***Project 3-Hidden Markov Model***

**Aim**: In modern world, understanding biological features of promoter’s gene remains one of the important and challenging problem within biology Science. Promoters can be modelled by identifying identical features with the help of network of connections associated. Generally, a promoter is a region of DNA in which transcription of a gene by RNA polymerase begins its origin. Promoter sequences helps us to describe the direction of transcription and point out which DNA strand will be transcribed. Let us create a predictive model to recognize promoter gene sequences from non-promoter gene sequences. These are promoter sequences with a sequence of nucleotides namely Thymine, Cytosine, Adenine and Guanine. These sequences are important in genetic process known as Gene transcription. This is the first step towards decrypting DNA information.

**Algorithm selected**: Hidden Markov Model machine learning algorithm.

Hidden Markov models can be termed as a formal foundation model for predicting probabilities sequence 'labeling' problems in general. This model provides a conceptual tool for building complex models just by drawing an intuitive picture. HMM is considered as a specific form of Bayesian networks, which are based on the probability theory of Bayes. [HMMs](https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/hidden-markov-model) are also statistical models to capture hidden information from observable sequential symbols (e.g., a nucleotide sequence in our case). HMM models are used in wide range of programs like gene -finding, profile match searches, multiple sequence alignment and regulatory site identification. In a [HMM](https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/hidden-markov-model), the system being modelled is assumed to be a Markov process with unknown parameters, and the main aim is to determine the hidden parameters from the observable parameters. A good HMM model hold accurately models the real-world source of the observed real data and can simulate the source by training the data set.

A picture containing clock

Description automatically generated

Background of HMM model

p = { p1, p2, ..., pn } is a sequence of states. Each pi takes a value from set Q. We do not observe p.

x = { x1, x2, ..., xn } is a sequence of emissions. Each xi takes a value from set ∑. We do observe x.

x2 is conditionally independent of everything else given p2.

p4 is conditionally independent of everything else given p3

Let’s dive in into the steps now

1. **DATA COLLECTION**: Data used in this project is promoter gene sequences from

https://archive.ics.uci.edu/ml/datasets/Molecular+ Biology+(Promoter+Gene+Sequences)

This data set has 106 instances and 3 features indicating as

V1-classified either as positive or negative sequence, indicating promoter and non-promoters

V2-identifier of the sequence,

V3-order of nucleotide sequences.

1. **DATA PREPARATION**:

We will start importing the csv data in R, and striping off white spaces to avoid spacings for this

* promoters=read.csv("promoters.data", header = F, dec = ",",strip.white = TRUE, stringsAsFactors = FALSE)

#To see data structure

* head(promoters)

*V1 V2 V3*

*1 + S10 tactagcaatacgcttgcgttcggtggttaagtatgtataatgcgcgggcttgtcgt*

*2 + AMPC tgctatcctgacagttgtcacgctgattggtgtcgttacaatctaacgcatcgccaa*

*3 + AROH gtactagagaactagtgcattagcttatttttttgttatcatgctaaccacccggcg*

*4 + DEOP2 aattgtgatgtgtatcgaagtgtgttgcggagtagatgttagaatactaacaaactc*

*5 + LEU1\_TRNA tcgataattaactattgacgaaaagctgaaaaccactagaatgcgcctccgtggtag*

*6 + MALEFG aggggcaaggaggatggaaagaggttgccgtataaagaaactagagtccgtttaggt*

This is the structure of the data. The V1 feature is categorial variable indicating promoters and non-promoters. V2 feature is an identifier, V3 feature is sequence of nucleotides.

1. **Data Cleaning:** First dividing the promoter sequences to positive and negative observations using feature V1.

* positive\_observations=subset(promoters, V1 == '+', 3)
* negative\_observations=subset(promoters, V1 == '-', 3)

Adding S and X in starting and ending of sequences in feature V3 to indicate start and end of sequence observations.

* positive\_observations=sapply(positive\_observations,function(x) paste("S", x, "X", sep=""))
* negative\_observations=sapply(negative\_observations,function(x) paste("S", x, "X", sep=""))
* positive\_observations

*V3*

*[1,] "StactagcaatacgcttgcgttcggtggttaagtatgtataatgcgcgggcttgtcgtX"*

*[2,] "StgctatcctgacagttgtcacgctgattggtgtcgttacaatctaacgcatcgccaaX"*

*[3,] "SgtactagagaactagtgcattagcttatttttttgttatcatgctaaccacccggcgX"*

*[4,] "SaattgtgatgtgtatcgaagtgtgttgcggagtagatgttagaatactaacaaactcX"*

*[5,] "StcgataattaactattgacgaaaagctgaaaaccactagaatgcgcctccgtggtagX"*

Splitting the sequence into single characters helps the model to identify nucleotide sequences. For this we will use strsplit() function

* positive\_observations=strsplit(positive\_observations, "")
* negative\_observations=strsplit(negative\_observations, "")
* head(positive\_observations[[1]], n = 15)

*[1] "S" "t" "a" "c" "t" "a" "g" "c" "a" "a" "t" "a" "c" "g" "c"*

**4.Data Preparation:** For training HMM, we must declare starting, emission and transition probabilities of states. Here there are 4 common emission symbols in all the cases, along with that S and X are added to symbols to make starting probability of all other states as 0.

* states=c("S", "X", "a", "c", "g", "t")
* symbols=c("S", "X", "a", "c", "g", "t")
* startingProbabilities= c(1,0,0,0,0,0)
* emissionProbabilities=diag(6)
* colnames(emissionProbabilities)=states
* rownames(emissionProbabilities)=symbols
* emissionProbabilities

*S X a c g t*

*S 1 0 0 0 0 0*

*X 0 1 0 0 0 0*

*a 0 0 1 0 0 0*

*c 0 0 0 1 0 0*

*g 0 0 0 0 1 0*

*t 0 0 0 0 0 1*

For transition matrix, we are going to create a function that adds up counts of state transitions. Input of the function will be states along with the data that contains concatenated vector sequences. Normalizing the matrix by dividing with row sum using sweep function ()

* calculateTransitionProbabilities=function(data, states) {
* transitionProbabilities=matrix(0, length(states), length(states))
* colnames(transitionProbabilities)=states
* rownames(transitionProbabilities)=states
* for(index in 1:(length(data) - 1)) {
* current\_state=data[index]
* next\_state=data[index + 1]
* transitionProbabilities[current\_state, next\_state]=transitionProbabilities[current\_state, next\_state] + 1

}

* transitionProbabilities=sweep(transitionProbabilities,1,rowSums(transitionProbabilities), FUN = "/")
* return(transitionProbabilities)
* }

There are only 53 observations of positive and negative observations each so dividing into test and training data set doesn’t yield better results. So, following a leave one out method, first we will be creating a transition probability matrix out of the negative observations and check the positive observation using cross validation technique.

* negative\_observation=Reduce(function(x, y) c(x, y),negative\_observations,c())

(transitionProbabilitiesNeg=calculateTransitionProbabilities(negative\_observation, states))

*S X a c g t*

*S 0 0.00000000 0.2264151 0.2830189 0.1320755 0.3584906*

*X 1 0.00000000 0.0000000 0.0000000 0.0000000 0.0000000*

*a 0 0.02168022 0.2113821 0.2696477 0.2506775 0.2466125*

*c 0 0.01256983 0.2500000 0.1634078 0.2667598 0.3072626*

*g 0 0.01958225 0.3133159 0.2480418 0.1919060 0.2271540*

*t 0 0.01622971 0.1885144 0.2434457 0.2946317 0.2571785*

In the start state (S), we can randomly move to a nucleotide state, but have zero probability of moving to the stop state (X) or staying in the start state. When in the nucleotide states, we can randomly transition to any state except back to the start state. Only valid transition from a stop state to the start state is for a new sequence.

**5.TRAINING DATA:** For this we are using initHMM () function in HMM library in R and giving all the variables which, we created earlier for negative hmm

* library("HMM")
* negative\_hmm=initHMM(states,symbols,startProbs=startingProbabilities,transProbs=transitionProbabilitiesNeg, emissionProbs=emissionProbabilities)

Next building the positive HMM, we follow the same leave one out method. This test observation will then be processed by the negative HMM model we trained in the previous step. If the positive HMM predicts a higher probability for the test observation than the negative HMM, our model will correctly classify the test observation. The following block of code performs a loop of these calculations for every positive observation.

We initiate a variable incorrect to store wrongly classified count. We train the positive list without the observation left out in each instance and test it.

* incorrect=0
* for(obs in 1:length(positive\_observations)) {
* positive\_observation=Reduce(function(x, y) c(x, y),positive\_observations[-obs],c())
* transitionProbabilitiesPos=calculateTransitionProbabilities(positive\_observation, states)
* positive\_hmm =initHMM(states,symbols,startProbs = startingProbabilities,transProbs = transitionProbabilitiesPos, emissionProbs = emissionProbabilities)
* test\_observation=positive\_observations[[obs]]
* final\_index=length(test\_observation)

* pos\_probs=exp(forward(positive\_hmm, test\_observation))
* neg\_probs=exp(forward(negative\_hmm, test\_observation))
* pos\_seq\_prob=sum(pos\_probs[, final\_index])
* neg\_seq\_prob=sum(neg\_probs[, final\_index])
* if(pos\_seq\_prob<neg\_seq\_prob)incorrect=incorrect + 1
* }
* incorrect

*[1] 13*

This means out of 53 positive observations;13 was misclassified and 40 were correctly done.

Now, doing the same loop with negative observations. Training negative observations leaving one out. We will then process this observation with both the positive HMM we just trained and the negative HMM trained without this observation in its training data

* positive\_observation=Reduce(function(x, y) c(x, y),positive\_observations,c())
* transitionProbabilitiesPos=calculateTransitionProbabilities(positive\_observation, states)
* positive\_hmm=initHMM(states,symbols,startProbs=startingProbabilities, transProbs = transitionProbabilitiesPos,
* emissionProbs = emissionProbabilities)
* for (obs in 1:length(negative\_observations)) {
* negative\_observation=Reduce(function(x, y) c(x, y),negative\_observations[-obs], c())
* transitionProbabilitiesNeg=calculateTransitionProbabilities(negative\_observation, states)
* negative\_hmm=initHMM(states, symbols, startProbs = startingProbabilities,
* transProbs = transitionProbabilitiesNeg,
* emissionProbs = emissionProbabilities)
* test\_observation=negative\_observations[[obs]]
* final\_index=length(test\_observation)
* pos\_probs=exp(forward(positive\_hmm,test\_observation))
* neg\_probs=exp(forward(negative\_hmm,test\_observation))
* pos\_seq\_prob=sum(pos\_probs[, final\_index])
* neg\_seq\_prob=sum(neg\_probs[, final\_index])
* if (pos\_seq\_prob > neg\_seq\_prob) incorrect=incorrect+1
* }
* incorrect

*[1] 25*

Overall number of misclassifications are 25

**6.EVALUATING THE MODEL:** Validation accuracy is calculated as number of misclassified observations by total number of observations.

* cross\_validation\_accuracy= 1 - (incorrect/nrow(promoters))

*[1] 0.7641509*

This gives 76% accuracy for leave one out approach,

As the data set is too small to do improvements as this estimate may have high variance

**Conclusion:**

We trained the model using leave one out method where we train the model without one observation and use left out one for validating. We used forward algorithm in HMM package and modeled positive and negative observations of promoter sequences and were able to achieve cross validation accuracy of 76%. Overall, 25 observations are wrongly classified.

**Limitations of HMM:**

* The HMM needs to be trained on a set of sequences and generally requires a larger dataset than the simple Markov models. The training involves repeated iterations of the algorithm like forward and viterbi which can be quite slow.
* Markov models follow first order Markov assumption, which states condition on present past and future are independent this is rather restrictive and might not be correct in real-world.