RESUMO

Anotações realizadas durante a pesquisa exploratória pré-qualificação.

Palavras-chave: Concept drift, change-point, detecção de novidades.

Capítulo 1

IMPLEMENTAÇÃO PETTITT - MOA

A fim de verificar a viabilidade de adaptar métodos estatísticos aplicados ao problema de *change-point* para detecção de *concept drifts* em *stream* de dados, foi realizada a implementação do método de Pettitt (PETTITT, 1979) no MOA (https://moa.cms.waikato.ac.nz/).

Observações sobre o método:

- *Nonparametric* - Dados não precisam estar numa distribuição normal. Os números observados são ordinais, indicando posição em um ranqueamento
- Hipótese **null**: não houveram mudanças
- Não requer conhecimento sobre a distribuição inicial

Datasets utilizados:

- Page (PAGE, 1954) - de forma contínua (números racionais...tem de -1.05 a 3.29) e como observações de Bernoulli (0, se $<= 0$, 1, se > 0). Testes exatos e conservadores.
- The Lindisfarne Scribes - binomial (contagem de palavras terminadas em *-s* e *-a*). Acreditava-se que autores diferentes faziam usos diferentes dessas terminações. Testes exatos e conservadores.
- Dados industriais. Percentual de uma material em uma sequências de 27 lotes produzidos. Testes aproximados.

Outras técnicas citadas por Pettitt. Algumas delas foram, de fato, adaptadas para técnicas de detecção de *concept drift*:

- Page - CUSUM (PAGE, 1954).

- Sen and Srivastava (SEN; SRIVASTAVA, 1975) - Testes no nível da média para um modelo normal.
- Hinkley (HINKLEY, 1970) - Probabilidade entre valor especificado de T e a estimativa de T.
- Smith (SMITH, 1975) considers a Bayesian approach to making inferences about the change-point.
- McGilchrist and Woodyer (MCGILCHRIST; WOODYER, 1975) consider a distribution-free CUSUM and

Obs: A maioria desses métodos assume conhecimentos sobre a distribuição inicial dos dados. O método proposto por Pettitt dispensa esse conhecimento prévio.

1.1 IMPLEMENTAÇÃO EM R

```
concept.drift<-function(x, plot=T){
    dataS <- length(x)
    vecSize <- 1:dataS
    dataRank <- rank(x)
    sumData <- sapply(vecSize,
        function(x) 2 * sum(dataRank[1:x]) - x * (dataS + 1))
    absSumData <- abs(sumData)
    maxAbsSumData <- max(absSumData)
    change.point<-vecSize[maxAbsSumData == absSumData]
    if(plot){
        plot(x, t="l", main=paste("Concept Drift:", change.point))
        abline(v=change.point, col="red")
    }
    change.point
}
```

1.2 IMPLEMENTAÇÃO EM JAVA/MOA

```
package moa.classifiers.core.driftdetection;

import com.github.javacliparser.FloatOption;
import com.github.javacliparser.IntOption;
import moa.core.ObjectRepository;
import moa.tasks.TaskMonitor;

import java.util.*;

/**
 * Drift detection method based in Pettitt
```

```
 *
 *
 * @author Ruivaldo Neto (rneto@rneto.net)
 * @version $Revision: 7 $
 */
public class Pettitt extends AbstractChangeDetector {

    private static final long serialVersionUID = 5210470661274384763L;

    public IntOption minNumInstancesOption = new IntOption(
            "minNumInstances",
            'n',
            "The minimum number of instances before permitting detecting change
            100, 0, Integer.MAX_VALUE);


    private ArrayList<Double> dataList;
    private Integer changePoint;
    private Integer nDataWhenChangePoint;

    public Pettitt() {
        resetLearning();
    }

    @Override
    public void resetLearning() {
        this.dataList = new ArrayList<Double>();

        this.changePoint = null;
        this.nDataWhenChangePoint = null;

        this.isChangeDetected = false;
        this.isInitialized = false;
    }

    @Override
    public void input(double inputData) {
        if (this.isChangeDetected) {
            this.isChangeDetected = false;
            dataList.add(inputData);
            return;
        }

        dataList.add(inputData);
```

```
int dataS = dataList.size();

int[] vecSize = new int[dataS];
for (int i = 1; i <= dataS; i++) {
    vecSize[i - 1] = i;
}

Double[] data = new Double[dataS];
dataList.toArray(data);

int[] dataRank = rank(data);

int[] dataRankSum = new int[dataS];
dataRankSum[0] = dataRank[0];
for (int i = 1; i < dataRank.length; i++) {
    dataRankSum[i] = dataRank[i] + dataRankSum[i - 1];
}

int[] sumData = new int[dataS];
for (int i = 1; i < sumData.length; i++) {
    sumData[i] = (2 * dataRankSum[i]) - (i * (dataS + 1));
}

int[] absSumData = new int[dataS];
for (int i = 0; i < absSumData.length; i++) {
    absSumData[i] = Math.abs(sumData[i]);
}

Integer maxAbsSumData = Arrays.stream(absSumData).max().getAsInt();

// Find Index
int newChangePoint = 0;
for (newChangePoint = 0; newChangePoint < absSumData.length; newChangePoint+
    if (absSumData[newChangePoint] == maxAbsSumData) {
        break;
    }
}

// First Index
if (this.changePoint == null) {
    this.changePoint = newChangePoint;
    this.nDataWhenChangePoint = this.dataList.size();
    return;
```

```
            }

            // If different, concept drift
            int changePointDelta = newChangePoint - this.changePoint;
            int nDataDelta = dataList.size() - this.nDataWhenChangePoint;

            if (changePointDelta >= this.minNumInstancesOption.getValue() && change
                this.changePoint = newChangePoint;
                this.nDataWhenChangePoint = this.dataList.size();

                this.isChangeDetected = true;

                return;
            }
        }

        private static int[] rank(Double[] x){
            int [] R = new int[x.length];
            if(x.length == 0)return R;
            Integer [] I = new Integer[x.length];
            for(int i = 0; i < x.length; i++) {
                I[i] = i;
            }
            Arrays.sort(I, (i0, i1) -> (int) Math.signum(x[i0]-x[i1]));
            int j = 0;
            for(int i = 0; i < x.length; i++){
                if(x[I[i]] != x[I[j]])
                    j = i;
                R[I[i]] = j;
            }
            return R;
        }

        @Override
        public void getDescription(StringBuilder sb, int indent) {
            // TODO Auto-generated method stub
        }

        @Override
        protected void prepareForUseImpl(TaskMonitor monitor,
                                         ObjectRepository repository) {
            // TODO Auto-generated method stub
        }
    }
```

Considerações a cerca da implementação e testes:

- Detecta mesmo quando não há drift;
- Muito sensível;
- Necessário adequar um método de janela (?)

Capítulo

2

OUTRA COISA...

teste

REFERÊNCIAS BIBLIOGRÁFICAS

HINKLEY, D. Inference about the change-point in a sequence of random variables. v. 57, 04 1970.

MCGILCHRIST, C. A.; WOODYER, K. D. Note on a distribution-free cusum technique. v. 17, p. 321–325, 08 1975.

PAGE, E. S. Continuous inspection schemes. *Biometrika*, v. 41, n. 1-2, p. 100–115, 1954. Disponível em: ⟨http://dx.doi.org/10.1093/biomet/41.1-2.100⟩.

PETTITT, A. A non-parametric approach to the change-point problem. v. 28, 01 1979.

SEN, A.; SRIVASTAVA, M. On tests for detecting change in mean. v. 3, 01 1975.

SMITH, A. F. M. A bayesian approach to inference about a change-point in a sequence of random variables. *Biometrika*, v. 62, n. 2, p. 407–416, 1975. Disponível em: ⟨http://dx.doi.org/10.1093/biomet/62.2.407⟩.