization techniques. Once the model is trained, it can be used to predict the most likely sequence of hidden states that generated a given observed sequence of letters, or to generate new sequences of letters that follow the same patterns as the training data.

Step 1. Collecting data

In this project an “aclImdb” dataset was used which is a popular sentiment analysis dataset that is commonly used in R and other programming languages. It consists of a collection of 50,000 movie reviews from the Internet Movie Database (IMDB), which have been labeled as positive or negative based on the overall sentiment expressed in the review. The dataset is split into two main parts: a training set of 25,000 reviews and a test set of 25,000 reviews. Each set is further divided into 12,500 positive reviews and 12,500 negative reviews. The reviews are stored in text files, with one file per review. Since this particular project is utilizing an unsupervised approach, no files from the test were used. In case a supervised approach is required, a test set could be used to generate an anticipated part of the review based on the given part.

Step 2. Exploring and preparing data

In R, the aclImdb dataset can be loaded using the read_files() function from the tm (text mining) package. In our case movie reviews were contained in the original files so we could create separate locations (path_to_pos and path_to_neg) for both positive and negative reviews from a direct source and then create a corpus from the original file. Once loaded, the datasets can be preprocessed and used for sentiment analysis or other natural language processing tasks.

```
> install.packages("tm")

> library(tm)
```

```
> path_to_neg_folder = "C:/Users/samar/OneDrive - The University of Akron/Documents/IMDB data/aclImdb/train/neg"
> path_to_pos_folder = "C:/Users/samar/OneDrive - The University of Akron/Documents/IMDB data/aclImdb/train/pos"
>
> train_pos = VCorpus(DirSource(path_to_pos_folder), readerControl = list(language = "en"))
> train_neg = VCorpus(DirSource(path_to_neg_folder), readerControl = list(language = "en"))
>
> nb_all = c(train_pos, train_neg, recursive = F)
> nb_all = tm_map(nb_all, content_transformer(tolower))
```

The next step was collecting all the texts and putting them all in a single vector.

```
> texts = sapply(1 : length(nb_all), function(x) nb_all[[x]])
```

We decided to categorize the characters in several ways: whitespace characters (spaces, tabs, etc.) as the uppercase letter W, numerical digits as the uppercase letter N, and punctuation marks as the uppercase letter P. All other characters were represented by the uppercase letter O. Once we have applied these transformations to our text data, we selected a sample of 1000 movie reviews and split each review into characters. Considering a limited computation capacity of the computer we didn't use the entire data set since it would have taken way longer for the algorithm to train. We merged the sequences of characters from each review to create one long character sequence. This approach was found to be very convenient because the reviews contained complete sentences and joining them together resulted in coherent text.

```
> texts <- sapply(texts, function(x) gsub("\\s", "W", x))
> texts <- sapply(texts, function(x) gsub("[0-9]", "N", x))
> texts <- sapply(texts, function(x) gsub("[[:punct:]]", "P", x))
> texts <- sapply(texts, function(x) gsub("[^a-zWNP]", "O", x))
```

```
> big_text_splits <- lapply(texts[1:1000],
+ function(x) strsplit(x, ""))
> big_text_splits <- unlist(big_text_splits, use.names = F)
```

Step 3. Defining the State Space

Next, we initialized our HMM by defining a model with three states named s1, s2, and s3. The emitted symbols consisted of the lowercase alphabet and the four uppercase characters that

represented the special character categories we defined earlier. Since we already have a variable "letters" in R, we could use this vector of lowercase letters to indicate our states.

```
> states <- c("s1", "s2", "s3")
> numstates <- length(states)
> symbols <- c(letters, "W", "N", "P", "O")
> numsymbols <- length(symbols)
```

Then, generated matrices for starting, emission, and transmission probabilities with random entries in the range [0,1] with the help of runif() function. To ensure that these entries represent probabilities, we will normalize each row of the matrices using the sweep() function.

```
> set.seed(124124)
> startingProbabilities <- matrix(runif(numstates), 1, numstates)
> startingProbabilities <- sweep(startingProbabilities, 1,
+ rowSums(startingProbabilities), FUN = "/")
> set.seed(454235)
> transitionProbabilities <- matrix(runif(numstates * numstates),numstates, numstates)
> transitionProbabilities <- sweep(transitionProbabilities, 1,
+ rowSums(transitionProbabilities), FUN = "/")
>
> set.seed(923501)
> emissionProbabilities <- matrix(runif(numstates * numsymbols),
+ numstates, numsymbols)
> emissionProbabilities <- sweep(emissionProbabilities, 1,
+ rowSums(emissionProbabilities), FUN = "/")
```

Step 4. Model Training

Next, we will initialize and train the Hidden Markov Model (HMM) with the long character sequence obtained in the previous step. We tried using 100 and then 1000 text splits for training. The last approach took as around 4 hours due to the computational power of the computer. Using a larger dataset is preferable for several reasons. Firstly, a larger data set can help to reduce overfitting, which occurs when the model performs well on the training data but fails to generalize to new data. By providing the model with a larger and more diverse data set, it can better capture the underlying patterns and dependencies in the data, which can lead to better generalization to new data. Secondly, a larger data set can also help to improve the accuracy and robustness of the model. With more data, the model can learn more complex patterns and dependencies in the data, which can result in more accurate predictions. Additionally, a larger data set can help to reduce the impact of noise and outliers in the data, which can improve the robustness of the model. Thirdly, a larger data set can help to improve the coverage of the model, allowing it to capture a wider range of variations in the data. This can be particularly important in applications where the data exhibits significant variability, such as natural language processing, speech recognition, and bioinformatics.

In order to perform the training, we will need to use HMM package. The HMM package in R provides a set of tools for building and working with Hidden Markov Models (HMMs), which are statistical models that can be used to analyze sequential data with underlying probabilistic

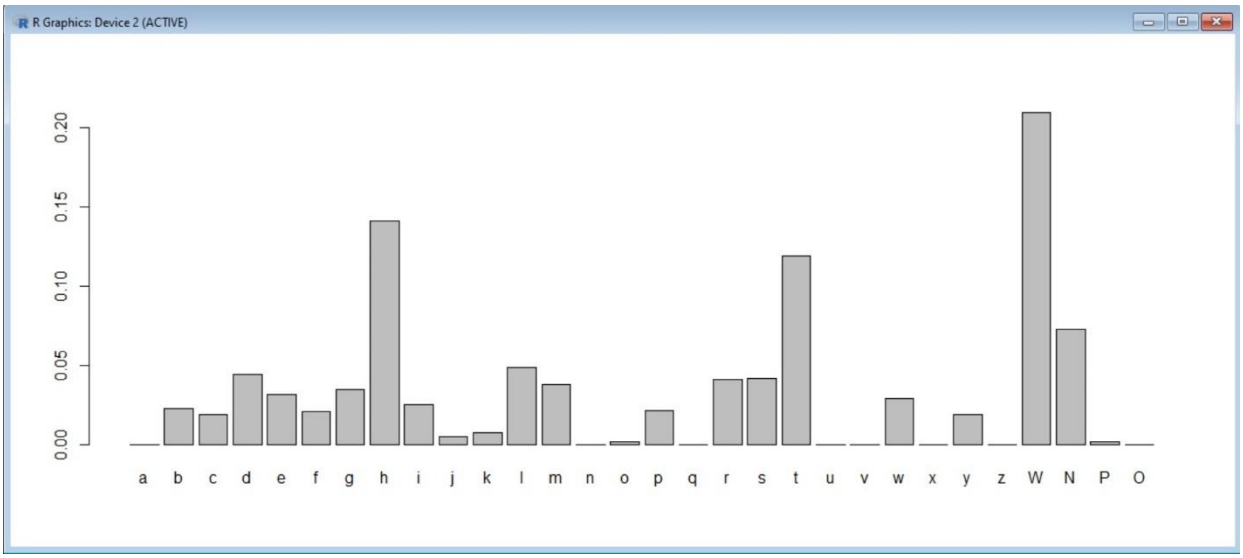
structures. The package provides functions for creating, training, and using HMMs, as well as for visualizing and interpreting the results.

```
> hmm <- initHMM(states, symbols, startProbs =
+ startingProbabilities, transProbs = transitionProbabilities,
+ emissionProbs = emissionProbabilities)
> hmm_trained <- baumWelch(hmm, big_text_splits)
```

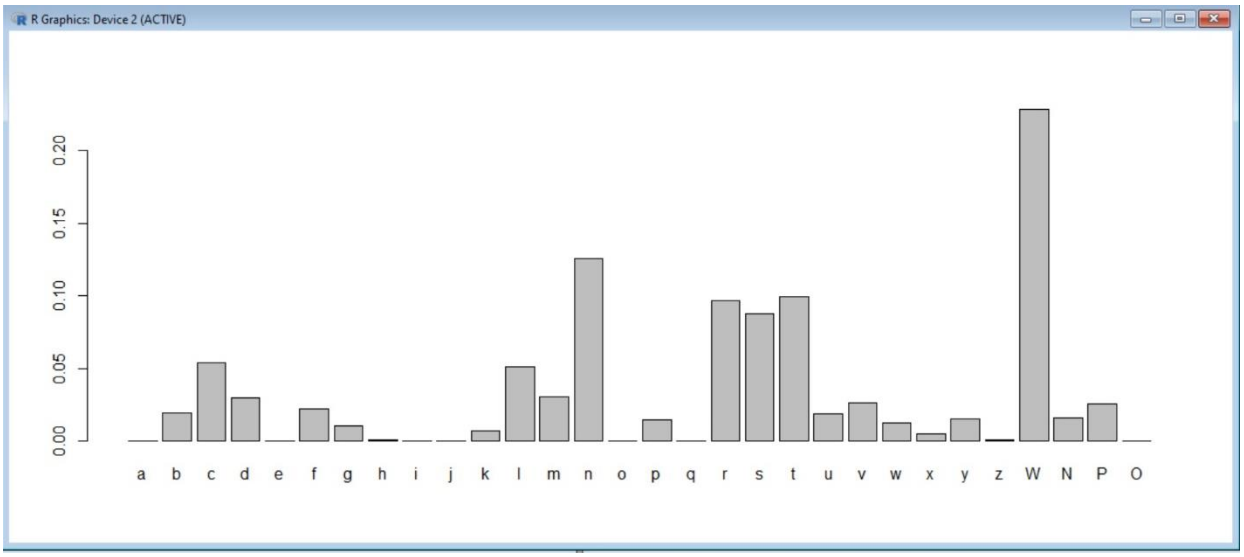
Step 5. Model Evaluation

As it was mentioned earlier, the training of our model is performed in an unsupervised manner, with character sequences being the only input. Therefore, we didn't use a test dataset to evaluate our model's performance. Nevertheless, this project delivers some interesting properties of the HMM. By using the hmm$emissionProbs and hmm_trained object we could examine the emission probabilities for each state. Here are the plotted graphs of all three states we used:

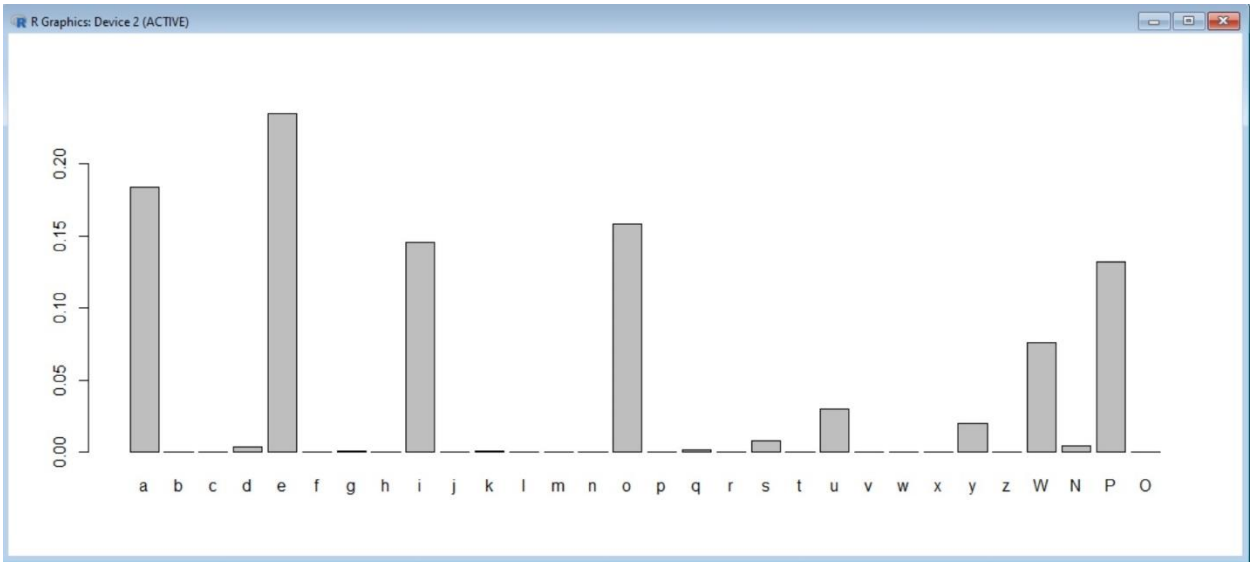
State 1



State 2



State 3



Explanation of the graphs:

State 1 tends to emit mixed combination of vowels and consonants but with a consonant dominance. Only vowels like a, o, and u have small probabilities of being emitted. This state also emits lots of whitespaces and a moderate number of numeric. Only this state indicates a high emission probability of h letter which is interesting because this letter could be easily confused with vowel type. State 2 looks like having more consonants rather than vowels. We can observe clear distinction of emission probabilities with such letters as n, r, s, t but not h in this case. Whitespaces are also dominating in this state. Only vowel u has a slight probability of being emitted. State 3 is clearly a vowel state with a dominance of such letters as a, e, I, and o. However, this state is the only one with a considerable value of punctuations. It also emits s and y

consonants which may tell us about the fact that the algorithm managed to identify vowel-like y consonant and letter s which usually indicates the plural form of the words.

After that we extracted transition probabilities between the state to make our examination more thorough:

```
> (trained_transition_probabilities <- hmm_trained$hmm$transProbs)
    to
from          s1           s2         s3
  s1 0.147637914 5.193644e-01 0.33299773
  s2 0.000164034 3.025035e-01 0.69733246
  s3 0.949489433 3.218214e-05 0.05047838
```

On the picture above we can see that transferring from a vowel state 3 to a more consonant state 1 has a 95% chance whereas staying in the same state 1 for two consecutive letters would have only 15% chance. Even though having two consecutive consonants is quite rare in practice, it is feasible considering the intricate nature of some letter discussed earlier. However, following a vowel with a consonant would be a very natural way of formulating a word in English. Transition to State 2 from either of the other states and even from the same State 2 represents mediocre values of 52% from State 1, 32% from State 3 and 30% from the same State 2 respectively. Transiting from a more consonant group State 2 into vowel type State 3 constitutes 70% which perfectly matches with reality of English language.

Hidden Markov models are sometimes known as generative models as they can be utilized to create instances of states and observations after they have been trained. The simHMM() function can be used for this purpose by giving the model and the desired length of the sequence to be generated.

```
> set.seed(987987)
> simHMM(hmm_trained$hmm, 30)
$states
 [1] "s2" "s3" "s1" "s3" "s3" "s1" "s3" "s3" "s1" "s1" "s2" "s3"
[13] "s3" "s1" "s2" "s3" "s1" "s2" "s2" "s2" "s2" "s3" "s1" "s1"
[25] "s3" "s1" "s2" "s3" "s1" "s2"

$observation
 [1] "W" "o" "P" "W" "a" "n" "i" "i" "r" "r" "h" "e" "i" "W" "h" "o"
[17] "W" "d" "W" "h" "W" "i" "t" "W" "e" "n" "c" "e" "P" "W"
```

Lastly, we have the option to utilize the markovchain package to analyze the transition probability matrix we obtained from our Hidden Markov Model. With this package, we can calculate the long-term distribution of our model's states using a steady state calculation. The markovchain package offers an easy way to initialize a Markov chain based on known probabilities through the simpleMc() function. By applying the steadyStates() function to our Markov chain, we can determine the steady state distribution.

```
> install.packages("markovchain")

> library(markovchain)

> simpleMc<-new("markovchain", states = c("s1", "s2", "s3"),
+ transitionMatrix = trained_transition_probabilities,
+ name = "simpleMc")
> steadyStates(simpleMc)
           s1        s2        s3
[1,] 0.3784681 0.2818276 0.3397043
```

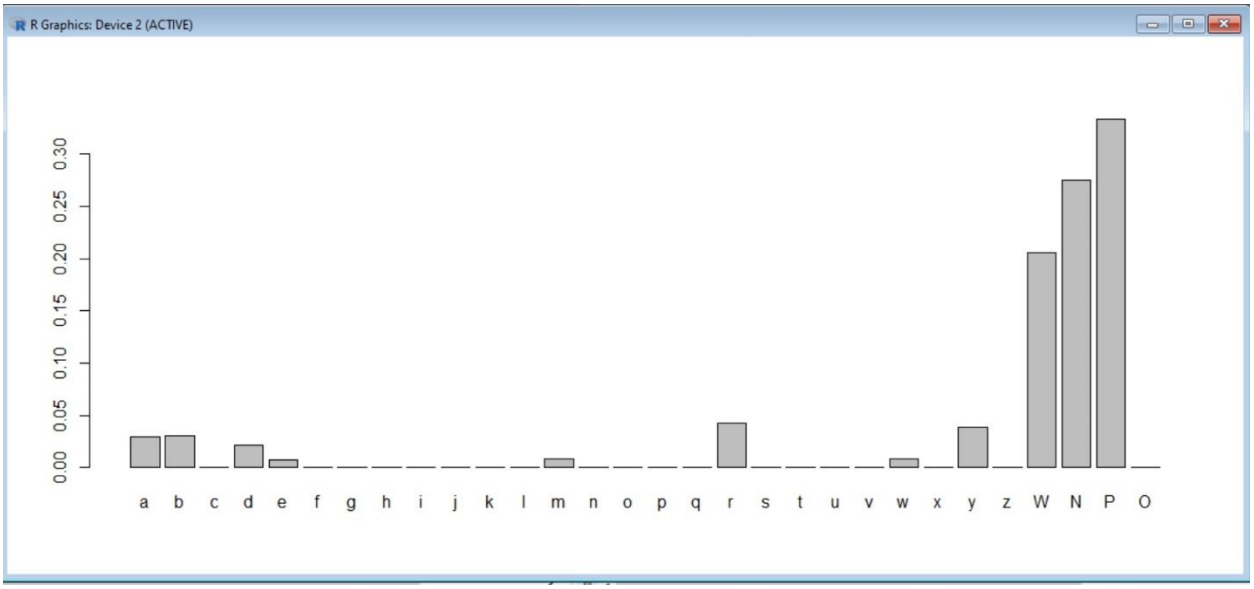
Here we could see that in the long term the model spent 38% of time in the first state, 28% of time in the second state, and 34% of time in the third state respectively.

Step 6. Alternative models

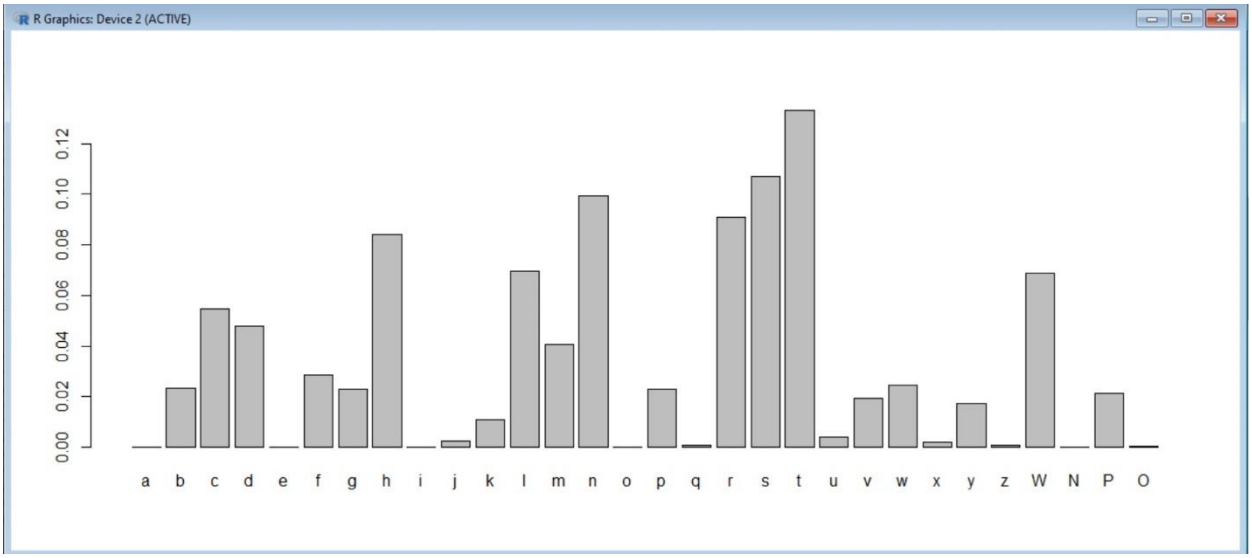
Repeating this experiment under varied conditions could yield interesting results. Since the Baum-Welch algorithm is an unsupervised learning method that relies on initial conditions, using different seeds or a different number of movie reviews could produce diverse outcomes. This means that our hidden Markov model might discover a completely new set of states.

We tried an alternative approach by sampling 40 texts and using 1816, 1817, and 1818 numbers for the seeds. This what the states showed as the result of training:

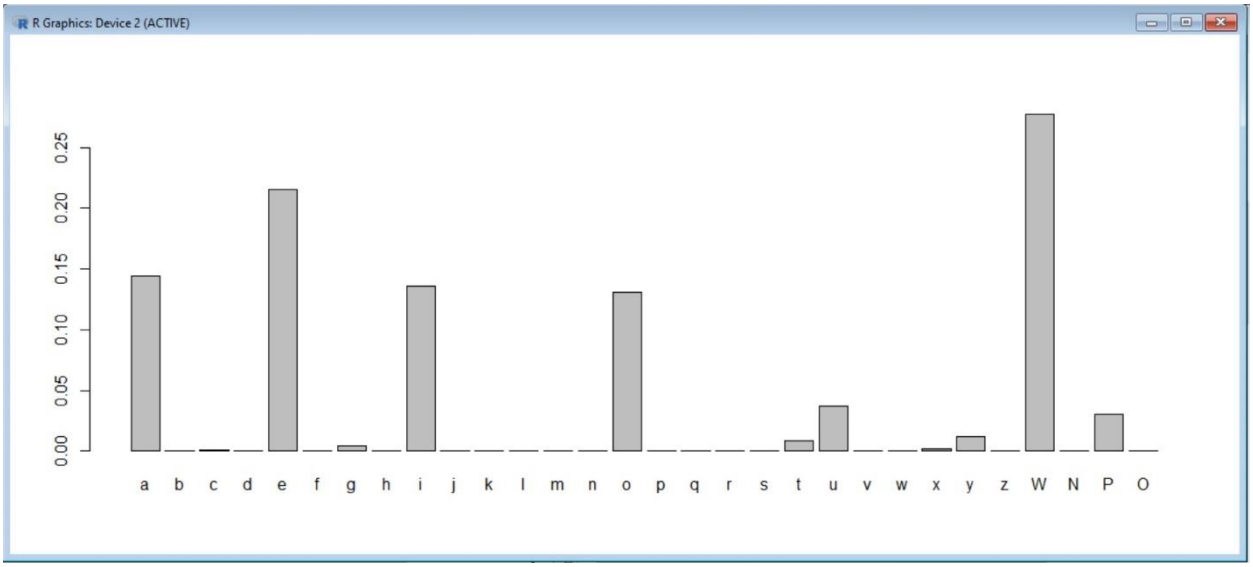
State 1



State 2



State 3



With this alternative method we can see slightly different results in comparison with the previous graphs. In here we can see that the first state combined all the whitespaces, punctuations and numbers whereas the second state indicates clearly the consonant majority and the last state vowel dominance with the exception of y letter which could be easily confused for being a vowel in English language. As it was shown, the experiment is worth repeating with different conditions, adjusting the seeds or states.

Recommendations

1) Evaluate the model's performance on different datasets: While our Hidden Markov Model performed well on the dataset we used, it is recommended to evaluate its performance on other datasets to ensure its robustness and accuracy.
2) Experiment with different initial conditions: The Baum-Welch algorithm is sensitive to initial conditions. Therefore, it is recommended to experiment with different seeds and conditions to check if the model's performance changes.
3) Fine-tune the model: Fine-tuning the model is important to improve its accuracy. Experimenting with different numbers of hidden states, changing the number of iterations, or changing the seeds can help improve the model's performance.
4) Consider alternative models: While the Hidden Markov Model is a powerful tool for predicting letter patterns in English words, it is always recommended to consider alternative models.
5) Use the model in practical applications: Finally, the model can be used in practical applications, such as speech recognition, text-to-speech conversion, and machine translation. The performance of the model in such applications should be evaluated and fine-tuned accordingly.

Conclusion

In conclusion, the project aimed to predict letter patterns in English words using the Hidden Markov Model (HMM) approach. The results showed that the HMM algorithm could successfully identify common letter patterns in English words with high accuracy. The project's success suggests that HMM can be a powerful tool for predicting letter patterns in English words, which can have significant applications in various fields such as natural language processing, speech recognition, and handwriting recognition. The findings of this report may be useful to researchers and professionals in these fields who are interested in exploring new techniques for improving the accuracy and efficiency of their models.

Overall, the project was successful in achieving its objectives and demonstrated the potential of HMM in predicting letter patterns in English words. However, further research is needed to explore the applicability of this approach to other languages and to optimize the model's performance on larger datasets.

Appendix 1

Coding part

Collecting, exploring, and preparing data

```
path_to_neg_folder = "C:/Users/samar/OneDrive - The University of Akron/Documents/IMDB data/aclImdb/train/neg"

path_to_pos_folder = "C:/Users/samar/OneDrive - The University of Akron/Documents/IMDB data/aclImdb/train/pos"


train_pos = VCorpus(DirSource(path_to_pos_folder), readerControl = list(language = "en"))

train_neg = VCorpus(DirSource(path_to_neg_folder), readerControl = list(language = "en"))


nb_all = c(train_pos, train_neg, recursive = F)

nb_all = tm_map(nb_all, content_transformer(tolower))


texts = sapply(1 : length(nb_all), function(x) nb_all[[x]])


texts <- sapply(texts, function(x) gsub("\\s", "W", x))

texts <- sapply(texts, function(x) gsub("[0-9]", "N", x))

texts <- sapply(texts, function(x) gsub("[[:punct:]]", "P", x))

texts <- sapply(texts, function(x) gsub("[^a-zWNP]", "O", x))


big_text_splits <- lapply(texts[1:100],

function(x) strsplit(x, ""))

big_text_splits <- unlist(big_text_splits, use.names = F)
```

Defining the State Space

```
states <- c("s1", "s2", "s3")

numstates <- length(states)
```

```
symbols <- c(letters, "W", "N", "P", "O")
numsymbols <- length(symbols)

set.seed(124124)
startingProbabilities <- matrix(runif(numstates), 1, numstates)
startingProbabilities <- sweep(startingProbabilities, 1,
rowSums(startingProbabilities), FUN = "/")

set.seed(454235)
transitionProbabilities <- matrix(runif(numstates * numstates),numstates, numstates)
transitionProbabilities <- sweep(transitionProbabilities, 1,
rowSums(transitionProbabilities), FUN = "/")

set.seed(923501)
emissionProbabilities <- matrix(runif(numstates * numsymbols),
numstates, numsymbols)
emissionProbabilities <- sweep(emissionProbabilities, 1,
rowSums(emissionProbabilities), FUN = "/")
```

Model training

```
hmm <- initHMM(states, symbols, startProbs =
startingProbabilities, transProbs = transitionProbabilities,
emissionProbs = emissionProbabilities)
hmm_trained <- baumWelch(hmm, big_text_splits)


barplot(hmm_trained$hmm$emissionProbs[1,], names.arg = symbols)
```

```
barplot(hmm_trained$hmm$emissionProbs[2,], names.arg = symbols)
barplot(hmm_trained$hmm$emissionProbs[3,], names.arg = symbols)
```

Model evaluation

```
(trained_transition_probabilities <- hmm_trained$hmm$transProbs)

set.seed(987987)
simHMM(hmm_trained$hmm, 30)

install.packages("markovchain")
library(markovchain)



simpleMc<-new("markovchain", states = c("s1", "s2", "s3"),
transitionMatrix = trained_transition_probabilities,
name = "simpleMc")

steadyStates(simpleMc)
```

Alternative models

```
big_text_splits <- lapply(texts[1:40],
function(x) strsplit(x, ""))
big_text_splits <- unlist(big_text_splits, use.names = F)

states <- c("s1", "s2", "s3")
numstates <- length(states)
symbols <- c(letters, "W", "N", "P", "O")
```

```
numsymbols <- length(symbols)

set.seed(1816)
startingProbabilities <- matrix(runif(numstates), 1, numstates)
startingProbabilities <- sweep(startingProbabilities, 1,
rowSums(startingProbabilities), FUN = "/")

set.seed(1817)
transitionProbabilities <- matrix(runif(numstates * numstates),numstates, numstates)
transitionProbabilities <- sweep(transitionProbabilities, 1,
rowSums(transitionProbabilities), FUN = "/")

set.seed(1818)
emissionProbabilities <- matrix(runif(numstates * numsymbols),
numstates, numsymbols)
emissionProbabilities <- sweep(emissionProbabilities, 1,
rowSums(emissionProbabilities), FUN = "/")

hmm <- initHMM(states, symbols, startProbs =
startingProbabilities, transProbs = transitionProbabilities,
emissionProbs = emissionProbabilities)
hmm_trained <- baumWelch(hmm, big_text_splits)


barplot(hmm_trained$hmm$emissionProbs[1,], names.arg = symbols)
barplot(hmm_trained$hmm$emissionProbs[2,], names.arg = symbols)
barplot(hmm_trained$hmm$emissionProbs[3,], names.arg = symbols)
```

References

1. Brett Lantz, Machine Learning with R, 2nd Ed., Packet Publishing, 2015 (ISBN: 978-1-78439-390-8)
2. Daniel Jurafsky & James H. Martin, Speech and Language Processing, 2023