马尔可夫链

隐马尔可夫模型

定义

隐马尔可夫模型（Hidden Markov Model；缩写：HMM）或称作隐性马尔可夫模型，是统计模型，它用来描述一个含有隐含未知参数的马尔可夫过程。它是马尔可夫链的一种。其难点是从可观察的参数中确定该过程的隐含参数。然后利用这些参数来作进一步的分析，例如模式识别。

在正常的马尔可夫模型中，状态对于观察者来说是直接可见的。这样状态的转换概率便是全部的参数。而在隐马尔可夫模型中，状态并不是直接可见的，但受状态影响的某些变量则是可见的。每一个状态在可能输出的符号上都有一概率分布。因此输出符号的序列能够透露出状态序列的一些信息。

三要素:隐马尔可夫模型是一个参数 λ (A, B, π)

- A 状态转移矩阵 $A = [\ a_{i,j}\ ]_{N,N} \quad i, j \in [1, 2, 3, 4, ..., N]$
  $a_{i,j} = P(\ i_{t+1} = q_i \mid i_t = q_i\ ), i = 1, 2, 3...N,\ j = 1, 2, 3...N$ 在 t 时刻处于 q_i 的条件下，在 $t+1$ 时刻转移到 qj 的概率
- B 观测概率矩阵 $B = [\ b_j(k)]_{N,M} \quad j \in [1, 2, 3, ..., N] \quad k \in [1, 2, 3, ..., M]$
  $b_j(k) = P(\ o_t = v_k \mid i_t = q_j\ )$ 在 t 处于状态 q_j 的条件下生成观测 v_k 的概率
- π $\pi = P(i_i = q_i) \quad i = 1, 2, ..., N$ 时刻 $t = 1$ 时处于状态 q_i 概率

两个基本假设

- 齐次马尔可夫假设，隐马尔可夫链 t 的状态只与 $t-1$ 时刻有关。
- 观测独立性假设，观测只与当前状态有关。

基于隐马尔可夫模型的文本生成器

token : 文本中的单词或者单词和符号所组成的标志

> this is my wife.
> 上述句子中的 token 序列依次为 'this' 'is' 'my' 'wife.' (即为对一行String的split空格得到的结果)

首先，对于文本中每个token，我们约定它只由前一个token决定，且我们观察到的token只由当前状态有关

pair

基于这个假设我们把相邻两个$token$的称为一个$pair =< first, second >=< token_i, token_j >$
实际上pair是一个序偶

```
/**
 * ConditionPair唯一的依据是first和second元素相同，不依赖于frequency
 * 下面的代码中hashcode和equals方法是为了HashMap判断两个ConditionPair是否相同的依据
*/
class ConditionPair {
    String first;
    String second;
    Double frequency;

    ConditionPair(String first, String second) {
        this.first = first;
        this.second = second;
        frequency = 1.0;
    }

    @Override
    public int hashCode() {
        return first.length() + second.length();
    }

    @Override
    public boolean equals(Object obj) {
        ConditionPair compare = (ConditionPair) obj;
        return compare.first.equals(this.first) & compare.second.equals(this.second);
    }

    @Override
    public String toString() {
        return first+":"+second+" f: "+ frequency;
    }
}
```

在把$pair$包含的条件概率算出来：

$$P(token_i, token_j) = (token_t = token_i | token_{t-1} = token_j)$$

计算的细节是

- 收集到有共同 $first$ 的 $pair$
- 根据统计出每个 $pair$ 出现的次数，算出上述概率。

```
/**
 * 数据结构
 * frequencyDistribution 用于储存每个token出现次数
 * conditionalDistribution 储存相同first的Pair集合
 * ConditionPair 就是文中Pair，不过多加了一个属性出现次数
*/
HashMap<String,Double> frequencyDistribution = null;
HashMap<String, HashSet<ConditionPair>> conditionalDistribution = null;
Double tokenCounts = 0.0;
```

> as long as you love me
> 其中含有 first 为 as 的 pair有<as,long> 和<as,you>
> 他们分别都出现了一次，也就是每个均是50%的概率。
> 换句话说as 后有两种可能，0.5的概率为long，0.5的概率为you

```
public void read(){
    StringBuffer line = new StringBuffer();
    String []tokens = null;
    String token =null;
    String nextToken = null;
    Double frequency = null;
    ConditionPair conditionPair = null;
    HashSet<ConditionPair> conditionPairsSet = null;
    while(in.hasNext()){
        line.append(in.nextLine()+" ");
    }
    tokens = line.toString().split(" ");
    tokenCounts = (double) tokens.length;
    for (int i = 0; i < tokens.length; i++) {
        token = tokens[i];
        if((frequency = frequencyDistribution.get(token))==null){
            frequencyDistribution.put(token,1.0);
        }else{
            frequencyDistribution.put(token,frequency+1.0);
        }
        if(i!=tokens.length-1){
            nextToken = tokens[i+1];
            conditionPair = new ConditionPair(token,nextToken);
            if((conditionPairsSet=conditionalDistribution.get(token))==null){
                conditionPairsSet = new HashSet<ConditionPair>();
                conditionPairsSet.add(conditionPair);
                conditionalDistribution.put(token,conditionPairsSet);
            }else if(conditionPairsSet.contains(conditionPair)){
                Iterator<ConditionPair> iterator = conditionPairsSet.iterator();
                while (iterator.hasNext()) {
                    ConditionPair next = iterator.next();
                    if(next.equals(conditionPair)){
                        next.frequency++;
                        break;
                    }
                }
            }else{
                conditionPairsSet.add(conditionPair);
            }
        }
    }
      System.out.println(frequencyDistribution);
      System.out.println(conditionalDistribution);
}
```

怎么得到第一个token?

第一个 $token$ 可以根据每个分布概率随机生成，或者指定，或者其他方式都可以，因为为们关注的是随机过程，只要保证不是固定 $token$ 就能得到相对理想的效果

```
/**
 * clac方法中对每个token的出现次数进行了归一化处理，为了方便操作。
 * 其实可以用rand与总token数(注意token的词频，如果为a a b总token数是3而不是2)的乘积
 * 与概率分布表比较
*/
public void calc(){
    HashMap<String,Double> temp = new HashMap<String, Double>();
    Set<Map.Entry<String, Double>> entries = frequencyDistribution.entrySet();
    for (Iterator<Map.Entry<String, Double>> iterator = entries.iterator(); iterator.ha
        Map.Entry<String, Double> entry = iterator.next();
        temp.put(entry.getKey(),entry.getValue()/tokenCounts);
    }
    frequencyDistribution = temp;
/**
 * P(token i): value(i-1)<value<=value(i)
 * */
public String getStart(){
    ArrayList<Map.Entry<String,Double>> list = new ArrayList(frequencyDistribution.entr
    double rand = Math.random(); // get a value [0,1)
    for (int i = 0; i < list.size(); i++) {
        rand -= list.get(i).getValue();
        if(rand<=0)
           return list.get(i).getKey();
    }
    return null;
}
```

怎么得到下一个token?

答案是根据条件概率概率随机生成，马尔可夫就是条件概率随机过程

```
public String getNext(String now){
    HashSet<ConditionPair> conditionPairs = conditionalDistribution.get(now);
    if(conditionPairs==null) return null;
    List<ConditionPair> condtionalPairs = new ArrayList(conditionPairs);
    double rand = Math.random();
    int all = 0;
    for (Iterator<ConditionPair> iterator = condtionalPairs.iterator(); iterator.hasNex
        ConditionPair conditionPair = iterator.next();
        all+=conditionPair.frequency;
    }
    rand = rand * all;
    for (int i = 0; i < condtionalPairs.size(); i++) {
        rand -= condtionalPairs.get(i).frequency;
        if(rand<0) return condtionalPairs.get(i).second;
    }
    return null;
}
```

某个 token 没有对应的 piar?

如果发现某个 $token$ 没有以它为 $first$ 的 $pair$ 我采用的策略是重新得到第一个$token$

生成代码

```
//当String 为null的时候即为根据概率生成，不指定第一个token
public void create(int cnt,String start){
    String step = start;
    while(cnt-- >0){
        if(step == null) step =getStart();
        else step = getNext(step);
        if(step==null)
            cnt++;
        else
            System.out.println(step+" ");
        if(cnt%10==0)
            System.out.println();
    }
}
```

主函数代码

```
public static void main(String args[]){
    Markov markov = new Markov("Economist.txt");
    markov.create(200);
    System.out.println();
    markov.create(200,"African");
}
```

效果展示

> travelers this year, according to more liberal visa policies.
> The Africa Union Commission, notes that opened up its borders
> to the continent, a landmark chapter in the continent, a
> regulatory environment that opened up its visa-on-arrival policy for Southern
> African Continental Free Trade Area and the continent, a landmark
> chapter in the continent, a new report says 2018 is
> ranked third on the latest Africa Union Commission, notes that
> there has said. The two frameworks are major milestones
> toward the growing momentum for Southern African Development Community (SADC)
> members.Despite the introduction of people continent-wide to have made the
> aspect of Addis Ababa by the table, Africans will need
> visas to the countries in opening up their score, it
> faster, less expensive and makes it adds. Rwanda, one
> of a regulatory environment that opened up their score, it
> adds. Forty-three countries in Africa’s most progress in Africa’s
> most visa-open countries continue to travel to travel to travel
> within Africa, and tangible trade and makes it adds.
> The two frameworks are put on the countries continue to

> the countries that promotes air connectivity and tangible trade and easier for Africans to travel to more liberal visa openness

完整代码

```
import java.io.*;
import java.util.*;

public class Markov {
    // IO Stream
    private File source =  null;
    private Scanner in = null;
    //Data Structure
    private HashMap<String,Double> frequencyDistribution = null;
    private HashMap<String, HashSet<ConditionPair>> conditionalDistribution = null;
    private Double tokenCounts = 0.0;

    Markov(String path){
        source = new File(path);
        frequencyDistribution   = new HashMap<String,Double>();
        conditionalDistribution = new HashMap<String,HashSet<ConditionPair>>();
        try {
            in = new Scanner(source);
        } catch (FileNotFoundException e) {
            System.out.println("File not found,plz check again.");
            System.exit(1);
        }
        read();
        calc();
    }

    private void calc(){
        HashMap<String,Double> temp = new HashMap<String, Double>();
        Set<Map.Entry<String, Double>> entries = frequencyDistribution.entrySet();
        for (Iterator<Map.Entry<String, Double>> iterator = entries.iterator(); iteratc
            Map.Entry<String, Double> entry = iterator.next();
            temp.put(entry.getKey(),entry.getValue()/tokenCounts);
        }
        frequencyDistribution = temp;
    }

    private void read(){
        StringBuffer line = new StringBuffer();
        String []tokens = null;
        String token =null;
        String nextToken = null;
        Double frequency = null;
        ConditionPair conditionPair = null;
        HashSet<ConditionPair> conditionPairsSet = null;
        while(in.hasNext()){
            line.append(in.nextLine()+" ");
        }
        tokens = line.toString().split(" ");
        tokenCounts = (double) tokens.length;
        for (int i = 0; i < tokens.length; i++) {
            token = tokens[i];
            if((frequency = frequencyDistribution.get(token))==null){
                frequencyDistribution.put(token,1.0);
            }else{
```

```
                frequencyDistribution.put(token,frequency+1.0);
            }
            if(i!=tokens.length-1){
                nextToken = tokens[i+1];
                conditionPair = new ConditionPair(token,nextToken);
                if((conditionPairsSet=conditionalDistribution.get(token))==null){
                    conditionPairsSet = new HashSet<ConditionPair>();
                    conditionPairsSet.add(conditionPair);
                    conditionalDistribution.put(token,conditionPairsSet);
                }else if(conditionPairsSet.contains(conditionPair)){
                    Iterator<ConditionPair> iterator = conditionPairsSet.iterator();
                    while (iterator.hasNext()) {
                        ConditionPair next = iterator.next();
                        if(next.equals(conditionPair)){
                            next.frequency++;
                            break;
                        }
                    }

                }else{
                    conditionPairsSet.add(conditionPair);
                }
            }
        }
//        System.out.println(frequencyDistribution);
//        System.out.println(conditionalDistribution);
    }

    public void create(int cnt,String start){
        String step = start;
        while(cnt-- >0){
            if(step == null) step =getStart();
            else step = getNext(step);
            if(step==null)
                cnt++;
            else
                System.out.print(step+" ");
            if(cnt%10==0)
                System.out.println();
        }
    }

    public void create(int cnt){
        String step = null;
        while(cnt-- >0){
            if(step == null) step =getStart();
            else step = getNext(step);
            if(step==null)
                cnt++;
            else
                System.out.print(step+" ");
            if(cnt%10==0)
                System.out.println();
        }
    }
```

```
    private String getNext(String now){
        HashSet<ConditionPair> conditionPairs = conditionalDistribution.get(now);
        if(conditionPairs==null) return null;
        List<ConditionPair> condtionalPairs = new ArrayList<ConditionPair>(conditionPai
        double rand = Math.random();
        int all = 0;
        for (Iterator<ConditionPair> iterator = condtionalPairs.iterator(); iterator.ha
            ConditionPair conditionPair = (ConditionPair)iterator.next();
            all+=conditionPair.frequency;
        }
        rand = rand * all;
        for (int i = 0; i < condtionalPairs.size(); i++) {
            rand -= condtionalPairs.get(i).frequency;
            if(rand<0) return condtionalPairs.get(i).second;
        }
        return null;
    }
    // token i : value(i-1)<value<=value(i)
    private String getStart(){
        ArrayList<Map.Entry<String,Double>> list = new ArrayList<>(frequencyDistributic
        double rand = Math.random(); // get a value [0,1)
        for (int i = 0; i < list.size(); i++) {
            rand -= list.get(i).getValue();
            if(rand<=0){
               return list.get(i).getKey();
           }
        }
        return null;
    }
    public static void main(String args[]){
        Markov markov = new Markov("Economist.txt");
        markov.create(200);
        System.out.println();
        markov.create(200, "African");
    }
}
class ConditionPair {
    String first;
    String second;
    Double frequency;

    ConditionPair(String first, String second) {
        this.first = first;
        this.second = second;
        frequency = 1.0;
    }

    @Override
    public int hashCode() {
        return first.length() + second.length();
    }

    @Override
    public boolean equals(Object obj) {
        ConditionPair compare = (ConditionPair) obj;
```

```
        return compare.first.equals(this.first) & compare.second.equals(this.second);
    }

    @Override
    public String toString() {
        return first+":"+second+" f: "+ frequency;
    }
}
```