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

ARTIGOS

2.1 DETECTA: ABRUPT CONCEPT DRIFT DETECTION IN NON-STATIONARY (ESCOVEDO et al., 2018)

- *Concept Drift* é bastante comum. Presente em parte significativa dos problemas do "mundo real"
- É fruto de ambientes não estacionários
- Os problemas de classificação são os mais afetados, pois as predições acabam invalidadas
- Atualizar o modelo frequentemente mitiga, mas é custoso
- Tipos de *Concept Drift*:
  - *Abrupt / Sudden* - Conceito A é substituído imediatamente pelo Conceito B
  - *Gradual* - Dados do Conceito B tornam-se gradativamente mais presentes, até tornarem-se maioria. Não há de fato a mudança cabal de A para B, apenas a maior presença de B. No começo, B pode ser entendido com ruído
- Formas de lidar com *Concept Drift*:
  - *Passive / Reactive*: Trabalha com a taxa de erro. Logo, tem que fazer predições erradas para poder começar a detectar a mudança
  - *Proactive*: Detecta a mudança antes da predição, evitando erros e podendo trazer resultados mais satisfatórios.
- O *DetectA* é uma continuação do trabalho **A2D2: A pre-event abrupt drift detection** (ESCOVEDO et al., 2015)
- Nomeclaturas utilizadas para se referir a *Concept Drift*: *Change Detection*

- Entendido como técnica para detectar *Change Point* e *Small Intervals - Drift*
- *Concept Drift Detectors*:
  - Metodologias para detecção de mudanças na distribuição dos dados
  - Pode utilizar dados da performance do classificador (*error rate*) ou os próprios dados
  - Normalmente observam as seguintes características:
    * Desvio Padrão
    * *Error rate* do modelo
    * *Instance Distribution*
    * Estabilidade
- Foco do trabalho é em Classificação. Mas *Concept Drift* também ocorre em regressões, séries temporais
- Outros métodos pró-ativos citados:
  - *PCA Feature Extraction for Change Detection in Multidimensional Unlabeled Data* (KUNCHEVA; FAITHFULL, 2014):
    * Extrai componentes (PCA)
    * Componentes com menor variância são mais sensíveis à mudança
    * Usa esses componentes através do método semi-paramétrico *log-likelihood* para detectar mudanças na média e variância, o que indicaria *CD*
  - *Proactive drift detection: Predicting concept drifts in data streams using probabilistic networks* (CHEN; KOH; RIDDLE, 2016):
    * Utiliza um histórico das taxas de mudança
    * Diminui o número de detecção de falsos positivos
    * Limitação: Usa a informação do intervalo do drift apenas para predizar novos locais de drift, comparando padrões
    * Entenda como campo minado: o histórico das posições das bombas é usado para "inferir"onde estarão as próximas (CDs)
- Resumo do método **DetectA**:
  - **Suporta metodologia reativa e proativa**
    * **Reativa:** Requer que os dados no instante t e $t+1$ tenham *labels*. Não utiliza algoritmos de clusterização. Simplesmente calcula e compara informações estatísticas entre t e $t+1$.
    * **Pró-ativa:** No instante $t+1$, não existem *labels*. A comparação é feita através da formação de clusters usando um algoritmo não-supervisionado aglomerativo (*k-means*).
      Observações sobre o uso do algoritmo não-supervisionado aglobmerativo:

        - Números de grupos a serem formados é conhecido (número de classes do problema)
        - Centróide inicial é a média condicional do vetor de cada classe
        - Mais eficiente que métodos divisivos

- Conclusões:
  - O detector é eficiente e compatível com datasets de grande dimensionalidade, blocos de tamanho médio, para qualquer proporção de drift e balanceamento das classes
  - O trabalho também propõe um procedimento para produzir datasets com drifts abruptos
- Trabalhos Futuros:
  - Metodologia para evitar o escolha ad-hoc do algoritmo de clusterização (teste com vários, análise da silhoueta ou outro índice)
  - Testar a técnica com um método de classificação mais complexo, *ensembles* de redes neurais ou abordagens neuro evolutivas
  - Abordagem híbrida, combinando método pró-ativo e reativo. Ao receber dados, o método pró-ativo é aplicado. Se não for detectado drift, classificação é realizada. Quando os labels corretos chegarem, o reativo é executado. Se drift for detectado, modelo é retreinado.

Capítulo

3

ALGORITMOS MOA

...

REFERÊNCIAS BIBLIOGRÁFICAS

CHEN, K.; KOH, Y. S.; RIDDLE, P. Proactive drift detection: Predicting concept drifts in data streams using probabilistic networks. In: *2016 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2016. p. 780–787. ISSN 2161-4407.

ESCOVEDO, T. et al. Detecta: abrupt concept drift detection in non-stationary environments. *Applied Soft Computing*, v. 62, p. 119 – 133, 2018. ISSN 1568-4946. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S1568494617306361⟩.

ESCOVEDO, T. et al. A2d2: A pre-event abrupt drift detection. *2015 International Joint Conference on Neural Networks (IJCNN)*, p. 1–8, 2015.

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

KUNCHEVA, L. I.; FAITHFULL, W. J. Pca feature extraction for change detection in multidimensional unlabeled data. *IEEE Transactions on Neural Networks and Learning Systems*, v. 25, n. 1, p. 69–80, Jan 2014. ISSN 2162-237X.

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⟩.