
Algoritmos *anytime* baseados em instâncias para
classificação em fluxo de dados

Cristiano Inácio Lemes

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Cristiano Inácio Lemes

**Algoritmos *anytime* baseados em instâncias para
classificação em fluxo de dados**

Dissertação apresentada ao Instituto de Ciências
Matemáticas e de Computação – ICMC-USP,
como parte dos requisitos para obtenção do título
de Mestre em Ciências – Ciências de Computação
e Matemática Computacional. *EXEMPLAR DE
DEFESA*

Área de Concentração: Ciências de Computação e
Matemática Computacional

Orientador: Prof. Dr. Gustavo Enrique de Almeida
Prado Alves Batista

USP – São Carlos
Fevereiro de 2016

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

Lemes, Cristiano Inácio
L551a Algoritmos *anytime* baseados em instâncias
para classificação em fluxo de dados / Cristiano
Inácio Lemes; orientador Gustavo Enrique de Almeida
Prado Alves Batista. - São Carlos - SP, 2016.
128 p.

Dissertação (Mestrado – Programa de Pós-Graduação
em Ciências de Computação e Matemática Computacional)
– Instituto de Ciências Matemáticas e de Computação,
Universidade de São Paulo, 2016.

1. Classificação baseada em instância.
 2. Algoritmo *anytime*.
 3. Algoritmo incremental.
 4. Fluxo de dados.
 5. Mudança de conceito.
- I. Batista, Gustavo Enrique de Almeida Prado Alves,
orient. II. Título.

Cristiano Inácio Lemes

Instance-based anytime algorithm to data stream classification

Master dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP, in partial fulfillment of the requirements for the degree of the Master Program in Computer Science and Computational Mathematics. *EXAMINATION BOARD PRESENTATION COPY*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Gustavo Enrique de Almeida Prado
Alves Batista

USP – São Carlos
February 2016

*Este trabalho é dedicado a todos que apenas sonharam em suas vidas,
Sonhar é suficiente, realizar este sonho é necessário.
Em especial, a todos que contribuiram para a realização deste sonho.*

AGRADECIMENTOS

Nos últimos meses venho pensando em palavras a serem ditas aqui, agradecimentos que gostaria de fazer. Confesso que não é fácil encontrar palavras para descrever este momento tão importante em minha vida. Traduzir sentimentos em palavras é uma tarefa difícil, pois em muitos casos não existe palavras para descrever aquilo que realmente estamos sentindo. Além disso, existe muitas pessoas que gostaria de agradecer. Diante esta dificuldade, tentarei de todas as maneiras, expressar meus sinceros agradecimentos às pessoas que contribuíram para a realização deste sonho.

Começo agradecendo meus pais, Almir e Elizabeth. Foram anos de muita luta, enfrentando diversas dificuldades, deixando de realizar seus próprios sonhos em prol da realização dos nossos, meu e de meu irmão Marcelo. Graças à oportunidade oferecida por nossos pais foi possível finalizar os estudos no curso superior em uma faculdade de qualidade. Graças a esta formação tive a oportunidade e capacidade de desenvolver este projeto de Mestrado. Um *muito obrigado* pode não representar toda gratidão que tenho pelos meus pais, por todo esforço e dedicação para nos oferecer uma vida digna e proporcionar uma boa formação acadêmica. O maior reconhecimento que posso oferecer a eles é sempre buscar por oportunidades melhores, realizando o trabalho da melhor maneira possível. Ofereço todas minhas conquistas a vocês! Agradeço ao meu irmão que me deu todo apoio e incentivo para que pudesse realizar este trabalho da melhor maneira possível. Quero deixar meus agradecimentos ao restante da minha família que contribuíram de forma ímpar no desenvolvimento do meu trabalho.

Quando fui aprovado no programa de mestrado, eu não conhecia pessoalmente meu orientador. Quando realizei minha matrícula que tive meu primeiro, de tantos contatos que teríamos no desenvolvimento deste trabalho. Agradeço o meu orientador Gustavo por aceitar me orientar antes mesmo de me conhecer pessoalmente. Obrigado pelo apoio, por cada dica e por toda ajuda oferecido no desenvolvimento deste trabalho. Nos momentos em que me encontrava mais perdido e sem saída, ele me mostrou um caminho diferente a ser trilhado, mudando minha forma de pensar. Sem dúvida estes novos caminhos possibilitaram a realização de muita coisa boa em minha vida pessoal e profissional. Graças a esta excelente orientação foi possível eu realizar vários sonhos e desenvolver bons trabalhos.

Parte do trabalho deste projeto foi realizada na Faculdade de Medicina da Universidade do Porto, em Portugal. Agradeço imensamente o Professor Pedro por ter aceitado me receber durante seis meses. Por ter dedicado uma parte importante do seu tempo no desenvolvimento do meu projeto e por ter me recebido da melhor maneira possível. Graças a este intercâmbio foi

possível realizar dois trabalhos importantes no meu mestrado. Ele foi uma pessoa incrível que me apoiou durante todo o tempo, em todos os aspectos. Ainda nos encontraremos!

Quando estava prestes a mudar para São Carlos no intuito de realizar este mestrado, eu estava um pouco perdido sobre como iniciar minha mudança, arrumar casa e tudo mais. Foi quando tive a excelente oportunidade de relembrar uma velha amizade. Ele que foi meu veterano durante a graduação e graças ao mestrado foi possível vivermos outros bons momentos de amizade. Obrigado Vinícius, por me receber na primeira semana em sua casa e me fazer companhia sempre que possível. Obrigado pelos fins de semana de muita risada, ao som de uma boa moda de viola e uma cerveja de qualidade e gelada! Sempre que precisar pode contar comigo meu amigo!

Não podia deixar de agradecer a todos meus amigos Labiquianos. Sem o apoio de vocês, seria difícil a realização de grande parte dos projetos desenvolvidos durante o mestrado. Obrigado pelos momentos do café, responsáveis por ótimos momentos e boas risadas. Um agradecimento especial ao Diego que me ajudou nos momento em que mais precisei e ao RG por sempre dar aquela ajuda com *Linux* ou *Latex*, ou até mesmo o *Matlab*! Peço desculpa por algo que tenha feito de ruim, garanto que eu não tinha a intensão de causar nada de ruim a ninguém. Viu Camila, o bombom foi só uma brincadeira! Nunca me esquecerei de nenhum de vocês, pois cada um contribuiu de alguma maneira no meu desenvolvimento profissional e pessoal.

Aos meus amigos da FMUP, muito obrigado por terem contribuído na realização do meu intercâmbio. Obrigado por ensinar e ajudar a conhecer mais sobre Porto e sobre como é viver nesta cidade incrível. Um agradecimento especial ao Daniel, que sempre esteve disposto a me ajudar nos momentos que precisei. Mostrou um pouco de Porto e que foram momentos que jamais esquecerei. Obrigado também pelos ótimos momentos que passamos durante o café da tarde. Foram muitas risadas que jamais esquecerei. Agradeço a todos que tive a oportunidade de conhecer morando em Porto. Sempre fazendo companhia e proporcionando bons momentos. Um agradecimento especial ao Matheus e ao Ciro que foram grandes amigos durante o período em que morei em Porto. Que esta amizade nunca se perca ao longo do tempo!

Muitos sabem a importância da dança na minha vida. Ela está presente em todos os momentos de minha vida. Sem a dança não seria possível aliviar a tensão nos momentos críticos, os momentos de estresse elevado. Além disso, a dança sempre proporcionou momentos únicos, ao lado de pessoas incríveis, histórias de vida que ajudam a criar nossos conceitos sobre a vida, o universo e tudo mais.

Quando me mudei para São Carlos, uma das primeiras coisas que procurei foi a dança. Lembro-me do meu segundo dia na cidade, quando conheci a turma da dança e desde então ficamos juntos durante muito tempo. Vivemos ótimos momentos, realizamos vários trabalhos relacionados à dança e celebramos a cada dia esta amizade dançando muito samba, zouk, kizomba e forró! Não podia deixar de agradecer a todos vocês, meus amigos da dança, que tive a oportunidade de conviver. Sem vocês não teria sido fácil realizar esta importante etapa na minha

vida. Obrigado a todos integrantes da CIACAASO de dança que contribuíram da melhor forma possível no meu desenvolvimento pessoal e da minha dança. Um obrigado especial ao Caio, por desenvolver um projeto de dança na USP São Carlos e pela iniciativa do desenvolvimento dos trabalhos da CIACAASO. Obrigado também a querida Yvone, minha aluna de kizomba, por sempre promover eventos dançantes e permitir a integração das pessoas em São Carlos e região. Obrigado a todos que tive oportunidade de conhecer em São Carlos e região graças à dança. Cada um de vocês contribuiu de alguma maneira no meu desenvolvimento pessoal e profissional.

Morei na minha querida cidade Uberlândia por muito tempo, e lá realizei o meu curso de graduação. Conheci muitas pessoas nesta época, Raíza, Fred, Rodrigo (vulgo Bodizim) entre tantos outros importantes na minha vida. Quero deixar meus agradecimentos por sempre estarem presentes em minha vida! Vivemos num período de grandes mudanças em minha vida, um período de grandes descobertas que vocês contribuíram de forma ímpar na minha formação pessoal e profissional. Agradeço pelos momentos de muita animação e felicidade ao som de uma boa música e uma cerveja gelada. Um agradecimento especial ao Cláudio, que sempre me incentivou a buscar experiências melhores, que sempre motivou a buscar projetos que eu desejava para minha vida, mas às vezes eu não acreditava, bastou apenas um empurrão para conquistar. “Às vezes temos que dar um passo para trás para alcançarmos um voo mais alto”, nunca me esquecerei desta frase.

Um grande agradecimento também aos meus professores e funcionários da Faculdade de Computação da Universidade Federal de Uberlândia. Sem o apoio e dedicação de todos vocês, não seria possível obter a formação de qualidade e com alto desempenho. Graças a esta formação foi possível à conquista de uma vaga no programa de mestrado de excelência e de alto nível na qual estou prestes a encerrar. Um agradecimento especial ao Marcelo pelo apoio e incentivo na conquista desta vaga de mestrado.

Por fim, quero agradecer imensamente as agências de fomento que possibilitaram o desenvolvimento de todo este projeto de mestrado. Começo agradecendo a CAPES que me apoiou nos primeiros meses do mestrado, possibilitou que eu pudesse morar em São Carlos para desenvolver este projeto. Agradeço duas vezes a FAPESP, pois ela me apoiou no restante do desenvolvimento do projeto até completar dois anos de apoio e possibilitou a realização do estágio de pesquisa na Universidade do Porto.

Muito obrigado!

*“Dancers are the athletes
of God”*
(Albert Einstein)

RESUMO

LEMES, C. I.. **Algoritmos *anytime* baseados em instâncias para classificação em fluxo de dados.** 2016. 128 f. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Aprendizado em fluxo de dados é uma área de pesquisa importante e que vem crescendo nos últimos tempos. Em muitas aplicações reais os dados são gerados em uma sequência temporal potencialmente infinita. O processamento em fluxo possui como principal característica a necessidade por respostas que atendam restrições severas de tempo e memória. Por exemplo, um classificador aplicado a um fluxo de dados deve prover uma resposta a um determinado evento antes que o próximo evento ocorra. Caso isso não ocorra, alguns eventos do fluxo podem ficar sem classificação. Muitos fluxos geram eventos em uma taxa de chegada com grande variabilidade, ou seja, o intervalo de tempo de ocorrência entre dois eventos sucessivos pode variar muito. Para que um sistema de aprendizado obtenha sucesso na aquisição de conhecimento é preciso que ele apresente duas características principais: (i) ser capaz de prover uma classificação para um novo exemplo em tempo hábil e (ii) ser capaz de adaptar o modelo de classificação de maneira a tratar mudanças de conceito, uma vez que os dados podem não apresentar uma distribuição estacionária. Algoritmos de aprendizado de máquina em lote não possuem essas propriedades, pois assumem que as distribuições são estacionárias e não estão preparados para atender restrições de memória e processamento. Para atender essas necessidades, esses algoritmos devem ser adaptados ao contexto de fluxo de dados. Uma possível adaptação é tornar o algoritmo de classificação *anytime*. Algoritmos *anytime* são capazes de serem interrompidos e prover uma resposta (classificação) aproximada a qualquer instante. Outra adaptação é tornar o algoritmo incremental, de maneira que seu modelo possa ser atualizado para novos exemplos do fluxo de dados. Neste trabalho é realizada a investigação de dois métodos capazes de realizar o aprendizado em um fluxo de dados. O primeiro é baseado no algoritmo *k*-vizinhos mais próximo *anytime* estado-da-arte, onde foi proposto um novo método de desempate para ser utilizado neste algoritmo. Os experimentos mostraram uma melhora consistente no desempenho deste algoritmo em várias bases de dados de *benchmark*. O segundo método proposto possui as características dos algoritmos *anytime* e é capaz de tratar a mudança de conceito nos dados. Este método foi chamado de *Algoritmo Anytime Incremental* e possui duas versões, uma baseado no algoritmo *Space Saving* e outra em uma *Janela Deslizante*. Os experimentos mostraram que em cada fluxo cada versão deste método proposto possui suas vantagens e desvantagens. Mas no geral, comparado com outros métodos *baselines*, ambas as versões apresentaram melhor desempenho.

Palavras-chave: Classificação baseada em instância, Algoritmo *anytime*, Algoritmo incremental, Fluxo de dados, Mudança de conceito.

ABSTRACT

LEMES, C. I.. **Algoritmos *anytime* baseados em instâncias para classificação em fluxo de dados.** 2016. 128 f. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Data stream learning is a very important research field that has received much attention from the scientific community. In many real-world applications, data is generated as potentially infinite temporal sequences. The main characteristic of stream processing is to provide answers observing stringent restrictions of time and memory. For example, a data stream classifier must provide an answer for each event before the next one arrives. If this does not occur, some events from the data stream may be left unclassified. Many streams generate events with highly variable output rate, i.e. the time interval between two consecutive events may vary greatly. For a learning system to be successful, two properties must be satisfied: (i) it must be able to provide a classification for a new example in a short time and (ii) it must be able to adapt the classification model to treat concept change, since the data may not follow a stationary distribution. Batch machine learning algorithms do not satisfy those properties because they assume that the distribution is stationary and they are not prepared to operate with severe memory and processing constraints. To satisfy these requirements, these algorithms must be adapted to the data stream context. One possible adaptation is to turn the algorithm into an anytime classifier. Anytime algorithms may be interrupted and still provide an approximated answer (classification) at any time. Another adaptation is to turn the algorithm into an incremental classifier so that its model may be updated with new examples from the data stream. In this work, it is performed an evaluation of two approaches for data stream learning. The first one is based on a state-of-the-art k -nearest neighbor anytime classifier. A new tiebreak approach is proposed to be used with this algorithm. Experiments show consistently better results in the performance of this algorithm in many benchmark data sets. The second proposed approach is to adapt the anytime algorithm for concept change. This approach was called *Incremental Anytime Algorithm*, and it was designed with two versions. One version is based on the *Space Saving* algorithm and the other is based in a *Sliding Window*. Experiments show that both versions are significantly better than baseline approaches.

Key-words: Instance-based classification, Anytime algorithm, Incremental algorithm, Data stream, Concept change.

LISTA DE ILUSTRAÇÕES

Figura 1 – Diferentes tipos de mudança de conceito (DONGRE; MALIK, 2014)	42
Figura 2 – Exemplo de atualização do <i>Stream Summary</i> no <i>Space Saving</i>	49
Figura 3 – Crescimento esperado da qualidade da resposta dos algoritmos <i>anytime</i>	54
Figura 4 – Critério de desempate <i>SimpleRank</i> e <i>DiversityTiebreak</i>	59
Figura 5 – Resultado obtido pelo conjunto de dados <i>Mfeat - Karhunen</i>	65
Figura 6 – Resultado obtido pelo conjunto de dados <i>Optical Recognition of Digits</i>	66
Figura 7 – Resultado obtido pelo conjunto de dados <i>Ionosphere</i>	66
Figura 8 – Estrutura <i>Stream Summary</i> alterada.	77
Figura 9 – Gráfico de espalhamento da configuração $m = 200$; $\alpha = 0.10$; $\beta = 0.20$	85
Figura 10 – Variação da taxa de erro da base <i>4CRE-V2</i> na configuração $m = 200$; $\alpha = 0.10$; $\beta = 0.20$	85
Figura 11 – Variação da taxa de erro da base <i>keystroke</i> na configuração $m = 200$; $\alpha = 0.10$; $\beta = 0.20$	86
Figura 12 – Gráfico de espalhamento da configuração $m = 200$; $\alpha = 0.10$; $\beta = 0.75$	88
Figura 13 – Variação da taxa de erro da base <i>2CDT</i> na configuração $m = 200$; $\alpha = 0.10$; $\beta = 0.75$	88
Figura 14 – Variação da taxa de erro da base <i>UG_2C_5D</i> na configuração $m = 200$; $\alpha = 0.10$; $\beta = 0.75$	89
Figura 15 – Gráfico de espalhamento da configuração $m = 200$; $\alpha = 0.90$; $\beta = 0.20$	90
Figura 16 – Variação da taxa de erro da base <i>5CVT</i> na configuração $m = 200$; $\alpha = 0.90$; $\beta = 0.20$	91
Figura 17 – Variação da taxa de erro da base <i>keystroke</i> na configuração $m = 200$; $\alpha = 0.90$; $\beta = 0.20$	91
Figura 18 – Gráfico de espalhamento da configuração $m = 200$; $\alpha = 0.90$; $\beta = 0.75$	93
Figura 19 – Variação da taxa de erro da base <i>5CVT</i> na configuração $m = 200$; $\alpha = 0.90$; $\beta = 0.75$	94
Figura 20 – Variação da taxa de erro da base <i>UG_2C_5D</i> na configuração $m = 200$; $\alpha = 0.90$; $\beta = 0.75$	95
Figura 21 – Exemplos de conjuntos de dados sintéticos gerados na avaliação sem mudança de conceito	111
Figura 22 – Variação da taxa de erro – sem mudança de conceito – configuração $\sigma: 0.1$; $d: 2$; $c: 2$; $m: 1000$; $\alpha: 0.05$; $\beta: -1.0$	113

Figura 23 – Variação da taxa de erro – sem mudança de conceito – configuração $\sigma: 0.1$; $d: 2; c: 2; m: 2500; \alpha: 0.30; \beta: 0.20$	114
Figura 24 – Variação da taxa de erro – sem mudança de conceito – configuração $\sigma: 0.1$; $d: 2; c: 2; m: 250; \alpha: 0.90; \beta: 0.10$	114
Figura 25 – Média e intervalo de confiança da taxa de erro – sem mudança de conceito – parâmetros m e β	115
Figura 26 – Média e intervalo de confiança da taxa de erro – sem mudança de conceito – parâmetros α e c	115
Figura 27 – Média e intervalo de confiança da taxa de erro – sem mudança de conceito – parâmetros β e d	116
Figura 28 – Gráfico de espalhamento da configuração $m = 200; \alpha = 0.90; \beta = 0.20$. .	118
Figura 29 – Variação da taxa de erro – com mudança de conceito – configuração $m = 200$; $\alpha = 0.90; \beta = 0.20$ – base $4CR$	118
Figura 30 – Variação da taxa de erro – com mudança de conceito – configuração $m = 200$; $\alpha = 0.90; \beta = 0.20$ – base UG_2C_5D	119
Figura 31 – Gráfico de espalhamento da configuração $m = 200; \alpha = 0.90; \beta = 0.75$. .	120
Figura 32 – Variação da taxa de erro – com mudança de conceito – configuração $m = 200$; $\alpha = 0.90; \beta = 0.75$ – base UG_2C_5D	121
Figura 33 – Variação da taxa de erro – com mudança de conceito – configuração $m = 200$; $\alpha = 0.90; \beta = 0.75$ – base MG_2C_2D	121
Figura 34 – Gráfico de espalhamento da configuração $m = 200; \alpha = 0.90; \beta = 0.20$. .	123
Figura 35 – Variação da taxa de erro – com mudança de conceito – nova fórmula – configuração $m = 200; \alpha = 0.90; \beta = 0.20$ – base $4CR$	124
Figura 36 – Variação da taxa de erro – com mudança de conceito – nova fórmula – configuração $m = 200; \alpha = 0.90; \beta = 0.20$ – base $4CRE-V2$	124
Figura 37 – Gráfico de espalhamento da configuração $m = 200; \alpha = 0.90; \beta = 0.75$. .	125
Figura 38 – Variação da taxa de erro – com mudança de conceito – nova fórmula – configuração $m = 200; \alpha = 0.90; \beta = 0.75$ – base UG_2C_5D	126
Figura 39 – Variação da taxa de erro – com mudança de conceito – nova fórmula – configuração $m = 200; \alpha = 0.90; \beta = 0.75$ – base $5CVT$	126

LISTA DE ALGORITMOS

Algoritmo 1 – <i>K</i> -Vizinhos mais Próximos Simples	46
Algoritmo 2 – Space-Saving	48
Algoritmo 3 – Vizinhos mais Próximos <i>Anytime</i>	55
Algoritmo 4 – Ordenação dos Índices – <i>SimpleRank</i>	56
Algoritmo 5 – Diversity Tiebreak	60
Algoritmo 6 – Space-Saving Anytime	78
Algoritmo 7 – Atualiza o <i>Stream Summary</i> usando o <i>Space Saving</i>	79
Algoritmo 8 – Atualiza o <i>Stream Summary</i> usando uma <i>Janela Deslizante</i> sobre o fluxo	80

LISTA DE TABELAS

Tabela 1 – Categorização das Mudanças de Conceito	43
Tabela 2 – Conjuntos de dados <i>benchmark</i>	61
Tabela 3 – Número de empates obtidos pelo <i>SimpleRank</i>	63
Tabela 4 – Área sob a curva de performance <i>anytime</i>	64
Tabela 5 – Conjuntos de dados que apresentam mudança de conceito.	81
Tabela 6 – Resultado <i>CountingRank</i> para configuração: $m = 200; \alpha = 0.10; \beta = 0.20$.	84
Tabela 7 – Resultado <i>CountingRank</i> para configuração: $m = 200; \alpha = 0.10; \beta = 0.75$.	87
Tabela 8 – Resultado <i>CountingRank</i> para configuração: $m = 200; \alpha = 0.90; \beta = 0.20$.	89
Tabela 9 – Resultado <i>CountingRank</i> para configuração: $m = 200; \alpha = 0.90; \beta = 0.75$.	92
Tabela 10 – Resultado – com mudança de conceito – configuração 200; alpha: 0.90; beta: 0.20	117
Tabela 11 – Resultado – com mudança de conceito – configuração $m = 200; \alpha = 0.90;$ $\beta = 0.75$	120
Tabela 12 – Resultado – com mudança de conceito – nova fórmula – <i>Configuração</i> (m: 200; alpha: 0.90; beta: 0.20)	123
Tabela 13 – Resultado – com mudança de conceito – nova fórmula – configuração $m =$ 200; $\alpha = 0.90; \beta = 0.75$	125

LISTA DE ABREVIATURAS E SIGLAS

- DTW *Dynamic Time Warping*
FLORA .. *Floating Rough Approximation*
IB3 *Instance-based 3*
ID3 Iterative Dichotomiser 3
iid independente e identicamente distribuídos
LWF *Locally-Weighted Forgetting*
PECS *Prediction Error Context Switching*
SGBD Sistema de Gerenciamento de Base de Dados
TWF *Time-Windowed Forgetting*

LISTA DE SÍMBOLOS

\Re^d — Espaço d -dimensional

φ — Função que verifica se dois valores são iguais

ϕ — Limiar que define se um elemento é frequente no *space saving*

ε_i — Valor para verificar se um elemento pertence ao *top-k* no *space saving*

Σ — Somatório

α — Probabilidade de interrupção

β — Taxa de exemplos verificados antes da interrupção

μ — Média da distribuição gaussiana na geração de dados sintéticos

σ — Sobreposição entre os exemplos sintéticos gerados pelas distribuições gaussianas

SUMÁRIO

1	INTRODUÇÃO	29
1.1	Justificativa e Motivação	30
1.2	Hipótese e Objetivos	31
1.3	Principais Contribuições	32
1.4	Organização do Trabalho	33
2	CONCEITOS RELEVANTES	35
2.1	Considerações iniciais	35
2.2	Fluxo de dados	35
2.3	Mudança de conceito	40
2.4	Classificação baseada em instância	44
2.5	<i>Space Saving</i>	46
2.6	Considerações Finais	49
3	CLASSIFICAÇÃO ANYTIME	51
3.1	Considerações iniciais	51
3.2	Características dos Algoritmos Anytime	52
3.3	O algoritmo <i>k</i>-Vizinhos mais Próximos <i>anytime</i>	53
3.3.1	<i>Framework k-Vizinhos mais Próximos Anytime</i>	54
3.3.2	<i>Determinação do Índice</i>	55
3.3.3	<i>Encontrando o pior exemplo</i>	56
3.4	O Algoritmo <i>k</i>-Vizinhos mais Proximos <i>anytime</i> com diversidade	57
3.5	Análise Experimental	61
3.5.1	<i>Configuração experimental</i>	61
3.5.2	<i>Empates obtidos pelo k-Vizinhos mais próximos anytime</i>	62
3.5.3	<i>Avaliação</i>	63
3.5.4	<i>Resultados</i>	64
3.6	Considerações Finais	66
4	CLASSIFICAÇÃO INCREMENTAL	69
4.1	Considerações iniciais	69
4.2	Características algoritmos incrementais	70
4.3	O Algoritmo <i>k</i>-Vizinhos Mais Próximo Incremental	73
4.4	O Algoritmo <i>k</i>-Vizinhos Mais Próximo <i>Anytime</i> Incremental	74

4.4.1	<i>Space Saving com CountingRank</i>	79	
4.4.2	<i>Janela Deslizante com CountingRank</i>	79	
4.5	Análise Experimental	80	
4.5.1	<i>Configuração experimental</i>	80	
4.5.2	<i>Avaliação</i>	82	
4.5.3	<i>Resultados</i>	83	
4.5.3.1	<i>Configuração 1: $m = 200; \alpha = 0.10; \beta = 0.20$</i>	83	
4.5.3.2	<i>Configuração 2: $m = 200; \alpha = 0.10; \beta = 0.75$</i>	86	
4.5.3.3	<i>Configuração 3: $m = 200; \alpha = 0.90; \beta = 0.20$</i>	88	
4.5.3.4	<i>Configuração 4: $m = 200; \alpha = 0.90; \beta = 0.75$</i>	92	
4.6	Considerações finais	94	
5	CONCLUSÃO	97	
5.1	Perpectivas Futuras	99	
REFERÊNCIAS		101	
APÊNDICE A		ANÁLISE EXPERIMENTAL SPACE SAVING COUNTINGRANK	109
A.1	Análise experimental – Sem mudança de conceito	109	
A.1.1	<i>Configuração experimental</i>	110	
A.1.2	<i>Avaliação</i>	112	
A.1.3	<i>Resultados</i>	113	
A.2	Análise experimental – Com mudança de conceito	116	
A.2.1	<i>Resultado Configuração 1: $m = 200; \alpha = 0.90; \beta = 0.20$</i>	117	
A.2.2	<i>Resultado Configuração 2: $m = 200; \alpha = 0.90; \beta = 0.75$</i>	119	
A.3	Análise experimental – Nova Fórmula	120	
A.3.1	<i>Resultado Configuração 1: $m = 200; \alpha = 0.90; \beta = 0.20$</i>	122	
A.3.2	<i>Resultado Configuração 2: $m = 200; \alpha = 0.90; \beta = 0.75$</i>	123	
A.4	Considerações Finais	126	



INTRODUÇÃO

Técnicas de aprendizado de máquina aplicadas a um fluxo de dados têm se tornado um tema de interesse entre diversos pesquisadores recentemente. Isso ocorre pelo fato de que em muitas aplicações os dados são gerados em fluxo. Algumas aplicações com esta característica são análise de sentimento em tempo real (KRANJC *et al.*, 2015), sistemas de prevenção e identificação de intrusos (KENKRE; PAI; COLACO, 2015) e integração de registros clínicos através de registros virtuais de pacientes (RODRIGUES; CORREIA, 2013). Nestas aplicações, os dados tipicamente ocorrem em um fluxo contínuo e potencialmente infinito.

A principal característica de uma técnica de aprendizado de máquina quando aplicada a um fluxo de dados é a necessidade de respostas que atendam a severas restrições de tempo. Por exemplo, um classificador aplicado a um fluxo de dados deve fornecer uma classificação a um determinado evento antes que o próximo evento ocorra. Caso isso não aconteça, alguns eventos do fluxo podem ficar sem classificação.

Neste trabalho, tem-se interesse investigar métodos capazes de atender às restrições de tempo impostas por um fluxo de dados. Especificamente, tem-se interesse em pesquisar métodos de classificação *anytime*. Esses classificadores são capazes de fornecer uma boa solução para aplicações onde os dados ocorrem em fluxo. É de interesse também investigar métodos de classificação *anytime* incremental, que possibilitam atualizar o modelo de aprendizado sem a necessidade de armazenar todos os dados passados.

O principal objetivo deste trabalho é pesquisar métodos baseados no algoritmo *k*-vizinhos mais próximos que apresentem características *anytime*. O objetivo secundário deste trabalho é encontrar métodos que apresentem características *anytime* e incremental. Os métodos discutidos neste trabalho operam de forma genérica e não tem vínculo com uma única aplicação. Foram utilizadas várias bases de dados reais e sintéticas nas avaliações dos métodos propostos nesta pesquisa.

1.1 Justificativa e Motivação

Muitas aplicações de fluxo de dados geram dados com intervalo de tempo variável entre eventos. Geralmente os classificadores tradicionais não são capazes de apresentar uma boa solução para esta característica do fluxo de dados. Tais algoritmos tipicamente possuem tempo de resposta fixo que é determinado pelo tempo necessário de analisar o modelo de classificação e obter uma resposta para o evento do fluxo. Porém, para muitas aplicações o intervalo de tempo entre um evento e outro pode variar de milissegundos a minutos, isso pode causar a perda de alguns eventos importantes para a aplicação. Por esta razão, existe a necessidade de pesquisar classificadores que utilizem da melhor forma possível o tempo disponível entre dois eventos consecutivos.

Um exemplo de algoritmo simples e muito investigado na literatura que apresenta tempo de resposta fixo é o *k*-vizinhos mais próximos (MAO *et al.*, 2015; ZUO *et al.*, 2015; BØHLING *et al.*, 2014). A quantidade de tempo necessário para este algoritmo fornecer uma resposta é proporcional à quantidade de exemplos presentes no conjunto de treino. Encontrar as *k* instâncias mais próximas do exemplo de teste na base de treino pode consumir muito tempo de processamento e em alguns casos memória também. Esta é uma técnica que mostra ser muito eficiente para tarefas de aprendizado, por exemplo, estimativa de densidade (BØHLING *et al.*, 2014) e classificação (SAMANTHULA; ELMEHDWI; JIANG, 2014).

Por outro lado, classificadores *anytime* têm apresentado bons resultados quando aplicados a problemas cujos os dados são gerados em fluxo (HU; CHEN; KEOGH, 2015; SERRÀ; ARCOS, 2015). Esses algoritmos são capazes de utilizar o intervalo de tempo entre cada evento e fornecer uma resposta, com uma boa qualidade, no instante em que for necessário. Por sua vez, a qualidade da resposta aumenta em função do tempo. No caso da classificação, a qualidade é medida pela probabilidade de classificação correta de um evento. Um algoritmo *anytime*, após um breve período de tempo de inicialização, pode ser interrompido a qualquer momento e retornar um resultado intermediário. Esta flexibilidade no tempo de resposta permite que esses algoritmos sejam utilizados com sucesso em vários problemas do mundo real que apresentam restrições de tempo variável.

Um exemplo de algoritmo *anytime* é a versão do algoritmo *k*-vizinhos mais próximos *anytime*, proposto por Ueno *et al.* (2006). Neste método as instâncias de treino são ordenadas de acordo com um valor de *rank* que é calculado para cada instância considerando toda base de treino. Para cada exemplo de treino é verificado se a sua classe é igual à classe do seu vizinho mais próximo. Se as classes forem iguais o seu vizinho mais próximo é considerado *associado* e caso contrário ele é considerado *inimigo*. Baseado nisso é realizado o cálculo do valor do *rank* dos exemplos, atribuindo uma bonificação para os exemplos considerados associados e penalizando aqueles que foram considerados inimigos.

Uma das restrições impostas por aplicações com características de fluxo de dados, não

tratada pelo algoritmo *anytime*, é a possibilidade de o algoritmo responder a possíveis mudanças de conceito dos dados. No algoritmo proposto por Ueno *et al.* (2006), para adaptar o modelo de aprendizado incluindo uma nova instância, seria preciso realizar vários cálculos para atualizar todo o modelo, pois esta nova instância pode influenciar no cálculo do valor do *rank* das demais instâncias. E o algoritmo *anytime* quando aplicados a um problema de fluxo, pode não ter disponível o tempo necessário para atualizar o modelo.

Algoritmos incrementais são algoritmos capazes de adaptar o modelo de classificação ao longo do tempo com as novas instâncias que ocorrem no fluxo de dados (GIRAUD-CARRIER, 2000). Com isso, estes algoritmos são capazes de responder a mudanças de conceitos nos dados e podem ser utilizados com sucesso em aplicações que apresentam esta característica. Dos algoritmos já estudados, o *k*-vizinhos mais próximos apresenta características naturais de ser incremental e de se adaptar o modelo de aprendizado. Este algoritmo parece ser uma boa solução para problemas cujo conceito a ser aprendido é complexo, onde existe a necessidade de tratar mudanças neste conceito. Particularmente, um exemplo influencia apenas suas instâncias mais próximas, então atualizar o modelo com um novo exemplo, pode ser restrito à região local na qual o exemplo foi observado.

Uma versão incremental do algoritmo *k*-vizinhos mais próximo foi proposta por Beringer e Hüllermeier (2007a). Seu algoritmo é flexível e capaz de se adaptar a um ambiente que esteja em rápida evolução, uma questão muito importante em problemas que envolvem fluxo de dados. A implementação deste algoritmo é simples e capaz de tratar dois tipos de mudança de conceito, a gradual e a repentina. Com este método é possível realizar a atualização do modelo gerado para o algoritmo *k*-vizinhos mais próximos incluindo novas instâncias e descartando as instâncias mais obsoletas.

Apesar de eficaz para algumas aplicações de fluxo de dados, este algoritmo não apresenta as características dos algoritmos *anytime*. Em alguns casos é interessante que o algoritmo de aprendizado apresente tais características. Na tentativa de mesclar em um único método as propriedades dos algoritmos *anytime*, com as propriedades dos algoritmos incrementais, surgiu a proposta do algoritmo *k-Vizinhos mais próximos Anytime Incremental*. Este algoritmo apresenta de forma simples as propriedades dos algoritmos *anytime* e também dos algoritmos incrementais. Sendo capaz de fornecer respostas intermediárias, possibilitando o processamento de todos os eventos, e também de adaptar o modelo de aprendizado permitindo que o algoritmo trate mudanças de conceitos nos dados.

1.2 Hipótese e Objetivos

A primeira hipótese levantada neste trabalho esta relacionada a um classificador *anytime* baseado em instância. A seguir está descrita esta hipótese.

Adicionar um critério de diversidade ao critério de desempate do algoritmo k -vizinhos mais próximos proposto por Ueno et al. (2006) pode melhorar o desempenho de classificação desse algoritmo

O principal objetivo em torno desta hipótese é investigar e avaliar métodos que possam garantir diversidade no critério de desempate do algoritmo proposto por Ueno et al. (2006). Em um pequeno experimento realizado com este método, pôde ser concluído que o critério de desempate utilizado pelo método original apresentava algumas desvantagens que serão discutidas nos capítulos a seguir. Daí surgiu a hipótese de que adicionando diversidade ao critério de desempate, o algoritmo *anytime* poderia obter melhor desempenho.

A segunda hipótese deste trabalho também está relacionada a um classificador *anytime* baseado em instância, porém capaz de tratar mudança de conceito em fluxo de dados. A seguir está descrita esta hipótese.

*Para lidar com mudanças de conceito em fluxo de dados é necessário tornar incremental o algoritmo k -vizinhos mais próximo *anytime**

O principal objetivo relacionado a esta hipótese é investigar e avaliar métodos capazes de tornar incremental o algoritmo k -vizinhos mais próximo *anytime*. Este algoritmo apresenta bom desempenho quando aplicado a um problema de classificação em fluxo de dados. Porém, com ele não é possível tratar mudança de conceito nos dados. Tornar este algoritmo incremental possibilita que ele trate mudança de conceito sem perder sua habilidade de utilizar, da melhor maneira, o tempo de processamento disponível entre dois eventos consecutivos.

Dada estas duas hipóteses, os objetivos deste trabalho são:

- Pesquisar métodos que garantam diversidade nos critério de desempate do algoritmo k -vizinhos mais próximo *anytime*;
- Avaliar os métodos do item anterior e comparar os resultados com os resultados obtidos pelo critério de desempate original utilizado pelo algoritmo k -vizinhos mais próximos;
- Pesquisar por técnicas para adaptar o algoritmo k -vizinhos mais próximos *anytime* tornando incremental este algoritmo;
- Avaliar os métodos pesquisados em domínios de aplicação reais e de *benchmark*.

1.3 Principais Contribuições

Este trabalho iniciou buscando por métodos capazes de ordenar os dados de treinamento e que pudesse ser utilizado pelo *framework* proposto por Ueno et al. (2006). O primeiro método analisado é baseado na distância dos exemplos para os centroides obtidos pelo algoritmo *kmeans*. Depois foi investigado um método baseado na distância dos exemplos para o *medoide* de cada

classe. Foi testado também um método baseado em um modelo de Gaussianas para ordenar os exemplos de acordo com suas probabilidades. Por fim, foi testada uma ordenação baseado na diversidade e no quanto cada exemplo está distribuído no espaço. Porém os resultados obtidos por todos estes métodos não foram pouco satisfatório. Todos apresentaram desempenho abaixo do método de ordenação aleatória, não apresentando nenhum ganho para o algoritmo *anytime*.

Analizando o número de empates obtidos pelo método *Simple Rank* foi observado que o método apresentava grande número de empates. Assim a ordenação do conjunto de treinamento sofria muita influência do critério de desempate adotado pelos autores. Daí surgiu a proposta de utilizar um dos métodos estudados como critério de desempate do algoritmo *k*-vizinhos mais próximos *anytime*. O método utilizado foi o baseado na diversidade dos exemplos no espaço e este método apresentou resultados melhores e consistentes do que o método *Simple Rank*. Estes resultados foram publicados na 13th International Conference on Machine Learning and Applications (LEMES *et al.*, 2014), e estão descritos no Capítulo 3, onde pode ser encontrada também a descrição do framework proposto por Ueno *et al.* (2006).

A segunda contribuição deste trabalho é um algoritmo *anytime* com propriedades incrementais. Com este algoritmo é possível tratar a alta variabilidade de tempo entre cada evento e ao mesmo tempo tratar mudanças de conceito que os dados sofrem ao longo do tempo. O algoritmo é baseado no *k*-vizinhos mais próximos *anytime* descrito no Capítulo 3, porém utilizando uma fórmula que possibilite atualizar facilmente o modelo de aprendizado. Foram propostos dois métodos para atualização do modelo de aprendizado utilizado pelo algoritmo, um baseado no algoritmo *Space Saving* descrito na Seção 2.5 do Capítulo 2, e outro baseado em uma *Janela Deslizante*. Estas propostas estão descritas no Capítulo 4 e o material para sua publicação está sendo organizado.

1.4 Organização do Trabalho

Este trabalho está organizado da seguinte maneira: no Capítulo 2 são descritos alguns dos conceitos básicos utilizados nas propostas realizadas neste trabalho; no Capítulo 3 é apresentado o algoritmo *k*-vizinhos mais próximos *anytime* e os experimentos e resultados obtidos pelo novo critério de desempate utilizado por este algoritmo proposto neste trabalho; no Capítulo 4 são apresentadas algumas características dos algoritmos incrementais e os experimentos e resultados obtidos pelo algoritmo *k*-vizinhos mais próximos *anytime* incremental proposto neste trabalho; o trabalho é concluído no Capítulo 5; no Apêndice A são apresentados estudos e avaliações experimentais adicionais realizadas durante a pesquisa dos métodos propostos no Capítulo 4.



CONCEITOS RELEVANTES

2.1 Considerações iniciais

Alguns conceitos da literatura motivaram a elaboração dos métodos propostos neste projeto. Entender tais conceitos pode auxiliar no melhor entendimento dos métodos propostos neste trabalho. Assim, na Seção 2.2 são exibidas algumas das propriedades dos fluxos de dados, assim como algumas características que os algoritmos de aprendizado de máquina devem possuir para obter bons resultados na classificação de dados em fluxo. Na Seção 2.3 são apresentadas algumas propriedades das mudanças de conceitos e também algumas categorias das mudanças que os dados podem sofrer.

O algoritmo *anytime* estudado neste projeto foi baseado no algoritmo *k*-vizinhos mais próximos. Então na Seção 2.4 é descrito este algoritmo, assim como alguma de suas propriedades. O algoritmo *anytime* incremental foi baseado em outro algoritmo que será descrito na Seção 2.5, o *Space Saving*. Este método foi proposto para realizar a contagem de elementos em um fluxo de dados.

2.2 Fluxo de dados

Muitos modelos de decisão são executados continuamente ao longo do tempo em um ambiente com restrições de recursos de máquina. Nesse cenário, pode ser necessário detectar e reagir a eventuais mudanças de conceitos nos dados gerados. Esta é uma característica comum em muitas aplicações reais, onde os dados são gerados através de um *fluxo de dados* (BABCOCK *et al.*, 2002; DOMINGOS; HULTEN, 2003; GAMA; SEBASTIÃO; RODRIGUES, 2013). O fluxo de dado é caracterizado por gerar dados continuamente, em grandes quantidades e com intervalo de tempo entre cada dado gerado altamente variado, além de ser possivelmente ilimitado.

Devido à grande quantidade de dados gerados pelo fluxo, realizar uma tarefa de aprendi-

zado de máquina pode utilizar muito tempo de processamento. Se os dados forem gerados mais rápidos que o tempo de processamento do classificador, alguns destes dados podem deixar de ser processados. Além disso, pode ser inviável armazenar todos os dados em memória, devido ao seu tamanho ilimitado. Com isso, os sistemas de aprendizado necessitam contemplar soluções para as restrições tempo e memória presente em um fluxo de dados. Exemplos de aplicações com esta característica são financeira (ZHOU *et al.*, 2015), monitoramento de trânsito rodoviário (WANG *et al.*, 2015), gerenciamento de redes de telecomunicação (DELATTRE; IMBERT, 2015), análise de sentimento em tempo real (KRANJC *et al.*, 2015), sistemas de prevenção e identificação de intrusos (KENKRE; PAI; COLACO, 2015) e integração de registros clínicos através de registros virtuais de pacientes (RODRIGUES; CORREIA, 2013).

Um fluxo de dados é uma sequência de itens de dados gerados em tempo real, continuamente e ordenada pelo tempo (GOLAB; ÖZSU, 2003). Em problemas de classificação de fluxo de dados, os exemplos podem ser representados por uma lista de atributo-valores. Alguns, ou até mesmo todos, desses dados não estão disponíveis *a priori* para serem acessados no disco ou na memória. Em algumas aplicações assume-se que os itens são independente e identicamente distribuídos (*iid*) e gerados por uma distribuição estacionária. Porém em muitos fluxos de dados isso não ocorre, sendo que os dados são gerados por uma distribuição não estacionária em processos que evoluí ao longo do tempo. Os fluxos de dados apresentam características peculiares quando comparados com o modelo relacional tradicional de armazenamento, onde Babcock *et al.* (2002) apresentaram algumas das características dos fluxo de dados.

- Os dados no fluxo são gerados de forma *online*;
- O sistema não tem controle sobre a ordem em que os elementos são gerados para serem processados, seja em apenas um fluxo de dados ou em conjunto de fluxos de dados;
- Um fluxo de dado é, possivelmente, de tamanho ilimitado;
- Uma vez que um elemento de um fluxo de dados foi processado ele é descartado ou arquivado – ele não pode ser recuperado facilmente a menos que ele esteja explicitamente armazenado em memória, onde tipicamente é relativamente pequena para o tamanho do fluxo de dados.
- Os dados gerados em instantes de tempo diferentes, podem apresentar características peculiares, mesmo pertencendo ao um mesmo grupo de itens. Isso caracteriza uma mudança de conceito, que é mais bem detalhada na Seção 2.3.

Sistemas de aprendizado baseados no tradicional gerenciador de dados Sistema de Gerenciamento de Base de Dados (SGBD) não são capazes de carregar todos os dados produzidos pelo fluxo e realizar uma extração de conhecimento de todos estes dados. Estes gerenciadores não foram projetados para carregar continuamente e rapidamente os dados gerados por um

fluxo apresentando bom desempenho e, além disso, realizar uma operação com estes dados. Outro problema em tais sistemas é que eles não suportam diretamente *consultas contínuas*, que é muito típico em aplicações de fluxo de dados. As consultas contínuas são semelhantes às consultas realizadas em sistemas de gerenciamento de dados tradicional, porém estas consultas são realizadas continuamente a cada dado gerado pelo fluxo (TERRY *et al.*, 1992; GOLAB; ÖZSU, 2003; ARASU; BABU; WIDOM, 2004).

Mesmo possuindo características que impossibilitam o uso completo destes sistemas de gerenciamento de dados em fluxo de dados, isso não exclui a possibilidade de armazenar alguns dados utilizando o modelo tradicional de gerenciamento. Frequentemente, o processamento em fluxo de dados pode realizar a junção entre dados provenientes do fluxo de dados e também dados armazenados no modelo de gerenciamento tradicional, como é o caso do algoritmo *k*-vizinhos mais próximos.

Realizar a extração de conhecimento de grandes volumes de dados oriundos de fluxos de dados exige que os métodos de mineração de dados realize o processamento dos dados na velocidade em que eles chegam utilizando os recursos disponíveis. É preciso também que estes dados atualizem o modelo de aprendizado utilizado pelo método. Baseado nisso, Domingos e Hulten (2003) identificaram algumas propriedades importantes que, idealmente, os sistemas de aprendizado devem apresentar para lidar com tais restrições.

- Deve utilizar um tempo de processamento pequeno e constante por item, caso contrário o método irá deixar de processar algum item gerado pelo fluxo;
- Deve utilizar uma quantidade fixa de memória principal, independente da quantidade de itens a ser processado;
- Deve ser capaz de construir o modelo realizando uma única leitura dos dados, uma vez que o método pode não ter tempo para realizar novas leituras de dados antigos, e os dados podem não estar todos disponíveis em um armazenamento secundário para serem lidos no futuro;
- Deve possuir um modelo utilizável disponível a qualquer momento durante o processamento dos itens, diferente da forma como é realizado pelos métodos tradicionais, onde os modelos são feitos processando os dados antes de iniciar o processo de aprendizado;
- Idealmente, ele deve produzir um modelo que é equivalente (ou pelo menos aproximadamente igual) ao modelo obtido pelo correspondente algoritmo de aprendizado, operando sem as restrições impostas pelo fluxo de dados;
- Quando ocorrer uma mudança no processo gerador de dados, ou seja, quando ocorrer uma mudança de conceito, o modelo de aprendizado deve ser atualizado, mas sem deixar de incluir informações dos itens processados no passado.

A primeira vista, parece pouco provável um sistema de aprendizado contemplar todas estas restrições simultaneamente. No caso deste trabalho, tem-se interesse em contemplar apenas algumas destas restrições. Gama, Sebastião e Rodrigues (2013) também identificaram algumas dimensões que influenciam o processo de aprendizado.

- *Espaço* – a quantidade de memória disponível é fixa;
- *Tempo de aprendizado* – processar os exemplos de entrada na velocidade em que eles chegam;
- *Poder de generalização* – o quanto efetivo é o modelo de aprendizado para fornecer respostas consistentes para o conceito a ser aprendido.

Muitos algoritmos consideram que os recursos computacionais são ilimitados, sendo possível, por exemplo, armazenar todos os dados na memória principal. Como os fluxos de dados são de tamanho ilimitado, a quantidade de espaço necessário para armazenar todos os dados gerados pode ser ilimitada também. Em várias aplicações, a quantidade de memória principal pode não ser suficiente para armazenar todos os dados gerados pelo fluxo. O tempo para calcular a resposta exata para uma consulta em um fluxo que produz grande quantidade de itens pode alcançar valores elevados. Existem algoritmos capazes de tratar conjuntos de dados maiores que a quantidade de memória disponível. Porém, tais algoritmos não são viáveis para aplicações de fluxo de dados, uma vez que tais algoritmos operam de forma muito lenta para fornecer uma resposta em tempo real.

O tempo para obter uma resposta, mesmo que aproximada, é um fator a se preocupar ao desenvolver sistemas inteligentes para operar em um fluxo de dados. Além do grande volume de dados, eles também são gerados a uma alta taxa de variabilidade ao longo do tempo. Novos dados podem ser gerados enquanto os dados antigos ainda estão sendo processados pelo sistema. Se o tempo de processamento de cada item for alto, alguns destes novos itens não serão processados pelo sistema. Em muitas aplicações é preciso que o sistema de aprendizado forneça uma resposta para todos os elementos, por exemplo, na aplicação onde se deseja identificar insetos através de um sensor ótico que gera dados em fluxo (SOUZA; SILVA; BATISTA, 2013).

Técnicas de aprendizado tradicionais utilizam conjuntos de dados de treino finito e geram modelos estáticos de apoio a decisão. Porém, estes métodos não apresentam boa solução quando utilizados em um fluxo de dados, pois o conjunto de dado não é finito e estará disponível ao longo do tempo, de forma online, e os modelos estáticos podem gerar soluções imprecisas devido a eventuais mudanças nas características dos dados. Para que um sistema de aprendizado obtenha sucesso diante um fluxo de dados, é importante que ele seja *adaptável* e capaz de fornecer um *resultado aproximado* quando requisitado (BABCOCK *et al.*, 2002). O sistema é *adaptável* quando ele atualiza o seu modelo com os novos dados que estão sendo gerados pelo fluxo. Para o sistema fornecer um *resultado aproximado* é preciso que ele busque por respostas aproximadas

baseados em um conjunto finito de dados, já que possivelmente não será possível calcular a resposta exata para um determinado item baseado em todos os dados gerados pelo fluxo. Porém, a qualidade do *resultado aproximada* deve ser alta para ser substituída pela resposta exata. Já os sistemas baseados em SGBD operam em sentido contrário, buscando respostas precisas baseados em consultas estáveis, sem se preocupar com o desempenho do processamento.

No intuito de encontrar métodos de aprendizado de máquina capazes de fornecer uma boa solução para as restrições impostas por um fluxo de dados, Gaber, Zaslavsky e Krishnaswamy (2005) categorizaram estas soluções em *baseado em dados (data-based)* e *baseado em tarefa (task-based)*. As soluções *baseadas em dados*, são técnicas onde o objetivo é utilizar apenas um subconjunto de todo o conjunto de dado ou transformar os dados na horizontal ou vertical para aproximar uma representação menor dos dados. Já as soluções *baseadas em tarefa*, são técnicas da teoria computacional que são adotadas para alcançar soluções eficientes em tempo e espaço.

A categoria *baseada em dado* pode ser dividida em outras duas subcategorias, as técnicas que resumem todos o conjunto de dados e as técnicas que selecionam um subconjunto dos dados gerados pelo fluxo para serem analisados. *Amostragem, load shedding e sketching techniques* são alguns métodos que representam a primeira subcategoria. *Synopsis data structures e aggregation* representam a segunda subcategoria.

- *Amostragem*: processo probabilístico de escolha dos itens de dados a serem processados ou não. Este método tem sido utilizado por um longo tempo. Pode ser utilizado em conjunto com alguma função de perda. O problema em aplicar esta técnica em um fluxo de dados é que o tamanho do conjunto de dado é desconhecido a priori. Outro problema é que seria importante checar por ruídos nos dados para uma melhor análise na extração de conhecimento do fluxo de dados. Além disso, este método não trata a variação de tempo em que os dados são gerados;
- *Load Shedding*: processo de quebrar uma sequencia de itens do fluxo de dados. Tem sido utilizado com sucesso em consultas de fluxo de dados, porém apresenta os mesmos problemas do método *amostragem*;
- *Sketching techniques*: processo de projetar aleatoriamente um subconjunto das características dos dados. Seria um processo de transformação vertical dos dados de entrada. É difícil usar esta técnica em um fluxo de dados devido ao tempo de processamento necessário para realizar esta transformação;
- *Synopsis Data Structures*: processo de resumo dos dados de entrada para uma análise mais detalhada. Quando esta técnica é utilizada, respostas aproximadas são produzidas, uma vez que os dados não são representados por todas as suas características;

- *Aggregation*: processo que calcula medidas estatísticas, tal como média e variância, que resumem os dados de entrada do fluxo. Seu problema é que não apresenta bom desempenho quando aplicado em uma distribuição de dados altamente variável.

As técnicas *baseadas em tarefa* são métodos que modificam alguns métodos existentes ou inventam novos métodos com o objetivo de solucionar os desafios computacionais de processar um fluxo de dados. *Algoritmos de aproximação* e *janela deslizante* são alguns exemplos de métodos que representam esta categoria.

- *Algoritmos de aproximação*: algoritmos que podem encontrar soluções aproximadas com baixo erro. Podem ser utilizados facilmente em problemas de aprendizado em fluxo de dados.
- *Janela Deslizante*: São métodos que utilizam os dados mais recentes produzidos pelo fluxo de dados para realizar o aprendizado de um determinado conceito. Pode utilizar versões resumidas de dados antigos para apoiar sua decisão.

Existe uma classe de algoritmos de aproximação capaz de lidar com a alta variabilidade de tempo entre cada item a ser processado pelo sistema de aprendizado. Esta classe de algoritmo é chamada de *Algoritmos Anytime* e será vista com mais detalhes no Capítulo 3. Os *algoritmos anytime* utilizam da melhor maneira o tempo de processamento disponível antes de enviar uma resposta aproximada, e quanto maior o tempo de processamento disponível, melhor será a qualidade da resposta. Na próxima seção serão apresentadas algumas características de fluxo de dados que utilizam uma distribuição não estacionária para gerar os dados, ou seja, um fluxo de dados que apresenta *mudança de conceito*.

2.3 Mudança de conceito

Em um fluxo de dados, geralmente os dados gerados não são estáticos, eles podem sofrer mudanças ao longo do tempo. Estas mudanças podem ocorrer de duas formas: ou o conceito que o dado representa muda ou os atributos do dado mudam. Esta alteração no conceito que o dado representa é geralmente conhecida por *mudança de conceito* (*concept drift*) (WIDMER; KUBAT, 1996; TSYMBAL, 2004). A origem desta mudança pode depender de um contexto desconhecido e externo ao próprio processo gerador dos dados. A segunda forma de mudança pode ser chamada de *mudança de conceito virtual* (*virtual concept drift*) (TSYMBAL, 2004). A origem desta mudança está associada a alguma mudança na distribuição responsável por gerar os dados, sendo que neste caso, esta distribuição é chamada de *distribuição não estacionária*. Em Salganicoff (1997), mudança de conceito virtual é referenciado por *sampling shift*, e mudança de conceito real é referenciado como *concept shift*. Exemplos de aplicações reais que apresentam mudança de conceito inclui sistemas de monitoramento (GEISLER; QUIX, 2014), detecção

de fraude financeira (MALEKIAN; HASHEMI, 2013), categorização de spam (JOU, 2013) e previsão meteorológica (BOJINSKI *et al.*, 2014).

Métodos de aprendizado de máquina quando aplicados em problemas com estas características devem ser capazes de tratar estas mudanças nos dados. Estas mudanças fazem com que o modelo de aprendizado construído com os dados antigos se torne inconsistente com os novos dados. Assim o método de aprendizado se torna ineficiente, conduzindo a muitas decisões erradas, o que reduz consideravelmente o desempenho do método. Essa perda de desempenho não representa que o método não é adequado em tais contextos. Ela apenas evidencia a necessidade de atualizar o modelo de aprendizado utilizado pelo método com os novos itens gerados pelo fluxo de dados. Nesse contexto é preciso de métodos especiais, diferentes das técnicas geralmente utilizadas, capazes de tratar os itens recém-produzidos pelo fluxo com a mesma importância na decisão do conceito final dos novos itens desconhecidos (TSYMBAL, 2004).

Porém, nem todas as alterações nos dados estão relacionadas a alguma forma de mudança de conceito. Algumas diferenças nos dados estão relacionadas à ruídos que os dados apresentam. Distinguir entre uma mudança real nos dados e um ruído não são uma tarefa fácil. Existem alguns métodos capazes de identificar ruídos nos dados, mas que não tratam mudança de conceito. Assim como existe alguns métodos que tratam mudança de conceito, mas não tratam ruídos nos dados. O método de aprendizado de máquina ideal deve conciliar estas duas funcionalidades.

Em Zliobaite (2009), os tipos de mudança foram relacionados aos padrões de configuração das origens dos dados sobre o tempo. Ela divide as mudanças em três grupos: *repentina*, *gradual* e *recorrente*. A mudança *repentina* (*sudden drift*), ocorre quando um determinado conceito é substituído por outro de um instante para o outro, sendo que esta mudança é instantânea e irreversível. *Recorrente* (*reoccurring concept*) é quando um conceito previamente ativo reaparece depois de algum tempo inativo. O período para um conceito reaparecer é incerto, ou seja, o seu ressurgimento é irregular. Porém existem algumas aplicações que o conceito reaparece periodicamente, como é o caso das estações do ano (primavera, verão, outono e inverno).

Existem duas formas de ocorrer a mudança *gradual* (*gradual drift*). A primeira se refere à mudança no conceito onde o novo conceito começa a surgir sem que o antigo deixe de existir, e os itens gerados pelo fluxo alternam entre os dois conceitos. Aos poucos o antigo conceito deixa de existir, restando apenas o novo conceito. Esta forma de mudança pode ser chamada apenas de *gradual*. O segundo tipo de mudança gradual ocorre quando existe uma transição gradativa entre o antigo e o novo conceito, onde esta transição é representada por conceitos intermediários que são diferentes do antigo e do novo conceito. Este tipo de mudança pode ser chamada de *incremental*.

É importante notar que os tipos de mudança discutidos neste trabalho não representam todas as possibilidades de mudança existentes. Existem vários tipos de mudança possíveis, e se pensar que pode haver mais de um gerador de fluxo de dados, são inúmeras as combinações possíveis de mudanças. Mais ainda, geralmente o tamanho de um fluxo de dados tende ao infinito,

isso faz com que sejam infinitas também as possibilidades de combinações entre os tipos de mudança nos dados gerados por este fluxo. Zliobaite (2009) definiu estes tipos de mudança para ajudar a desenvolver estratégias de adaptação do modelo de aprendizado.

Já em Dongre e Malik (2014), as mudanças de conceito foram categorizadas em *Repentina (Sudden)*, *Incremental*, *Gradual*, *Recorrente (Recurring)*, *Pequeno ponto (Blip)* e *Ruído (Noise)*. A Figura 1 mostra estes seis tipos de mudança. As mudanças *Repentina*, *Incremental*, *Gradual* e *Recorrente* tem a mesma definição discutida anteriormente. A mudança *Pequeno ponto* representa um “evento raro”, que pode ser considerado como um ruído em uma distribuição estacionária. Nesta mudança o conceito é alterado num instante de tempo t e logo em seguida ele volta ao seu estado anterior. Já o *Ruído*, representa as mudanças que ocorrem de forma aleatória e sem nenhum padrão.

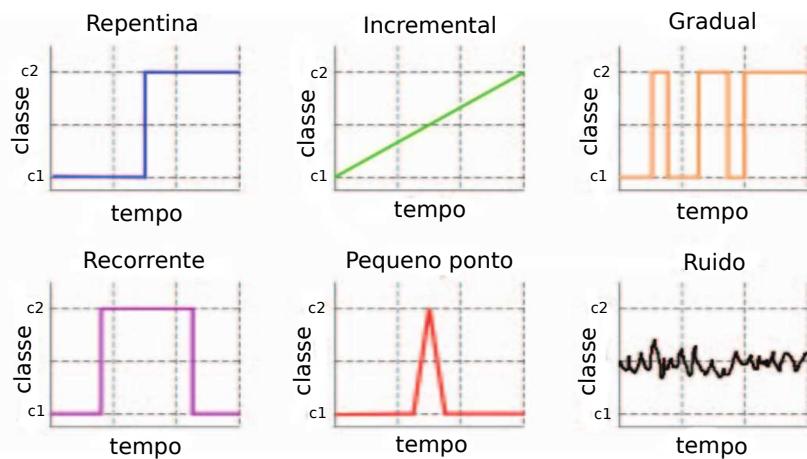


Figura 1 – Diferentes tipos de mudança de conceito (DONGRE; MALIK, 2014)

Além dessas tentativas de categorizar as mudanças, existe um trabalho que categoriza os tipos de mudança em categorias mutuamente exclusivas, baseados no número de recorrência, gravidade, velocidade e previsibilidade (MINKU; WHITE; YAO, 2010). De acordo com os autores, é preciso categorizar as mudanças em categorias menos heterogêneas, dividindo as mudanças em diferentes tipos de acordo com diferentes critérios, e criando categorias mutuamente exclusivas e não vagas considerando um critério particular. Eles dividiram os critérios para categorização das mudanças em dois grupos principais, *Alteração Isolada* e *Alteração Sequencial*, como pode ser visto na Tabela 1.

Analizando uma alteração isolada, pode se observar que diferentes alterações podem causar uma quantidade diferente de mudanças nos dados, onde estas podem demorar mais ou menos tempo para serem finalizadas por completo. Para categorizar uma alteração isolada, os autores utilizaram a *Severidade* (quantidade de mudanças que um novo conceito causa) e *Velocidade* (o inverso do tempo gasto para um novo conceito substituir completamente o antigo conceito) como critério de categorização.

O critério *Severidade* ainda foi dividido em *Classe* e *Característica*. Do critério severi-

Tabela 1 – Categorização das Mudanças de Conceito (MINKU; WHITE; YAO, 2010)

Critério			Categoria	
Alteração isolada	Severidade	Classe	Rigorosa	
			Interseção	
		Característica	Rigorosa	
	Velocidade		Interseção	
			Repentina	
		Gradual	Probabilística	Contínua
Alteração Sequencial	Previsibilidade	Previsível		
		Aleatória		
	Frequência	Periódica		
		Não periódica		
	Recorrência	Recorrente	Cíclica	Desordenada
		Não Recorrente		

dade por classe resulta as categorias *Rigorosa* e *Interseção*. A mudança é considerada *Rigorosa* quando nenhum exemplo mantém a classe antiga no novo conceito, ou seja, 100% do espaço de entrada muda para a nova classe quando a mudança ocorre. Se parte do espaço de entrada mantém a classe antiga durante o novo conceito, a alteração é de *Interseção*.

O critério severidade por classe reflete principalmente mudanças nos conceitos que os dados pertencem, não refletindo as mudanças que os dados sofrem em seus atributos. Assim, os autores introduziram o critério de severidade por característica, que representa a quantidade de mudanças na distribuição dos atributos de entrada. Assim como no critério anterior, a categoria deste critério pode ser *Rigorosa*, se todos os atributos de entrada sofrem alteração em suas distribuições, ou *Interseção*, se parte do espaço de entrada mantém a mesma distribuição.

O critério velocidade deve ser interpretado como sendo o inverso do tempo gasto em uma alteração completa para um novo conceito. Ou seja, quanto maior a velocidade, menor será o intervalo de tempo em que uma mudança ocorre, e quanto menor a velocidade, maior o intervalo de tempo para finalizar uma mudança por completo. Assim as mudanças podem ser categorizadas como *Repentina*, quando a mudança completa ocorre em apenas um instante de tempo, ou *Gradual*, caso contrário. A categoria *Gradual* ainda foi dividida em *Probabilística* ou *Contínua*. A mudança gradual probabilística ocorre quando o conceito muda gradualmente, criando um período de incerteza entre os conceitos. O novo conceito substitui o antigo gradualmente e alguns itens ainda devem ser classificados de acordo com o antigo conceito. Já na mudança gradual contínua o conceito muda gradualmente e continuamente do antigo para o novo conceito, sofrendo modificações a cada intervalo de tempo consecutivo. Neste caso os novos itens não necessitam ser classificados utilizando o antigo conceito.

Para categorizar as alterações sequenciais, os autores utilizaram a *Previsibilidade* (se a alteração é completamente aleatória ou segue algum padrão), *Frequência* (o quanto frequente a

alteração ocorre e se elas podem ter um comportamento periódico) e *Recorrência* (possibilidade de um conceito antigo retornar) como critério de categorização.

O critério previsibilidade pode ser dividido em duas categorias, *Aleatória* ou *Previsível*. Um exemplo de mudança previsível é um conceito representado por um círculo com raio fixo, mas que o ponto central move em uma linha reta. Se este mesmo conceito mover em uma direção desordenada, tem-se uma mudança aleatória.

O critério recorrência é dividido em categoria *recorrente*, quando um conceito antigo reaparece, e *não recorrente* caso contrário. A categoria recorrente é dividida em categoria *cíclica*, quando o conceito reaparece periodicamente, ou *desordenada*, quando o conceito reaparece a qualquer instante de tempo. É interessante notar que, embora um determinado conceito volte a reaparecer, este é similar ao conceito anterior, mas não o mesmo.

O critério frequência pode ser categorizado em *periódico* e *não periódico*. Se um novo conceito ocorre durante todo intervalo de tempo t , a alteração é considerada periódica. Caso contrário ela é não periódica. Vale notar aqui também que, embora os critérios *recorrente* e *frequência* sejam distintos, muitas mudanças que apresentam comportamento *recorrente* são também *frequente periódico*.

2.4 Classificação baseada em instância

Aprendizado baseado em instância é o nome atribuído a um *framework* e metodologia capaz de realizar a classificação usando instâncias específicas em um conjunto de dados (AHA; KIBLER; ALBERT, 1991). Os métodos que utilizam esta técnica estendem a ideia da classificação pelo algoritmo vizinho mais próximo, proposto por Cover e Hart (1967). Algumas aplicações tem obtido sucesso quando utilizam esta técnica, tais como o problema de diagnóstico da doença do *Mal de Parkinson's* (CHEN *et al.*, 2015), a identificação de deslizamento de terra em pistas em montanhas (CHENG; HOANG, 2015), monitoração e categorização de *tweet* (GOBEILL; GAUDINAT; RUCH, 2014) e identificação de insetos de interesse através de sensor laser (SOUZA; SILVA; BATISTA, 2013).

Classificação pelo vizinho mais próximo considera que regiões próximas no espaço pertencem à mesma categoria (COVER; HART, 1967). Isso ocorre pelo fato de estas regiões apresentarem características semelhantes no espaço d -dimensional (\mathbb{R}^d). Também conhecido como *k-vizinhos mais próximo* (*k-nearest neighbor*), esta técnica é bem simples e tem apresentado bons resultados quando aplicados a problemas reais (ZUO *et al.*, 2015; MAO *et al.*, 2015; BØHLING *et al.*, 2014). Este algoritmo utiliza um conjunto de dados de treinamento para determinar a categoria de exemplos desconhecidos. Quando um exemplo desconhecido é apresentado, um conjunto de dados similares é recuperado do conjunto de treinamento e utilizado para definir a categoria do exemplo desconhecido. O algoritmo utiliza também um parâmetro k para determinar a quantidade de exemplos do conjunto de treinamento que deve ser recuperada

para verificar sua similaridade com o exemplo desconhecido.

Apesar de simples, este algoritmo apresenta alguns problemas que devem ser tratados para que possa ser utilizado em aplicações cujos dados são provenientes de um fluxo. Um deles é o alto consumo de recursos computacionais para realizar a classificação de um novo exemplo. Para utilizar este método é preciso armazenar em memória todas as instâncias de treino, pois todas elas são utilizadas no processo de classificação de um novo exemplo. O número de instâncias geradas pelo fluxo pode ser grande e por consequência a quantidade de memória pode ser insuficiente para armazenar todos estes dados. O processo de classificação de um novo exemplo deve comparar este exemplo com todo o conjunto de treinamento, diferente de outros métodos que geram modelos baseados nestes dados e utilizam este modelo para classificar novas instâncias. Portanto o algoritmo necessita de uma grande quantidade de tempo para realizar o processamento de um novo exemplo.

Algumas versões do algoritmo *k*-vizinho mais próximo foram propostas no intuito de contornar o consumo excessivo de recursos computacionais (HART, 1968; GATES, 1972; DASARATHY, 1980; AHA; KIBLER; ALBERT, 1991). A essência destas versões é selecionar e armazenar apenas as instâncias selecionadas para realizar a classificação de um novo exemplo. Nestes trabalhos é demonstrado que a versão alterada do algoritmo *k*-vizinho mais próximo pode reduzir a quantidade de memória e o tempo de processamento necessário para realizar a classificação, sem que haja uma grande perda no desempenho do algoritmo, ou seja, sem que haja uma grande perda na acurácia de classificação.

A ideia principal do algoritmo *k*-vizinho mais próximo é atribuir a um exemplo desconhecido a mesma classificação do seu exemplo vizinho que pertence a um conjunto de amostras conhecidas previamente. De forma genérica, para classificar um exemplo x em uma categoria y é preciso somente de um conjunto de exemplos corretamente classificados $\{(x_1, y_1), \dots, (x_n, y_n)\}$. Considera-se que todos os exemplos sejam definidos através de atributos no espaço d -dimensional (\mathbb{R}^d), caso contrário não seria possível verificar a verossimilhança entre os exemplos. Para verificar a dissimilaridade entre os exemplos é utilizada, frequentemente, a distância euclidiana, de acordo com a seguinte fórmula.

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^d (x_{ir} - x_{jr})^2} \quad (2.1)$$

onde, x_i e x_j são os exemplos a serem verificados a dissimilaridade. O valor d representa a dimensionalidade dos exemplos e x_{ir} e x_{jr} são os respectivos valores de cada exemplo na r -ésima dimensão.

Um exemplo de treino x_i será considerado vizinho mais próximo do exemplo desconhecido x , se a sua dissimilaridade for a menor entre todos os exemplos de treino. Ou seja, se $d(x_i, x) \leq d(x_j, x)$, $\forall j \in \{1, 2, \dots, n\}$ e $j \neq i$.

O algoritmo recebe como parâmetro um conjunto de exemplos de treino *database*, representado pela lista $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$. O valor y_i representa a classe correta do i -ésimo exemplo de treino e pode assumir qualquer valor da lista de classes $\{C_1, C_2, \dots, C_{n\text{class}}\}$. O valor $n\text{class}$ representa a quantidade de classes do problema. Também recebe um exemplo desconhecido x , na qual se deseja inferir uma categoria dentre as existentes para o problema proposto. Por fim, o algoritmo recebe como parâmetro o número k que determina a quantidade de vizinhos serão utilizadas para o cálculo da categoria a ser atribuída ao exemplo desconhecido. Com isso, pode-se definir o Algoritmo 1.

Algoritmo 1: Algoritmo K -Vizinhos mais Próximos Simples (BATISTA, 2003)

Entrada: *database* – conjunto corretamente classificado,
 x – objeto desconhecido,
 k – número de vizinhos mais próximo
Saída: y_{nn} – classe a ser atribuída ao objeto x

- 1 **início**
- 2 Identifica os k vizinhos mais próximos de x em *database*;
- 3
$$y_{nn} = \arg \max_{c \in C} \sum_{i=1}^k \varphi(c, \text{classe}(x_i)) \quad (2.2)$$
- 4 onde, $\varphi(a, b) = 1$, se $a = b$; e $\varphi(a, b) = 0$, caso contrário;

O algoritmo retorna a classe y_{nn} mais frequente entre os k exemplos vizinhos a x . A quantidade de vizinhos mais próximos a ser utilizado está diretamente relacionada ao problema a ser aplicado o algoritmo, dependendo também da quantidade de exemplos de treino disponíveis. Para problemas de classe quantitativa, ou seja, classes que são representadas por valores reais, é preciso alterar a Equação 3 que determina y_{nn} , pela Equação 2.3 a seguir.

$$y_{nn} = \frac{\sum_{i=1}^k \text{classe}(x_i)}{k} \quad (2.3)$$

onde, $\text{class}(x_i)$ representa o valor real da classe do exemplo x_i . O algoritmo neste caso retorna a média dos k vizinhos mais próximos de x e atribui esta média ao exemplo desconhecido.

Neste trabalho tem-se o interesse em investigar métodos baseados no algoritmo k -vizinhos mais próximos aplicados a problemas de classe qualitativa.

2.5 *Space Saving*

O método *Space Saving* foi proposto na tentativa de realizar a contagem de palavras em um fluxo de dados (METWALLY; AGRAWAL; ABBADI, 2005). Neste método é definida uma quantidade limite de itens diferentes a serem contados no fluxo de dados. Então, na medida em que os dados são gerados pelo fluxo, os contadores de cada item são incrementados e os itens

com menor ocorrência são removidos, liberando espaço para os novos itens que são gerados pelo fluxo e não pertence à lista de contagem de elementos.

O *Space Saving* é um algoritmo para contagem de elementos associado à estrutura de dados *Stream Summary*. Este método foi proposto por Metwally, Agrawal e Abbadi (2005) para contemplar duas funcionalidades. A primeira delas se trata de estimar com precisão a frequência dos elementos significantes¹ em um fluxo de dados. Já a segunda de armazenar estes termos frequentes, e suas respectivas frequências, em uma estrutura que estará sempre ordenada. Esta técnica é eficiente e exata quando aplicada a problemas com alfabeto pequeno. Quando aplicado em problemas com alfabeto maiores este algoritmo é eficiente considerando o espaço dos exemplos. Este método é capaz de identificar os top- k elementos, assim como os elementos mais frequentes com baixa taxa de erro.

Formalmente, dado um alfabeto A , um *elemento frequente*, E_i , é um elemento cuja frequência, F_i , em um fluxo de dados S de um tamanho N , excede o limiar especificado pelo usuário de ϕN , onde $0 \leq \phi \leq 1$. Além disso, um elemento E_i pertence aos *elementos top- k* , que são os k elementos garantidamente com as maiores frequências, se a sua frequência, F_i , subtraindo um valor de *threshold*, ε_i , exceder o mesmo limiar especificado pelo usuário, $F_i - \varepsilon_i > \phi N$.

A ideia principal em torno do algoritmo *Space Saving* é manter informações parciais de interesse, isto é, monitorar apenas os m elementos de interesse em um fluxo de dados. Os contadores são atualizados de tal forma que seja capaz de estimar com precisão a frequência dos elementos significantes, sendo que é utilizada uma estrutura de dados compacta que mantém os elementos ordenados por suas respectivas frequências. Em uma situação ideal, qualquer elemento de interesse, E_i , com *rank* i , que tenha ocorrido F_i vezes no fluxo, deve ser acomodado no i -ésimo contador. O contador na i -ésima posição na estrutura de dado será simbolizado por $count_i$ e estima a frequência f_i do elemento e_i .

O algoritmo opera de forma simples. Se um elemento e for gerado pelo fluxo de dados e este elemento já tenha sido monitorado, o seu contador é incrementado. Caso e não tenha sido monitorado, a este será dado o benefício de fazer parte da lista dos elementos monitorado pelo algoritmo. Assim este elemento substituirá o elemento e_m , que apresenta o menor contador entre os elementos monitorados, $count_m = min$, no instante em que e é monitorado. Ao novo contador do elemento e , $count_m$, será atribuído o valor $min + 1$. Para cada elemento monitorado e_i , será mantido um valor *threshold*, ε_i , que representa o valor do contador do elemento no qual o elemento e_i substituiu na estrutura, ou seja, representa o valor min antes da inclusão do novo elemento. Esta informação é utilizada para verificar se o elemento mais frequente está garantidamente entre os top- k elementos mais frequentes monitorados no fluxo de dados. O algoritmo está descrito em Algoritmo 2.

Para implementar este algoritmo de forma eficaz, é necessário uma estrutura de dado

¹ Os elementos significantes são elementos de interesse que podem ser obtidos através de consultas significativas sobre o top- k elementos ou elementos frequentes.

Algoritmo 2: Algoritmo Space-Saving (METWALLY; AGRAWAL; ABBADI, 2005)

Entrada: m contadores, S fluxo de dados

1 **início**

```

2   para cada elemento  $e$  em  $S$  faça
3     se  $e$  foi monitorado então
4       incrementa o contador de  $e$ ;
5     senão
6       faça  $e_m$  ser o elemento com menor número de acessos,  $min$ ;
7       substitua  $e_m$  por  $e$ ;
8       incrementa  $count_m$ ;
9       atribua a  $\varepsilon_m$  o valor  $min$ ;

```

que seja capaz de incrementar os contadores de forma rápida e eficaz sem violar sua ordem e que permita recuperar os elementos em tempo constante. Baseado nisso, Metwally, Agrawal e Abbadi (2005) propuseram a estrutura de dados *Stream Summary*.

Na estrutura *Stream Summary*, todos elementos com mesmo contador são agrupados juntos em uma lista encadeada. Todos eles apresentam um ponteiro para o seu respectivo contador. Todos os elementos ligados a um contador tem o mesmo valor de contador. Todo contador possui um ponteiro para um elemento da lista de elementos associados a este contador. Os contadores são mantidos em uma lista duplamente encadeada e ordenada por seus valores. Inicialmente todos os elementos monitorados são vazios e estão ligados a um único contador com valor igual a zero. Os elementos podem ser armazenados em uma tabela *hash* para que o custo de acesso seja constante e não seja muito alto, ou em uma memória associativa para que o custo de acesso seja constante mesmo no pior caso. A estrutura *Stream Summary* pode ser percorrida sequencialmente como uma lista ordenada, desde que a lista dos contadores esteja ordenada. Assim com esta estrutura é possível identificar os elementos mais e menos significativos em um fluxo de dados, ou seja, o termo mais e menos frequente respectivamente.

Na Figura 2 é possível verificar o comportamento da estrutura *Stream Summary* com relação a inclusão de novos elementos na estrutura. Quando o contador de um elemento é incrementado, é verificado se o próximo contador possui o novo valor do contador deste elemento. Se o valor deste contador é igual ao novo valor do contador do elemento, então este elemento é removido da lista do seu antigo contador e é inserido na lista de elementos do novo contador. Caso contrário, um novo contador é criado com o valor do novo contador do elemento e este contador é adicionado à lista de contadores da estrutura na posição correta. Em seguida o elemento é adicionado a lista de elementos deste novo contador. O contador antigo que não possui nenhum elemento associado a ele é removido da estrutura.

Por exemplo, assuma que $m = 2$ e $A = \{X, Y, Z\}$. A situação da estrutura *Stream Summary*, após a ocorrência do fluxo $S = X, Y$, está descrita na Figura 2(a), depois de os elementos observados serem associados ao contador de valor 1. Quando ocorre outro elemento Y , um novo

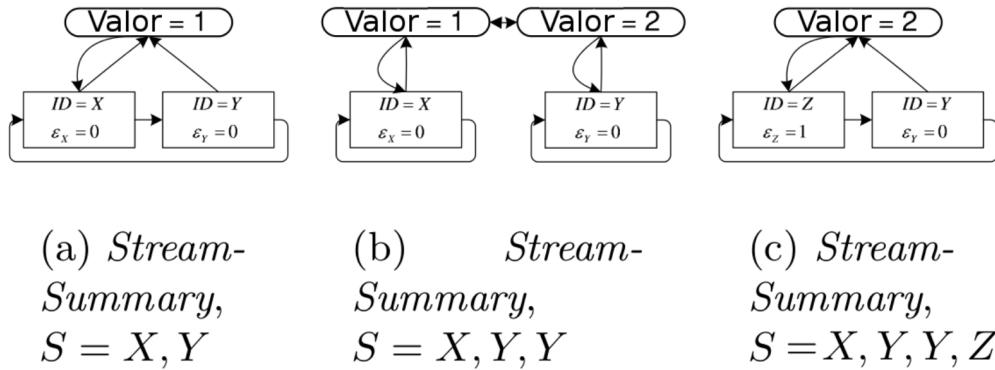


Figura 2 – Exemplo de atualização da estrutura *Stream Summary* pelo algoritmo *Space Saving* (METWALLY; AGRAWAL; ABBADI, 2005)

contador é criado com o valor 2, e Y é removido do contador de valor 1 e é inserido na lista de elementos do contador de valor 2, Figura 2(b). Quando ocorre Z , o elemento com o menor contador, X , será substituído por Z . Então o valor do contador de X é associado ao *threshold* de Z , $\varepsilon_z = 1$, e contador de Z é incrementado, $count_z = 2$. O contador de valor 1 não terá nenhum elemento associado a ele, então ele é removido da estrutura. O *Stream-Summary* final, após o fluxo $S = X, Y, Y, Z$ é mostrado na Figura 2(c).

2.6 Considerações Finais

Em muitas aplicações reais os dados são gerados por um fluxo contínuo e potencialmente infinito. Estes dados são gerados em sequência e em grande quantidade, e por esta razão não é possível armazena-los em memória. Um algoritmo de aprendizado de máquina, quando aplicado em aplicações com estas características, é preciso tratar estas restrições de tempo e espaço impostas pelo fluxo de dados. Uma classe de algoritmos capaz de obter bons resultados neste cenário são os *algoritmos anytime*. Neste projeto foi proposto um novo método de desempate para ser utilizado em conjunto com o algoritmo *anytime* baseado em instância proposto por Ueno *et al.* (2006). No Capítulo 3 pode ser encontrado os detalhes deste método e sobre as propriedades do algoritmo *anytime*.

Os dados gerados pelo fluxo geralmente sofrem mudanças ao longo do tempo. As origens destas mudanças podem ocorrer por vários motivos e o algoritmo de aprendizado de máquina quando aplicados neste contexto devem ser capazes de identificar e tratar estas mudanças. Existem vários trabalhos que categorizam estas mudanças para auxiliar a extração de conhecimento em tais aplicações. As mudanças podem ser divididas em apenas 3 categorias (ZLIOBAITE, 2009), onde as categorias são heterogêneas e duas diferentes mudanças, mesmo apresentando características distintas, são categorizadas em uma mesma categoria. Ou as mudanças podem ser categorizadas em várias categorias baseados em vários critérios (MINKU; WHITE; YAO, 2010), onde as categorias são homogêneas e as mudanças podem ser melhor categorizadas de acordo com suas características. Neste projeto foi proposto um método capaz de tratar, ao mesmo tempo, as

restrições impostas por um fluxo de dados, utilizando o algoritmo *anytime*, e as alterações que os dados sofrem ao longo do tempo, utilizando um algoritmo *incremental* que é capaz de atualizar o modelo de aprendizado utilizado pelo algoritmo de aprendizado de máquina. Os detalhes deste método pode ser encontrado no Capítulo 4.

O método de classificação baseado em instância é um método simples e que utiliza todos os exemplos de treino para realizar a classificação de itens desconhecidos. Ou seja, este método não gera nenhum modelo de aprendizado a partir dos dados. Ele apresenta bons resultados quando utilizado em problemas cujos dados podem ser armazenados e processados a qualquer momento. Porém para aplicações cujos dados são gerados em fluxo contínuo, este algoritmo em sua definição inicial não apresenta bons resultados. Necessitando de algumas alterações para ser utilizado em tais aplicações. Neste projeto o algoritmo baseado em instância foi utilizado em dois métodos distintos. Estes métodos serão descritos a seguir nos próximos capítulos.

O método *Space Saving* foi proposto para realizar a contagem de palavras em um fluxo de dados. Com este método é possível identificar os elementos mais frequentes em um fluxo de dados, assim como os elementos que pertencem ao *top-k* dos elementos garantidamente mais frequentes no fluxo. Ele utiliza de uma estrutura capaz de armazenar as palavras em ordem crescente pelo número de ocorrências no fluxo. Esta estrutura armazena também informações utilizadas para verificar se um determinado elemento pertence à lista dos elementos garantidamente mais frequentes. Este método foi adaptado para ser utilizado com conjunto com o algoritmo *anytime* baseado em instâncias, onde ele auxilia na atualização dos exemplos de treino utilizados pelo algoritmo. Além disso, ele também mantém uma lista ordenada destes exemplos de treino que será utilizada durante a classificação do algoritmo *anytime*. Mais detalhes desta proposta podem ser encontrados no Capítulo 4.



CLASSIFICAÇÃO ANYTIME

3.1 Considerações iniciais

Em muitos problemas do mundo real a classificação deve ser realizada sob restrições de recursos computacionais. Por exemplo, para classificar instâncias oriundas de um fluxo de dados com taxa de ocorrência variável (SHAH; KRISHNASWAMY; GABER, 2005; AGGARWAL *et al.*, 2004), o tempo para retornar o rótulo de uma nova instância pode variar consideravelmente. Por exemplo, em um estudo que foi realizado com um conjunto de dados de classificação de insetos (SOUZA; SILVA; BATISTA, 2013), foi verificado que o tempo entre dois eventos consecutivos variava de 0.01 segundos até 40 minutos.

De acordo com Shieh e Keogh (2010), a classificação de fluxos de dados é frequentemente mais difícil do que a classificação em lote porque o algoritmo deve ser executado sensível ao tempo. Classificadores em lote geralmente apresentam tempo de classificação constante ou que não pode ser ajustado pelo usuário. Desta maneira, estes algoritmos não são adequados para problemas de fluxo de dados, pois o tempo disponível para rotular um novo evento pode não ser suficiente para realizar a classificação. Por consequência alguns eventos do fluxo podem não ser classificados.

Para desenvolver sistemas que sejam capazes de classificar eventos oriundos de fluxo de dados, é necessário entender como projetar algoritmos eficientes com restrição de tempo. Isso ocorre pelo fato de o número de eventos que o sistema deve processar variar muito e o intervalo de tempo em que os eventos ocorrem também. Então o tempo disponível para classificar um evento é variado e dependente do instante em que o evento seguinte ocorre. Ou seja, classificação de fluxo de dados com alta variação de ocorrência de eventos deve fazer o melhor uso do tempo disponível para rotular um evento.

Algoritmos de classificação em lote carecem de um mecanismo para fornecer um resultado intermediário antes de finalizar o seu processamento. Por isso são incapazes de classificar

eventos separados por um intervalo de tempo pequeno.

Por outro lado, algoritmos que apresentam características *anytime* (GRASS; ZILBERSTEIN, 1996) tem demonstrado bons resultados quando aplicados a problemas de classificação em fluxo de dados (ESMEIR; MARKOVITCH, 2005; HUI; YANG; WEBB, 2009; KRANEN; SEIDL, 2009; LIN *et al.*, 2004; UENO *et al.*, 2006; YANG *et al.*, 2007). Classificadores *anytime* são capazes de fornecer uma classificação com diferentes tempos de processamento, sendo que a qualidade da resposta melhora com a quantidade de tempo de processamento. Mais concretamente, um classificador *anytime*, depois de um curto período de inicialização, pode sempre ser interrompido para retornar algum resultado intermediário. Essa flexibilidade no tempo de resposta permite que algoritmos *anytime* possam ser utilizados com sucesso em problemas de classificação de fluxo de dados com taxa de ocorrência de eventos variável.

Pesquisadores têm proposto formas de adaptar classificadores em lote para que possam ser aplicados a problemas de fluxo de dados. Alguns classificadores adaptados para o contexto *anytime* são: técnicas *boosting* (MYERS *et al.*, 2000), árvores de decisão (ESMEIR; MARKOVITCH, 2005), Máquinas de Suporte Vetorial (DECOSTE, 2002), *k*-vizinhos mais próximos (UENO *et al.*, 2006) e redes bayesianas (LIU; WELLMAN, 1996; HULTEN; DOMINGOS, 2002). Neste trabalho tem-se interesse em investigar variações do algoritmo *k*-vizinhos mais próximos.

3.2 Características dos Algoritmos *Anytime*

O termo *anytime* surgiu na década de 80 com o trabalho de Dean e Boddy (1988) sobre planejamento com dependência de tempo. Para realizar o planejamento, o tempo disponível para retornar o resultado para um determinado evento pode variar significativamente. Por isso, existe a necessidade de elaborar algoritmos capazes de disponibilizar uma resposta utilizando o tempo disponível. Esta classe de algoritmos foi chamada de *algoritmos anytime*.

As características mais importantes desses algoritmos são que (*i*) eles podem ser interrompidos a qualquer momento e podem voltar a ser executados com um custo insignificante, (*ii*) eles podem ser finalizados a qualquer momento e retornar alguma resposta e (*iii*) a resposta retornada melhora em função do tempo. As duas últimas características que realmente distinguem os algoritmos *anytime* dos demais algoritmos tradicionais (DEAN; BODDY, 1988). De uma forma geral, as características que os algoritmos considerados *anytime* apresentam são (GRASS; ZILBERSTEIN, 1996; ZILBERSTEIN; RUSSELL, 1995):

- *Qualidade mensurável*: a qualidade do resultado aproximado pode ser medida com precisão, desde que o resultado correto possa ser determinado;
- *Qualidade reconhecida*: a qualidade do resultado pode ser determinada em tempo de execução (com tempo constante). Em alguns casos a qualidade pode ser medida, mas não

pode ser reconhecida (por exemplo, problema de otimização combinatória);

- *Monotonicidade*: a qualidade da resposta é uma função não decrescente do tempo e da qualidade da entrada. Quando a qualidade da resposta é reconhecida, o algoritmo pode garantir a monotonicidade simplesmente retornando o melhor resultado, ao invés de retornar o último resultado gerado;
- *Consistência*: a qualidade do resultado está relacionada ao tempo e a qualidade da entrada. Em geral, algoritmos não garantem uma qualidade da resposta retornada para uma dada quantidade de tempo. Um classificador *anytime* produz resultados melhores quanto mais tempo for concebido para ser computado;
- *Retornos decrescentes*: a taxa de crescimento da qualidade da resposta melhora consideravelmente no início da execução do algoritmo e diminui ao longo do tempo;
- *Interruptibilidade*: depois de um curto período de inicialização, o algoritmo pode ser parado e um resultado intermediário pode ser retornado a qualquer momento até a finalização de sua execução;
- *Preemptividade*: após a interrupção do algoritmo, ele pode ser retomado para melhorar a qualidade do resultado sem sobrecarga significativa.

A interruptibilidade é a característica que permite aos algoritmos *anytime* fornecer uma resposta tão logo um novo evento ocorra. Por outro lado, a preemptividade pode não ser necessária nesse contexto, dado que fluxos de dados podem ter uma alta taxa de chegada de novos exemplos, o que não permitiria retomar o processamento de um exemplo já classificado.

A Figura 3 ilustra a relação esperada entre a qualidade do resultado e o tempo de execução em um algoritmo *anytime*. Logo nos primeiros instantes da execução, a qualidade da resposta de classificação está próxima da melhor resposta. E com mais tempo, a qualidade da resposta chega próxima à de classificadores com tempo ilimitado para a classificação (ZILBERSTEIN, 1996; UENO *et al.*, 2006). Esta flexibilidade é vantajosa quando o tempo de classificação disponível não é conhecido inicialmente.

Devido à sua utilidade em problemas do mundo real, os algoritmos *anytime* têm sido extensivamente estudados e têm encontrado aplicações em um número diversificado de domínios. Aplicações que vão de planejamento em tempo real de jogos de estratégia (BUTT; JOHANSSON, 2009) até o agrupamento de séries temporais (LIN *et al.*, 2004).

3.3 O algoritmo k -Vizinhos mais Próximos *anytime*

Um dos métodos criados para adaptar o algoritmo k -vizinhos mais próximos na versão *anytime* foi proposto por Ueno *et al.* (2006). Durante sua pesquisa do estado da arte, foi

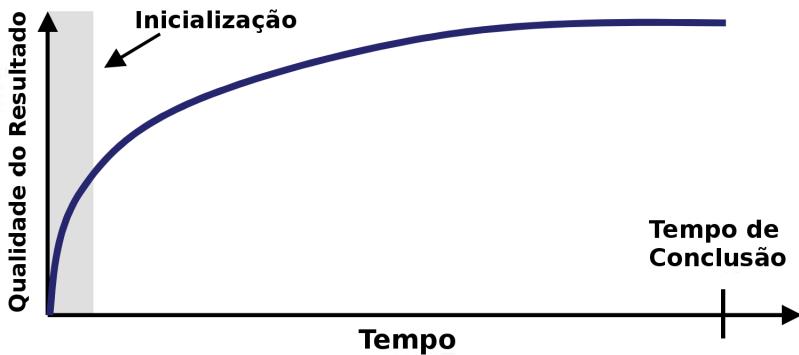


Figura 3 – Algoritmos *anytime* são interrompíveis depois da inicialização. Este gráfico mostra o crescimento esperado da qualidade da resposta com a adição de tempo computacional (SHIEH; KEOGH, 2010)

notado que, em algumas situações, ordenar os dados de treinamento apresenta bons resultados (WEBB *et al.*, 2005; GRUMBERG; LIVNE; MARKOVITCH, 2003). Porém, esta técnica havia sido aplicada apenas à árvore de decisão (GRUMBERG; LIVNE; MARKOVITCH, 2003) e classificador Bayes (WEBB *et al.*, 2005). E nenhum método havia sido reportado para os vizinhos mais próximos.

Ueno *et al.* (2006) propõe um *framework* genérico para a versão *anytime* do algoritmo *k*-vizinhos mais próximos. Esse *framework* permite especificar diversas heurísticas de ordenação de exemplos de treinamento. Essas heurísticas podem ser ajustadas ao problema que se deseja aplicar este *framework*. Eles propuseram uma heurística genérica para organizar os índices dos dados de treinamento para ser utilizada pelo algoritmo *k*-vizinhos mais próximos *anytime*.

3.3.1 Framework *k*-Vizinhos mais Próximos *Anytime*

Considere m o tamanho do conjunto de treinamento o qual será referido por *Database*. É possível acessar as características do i -ésimo exemplo de treinamento com *Database.objeto*(i), da mesma forma que é possível acessar o rótulo que o exemplo pertence com *Database.rotulo*(i). As instâncias da base de dados podem ser qualquer objeto que se tenha interesse em realizar a classificação. Por exemplo, bases de dados tradicionais, sequência de caracteres, grafos, séries temporais, entre outras.

Assuma que *Indice* é uma permutação de números inteiros que vai de 1 ate m . Lembrando que m é o tamanho da *Database*. Também assuma que O é um objeto desconhecido na qual se deseja classificar usando a versão *anytime* do algoritmo vizinhos mais próximos, utilizando *Database* como dados de treinamento do algoritmo. Dada esta notação, o Algoritmo 3 mostra a proposta do algoritmo *k*-vizinhos mais próximos *anytime*.

Nas dez primeiras linhas do algoritmo o novo objeto O é comparado com 1 exemplo de treinamento referente a cada classe do problema na tentativa de encontrar a classe do exemplo mais próximo. Durante a execução dessas linhas o algoritmo não pode ser interrompido, sendo esse período conhecido como o período de inicialização do algoritmo *anytime*. Após este breve

Algoritmo 3: Vizinhos mais Próximos Anytime

Entrada: Database, Indice, O
Saída: melhor_resposta

```

1 início
2   melhor_valor =  $\infty$ ;
3   melhor_resposta = indefinido;
4   para  $p = 1$  até numero_de_classes(Database) faz
5      $D = \text{distancia}(\text{Database}.objeto(\text{Index}[p]), O);$ 
6     se  $D < \text{melhor\_valor}$  então
7        $\text{melhor\_valor} = D;$ 
8        $\text{melhor\_resposta} = \text{Database}.rotulo(\text{Index}[p]);$ 
9
10    print('A partir de agora o algoritmo pode ser interrompido');
11     $p = \text{numero\_de\_classes}(Database) + 1;$ 
12    enquanto usuario_nao_interrompeu  $E p < \max(\text{Index})$  faz
13       $D = \text{distancia}(\text{Database}.objeto(\text{Index}[p]), O);$ 
14      se  $D < \text{melhor\_valor}$  então
15         $\text{melhor\_valor} = D;$ 
16         $\text{melhor\_resposta} = \text{Database}.rotulo(\text{Index}[p]);$ 
17       $p = p + 1;$ 
18      usuario_nao_interrompeu = teste_interrupcao_usuario();

```

período de inicialização, o algoritmo pode então ser interrompido a qualquer momento.

Nas linhas restantes do algoritmo, o objeto O será comparado com o restante da base de dados até o algoritmo ser interrompido ou até ser comparado com todos os exemplos da base. Em cada passo poderá ser atualizado o rótulo do novo objeto, refletindo o resultado intermediário da classificação. A ordem em que os exemplos de treinamento serão usados para comparar com o objeto será a ordem dada pela variável *Indice*. Mais detalhes sobre esta variável encontram-se na Seção 3.3.2.

A função de distância que foi utilizada nas linhas 5 e 14 do Algoritmo 3 pode ser implementada como qualquer medida de distância presente na literatura, por exemplo distância euclidiana, distância de Manhattan, correlação, entre outras. A complexidade do tempo e espaço também não é alterada com relação ao algoritmo tradicional, apesar de que se deve realizar o pré-processamento para obter a variável *Indice*.

3.3.2 Determinação do Índice

Baseado no algoritmo apresentado anteriormente, produzir a versão *anytime* do algoritmo k -vizinhos mais próximos se resume a encontrar a ordem de processamento de exemplo dada pela variável *Indice* que fornece bons resultados para uma aplicação. Porém existem diversas maneiras de realizar esta ordenação. Além disso, para cada base de dados de um determinado problema existe uma melhor ordem dos dados para esta variável.

Algoritmo 4 mostra de forma genérica como os dados são ordenados. Ele inicia criando um vetor para armazenar os índices dos exemplos de treinamento (linha 2). Em seguida é

feita uma busca pela posição do “pior” exemplo de treinamento (linha 4), e esta posição é armazenada na variável *Indice* criada anteriormente (linha 5). Para garantir que o exemplo não seja selecionado novamente na busca, é atribuído valor vazio na sua posição da base de treinamento (linha 6). Essa busca pelo pior exemplo é feita até que sejam preenchidas todas as posições restantes no vetor de índices.

Algoritmo 4: Ordenação dos Índices

Entrada: Base_de_Treinamento
Saída: Indice

```

1 início
2   // m é o tamanho da Base_de_Treinamento
3   Indice = cria_vetor_tamanho(m);
4   para p = 0 até m - (1 + numero_de_classes(Base_de_Treinamento)) faca
5     posicao = encontra_posicao_pior_exemplo(Base_de_Treinamento);
6     Indice[m - p] = posicao;
7     Base_de_Treinamento (posicao) = null;

```

Entende-se por “pior” exemplo aquela instância que menos representa a classe na qual pertence. Neste ponto podem-se explorar várias possibilidades de definir a representatividade da instância para a classe que esta pertence. Também permite que qualquer um use esse algoritmo definindo apenas como encontrar o “pior” exemplo na base de treinamento.

Os primeiros exemplos na lista de índices ordenada devem obrigatoriamente pertencer a diferentes classes do problema na qual se deseja realizar a classificação. Isso ocorre devido ao período de inicialização do algoritmo *anytime*, que deve garantir que todos os rótulos possíveis do problema tenham sido analisados. Assim pode-se obter uma boa resposta em um curto período de tempo.

O algoritmo busca pelo “pior” exemplo e não pelo “melhor” exemplo. Pode-se perfeitamente realizar a busca pelo sentido inverso (busca pelo “melhor” exemplo). Porém há certa dificuldade em definir as “melhores” instâncias nos primeiros estágios do algoritmo. Por esta razão os autores decidiram realizar a busca pelo “pior” exemplo.

3.3.3 Encontrando o pior exemplo

Para finalizar a definição do algoritmo *anytime* é preciso definir como encontrar os “piores” exemplos da base para criar a lista ordenada de índices a ser utilizada pelo algoritmo *k*-vizinhos mais próximos. Ueno *et al.* (2006) propuseram o algoritmo *SimpleRank* para ordenar as instâncias em classificação *anytime*.

A ideia do algoritmo **SimpleRank** é atribuir a cada instância um *rank* de acordo com a sua contribuição para a classificação correta das demais instâncias do conjunto de treinamento. Esse algoritmo é baseado no método *Naive-Rank* (XI *et al.*, 2006) que inicialmente foi proposto para aumentar o desempenho da classificação do 1-Vizinho mais próximo com distância *Dynamic*

Time Warping (DTW). Embora o método *Naive-Rank* tenha sido proposto para a distância DTW, o *SimpleRank* pode ser utilizado com qualquer função de distância.

O método *SimpleRank* inicia realizando a classificação de 1 instância do conjunto de treinamento utilizando o 1-vizinho mais próximo sobre as instâncias remanescentes. Após a classificação, é verificado se o vizinho mais próximo da instância analisada é *inimigo* ou *associado*. O vizinho da instância é considerado *inimigo* se seu rótulo é diferente do rótulo da instância, e é considerado *associado* caso contrário.

Com essa informação para todo o conjunto de treinamento o *rank* de uma instância x pode ser calculado de acordo com a Equação 3.1.

$$\text{rank}(x) = \sum_j \begin{cases} 1 & , \text{if } \text{classe}(x) = \text{classe}(x_j); \\ -2/(\text{num_de_classes} - 1) & , \text{caso contrário.} \end{cases} \quad (3.1)$$

onde x é a instância a ser calculado o *rank* e x_j são as instâncias que tem x como vizinho mais próximo.

Em outras palavras, x terá seu score incrementado em uma unidade para cada instância da qual x é associado. Por outro lado, x terá seu score decrescido de um fator constante dependente do número de classes para cada instância na qual x é inimiga. Esta função geralmente produz muitos valores iguais para o *rank* dos piores exemplos. Ueno *et al.* (2006) define um critério de desempate no qual os exemplos são ordenados de acordo com a distância do seu vizinho mais próximo de mesma classe. Os exemplos mais distantes serão adicionados ao final da lista de índices.

Ueno *et al.* (2006) escolheram o peso -2 para os vizinhos *inimigos* da instância, pois estes devem ser punidos severamente no cálculo do *rank* a fim de identificar os piores exemplos (os que menos representam a classe ao qual pertencem). Este peso funciona bem para classificação binária, onde se tem apenas dois rótulos. Para problemas multiclasses, os autores obtiveram bons resultados utilizando o peso $-2/(\text{num_de_classes} - 1)$.

Depois de realizado o cálculo do *rank* das instâncias é verificado qual é o pior exemplo de treinamento e este é adicionado ao final da lista dos índices que será utilizada no algoritmo *anytime*. Este exemplo é removido da lista e continua-se buscando os piores exemplos até que todos os exemplos tenham sido adicionados ao índice.

3.4 O Algoritmo *k*-Vizinhos mais Proximos *anytime* com diversidade

Apesar dos bons resultados demonstrados por Ueno *et al.* (2006) para o algoritmo *k*-vizinhos mais próximos *anytime*, este ainda possui algumas desvantagens. A primeira delas é que

vários exemplos da mesma classe podem aparecer em sequência no *ranking*. Assim, o método pode calcular a distância do exemplo a ser classificado com exemplos da mesma classe por um longo período. Desse modo, a estimativa da solução não é melhorada durante todo esse período.

Outra dificuldade relacionada a esse algoritmo é o fato de que o método *SimpleRank* comumente enfrenta o problema do empate entre os valores do *rank* de dois ou mais exemplos distintos. O método adotado pelos autores para corrigir esses empates foi ordenar as instâncias com mesmo *ranking* de acordo com a distância do seu vizinho mais próximo de mesma classe.

Note que o *rank* calculado a partir da equação 3.1 pode gerar diversos empates. Isso acontece pelo fato de que o número de vezes que um exemplo é considerado *amigo* ou *inimigo* depende do número (limitado) de exemplos em que ele é vizinho mais próximo. Para comprovar essa suposição, foi executado um experimento simples com o *SimpleRank*. Nesse experimento foi utilizado o conjunto de dados *MAGIC Gamma Telescope*, o qual possuía 17,118 exemplos no conjunto de treinamento. O método *SimpleRank* sem desempate gerou 14 *ranks* diferentes. Em outras palavras, os 17,118 exemplos foram agrupados em apenas 14 grupos. Na média, cada exemplo obteve o mesmo *rank* que outros 1222 exemplos do conjunto de treinamento.

O critério de desempate proposto originalmente para resolver este problema é ordenar os exemplos de treinamento empatados de acordo com a distância de cada um deles ao seu vizinho mais próximo de mesma classe. Porém, esse critério de desempate acaba por agrupar exemplos de regiões muito próximas no espaço de exemplos em posições próximas no *ranking* dos exemplos de treino.

Outro problema relacionado à estratégia de desempate original é o fato que, caso uma classe possua uma distribuição em que os exemplos sejam mais espalhados no espaço do que outras, o algoritmo beneficia a classe mais compacta.

A Figura 4 exemplifica ambos os fatos. Nessa figura são exibidos graficamente os dez primeiros exemplos do *ranking* obtido pelo método *SimpleRank* original e pelo método *SimpleRank com DiversityTiebreak* (LEMES *et al.*, 2014), que será descrito logo a seguir. Na Figura 4(a), pode-se ver que os exemplos escolhidos pelo critério de desempate do método original se concentram em uma região limitada do espaço, o que não acontece com o novo critério de desempate (Figura 4(b)). Quando uma classe possui os exemplos mais compactos no espaço do que a outra, o método original prioriza tal classe (Figura 4(c)), diferentemente do método *SimpleRank com DiversityTiebreak* (Figura 4(d)).

As estratégias de ordenação e desempate do método *SimpleRank* não garantem uma cobertura uniforme do espaço onde estão definidos os exemplos. Porém, esta cobertura uniforme do espaço pode ser vantajosa para o algoritmo *k-Vizinhos mais próximos anytime*, uma vez que será mais provável encontrar um vizinho mais próximo que é verdadeiramente próximo a uma instância desconhecida. Entretanto, ainda é importante considerar a contribuição para a classificação correta de exemplos desconhecidos que cada exemplo de treino possui, assim como

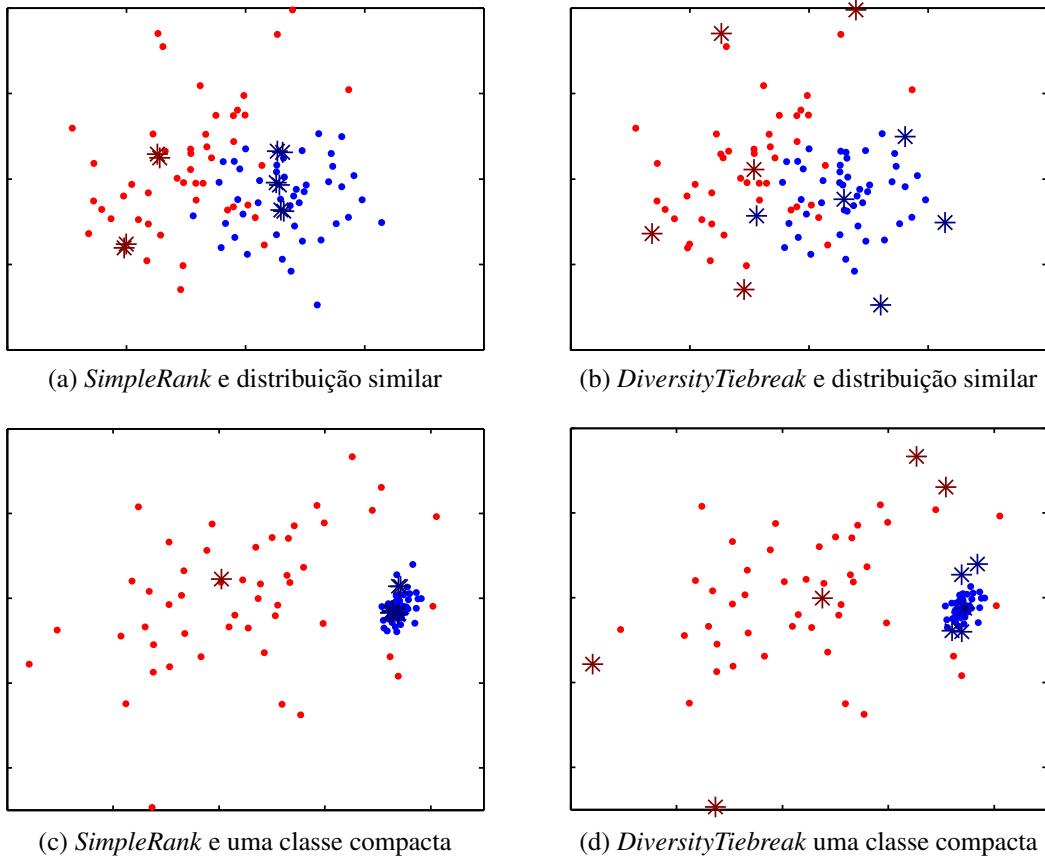


Figura 4 – Os primeiros dez exemplos que obtiveram o mesmo valor de *rank* (marcados com ‘*’) desempatados pelo *SimpleRank* original – (a) e (c) – e de acordo com o método *DiversityTiebreak* – (b) e (d) – para um conjunto de dados sintético com distribuição espacial similar para ambas as classes – (a) e (b) – e com uma classe mais compacta que a outra – (c) e (b).

é implementado pelo *SimpleRank*. Desta forma, foi proposto o método de ordenação baseado no *SimpleRank*, porém utilizando o algoritmo *DiversityTieBreak* como critério de desempate (LEMES *et al.*, 2014).

De forma sucinta, esta proposta mantém a estratégia de cálculo do valor do *rank* utilizada pelo *SimpleRank*, porém utilizando um novo método como critério de desempate. A estratégia de desempate utilizada garante uma melhor cobertura do espaço onde estão definidos os exemplos. Ela foi aplicada considerando duas etapas. A primeira é a ordenação das instâncias de treino considerando este novo critério, *DiversityTiebreak*, descrito no Algoritmo 5.

Este algoritmo define diferentes listas ordenadas para cada classe c_i . O primeiro elemento da cada lista é o vizinho mais próximo do centroide da classe. Antes de realizar a busca por este exemplo, na linha 4 é calculado o centroide da classe c_i considerando todos os exemplos que possui o rótulo da classe c_i . Em seguida, na linha 5, é verificado qual é o índice do exemplo mais próximo do centroide da classe. O valor deste índice é adicionado na primeira posição da lista de índices da classe atual, linha 6.

Os índices das próximas instâncias a ser adicionada a lista ordenada da classe atual são

Algoritmo 5: Diversity Tiebreak

Entrada: *database* conjunto de treino
Saída: *diversity_list* – lista ordenada de índices das instâncias

```

1 início
2   para cada Classe  $c_i$  faz
3      $index\_list_{c_i} = <>;$ 
4      $k = centroid(c_i);$ 
5      $idx\_add =$  índice do vizinho mais próximo de  $k$  que pertence a  $c_i;$ 
6      $index\_list_{c_i} = index\_list_{c_i} + < idx\_add >;$ 
7     enquanto  $|index\_list_{c_i}| < |instances(c_i)|$  faz
8        $idx\_add =$  instância mais distante entre todas instâncias em  $index\_list_{c_i};$ 
9        $index\_list_{c_i} = index\_list_{c_i} + < idx\_add >;$ 
10    $diversity\_list =$  junção das listas  $index\_list_{c_i}$  de cada classe  $c_i;$ 
11   return diversity_list;

```

os índices das instâncias que pertencem à classe atual e possuem as maiores distâncias entre todas as instâncias já adicionadas na lista antes. Em outras palavras, o índice da instância com a maior soma das distâncias para as instâncias que já tiveram seus índices adicionados antes na lista ordenada deve ser o próximo índice a ser adicionada a lista ordenada da classe. Este processo é repetido até que todos os exemplos pertencentes a classe c_i sejam adicionados a lista ordenada desta classe.

Na última etapa deste algoritmo, na linha 11, é realizada a junção das listas de índices ordenadas de cada classe originando uma lista de índices de todo o conjunto de dados de treino. O método mais natural de realizar esta junção é selecionar um índice de cada lista alternadamente e adicionar este valor no final da lista ordenada de todo o conjunto de dados. Mas, este método pode ser inapropriado quando aplicado a problemas com classes desbalanceadas. Para contornar esta questão, neste algoritmo foi utilizado um método de junção que considera a probabilidade a priori de cada classe. Ou seja, se a classe a tem N exemplos e a classe b tem $2N$, será selecionado um exemplo da classe a e dois da classe b em cada etapa da junção.

Depois de criado a lista de desempate baseado na diversidade, é realizada a segunda etapa da proposta. Nesta etapa, é calculado o valor do *rank* das instâncias pelo *SimpleRank*. As instâncias que obtiverem o mesmo valor de *rank* serão desempatadas considerando a sua posição obtida pela ordenação do *DiversityTiebreak*. Assim, a lista de índice ordenada utilizada pelo algoritmo *k-Vizinhos mais próximos anytime* considera a importância de cada exemplo na classificação correta de novas instâncias e também a diversidade dos exemplos no espaço nos casos de empate.

3.5 Análise Experimental

No intuito de verificar a eficácia do método *SimpleRank com DiversityTiebreak*, foi realizado uma abrangente avaliação experimental. O desempenho do algoritmo *k-vizinhos mais próximos anytime* foi comparada usando, além deste método, outros métodos de ordenação: **Aleatório e SimpleRank**.

O método *Aleatório* é o método mais simples de ordenar as instâncias de treino. Para gerar a lista ordenada de índices das instâncias, é selecionado um exemplo de cada classe para compor as primeiras posições da lista que será utilizada pelo algoritmo *k-vizinhos mais próximos anytime* durante o período de inicialização. Após definir as primeiras posições da lista, os demais exemplos são adicionados à lista em ordem completamente aleatória.

Essa estratégia, apesar de simples, não leva em consideração a qualidade dos exemplos escolhidos para a construção da lista de índices ordenada. Desse modo, é esperado que o ganho de acurácia não seja tão grande nos primeiros passos da execução do algoritmo *anytime*.

O método *SimpleRank* foi descrito na Seção 3.3. Sua implementação foi baseada na descrição do trabalho em Shieh e Keogh (2010).

3.5.1 Configuração experimental

Foram utilizados 15 conjuntos de dados de *benchmark* para avaliar e comparar os resultados entre os métodos analisados. Os conjuntos de dados podem ser encontrados no *UCI Machine Learning Repository* (LICHMAN, 2013). Os conjuntos de dados utilizados neste trabalho possuem diferentes características, resumidas na Tabela 2.

Tabela 2 – Breve descrição dos conjuntos de dados de *benchmark* utilizados na avaliação experimental

Conjunto de dados	Nº de Atributos	Nº de Classes	Nº de Exemplos
EEG Eye State	15	2	14980
Image Segmentation	19	7	2310
Ionosphere	34	2	351
Letter Recognition	16	26	20000
MAGIC Gamma Telescope	11	2	19020
Mfeat - Factors	216	10	2000
Mfeat - Fourier	76	10	2000
Mfeat - Karhunen	64	10	2000
Mfeat - Morphological	6	10	2000
Mfeat - Zernike	47	10	2000
Optical Recognition of Digits	64	10	5620
Page Blocks Classification	10	5	5473
Pen-Based Recognition of Digits	16	10	10992
Spambase	57	2	4601
Waveform Generator (Version 2)	40	3	5000

Foi utilizado também o método de validação cruzada de tamanho 10 (*10-fold cross validation*) para estimar a performance de classificação dos métodos nos conjuntos de dados selecionados. Os atributos foram normalizados linearmente para o intervalo entre 0 e 1. Este é um procedimento padrão na classificação utilizando o algoritmo *k*-vizinhos mais próximos para tornar a função de distância invariante a escala dos atributos.

No procedimento de avaliação, o algoritmo *k*-vizinhos mais próximos *anytime* é interrompido depois de processar cada instância de treino seguindo a ordem determinada pela lista ordenada de índices da instâncias de treino. Esta lista é definida por cada um dos métodos analisados. Em cada interrupção é verificado o desempenho no conjunto de teste assumindo que o algoritmo *anytime* teve tempo para processar somente as primeiras j instâncias na lista ordenada. O valor inicial de j é o valor do número de classes e ele varia até o total de número de exemplos presentes no conjunto de treinamento.

O período de inicialização é o tempo necessário para processar o número de instâncias equivalente ao número de classes presentes no conjunto de dados. Isso ocorre, pois nas primeiras posições da lista ordenada utilizada pelo algoritmo está presente uma instância pertencente a cada classe possível no conjunto de dados. Cada método de ordenação deve garantir esta condição e este é o mínimo de informação que deve ser fornecido para o bom funcionamento do algoritmo *anytime*. O período de inicialização é muito pequeno e pode ser negligenciado na maioria das situações práticas, uma vez que o número de classes é frequentemente muito pequeno comparado ao número de instâncias no conjunto de treinamento.

Tal procedimento de avaliação caracteriza o desempenho dos métodos analisados em uma lista de valores de acurácia para todas as possibilidades de interrupção. Como são discutidos nas seções seguintes, estes valores de acurácia podem ser graficamente comparada por meio de um gráfico da curva de desempenho. Esta curva de desempenho pode ser resumida em um único valor, calculando a área sob a curva.

3.5.2 Empates obtidos pelo *k*-Vizinhos mais próximos *anytime*

A primeira etapa da avaliação experimental é realizar um experimento para calcular o número de empates obtidos pelo método *SimpleRank*. Obviamente, se o número de empates for insignificante, a proposta de um novo critério de desempate para o *SimpleRank* se torna desnecessária.

Em uma das seções anteriores foi antecipado um dos resultados obtidos com um experimento similar sobre uma base de dados específica. Nesta seção são descritos todos os resultados obtidos em todos os conjuntos de dados de *benchmark* utilizados.

A Tabela 3 mostra uma análise resumida do número de empates para cada conjunto de dados utilizados no trabalho. A coluna “Média de score distintos” representa a média de números de diferentes valores de *rank* obtidos através dos conjunto de treinamento. A coluna “Média

de instâncias com o mesmo *score*” representa o número médio de exemplos que obtiveram o mesmo *score* com o *SimpleRank* sem desempate.

Tabela 3 – Análise do número de empates obtidos pelo método *SimpleRank*

Conjunto de dados	Média de <i>score</i> distintos	Média de instâncias com o mesmo <i>score</i>
EEG Eye State	11.70	1152.30
Image Segmentation	11.90	174.71
Ionosphere	12.30	25.68
Letter Recognition	30.10	598.00
MAGIC Gamma Telescope	14.40	1188.80
Mfeat - Factors	15.20	118.42
Mfeat - Fourier	20.70	86.96
Mfeat - Karhunen	13.70	131.39
Mfeat - Morphological	12.30	146.34
Mfeat - Zernike	16.10	111.80
Optical Recognition of Digits	14.10	358.72
Page Blocks Classification	13.70	359.54
Pen-Based Recognition of Digits	12.10	817.58
Spambase	21.10	196.25
Waveform Generator (Version 2)	24.40	184.43

Pode-se observar que mesmo em problemas com poucos exemplos de treinamento, o *SimpleRank* gerou uma quantidade significativa de valores de *rank* repetidos. Este experimento evidencia a dificuldade que o *SimpleRank* enfrenta em relação ao número de valores iguais no cálculo do *rank* e em sua ordenação a ser utilizada pelo algoritmo *anytime*. Por isso existe a necessidade de um bom critério de desempate.

3.5.3 Avaliação

Foram utilizadas neste trabalho duas formas de avaliar os resultados. A primeira é um gráfico do desempenho versus o tempo de processamento. A principal medida de desempenho adotada foi a *acurácia* por se tratar de uma medida que trata os erros e acertos de cada método de uma forma igualitária. Devido ao fato da necessidade de caracterizar o desempenho dos métodos para todas as interrupções possíveis, foi escolhido interromper o algoritmo *k-vizinhos mais próximos anytime* após o processamento de cada instância de treino, como foi discutido na seção anterior.

Este procedimento gera um gráfico que avalia a acurácia de classificação com relação ao número de exemplos de treino analisados antes de uma interrupção. Idealmente, esta análise deveria ser feita com relação ao tempo de processamento. Entretanto, o tempo de processamento é completamente dependente da otimização do código e da configuração de *hardware* utilizada durante o processo de avaliação, sendo difícil realizar uma comparação justa entre os diferentes métodos. Portanto, é um procedimento padrão na literatura substituir o tempo de processamento

do gráfico por outra informação que permite de uma forma genérica avaliar o desempenho do algoritmo *anytime*. Neste trabalho foi utilizado o número de exemplos processados, semelhante ao que foi utilizado por (UENO *et al.*, 2006).

Como uma segunda forma de avaliação, foi utilizado também um único valor numérico que representa o desempenho geral de cada método. Este valor tipicamente fornece menos informação que o gráfico, entretanto é conveniente resumir os resultados de todos os conjuntos de dados em uma única tabela. Então, para o cálculo deste valor foi utilizado a área sob a curva de desempenho do algoritmo *anytime*. Formalmente esta área é definida pela Equação 3.2.

$$AUC_{anytime} = \frac{1}{n - \#classes} \sum_{i=\#classes}^n acc_i \quad (3.2)$$

na qual acc_i é o valor da acurácia obtida quando o algoritmo foi interrompido após o processamento do i -ésimo exemplo e n é o número total de exemplos de treino. Devido ao período de inicialização do algoritmo *anytime*, o valor inicial de i é o número de classes.

3.5.4 Resultados

A Tabela 4 apresenta as áreas sob a curva de performance obtidas pelo algoritmo *k-vizinhos mais próximos anytime* utilizando os três métodos de ordenação para os 15 conjuntos de dados utilizados.

Tabela 4 – Área sobre a curva de performance do algoritmo *knn anytime* utilizando três diferentes métodos

Conjunto de dados	Aleatório	SimpleRank	SimpleRank com DiversityTiebreak
EEG Eye State	0.7986	0.7992	0.8028
Image Segmentation	0.9388	0.9417	0.9441
Ionosphere	0.8444	0.8721	0.8821
Letter Recognition	0.9106	0.9083	0.9173
MAGIC Gamma Telescope	0.7972	0.8177	0.8173
Mfeat - Factors	0.9399	0.9414	0.9472
Mfeat - Fourier	0.7684	0.7834	0.7883
Mfeat - Karhunen	0.9288	0.9297	0.9382
Mfeat - Morphological	0.6749	0.6842	0.6824
Mfeat - Zernike	0.7737	0.7857	0.7920
Optical Recognition of Digits	0.9767	0.9773	0.9806
Page Blocks Classification	0.9480	0.9518	0.9538
Pen-Based Recognition of Digits	0.9857	0.9852	0.9872
Spambase	0.8791	0.8810	0.8825
Waveform Generator (Versão 2)	0.7178	0.7585	0.7556
Vitórias	0	3	12

Os resultados indicam a eficácia do método proposto. Quando analisado a área sob a curva de performance do algoritmo *anytime* para os diferentes métodos de ordenação, nota-se

que o método *SimpleRank com DiversityTiebreak* obteve melhores resultados em 12 dos 15 conjuntos de dados analisados.

Para avaliar a eficácia do método proposto, foi realizado um teste de hipótese. Foi escolhido o teste *Friedman com pós-teste de Li* e 95% de intervalo de confiança, como sugerido em (TRAWIŃSKI *et al.*, 2012). O teste rejeitou a hipótese nula que o *SimpleRank com DiversityTiebreak* tem performance similar com os demais métodos analisados.

A seguir serão apresentados alguns gráficos da curva de desempenho que representam os resultados. A Figura 5 mostra o resultado obtido pelo método obtido aplicado ao conjunto de dados *Mfeat - Karhunen*. Nesta figura, pode-se observar que a curva de desempenho está de acordo com o resultado que se espera para um algoritmo *anytime*. Mais especificamente, o ganho da acurácia no início da execução é bem notável, especialmente para o método *SimpleRank com DiversityTiebreak*. Pode-se notar que este método fornece um resultado mais consistente para a acurácia, com pequena variabilidade comparado com o *SimpleRank*.

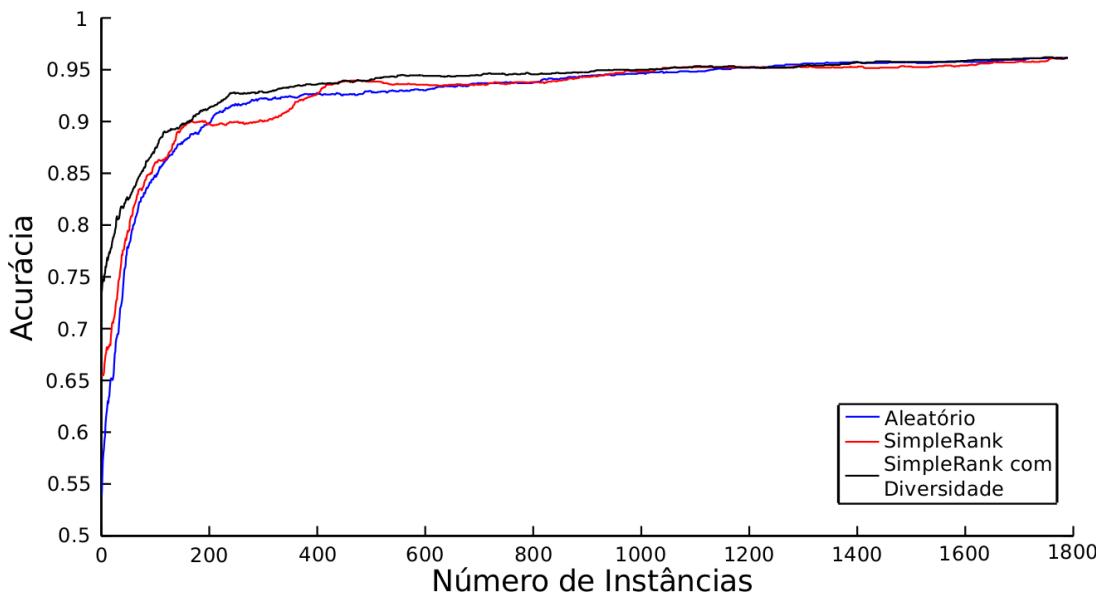


Figura 5 – Resultado obtido pelo conjunto de dados *Mfeat - Karhunen*

Embora a diferença de desempenho entre os métodos não seja muito grande, ela é consistente durante toda a curva. Em geral, o método *SimpleRank com DiversityTiebreak* apresenta resultados com menos variabilidade que os demais métodos. Por exemplo, apenas em alguns momentos que o *SimpleRank* apresenta performance melhor que o método *Aleatório*. Isso raramente acontece com o método proposto.

A Figura 6 apresenta o resultado para o conjunto de dados *Optical Recognition of Digits*. Neste conjunto de dados a diferença de desempenho é muito pequena. Isto ocorre pelo fato de que o método *Aleatório* já alcança bons resultados neste conjunto, deixando pouco espaço para uma performance melhor. Note como a curva é muito íngreme mesmo durante o processamento das primeiras instâncias da lista ordenada. E mesmo assim, o *SimpleRank com DiversityTieBreak*

alcançou consistentemente um melhor resultado que os outros dois métodos.

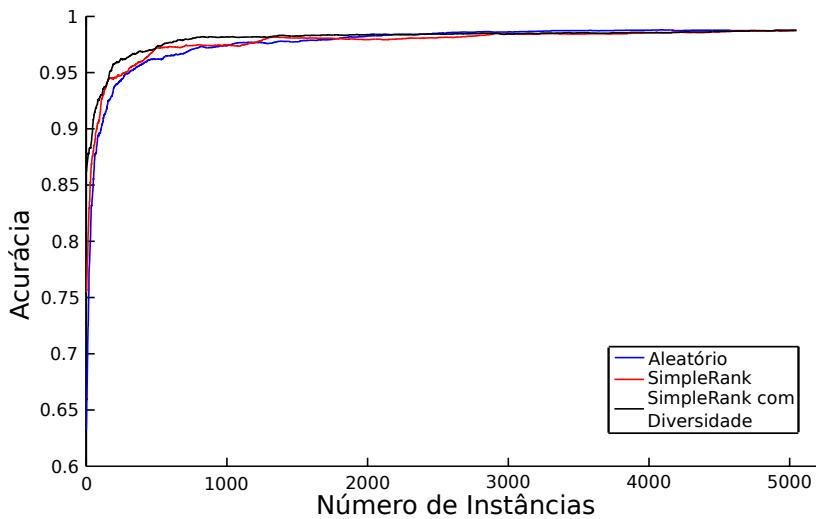


Figura 6 – Resultado obtido pelo conjunto de dados *Optical Recognition of Digits*

A Figura 7 apresenta os resultados obtidos para o conjunto de dados *Ionosphere*. Este é um conjunto de dados pequeno na qual a diferença de desempenho é muito notável. Mais uma vez, *SimpleRank com DiversityTiebreak* apresenta melhores resultados entre os métodos comparados para todo o número de instâncias de treinamento que o conjunto de dados apresenta.

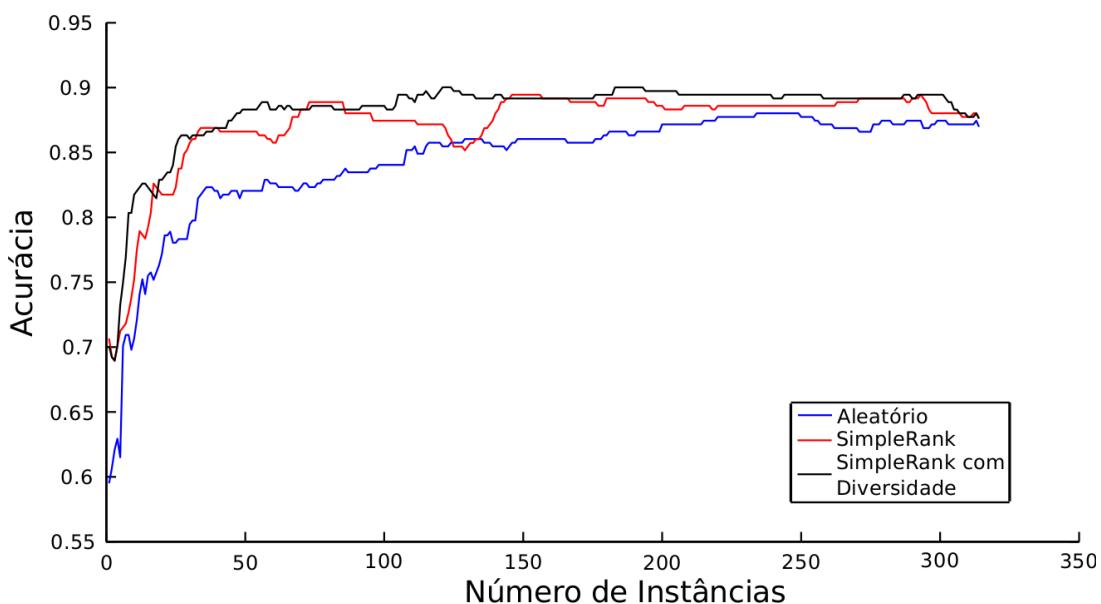


Figura 7 – Resultado obtido pelo conjunto de dados *Ionosphere*

3.6 Considerações Finais

Realizar tarefas de aprendizado de máquina em fluxo de dados tem se tornado muito importante em várias áreas de pesquisa. Isso ocorre pelo fato de muitas aplicações que se deseja

extrair conhecimento automatizado apresentarem propriedades características de fluxo de dados. Classificadores tradicionais não apresentam boas soluções quando aplicados em problemas com estas características. Por outro lado, os classificadores *anytime* tem apresentado bons resultados em tais problemas devido a sua flexibilidade em fornecer uma resposta intermediária e aumentar a qualidade da resposta ao longo do tempo.

Neste capítulo foi apresentado uma versão *anytime* de um algoritmo simples e bem conhecido, o algoritmo *k*-vizinhos mais próximos. Devido a sua simplicidade de implementação e funcionamento, é fácil definir uma versão *anytime* para este algoritmo. Sendo necessário apenas definir a melhor ordem de verificação da base de treino e utilizar esta ordem durante o processo de classificação de instâncias desconhecidas.

O algoritmo proposto por Ueno *et al.* (2006) é simples e apresentou bons resultados comparados a outros métodos. Mesmo assim, este método apresentou alguns pontos na qual poderia ser melhorado. Como o grande número de empates obtidos ao determinar a melhor ordem de verificação do conjunto de treino. Também vale lembrar o privilégio dado a classes compactas pelo critério de desempate adotado pelos autores que propuseram esta versão *anytime* do algoritmo *k*-vizinhos mais próximos.

Na tentativa de melhorar o algoritmo proposto anteriormente, Lemes *et al.* (2014) propuseram um método de desempate para ser utilizado como parte do algoritmo *k*-vizinhos mais próximo *anytime*. Tal método considera a diversidade das instâncias no espaço entre os exemplos de mesma classe como critério de desempate. Este método realiza uma melhor cobertura do espaço de exemplos de treino, uma vez que ele prioriza os exemplos mais distantes na construção da lista de índices ordenada (que é utilizada pelo algoritmo *anytime*). Além disso, o procedimento de junção das listas de diversidade por classe considera a distribuição das classes ao montar a lista de índices final, evitando que os exemplos de mesma classe se posicionem em longas sequências, o que levaria ao algoritmo *anytime* verificar por um longo período exemplos de uma única classe.

Foi apresentado também que o método de desempate proposto é capaz de melhorar consistentemente a performance do algoritmo *SimpleRank*. Considerando os 15 conjuntos de dados utilizados para avaliar o método proposto, o *SimpleRank com DiversityTiebreak* alcançou melhores resultados em 12 conjuntos de dados quando comparado com a performance do algoritmo *SimpleRank*. De fato, o teste de hipótese demonstrou a superioridade do método.

Os algoritmos *anytime* apresentados aqui alcançam bons resultados quando aplicados a fluxos de dados estacionário. A principal característica de tais fluxos é que os dados não sofrem mudança de conceito. Porém existe uma classe de aplicações na qual o algoritmo deve realizar a classificação de um fluxo de dados não estacionário. Neste caso o algoritmo deve ser capaz de tratar as mudanças de conceito que os dados sofrem. Os algoritmos *anytime* não possuem a habilidade de tratar as mudanças de conceito. Por outro lado, algoritmos incrementais alcançam bons resultados quando aplicados a um fluxo de dados não estacionário. Estes algoritmos são

capazes de construir modelos de aprendizado na medida que os dados são gerados pelo fluxo de dados. No próximo capítulo será proposto um classificador, capaz de possuir as características dos algoritmos *anytime* e *incremental*. Além disso será apresentado os resultados obtidos por este classificador quando aplicados a um fluxo de dados com mudança de conceito.



CLASSIFICAÇÃO INCREMENTAL

4.1 Considerações iniciais

Aprendizado de máquina explora o estudo e a construção de algoritmos que podem realizar o aprendizado e a predição sobre dados (KOHAVI; PROVOST, 1998). Em muitos destes algoritmos assume-se implicitamente que os exemplos de treino estejam disponíveis a priori para que o processo de aprendizado possa ser realizado. Porém em muitos problemas reais os dados de treino não estão totalmente disponíveis antes de iniciar o processo de aprendizado. Geralmente estes dados são gerados ao longo do tempo através de um fluxo de dados. Neste contexto, é preciso que o algoritmo analise os dados de forma *online* e de uma maneira incremental.

Realizar o aprendizado sobre um fluxo de dados requer que o algoritmo trate algumas restrições, por exemplo, de tempo de processamento e espaço de armazenamento. Em diversas aplicações de fluxo de dados, um evento pode ocorrer a qualquer momento, sendo que o intervalo de tempo entre dois eventos pode ser altamente variado. Uma técnica de aprendizado de máquina deve ser capaz de utilizar o tempo disponível entre eventos para realizar o processamento do cada evento do fluxo de dados. A quantidade de dados gerados por um fluxo não é constante e em alguns casos este valor tende ao infinito. Assim é inviável armazenar todos os dados gerados em memória para realizar o aprendizado dos novos eventos que são gerados pelo fluxo.

Um fluxo de dados pode apresentar mudanças nas características dos dados. O algoritmo de aprendizado deve ter a habilidade de atualizar o seu modelo de aprendizado para obter sucesso neste contexto. Para isso, o algoritmo deve identificar um exemplo está desatualizado, e removê-lo do seu modelo. E deve incluir novos exemplos no modelo para que este fique consistente ao conceito atual.

Algoritmos incrementais são algoritmos capazes de tratar mudanças de conceito nos dados. Porém, estes algoritmos não são capazes de tratar a alta variabilidade de tempo entre um evento e outro. Assim alguns eventos de interesse podem não ser processados pelo sistema

de aprendizado. Algumas técnicas foram propostas para diminuir o tempo de obtenção de resposta por um algoritmo de aprendizado de máquina (LAW; ZANIOLO, 2005; BERINGER; HÜLLERMEIER, 2007a). Mesmo assim, o tempo entre dois eventos distintos pode ser menor que o tempo necessário que a técnica necessita pra obter uma resposta.

Por outro lado, o algoritmo *anytime* proposto por Ueno *et al.* (2006) é capaz de se adaptar ao tempo disponível entre eventos e prover uma classificação em tempo hábil para cada evento do fluxo. Entretanto atualizar o modelo de aprendizado utilizado por este algoritmo não é uma tarefa fácil. Um exemplo de treino pode influenciar o valor do *rank* de vários exemplos próximos de sua região do espaço. Para remover este exemplo seria preciso remover a influência que este exemplo causa no valor do *rank* de seus vizinhos mais próximos. Além disso, para incluir um novo exemplo de treino é preciso verificar quais exemplos no conjunto de treino este novo exemplo influencia. Ou seja, seria necessário realizar vários cálculos do valor do *rank* de vários exemplos novamente. Sendo que o tempo de processamento deste cálculo é alto, já que para cada exemplo de treino é preciso verificar em todo o conjunto quais exemplos são influenciados por este exemplo.

Um algoritmo que seja *anytime* e incremental seria o ideal a ser utilizado em aplicações que apresentam as características descritas anteriormente. Isso motivou a elaboração da proposta de um algoritmo que apresentasse as propriedades *anytime*, descritas no capítulo anterior, e ao mesmo tempo fosse um algoritmo incremental. O algoritmo utilizado nesta proposta foi o baseado em instância. Inicialmente foi definida uma função de ranqueamento de exemplos de treino a ser utilizada para ordenar as instâncias. Em seguida, foram elaboradas duas propostas para atualização do modelo de aprendizado utilizado pelo algoritmo, uma baseado no algoritmo *Space Saving*, e outra baseado em uma *Janela Deslizante*.

A seguir na Seção 4.2 é realizado uma revisão sobre algumas características que os algoritmos devem possuir para serem considerados incrementais. Na Seção 4.3 são descritos brevemente alguns algoritmos incrementais que são baseados em instâncias. Na Seção 4.4 é descrita a função de ranqueamento e as duas propostas de atualização do modelo de aprendizado. Uma análise experimental deste método é relatada na Seção 4.5. E por fim, temos uma consideração final na Seção 4.6.

4.2 Características algoritmos incrementais

Em geral, os algoritmos podem ser divididos em duas categorias: algoritmos *batch* (GANTER, 2010; OUTRATA; VYCHODIL, 2012) e algoritmos incrementais (MERWE; OBIEDKOV; KOURIE, 2004; KOURIE *et al.*, 2009). Nos algoritmos *batch* os modelos de aprendizado utilizados pelo algoritmo usualmente são construídos a partir de todos os dados. Já nos algoritmos incrementais o modelo de aprendizado é construído adicionando novos objetos (ou atributos) um por um na medida em que são apresentados ao algoritmo. Uma propriedade importante desta

última categoria de algoritmo é que qualquer informação acerca das características dos dados que ainda não foram processados permanece desconhecida pelo algoritmo até que o dado seja apresentado ao algoritmo (ZOU; ZHANG; LONG, 2015).

O termo incremental pode ser utilizado tanto para tarefas de aprendizado, assim como para algoritmos de aprendizado. Para conceituar esses termos Giraud-Carrier (2000) forneceu uma definição formal para tarefas incrementais e exemplificou algumas aplicações com estas características. Ele também forneceu uma definição para algoritmos incrementais e exemplificou alguns algoritmos que possuem esta propriedade. A primeira definição caracteriza o conceito incremental, quando aplicadas a tarefas de aprendizado.

Uma tarefa de aprendizado é incremental se os exemplos de treinamento usados para resolvê-la venham a estar disponíveis ao longo do tempo, geralmente um de cada vez (GIRAUD-CARRIER, 2000).

Se uma tarefa de aprendizado incremental pode esperar por um longo período de tempo a sua solução, então esta tarefa pode, a priori, ser considerada uma tarefa não incremental. Essa tarefa pode apresentar mudanças nas características dos dados, por consequência ela pode esperar para sempre antes de realizar o aprendizado em tais situações. Então, para realizar a tarefa de aprendizado incremental, é necessário assumir que não é possível esperar por muito tempo até a obtenção de uma solução, ou que este tempo de espera seja impraticável. Além disso, em muitas aplicações é inviável o tempo de espera para gerar uma grande quantidade de exemplos representativos, isso devido à mudanças no ambiente ou devido a geração lenta dos exemplos. A seguir alguns exemplos de tarefas de aprendizado incremental.

- *Modelagem de perfis de usuário.* O mapeamento de competências é o processo de identificação das competências chaves para uma organização, as tarefas e funções dentro dela. Esta é uma atividade essencial para uma organização, permitindo que ela direcione da melhor forma possível seus profissionais para as áreas em que suas competências serão melhor utilizadas (JANANI; GOMATHI, 2015).
- *Robótica.* Em tais aplicações o ambiente está em mudanças constantes e geralmente imprevisíveis, mesmo nos casos mais simples. Para obter sucesso, a tarefa deve reagir e adaptar de forma incremental aos estímulos do ambiente. Podendo ser aplicado a cenários onde os obstáculos estão parados ou movendo e também reagir num ambiente com vários veículos (HOY; MATVEEV; SAVKIN, 2015).
- *Agentes inteligentes.* São caracterizados por apresentar reatividade e proatividade. Incrementabilidade é inerente no aprendizado de agentes. Muitas empresas têm adotado agentes inteligentes, pois necessitam tratar as constantes transformações que a teoria e a prática de marketing vêm sofrendo através do aumento de sua complexidade devido à disponibilidade de informações, ao seu alto alcance e interações, e a alta velocidade das transações (KUMAR *et al.*, 2015).

Em síntese, uma tarefa de aprendizado incremental apresenta duas principais características. A primeira é a indisponibilidade dos exemplos de treino, sendo que eles ficam disponíveis ao longo do tempo, sendo gerado um exemplo por vez. A segunda é que o aprendizado pode precisar continuar (quase) indefinidamente.

O algoritmo de aprendizado incremental é definido a seguir.

Um algoritmo de aprendizado é incremental se para um dado conjunto de treinamento e_1, \dots, e_n , ele produz a sequencia de hipóteses h_0, h_1, \dots, h_n , tal que h_{i+1} depende somente de h_i e o exemplo atual e_i . (GIRAUD-CARRIER, 2000)

Alguns algoritmos se baseiam nesta definição para permitir que a próxima hipótese dependa somente da hipótese anterior e um pequeno conjunto de exemplos de treino, ao invés de um único exemplo. Outros algoritmos, como os baseados no teorema de *bayes*, utilizam mais fielmente esta definição, onde apenas o resultado da iteração atual é utilizado para determinar o valor da próxima iteração. De acordo com Giraud-Carrier (2000), por razões de coerência com as tarefas de aprendizado, o termo incremental é melhor aplicado para algoritmos. Alguns exemplos de algoritmos de aprendizado incremental são apresentados de forma resumida na Seção 4.3.

Muitos algoritmos conhecidos na literatura, tais como *Iterative Dichotomiser 3* (ID3) (QUINLAN, 1986), CN2 (CLARK; NIBLETT, 1989) e *back-propagation* (RUMELHART *et al.*, 1988) não são incrementais. Eles assumem que o conjunto de treino pode ser construído a priori e que os exemplos podem ser processados várias vezes e também que o aprendizado para por um tempo após todo o conjunto de treino ser processado.

De acordo com Giraud-Carrier (2000) existem duas propriedades principais de um algoritmo de aprendizado incremental. A primeira propriedade é que os exemplos não necessitam ser reprocessados pelo algoritmo. A segunda propriedade é que cada hipótese h_i pode ser considerada a melhor resposta para uma determinada aplicação. O algoritmo pode a qualquer momento, produzir respostas para uma consulta e a qualidade da resposta aumenta ao longo do tempo.

Vale observar que um algoritmo não incremental pode ser utilizado para resolver uma tarefa de aprendizado incremental. Porém, em tais algoritmos é preciso que os exemplos anteriores estejam disponíveis para viabilizar o aprendizado, através da construção do modelo de aprendizado utilizado pelo algoritmo. Ou seja, nestes algoritmos é preciso realizar o reprocessamento dos exemplos anteriores a cada novo exemplo que lhe for apresentado. Vale lembrar também que a incrementabilidade está incluída em muitas aplicações reais e que a forma mais natural de contemplar uma tarefa de aprendizado incremental é utilizando um algoritmo de aprendizado incremental.

4.3 O Algoritmo *k*-Vizinhos Mais Próximo Incremental

Em uma aplicação cujos dados são gerados por um fluxo, geralmente é produzida uma grande quantidade de dados (BERINGER; HüLLERMEIER, 2007b). Devido a esta grande quantidade de dados, nestas aplicações não é possível simplesmente armazenar os dados da maneira tradicional, através de um sistema de armazenamento de dados e após isso executar alguma tarefa de extração de conhecimento destes dados. Por isso, é importante que o processo de aprendizado, quando aplicado a um fluxo de dado, ocorra de maneira *online* com o intuito de garantir que os resultados estejam atualizados com os dados gerados pelo fluxo.

Outra consequência inerente no aprendizado em fluxo de dados é a necessidade de tratar eventuais mudanças de conceito nos dados (Seção 2.3). Estas mudanças ocorrem devido a fatores externos e fora de controle do método de aprendizado utilizado em tais aplicações. Para tratar estas mudanças é preciso que o modelo de aprendizado seja atualizado ao longo do tempo, extraíndo novos conhecimentos dos exemplos que são gerados pelo fluxo de dados. Muitos algoritmos conhecidos na literatura que utilizam modelos de aprendizado criados a partir dos exemplos, foram adaptados para contemplar esta necessidade de atualizar o modelo de aprendizado (UTGOFF, 1988; CLI; HARVEY; HUSBANDS, 1992; DIEHL; CAUWENBERGHS, 2003). Porém esta tarefa de atualização em tais algoritmos é muito custosa em termos de tempo de processamento e memória utilizada. Isso acontece pelo fato de seus modelos serem complexos e exigirem uma quantidade de informação adicional considerável para realizar a atualização de seu modelo.

Já o algoritmo *k*-vizinhos mais próximos apresenta propriedade natural de ser incremental, uma vez que para atualizar o modelo basta adicionar e/ou remover os exemplos gerados pelo fluxo de dados. Ou seja, realizar a atualização do modelo utilizado pelo algoritmo é uma tarefa simples e barata, em termos de recurso de máquina, quando comparado com os algoritmos que geram modelos de aprendizado a partir dos dados. Porém, comparado com os métodos baseados em modelos de aprendizado, o algoritmo vizinhos mais próximos apresenta alto custo computacional para fornecer uma resposta para exemplos desconhecidos. Este fato pode ser notado facilmente, uma vez que, para encontrar os *k* exemplos de treino vizinhos a um exemplo desconhecido, é preciso que o algoritmo compare este exemplo com todos os exemplos no conjunto de treino. Armazenar e utilizar os exemplos mais representativos, além de contribuir para a classificação correta dos exemplos desconhecidos, evita também comparações desnecessárias durante o processo de previsão, o que também pode evitar um gasto excessivo de recursos computacionais.

Mesmo apresentando alto custo para realizar a classificação de novos exemplos, este algoritmo tem sido utilizado com sucesso em várias aplicações de fluxo de dados. Isso acontece pelo fato de sua implementação ser simples e a facilidade de atualização de seu modelo de aprendizado, aliado aos bons resultados que este algoritmo apresenta. Com ele é possível reagir a uma mudança de conceito de forma flexível e também utilizar uma boa estratégia de adaptação. Manter os melhores exemplos observados ajuda a manter a descrição dos conceitos atuais e

também facilita a identificação de exemplos desatualizados.

Dos algoritmos incrementais conhecidos na literatura, o algoritmo *Instance-Based 3* (IB3) (AHA; KIBLER; ALBERT, 1991) é o mais simples. Ele utiliza como essência o algoritmo *k*-vizinhos mais próximo proposto por Cover e Hart (1967). Porém, nele os atributos são normalizados, as instâncias são processadas incrementalmente e existe tolerância para valores nulos. As instâncias classificadas erradas são adicionadas ao modelo de aprendizado. E ele utiliza um método para detecção e remoção de ruídos nos dados.

Outro algoritmo incremental conhecido é o *Floating Rough Approximation* (FLORA). Este algoritmo mantém uma janela de exemplos confiáveis armazenados em conjuntos positivos, negativos e mistos. O algoritmo FLORA é o mais simples nesta família de algoritmos e tenta manter um conjunto consistente de conceitos dentro de uma janela deslizante. O algoritmo FLORA 2 possui a habilidade de ajustar dinamicamente o tamanho da janela deslizante durante o processo de aprendizado. FLORA 3 armazena um conjunto de conceitos antigos que podem ser reutilizados caso estes conceitos deixem de existir na janela deslizante atual e volte a existir numa janela de exemplos no futuro. Por fim, o algoritmo FLORA 4 foi desenvolvido para tratar possíveis ruídos nos dados de entrada do algoritmo.

O algoritmo *Prediction Error Context Switching* (PECS) (SALGANICOFF, 1997) mescla as características de outros dois algoritmos, o TWF e o LWF. O algoritmo *Time-Windowed Forgetting* (TWF) permite realizar o aprendizado através de uma função que varia de acordo com o tempo. Porém, esta versão sofre alguns problemas quando a distribuição do espaço amostral do conjunto de aprendizado sofre variação no tempo. O *Locally-Weighted Forgetting* (LWF) corrige alguns aspectos negativos do algoritmo TWF, mantendo no conjunto de treinamento apenas os exemplos que possuem atributos similares aos exemplos subsequentes a ele. Entretanto, o LWF tem problema quando não existe variação dos exemplos. Ambas as versões foram propostas por Salganicoff (1997). O algoritmo PECS, usa como princípio o LWF, mas apenas desativa o exemplo que pode ser reutilizado futuramente.

4.4 O Algoritmo *k*-Vizinhos Mais Próximo *Anytime Incremental*

A versão do algoritmo *k*-vizinhos mais próximos *anytime* apresentado no Capítulo 3 não apresenta a propriedade incremental. Assume-se que os exemplos de treino estejam disponíveis a priori, sendo que a ordenação das instâncias é feita antes do algoritmo ser utilizado. Nesta versão do algoritmo, atualizar o modelo com a inclusão de um novo exemplo seria uma operação muito custosa devido a quantidade de operações a serem realizadas nesta tarefa. Para atualizar o modelo gerado por este método incluindo um novo exemplo de treino, além de realizar o cálculo do *rank* do novo exemplo, é preciso recalcular o *rank* de todos os exemplos da base de treino que são influenciados, direta ou indiretamente, pelo novo exemplo. Da mesma maneira que, para

remover um exemplo de treino é preciso também recalcular os *ranks* das instâncias para que este permaneça consistente em todo conjunto de treino.

Por outro lado, algoritmos incrementais possuem a habilidade de construir e atualizar o modelo de aprendizado durante o processo de aprendizado. A ideia de mesclar em uma única técnica as propriedades das duas classes de algoritmos motivou a elaborar a proposta deste trabalho. O objetivo desta proposta é encontrar um método capaz de possuir ao mesmo tempo as propriedades dos algoritmos incrementais e *anytime*. O primeiro passo desta proposta foi definir uma fórmula que possibilitasse o ranqueamento dos exemplos de treino. Em seguida foram definidas duas maneiras de organizar os exemplos de treino a serem utilizadas pelo algoritmo *anytime* incremental. A primeira é baseada no algoritmo *Space Saving* e a segunda é baseada em uma *Janela Deslizante*.

O algoritmo *anytime*, mencionado no capítulo anterior, utiliza a ideia de que os exemplos de treino que mais contribuem para a classificação correta dos exemplos desconhecidos, devem ser avaliados nos primeiros estágios do algoritmo *k*-vizinhos mais próximo. Estes exemplos melhor representam as classes a qual pertencem. Isso faz com que a performance do algoritmo seja maior nos primeiros estágios do processamento, propriedade importante no algoritmo *anytime*. O método *SimpleRank* realiza uma ordenação das instâncias através de uma fórmula que bonifica as instâncias que contribuem para a classificação correta de instâncias desconhecidas e penaliza aquelas instâncias que não contribuem.

Foi utilizada esta ideia para definir a fórmula a ser utilizada pelo algoritmo *anytime* incremental. A nova equação é capaz de garantir benefício aos exemplos que contribuem para a classificação correta, de tal forma que seja simples de ser calculada. Esta equação foi chamada de *CountingRank* e ela substitui a equação de ranqueamento utilizada pelo algoritmo *anytime* definido no capítulo anterior. Nesta fórmula é preciso apenas identificar o vizinho mais próximo de um exemplo de treino e incrementar o seu contador de vizinhos mais próximos. Caso o seu vizinho seja de mesma classe, o contador será incrementado em uma unidade. Caso contrário, o contador será decrementado em duas unidades, como é definido na Equação 4.1.

$$counter_t(x) = counter_{t-1}(x) + \begin{cases} 1 & , \text{if } classe(x) = classe(x_j); \\ -2 & , \text{caso contrário.} \end{cases} \quad (4.1)$$

onde x é a instância que terá o seu contador atualizado e x_j é a instância gerada pelo fluxo de dados que tem a instância x como o seu vizinho mais próximo. $counter_{t-1}(x)$ é o contador da instância x antes de x_j ser adicionado à base de treino e $counter_t(x)$ é o contador após a adição de x_j .

Outra vantagem desta fórmula é que não é preciso nenhum processo extra para identificar o vizinho mais próximo do exemplo desconhecido e assim incrementar o contador do vizinho. Quando o algoritmo *anytime* realiza a classificação de um exemplo desconhecido (x_j), ele identifica qual o seu vizinho mais próximo (x). Com esta informação é possível atualizar o

contador do exemplo de treino, x , utilizando a equação proposta aqui.

Para incrementar o contador de um exemplo de treino x_i seria preciso verificar se este exemplo é o mais próximo do exemplo desconhecido entre todos os exemplos do conjunto de treinamento. Porém, o processamento de um algoritmo *anytime* pode ser interrompido a qualquer momento. Assim no caso em que ocorre interrupção, não é possível comparar o exemplo desconhecido com todo o conjunto de treino. Se o exemplo de treino que é realmente mais próximo do exemplo desconhecido não for processado antes da interrupção, o seu contador não pode ser atualizado. Neste caso, os métodos propostos aqui atualizam o contador do exemplo de treino mais próximo do exemplo desconhecido antes de ocorrer a interrupção. Mesmo que este exemplo não seja realmente o mais próximo do exemplo gerado pelo fluxo.

Para que o algoritmo possa atualizar o contador dos exemplos de treino, é preciso que ele conheça a classe correta do exemplo desconhecido. Então, assume-se neste trabalho que, imediatamente após a classificação do exemplo gerado pelo fluxo, o seu verdadeiro rótulo passe a estar disponível. Essa é uma suposição comum em aplicações de classificação em que os dados são gerados por um fluxo (WIDMER; KUBAT, 1996; GAMA; RODRIGUES; SEBASTIÃO, 2009).

Os métodos propostos aqui utilizam uma estrutura que auxilia no armazenamento dos exemplos de treino. O algoritmo *Space Saving*, apresentado no Capítulo 2, utiliza uma estrutura de dados que mantém os elementos do fluxo ordenado, o *Stream Summary*. Além de manter ordenado, uma característica importante desta estrutura é a possibilidade de atualizar a ordenação dos elementos de forma fácil e rápida. A estrutura *Stream Summary* foi adaptada para ser utilizada nas propostas deste trabalho.

Na nova estrutura, os exemplos com o mesmo valor de contador são agrupados em uma lista duplamente encadeada circular. Com apenas um ponteiro é possível acessar o primeiro e o último elemento desta lista. Todo elemento possui um ponteiro para o seu contador. Atualizar o contador de um elemento é fácil, pois se pode ter acesso direto ao próximo contador que deve armazenar este elemento. Uma lista de elementos está vinculada a apenas um contador.

Os contadores são armazenados em uma lista duplamente encadeada, semelhante a estrutura original. Cada contador possui um ponteiro para o último item da lista de elementos vinculada a ele. Com esse ponteiro é possível percorrer a sua lista de elementos em ordem crescente ou decrescente.

A estrutura adaptada mantém um ponteiro para o primeiro e para o último contador da lista. O primeiro ponteiro é para facilitar o acesso ao último elemento da lista ordenada, considerando todos os elementos presentes na estrutura. O segundo é para facilitar o acesso ordenado de forma decrescente dos exemplos de treino presentes na estrutura. Este segundo ponteiro será utilizado pelo algoritmo *anytime* para recuperar os exemplos de treino presentes na estrutura, e assim realizar a classificação do exemplo desconhecido gerado pelo fluxo.

O identificador dos elementos na nova estrutura, é representado pelas informações do exemplo de treino, *lista de atributos* e *classe*. Os elementos são armazenados em uma tabela *hash* para possibilitar acesso de baixo custo aos exemplos de treino em eventuais leituras. Após o exemplo ser gerado pelo fluxo e classificado pelo algoritmo *anytime*, ele é incluído na estrutura. Um exemplo de treino será removido e este novo exemplo de treino é associado ao elemento do exemplo removido. A estrutura armazena no máximo m elementos que irão compor o conjunto de treinamento utilizado pelo algoritmo.

A Figura 8 mostra uma parte da situação da estrutura *Stream Summary* modificada. Em azul é exibido os contadores e em verde os elementos pertencentes ao contador de valor igual a 5. Em preto é mostrado as ligações entre os objetos presentes na estrutura. A direção das setas indica qual item possui o ponteiro para o objeto na qual está referindo. Os elementos ligados aos outros contadores estão organizados da mesma maneira que está exibido na figura. Os ponteiros *primeiro* e *último* permitem o acesso aos contadores e elementos da estrutura.

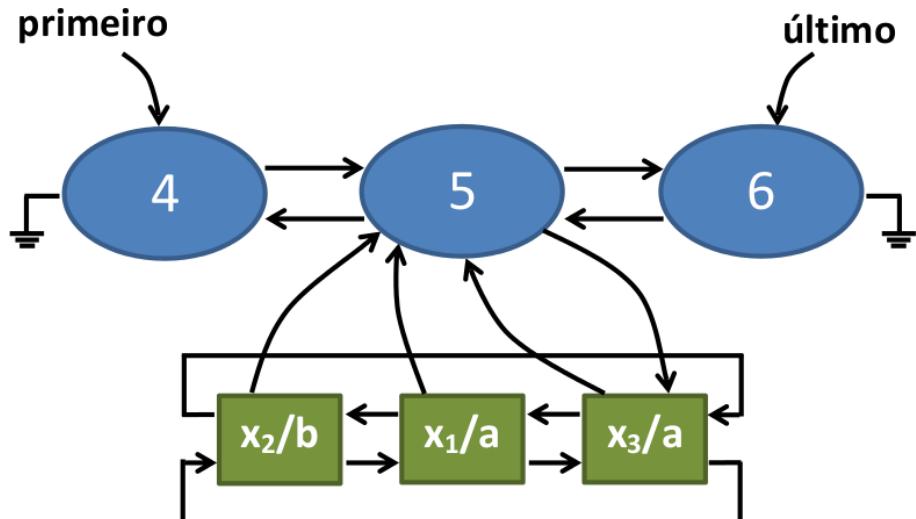


Figura 8 – Exemplo da estrutura *Stream Summary* modificada para ser utilizada pelo algoritmo k -vizinhos mais próximo *anytime* incremental.

Algoritmo 6 descreve o algoritmo k -vizinhos mais próximos *anytime* incremental em detalhes. O algoritmo k -vizinhos mais próximos *anytime* é executado utilizando os exemplos de treino armazenados na estrutura *Stream Summary* adaptada. O algoritmo retorna a instância x_{nn} mais próxima da nova instância x_e , rotulando a instância x_e com o mesmo rótulo de seu vizinho mais próximo.

Poucos instantes após a classificação, o algoritmo tem acesso à classe real da instância x_e . Se a classe da instância x_e for igual a classe do seu vizinho mais próximo x_{nn} , o contador referente à x_{nn} é incrementado em uma unidade. Isso indica que este elemento tem mais um vizinho mais próximo de mesma classe e assim considera-se que esta instância contribui para a classificação correta de instâncias desconhecidas. Então esta instância deve ser posicionada mais ao topo da lista de contadores da estrutura *Stream Summary* e deve ser utilizada nos estágios iniciais do algoritmo *anytime*.

Algoritmo 6: Space-Saving Anytime

Entrada: m contadores, S fluxo de dados

1 **início**

```

2   para cada instância  $x_e$  em  $S$  faça
3      $x_{nn} = algoritmo\_k - nn\_anytime(m, x_e);$ 
4     se classe( $x_{nn}$ ) == classe( $x_e$ ) então
5       incrementa em uma unidade o contador do exemplo  $x_{nn}$ ;
6     senão
7       decrementa em duas unidades o contador do exemplo  $x_{nn}$ ;
8     atualiza_Stream_Summary( $x_e$ );

```

Se a classe da instância x_e não for igual a classe da instância x_{nn} , o contador referente à x_{nn} é decrementado em duas unidades. Isso significa que este exemplo deve ser penalizado por não contribuir para a classificação correta de novas instâncias. Por isso, este exemplo deve ser posicionado mais ao final da lista ordenada pela estrutura adaptada. Com esta penalização é possível que o contador de alguns exemplos seja negativo, porém isso não afeta o comportamento do algoritmo.

Os contadores são atualizados de tal forma que seja possível estimar com precisão a quantidade de vizinhos mais próximos de um exemplo de treino. Qualquer exemplo de treino, x_i , que tenha NN_i número de vizinhos mais próximos, deve ser acomodado no i -ésimo contador, cujo valor será igual a NN_i .

O período de *setup* do algoritmo *anytime* é importante para garantir a primeira resposta parcial do algoritmo, sendo que o algoritmo pode ser interrompido após esta resposta estiver disponível. Por esta razão, é preciso garantir que nas primeiras posições da lista de exemplos de treino utilizada pelo algoritmo esteja presente um exemplo de treino de cada classe possível do problema em questão. No Algoritmo 6, isso é garantido mantendo uma tabela *hash* com uma instância de cada classe que apresenta o maior contador entre as instâncias de mesma classe. Quando o contador de certa instância é incrementado e o seu valor supera o valor do contador da instância de mesma classe na tabela *hash*, estas instâncias são substituídas. O algoritmo *anytime* utiliza primeiro os exemplos de treino presentes nesta tabela *hash*. Em seguida utiliza os demais exemplos presentes na estrutura *Stream Summary*.

Para atualizar o modelo de aprendizado e garantir a incrementabilidade, basta definir a estratégia de atualização do modelo utilizado pelo algoritmo. Esta estratégia está definida pela função *atualiza_Stream_Summary()* no Algoritmo 6. Neste trabalho foi proposto duas estratégias, uma baseada no algoritmo *Space Saving* e outra baseada em uma *Janela Deslizante* sobre o fluxo de dados.

4.4.1 Space Saving com CountingRank

A estratégia de atualização da estrutura *Stream Summary* baseada no *Space Saving* utiliza a mesma ideia que o algoritmo original, descrito no Capítulo 2. Remove o exemplo de treino que possuir o menor valor de contador na estrutura *Stream Summary*, e inclui o novo exemplo na mesma posição do exemplo removido, incrementando o seu contador em uma unidade. Esta proposta será chamada de *Space Saving CountingRank* na análise experimental realizada neste trabalho.

O Algoritmo 7 descreve esta estratégia. Primeiro é identificado o elemento associado ao exemplo de treino que apresenta o menor valor de contador de vizinhos, e_{min} . Substitui o exemplo associado a este elemento, x_{min} , pelo novo exemplo de treino x_e . O novo elemento entra na estrutura possuindo o mesmo valor do contador do exemplo removido, min . Atribui ao *threshold* ϵ_{min} o valor do contador do novo exemplo de treino. Em seguida o contador do novo exemplo é incrementado em uma unidade.

Algoritmo 7: *atualiza_Stream_Summary()* – *Space Saving*

Entrada: instância x_e

1 **início**

- 2** identifica e_{min} , o elemento com menor número de vizinhos mais próximo, min ;
 - 3** substitua x_{min} , associado à e_{min} , por x_e ;
 - 4** atribua a ϵ_{min} o valor min ;
 - 5** incrementa o contador de x_e em 1 unidade;
-

Caso haja empate entre dois exemplos, foi utilizado o valor do ϵ para desempatar os exemplos empatrados. Se persistir o empate, o último elemento adicionado na lista de elementos do contador terá prioridade dentre os demais elementos.

4.4.2 Janela Deslizante com CountingRank

Nessa estratégia de atualização é utilizada uma *Janela Deslizante* sobre o fluxo. O exemplo de treino mais antigo presente nesta janela é removido. O novo exemplo de treino é adicionado na estrutura com um contador constante. Esta proposta será chamada de *Janela Deslizante CountingRank* na análise experimental.

O Algoritmo 8 descreve esta estratégia. Inicialmente identifica o elemento associado ao exemplo de treino mais antigo, e_{ant} , presente na janela deslizante. Esse elemento é removido da estrutura *Stream Summary*. Em seguida adiciona um novo elemento e associado ao novo exemplo de treino x_e na estrutura com um contador igual a zero (0). O parâmetro ϵ nesta proposta será constante igual a zero (0).

Em caso de empate, o exemplo mais recente, ou o último exemplo adicionado na lista do contador, terá preferência.

Algoritmo 8: *atualiza_Stream_Summary() – Janela Deslizante*

Entrada: instância x_e

- 1 **início**
- 2 seja e_{ant} ser o exemplo mais antigo dentro da *janela deslizante*;
- 3 remove o elemento e_{ant} e o exemplo x_{ant} associado a este elemento;
- 4 adiciona o elemento e associado ao exemplo x_e com contador igual a zero (0);

4.5 Análise Experimental

Nossa análise experimental está dividida em duas partes. Na primeira análise foi utilizada bases de dados com e sem mudanças de conceito avaliando apenas o método *Space Saving CountingRank*. Na segunda os experimentos utilizam bases de dados com mudanças de conceitos avaliando ambos os métodos propostos neste trabalho. Por motivos de espaço, e pelo fato de que as análises com mudança de conceito serem mais desafiadoras, apresentamos neste capítulo apenas os resultados com bases de dados com mudanças de conceito avaliando os métodos *Space Saving CountingRank* e *Janela Deslizante CountingRank*. Reservamos o Apêndice A para apresentar os resultados da avaliação do método *Space Saving CountingRank* nas bases de dados com e sem mudanças de conceito.

Para avaliar a eficácia dos métodos propostos neste capítulo, eles foram comparados com os métodos *Aleatório* e *Janela Deslizante*. Ambos os métodos tem como objetivo principal selecionar as m instâncias do fluxo de dados que irão compor a base de treinamento utilizada nas classificações do algoritmo k -vizinhos mais próximos *anytime*.

O método **Aleatório** atualiza a base de treinamento retirando uma instância de forma aleatória da lista de exemplos de treino e incluindo uma nova instância no final desta lista. Este método não utiliza nenhuma informação sobre as características dos dados para atualizar o modelo de aprendizado.

O método **Janela Deslizante** mantém como conjunto de treinamento os últimos m exemplos gerados pelo fluxo de dados. Assim é utilizada uma janela deslizante sobre os exemplos gerados pelo fluxo de dados, como sendo os exemplos de treino a serem utilizados pelo algoritmo. Esta técnica considera que as instâncias mais antigas devem ser removidas, pois não representam bem o conceito atual dos dados gerados pelo fluxo. Para atualizar o conjunto de treinamento, basta remover a instância mais antiga e incluir a nova instância no final da lista, ou seja, basta remover a instância mais ao topo da lista e adicionar a nova instância no final da lista.

4.5.1 Configuração experimental

Os métodos propostos foram avaliados utilizando diversas bases sintéticas que apresentam várias formas de mudança de conceito (SOUZA *et al.*, 2015). Foi possível também avaliar os métodos em uma base de dados real que apresenta mudança de conceito. Em geral,

esses conjuntos de dados apresentam mudanças de conceito incremental e gradual. A Tabela 5 representa informações sobre cada um dos conjuntos de dados e está disponível na página web *Nonstationary Environments - Archive*¹.

Tabela 5 – Conjuntos de dados que apresentam mudança de conceito.

Conjunto de dados	Número de classes	Número de atributos	Número de exemplos	Mudança a cada #exemplos	Tipo
1CDT	2	2	16,000	400	Sintético
2CDT	2	2	16,000	400	Sintético
1CHT	2	2	16,000	400	Sintético
2CHT	2	2	16,000	400	Sintético
4CR	4	2	144,400	400	Sintético
4CRE-V1	4	2	125,000	1,000	Sintético
4CRE-V2	4	2	183,000	1,000	Sintético
5CVT	5	2	40,000	1,000	Sintético
1CSurr	2	2	55,283	600	Sintético
4CE1CF	5	2	173,250	750	Sintético
UG_2C_2D	2	2	100,000	1,000	Sintético
MG_2C_2D	2	2	200,000	2,000	Sintético
FG_2C_2D	2	2	200,000	2,000	Sintético
UG_2C_3D	2	3	200,000	2,000	Sintético
UG_2C_5D	2	5	200,000	2,000	Sintético
GEARS_2C_2D	2	2	200,000	2,000	Sintético
Keystroke	4	10	1,600	200	Real

Para simular uma avaliação de um algoritmo *anytime*, é necessário decidir se uma determinada classificação deve ou não ser interrompida e quando deve ser realizada a interrupção. Para isso, a avaliação também inclui os parâmetros α e β . O primeiro define se o fluxo de dados deve gerar um novo exemplo antes de o anterior ser completamente processado. Ou seja, este parâmetro define se o algoritmo *anytime* deve ou não ser interrompido no meio do seu processamento. Já o segundo define quantos exemplos de treino devem ser processados pelo algoritmo *anytime* antes dele ser interrompido.

Estes dois parâmetros foram utilizados nos experimentos com as seguintes configurações:

- α : indica a probabilidade de interrupção do algoritmo *anytime* para um determinado exemplo de teste. Ou seja, este parâmetro determina se o algoritmo *anytime* será executado por completo ou interrompido em algum momento. As probabilidades de interrupção utilizadas foram $\{0.0, 0.01, 0.05, 0.1, 0.3, 0.5, 0.7, 0.9\}$.
- β : indica a quantidade de exemplos processados antes de interromper o algoritmo *anytime*. Quanto maior seu valor, maior será a quantidade de exemplos verificados antes da interrupção. Seu valor varia entre $\{0.1, 0.2, 0.5, 0.75, 0.95\}$. O valor de β é multiplicado pelo valor de m (número de exemplos do conjunto de treinamento) para calcular o número de exemplos a serem processados. Para este parâmetro foi utilizada também uma distribuição

¹ <https://sites.google.com/site/nonstationaryarchive/>

normal. Assim é possível simular um fluxo de dados real, onde não se conhece a priori a quantidade de instâncias que o algoritmo deve verificar antes da interrupção.

Algumas bases não apresentam grande quantidade de exemplos e as mudanças de conceito ocorrem após uma quantidade fixa de exemplos. Valores elevados para o parâmetro m teriam como consequência a presença de vários conceitos diferentes no conjunto de treinamento utilizado pelo algoritmo *anytime*. Para avaliar o comportamento dos métodos foram utilizados os valores de m no conjunto $\{100, 200, 300, 400, 500\}$.

4.5.2 Avaliação

Os resultados dos experimentos são reportados com o *prequential error* (GAMA; RODRIGUES; SEBASTIÃO, 2009). Em resumo, os exemplos são primeiramente classificados e logo em seguida se recebe o rótulo correto e se atualiza o valor do erro de classificação. O erro é calculado considerando os erros ocorridos em uma janela deslizante sobre o fluxo de dados, de acordo com a Equação 4.2. Assim é possível observar o desempenho dos classificadores em cada parte do fluxo de dados. A ocorrência de mudanças de conceito pode fazer com que haja mudanças repentinas nos desempenhos dos classificadores. Tais mudanças de desempenho poderiam passar despercebidas caso o cálculo do erro levasse em consideração todas as instâncias do fluxo.

$$\text{erro} = \frac{1}{n} \sum_{i=1}^n \text{errado}(x_i) \quad (4.2)$$

onde x_i representa a instância classificada na i -ésima posição da janela deslizante. A função $\text{errado}(x)$ retorna 0 caso a instância x tenha sido classificada corretamente, ou 1 caso contrário. n representa o tamanho da janela deslizante e o valor erro representa o erro obtido na janela deslizante.

Foram utilizadas três formas de análise dos resultados nesta avaliação experimental. A primeira é um gráfico da taxa de erro *versus* o número de exemplos gerados pelo fluxo. Com este gráfico é possível ver o comportamento dos métodos durante todo o fluxo de dados. Assim é possível verificar em que momento do fluxo o método apresentou melhor desempenho. Pois em alguns casos, o método pode apresentar altíssimo desempenho em pequenos instantes do fluxo, e no geral pode ser considerado um bom método. Espera-se que o método apresente bom desempenho durante a maior parte de tempo do fluxo.

A segunda forma de análise sumariza em apenas um valor a taxa de erro obtida pelos métodos. Mesmo que este valor forneça menos informação que o gráfico, com ele é possível verificar o comportamento de uma forma geral dos métodos nas diversas bases utilizadas nesta análise experimental. Baseado no desempenho obtido pode-se visualizar o gráfico da taxa de erro para encontrar mais detalhes sobre o comportamento dos métodos ao longo do tempo. Para este

valor foi realizado o cálculo da área sobre a curva da taxa de erro, de acordo com a Equação 4.3.

$$AUC_{erro} = \frac{1}{n} \sum_{i=1}^n erro_i \quad (4.3)$$

onde $erro_i$ é o valor do erro obtido para a primeira janela deslizante sobre o fluxo de dados. n é o número total de janelas deslizantes sobre o fluxo de dados.

A terceira forma de análise utiliza um gráfico de espalhamento. Neste gráfico é possível analisar a relação de desempenho entre dois métodos. Assim é possível verificar o quanto bom é um método comparado com outro método. Será utilizado o valor da área sobre a curva do erro para gerar este gráfico. Esse gráfico é dividido por uma linha diagonal que se inicia no ponto $(0, 0)$ e termina no ponto $(1, 1)$. Os pontos que estiverem sobre esta linha representam desempenho equivalente entre os métodos. Os pontos presentes no triângulo inferior indicam que o método do eixo y apresenta melhor performance do que o método do eixo x . Os pontos presentes no triângulo superior, indicam que o método do eixo x apresenta melhor desempenho do que o método do eixo y .

4.5.3 Resultados

Devido à grande quantidade de configurações experimentais é inviável apresentar todos os resultados obtidos pelos métodos. Será apresentado o resultado para quatro configurações diferentes de fluxo de dados. A *configuração 1* apresenta baixa probabilidade de interrupção e baixo número de exemplos de treino verificados em uma interrupção. A *configuração 2* apresenta baixa probabilidade de interrupção e alto número de exemplos de treino verificados em uma interrupção. A *configuração 3* apresenta alta probabilidade de interrupção e baixo número de exemplos de treino verificados em uma interrupção. E, por fim, a *configuração 4* apresenta alta probabilidade de interrupção e alto número de exemplos de treino verificados em uma interrupção.

Os resultados apresentados utilizam o tamanho da base $m = 200$, para possibilitar visualizar os resultados em todas as bases de dados. O tamanho da janela deslizante sobre o fluxo para o cálculo do erro foi de 7 vezes o tamanho da base m .

4.5.3.1 Configuração 1: $m = 200$; $\alpha = 0.10$; $\beta = 0.20$

Um fluxo de dados com estas características apresenta 10% de probabilidade de ocorrer uma interrupção da execução do algoritmo *anytime*. Isso representa uma baixa quantidade de interrupções do algoritmo. Nos casos em que ocorre interrupção, o algoritmo verifica 20% do tamanho da base de treino. Isso representa que o algoritmo verifica poucos exemplos de treino para disponibilizar a resposta em uma interrupção. A Tabela 6 apresenta as áreas sobre a curva do erro obtidas pelos métodos analisados neste trabalho para a configuração 1.

Tabela 6 – Área sobre a curva da taxa de erro obtida pelos métodos para a *Configuração*: [$m = 200$; $\alpha = 0.10$; $\beta = 0.20$]. O tamanho da janela deslizante para o cálculo do erro é de 7 vezes o tamanho da base (m).

Base de Dado	Aleatório	Janela Deslizante	Space Saving CountingRank	Janela Deslizante CountingRank
1CDT	0.000816	0.000685	0.000503	0.000651
1CHT	0.004927	0.004015	0.003143	0.003916
1CSurr	0.020043	0.017475	0.017324	0.017120
2CDT	0.081052	0.061765	0.051427	0.058493
2CHT	0.160490	0.144067	0.122916	0.139582
4CE1CF	0.030484	0.030779	0.025151	0.029947
4CR	0.000161	0.000170	0.000159	0.000157
4CRE-V1	0.026989	0.023687	0.020153	0.022653
4CRE-V2	0.108633	0.108782	0.090094	0.106144
5CVT	0.144268	0.128853	0.107631	0.123501
FG_2C_2D	0.062555	0.063146	0.051338	0.061495
GEARS_2C_2D	0.003736	0.003551	0.002710	0.003175
MG_2C_2D	0.097326	0.098336	0.080755	0.096033
UG_2C_2D	0.058484	0.058020	0.047353	0.056708
UG_2C_3D	0.072498	0.073067	0.059163	0.071176
UG_2C_5D	0.105034	0.104821	0.084386	0.102213
keystroke	0.106749	0.094691	0.099067	0.089355
Vitórias	0	0	14	3

Nesta configuração, o método *Janela Deslizante CountingRank* apresentou o melhor valor da área sob a curva do erro em 3 das bases analisadas. O método *Space Saving CountingRank* obteve melhor valor da área sob a curva de erro para 14 das 17 bases de dados analisadas. Isso indica que em um fluxo de dados que apresenta as características desta configuração, o método *Space Saving CountingRank* é o mais indicado.

A Figura 9 apresenta os gráficos de espalhamento que mostram a relação entre os métodos. Nas três primeiras figuras, pode ser observado que o método *Space Saving CountingRank* obteve melhor desempenho que os demais métodos na grande maioria das bases analisadas. A superioridade do método pode ser notada pela presença dos pontos no triangulo inferior do gráfico e suas distâncias da linha diagonal. Nas duas últimas figuras, é apresentada a relação de performance do método *Janela Deslizante CountingRank*. Este método obteve desempenho um pouco semelhante ao método *Janela Deslizante*, apresentando o valor da área da curva do erro um pouco menor em algumas bases. O critério de desempate adotado por este método pode ter contribuído para este resultado.

A Figura 10 mostra a variação da taxa de erro ao longo do fluxo de dados para a base de dados *4CRE-V2*. Apesar das curvas estarem próximas em parte do tempo do fluxo de dados, nota-se que a curva do método *Space Saving CountingRank* permaneceu abaixo das curvas dos demais métodos nos momentos importantes do fluxo. Nos instantes em que ocorre mudança de conceito, esse método aumenta o valor do erro lentamente comparado aos demais métodos. Sendo que ao finalizar a mudança de conceito, o valor do erro para este método permanece

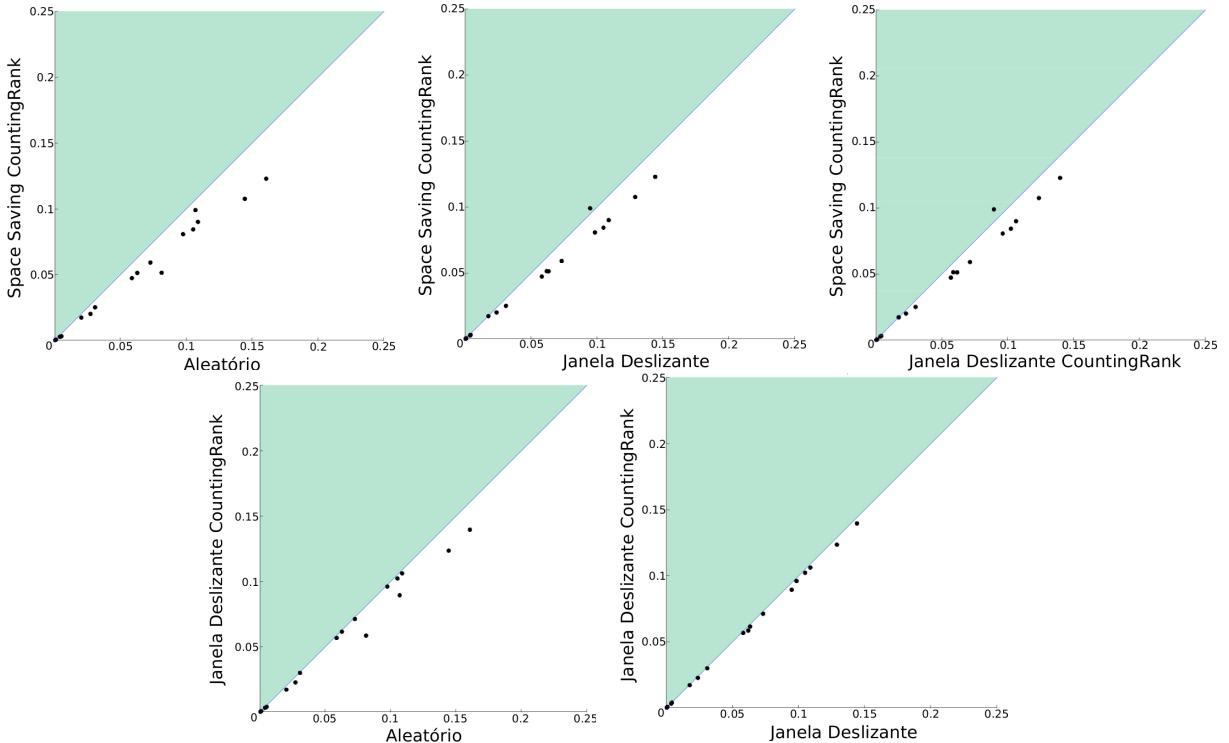


Figura 9 – Gráfico de espalhamento da relação do erro entre os métodos para a configuração: $m = 200$; $\alpha = 0.10$; $\beta = 0.20$. Os três primeiros gráficos mostram a relação do método *Space Saving CountingRank* com os demais métodos. Os dois gráficos restantes mostram a relação entre o método *Janela Deslizante CountingRank* e os demais métodos.

aproximadamente 5% abaixo dos demais métodos. Esse é um bom resultado, pois o método foi capaz de tratar as mudanças de conceito para esta base e manter uma baixa taxa de erro para a classificação das novas instâncias.

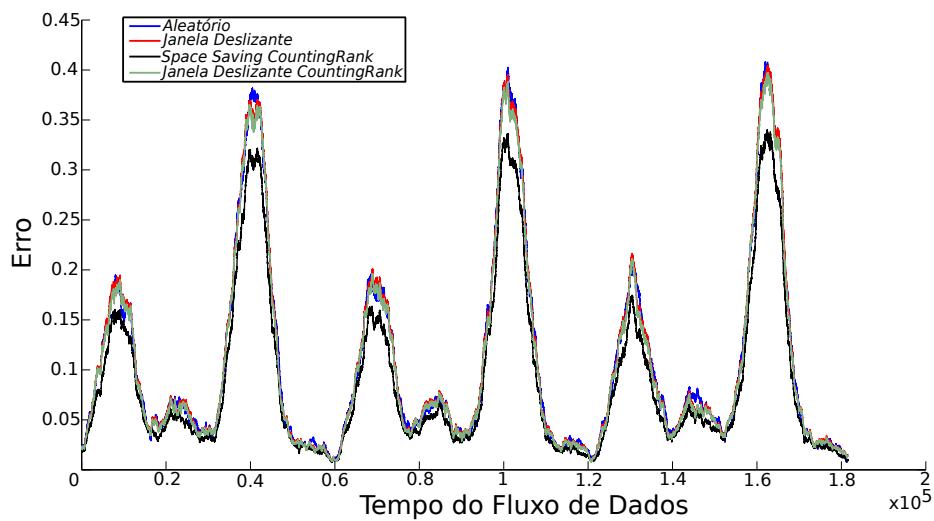


Figura 10 – Variação da taxa de erro ao longo do fluxo de dados no conjunto de dados 4CRE-V2 para a configuração $m = 200$; $\alpha = 0.10$; $\beta = 0.20$

A Figura 11 mostra a variação da taxa de erro para a base *keystroke*. O método *Space Saving CountingRank* já não obteve o mesmo desempenho que obteve na base anterior. Nessa

base, o método *Janela Deslizante CountingRank* obteve menor variação da taxa de erro. Em alguns momentos do fluxo a diferença para a segunda melhor técnica alcança quase 1% de diferença da taxa de erro. Isso mostra que em uma aplicação que apresente as características desta base, a técnica *Janela Deslizante CountingRank* pode ser a mais indicada.

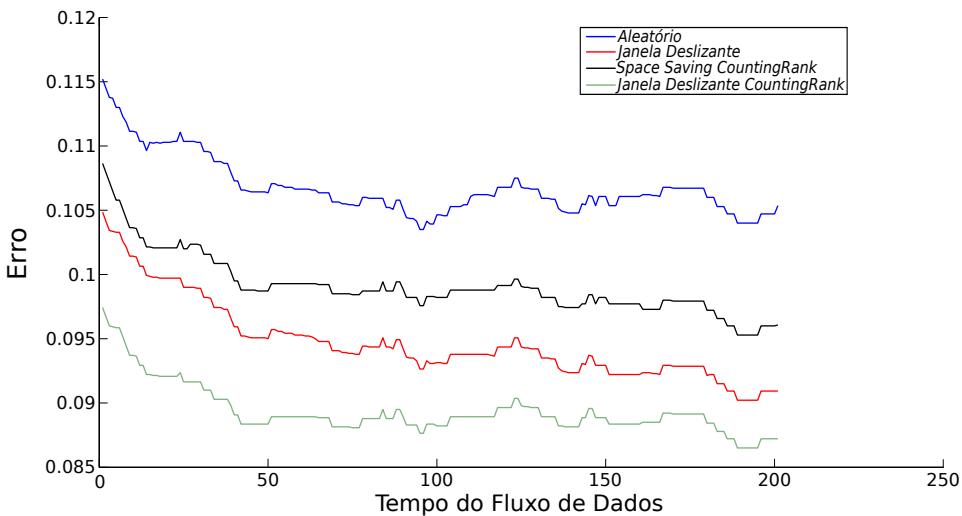


Figura 11 – Variação da taxa de erro ao longo do fluxo de dados no conjunto de dados *keystroke* para a configuração $m = 200; \alpha = 0.10; \beta = 0.20$

4.5.3.2 Configuração 2: $m = 200; \alpha = 0.10; \beta = 0.75$

Um fluxo de dados com estas características apresenta 10% de probabilidade de ocorrência de interrupção da execução do algoritmo *anytime*. A quantidade de interrupções do algoritmo continua baixa de maneira similar à configuração anterior. Porém, nos casos em que ocorre interrupção, o algoritmo verifica 75% do tamanho da base de treino. Assim o algoritmo verifica grande parte dos exemplos de treino para disponibilizar uma resposta. A Tabela 7 apresenta as áreas sobre a curva do erro obtidas pelos métodos analisados neste trabalho para a configuração 2.

Nesta configuração o método *Space Saving CountingRank* também apresentou melhor desempenho na maioria dos conjuntos de dados. Foram 13 do total de 17 conjuntos analisados. Em alguns conjuntos de dados, a diferença do valor da área da curva deste método para o segundo melhor alcançou 2%. O método *Janela Deslizante CountingRank* apresentou o segundo melhor desempenho nesta configuração.

A Figura 12 apresenta os gráficos de espalhamento que mostra a relação de performance entre os métodos. Nos três primeiros gráficos, pode-se confirmar a superioridade do método *Space Saving CountingRank*. A maioria dos pontos está presente no triângulo inferior e com uma distância notável. Nos dois últimos gráficos tem-se a relação do método *Janela Deslizante CountingRank* e os demais métodos. Nessa configuração, o desempenho do método foi ainda mais semelhante com o método *Janela Deslizante*. No gráfico, os pontos estão praticamente

Tabela 7 – Área sobre a curva da taxa de erro obtida pelos métodos para a *Configuração*: [$m = 200$; $\alpha = 0.10$; $\beta = 0.75$]. O tamanho da janela deslizante para o cálculo do erro é de 7 vezes o tamanho da base (m).

Base de Dado	Aleatório	Janela Deslizante	Space Saving CountingRank	Janela Deslizante CountingRank
1CDT	0.000799	0.000696	0.000450	0.000662
1CHT	0.004990	0.003988	0.003122	0.003935
1CSurr	0.018420	0.016305	0.016686	0.016272
2CDT	0.074817	0.060793	0.049218	0.059238
2CHT	0.157175	0.142873	0.122064	0.142246
4CE1CF	0.029362	0.029640	0.025096	0.029628
4CR	0.000146	0.000153	0.000159	0.000158
4CRE-V1	0.025799	0.023185	0.019802	0.022885
4CRE-V2	0.107483	0.107804	0.090005	0.107351
5CVT	0.137176	0.126029	0.105959	0.124579
FG_2C_2D	0.061803	0.062398	0.051152	0.062025
GEARS_2C_2D	0.002038	0.001851	0.001902	0.001918
MG_2C_2D	0.096889	0.097852	0.080616	0.097384
UG_2C_2D	0.057994	0.057824	0.047584	0.057689
UG_2C_3D	0.072066	0.072573	0.059167	0.072173
UG_2C_5D	0.103645	0.103545	0.084851	0.102923
keystroke	0.095092	0.088645	0.098723	0.089052
Vitórias	1	2	13	1

na linha diagonal. Como ocorrem poucas interrupções e o algoritmo processa grande parte dos exemplos de treino podia-se esperar um comportamento semelhante entre esses métodos.

A Figura 13 apresenta a variação da taxa de erro ao longo do fluxo de dados para a base *2CDT*. Nota-se nesse gráfico que a curva do erro do método *Janela Deslizante CountingRank* se assemelha muito com a curva do método *Janela Deslizante*. Já a curva do método *Space Saving CountingRank* permanece abaixo das demais durante grande parte do fluxo de dados. Apenas no início da curva que as taxas de erro se aproximam dos demais métodos. Nesta fase inicial do fluxo os conceitos se alteram gradativamente. Após certo tempo em que os conceitos estão bem diferentes do início do fluxo, o método mantém baixas taxas de erro, enquanto os demais apresentam alta taxa de erro.

A Figura 14 apresenta a variação da taxa de erro ao longo do fluxo de dados para a base *UG_2C_5D*. No início do fluxo os métodos apresentam taxas de erro bem próximas. Após o fluxo gerar aproximadamente 40000 exemplos, o método *Space Saving CountingRank* apresenta baixa taxa de erro em relação aos demais métodos. A curva da taxa de erro não apresenta grandes variações, se mantendo abaixo da curva dos demais métodos até aproximadamente 160000 exemplos gerados. Nesse momento, as taxas de erro dos métodos voltam a ser bem próximas. No final do fluxo este método volta a apresentar baixas taxas de erro. Isso mostra que esse método também apresenta boa performance quando aplicado a um fluxo com as propriedades desta configuração.

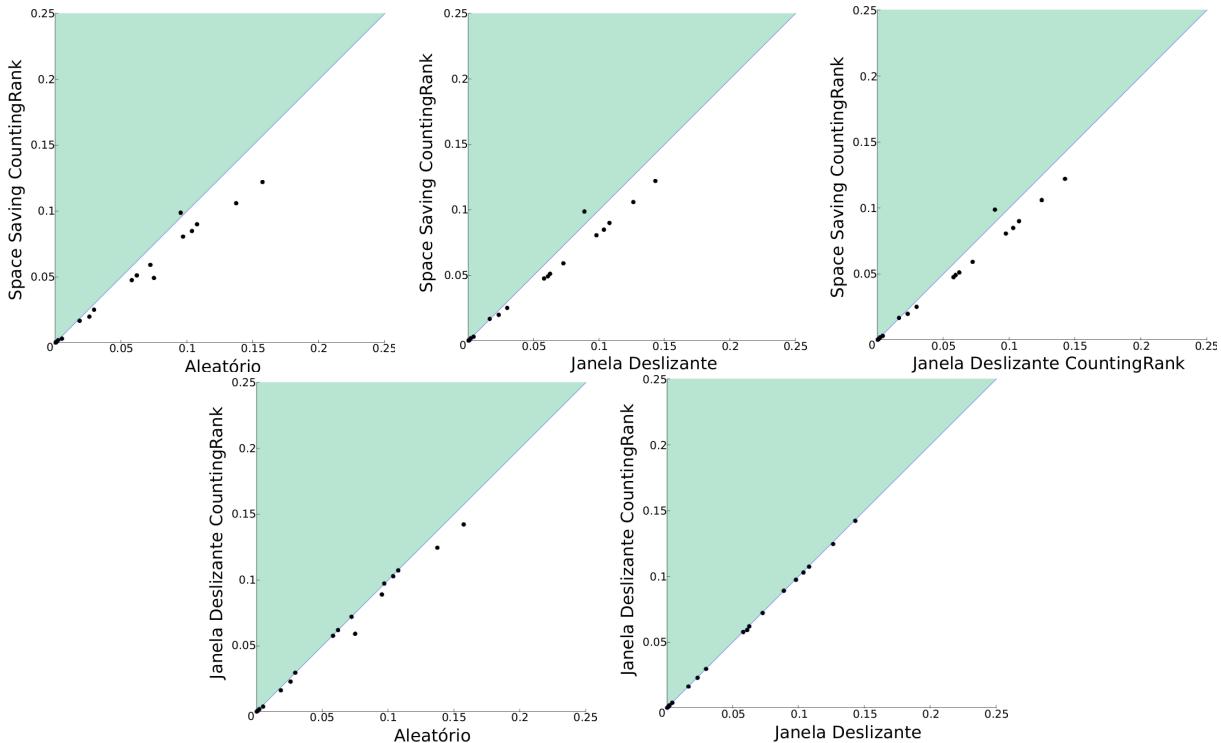


Figura 12 – Gráfico de espalhamento da relação entre o erro entre dois métodos para a configuração: $m = 200$; $\alpha = 0.10$; $\beta = 0.75$. Os três primeiros gráficos mostram a relação do método *Space Saving CountingRank* com os demais métodos. Os dois restantes mostram a relação entre o método *Janela Deslizante CountingRank* e os demais métodos.

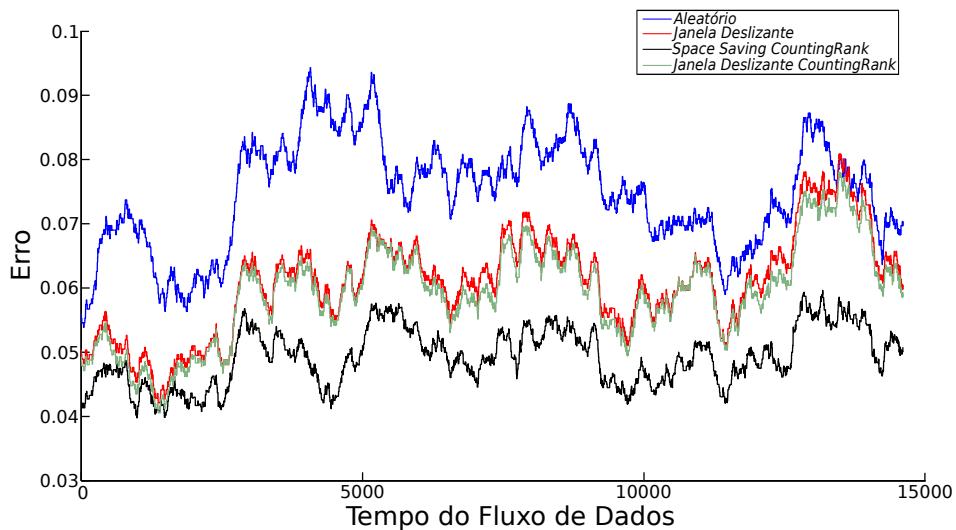


Figura 13 – Variação da taxa de erro ao longo do fluxo de dados no conjunto de dados 2CDT para a configuração $m = 200$; $\alpha = 0.10$; $\beta = 0.75$

4.5.3.3 Configuração 3: $m = 200$; $\alpha = 0.90$; $\beta = 0.20$

Um fluxo de dados com estas características apresenta 90% de probabilidade de ocorrência de interrupção da execução do algoritmo *anytime*. Isso representa uma grande quantidade de interrupções do algoritmo. Nos casos em que ocorre interrupção, o algoritmo verifica apenas 20% do tamanho da base de treino. Isso representa também que o algoritmo verifica poucos

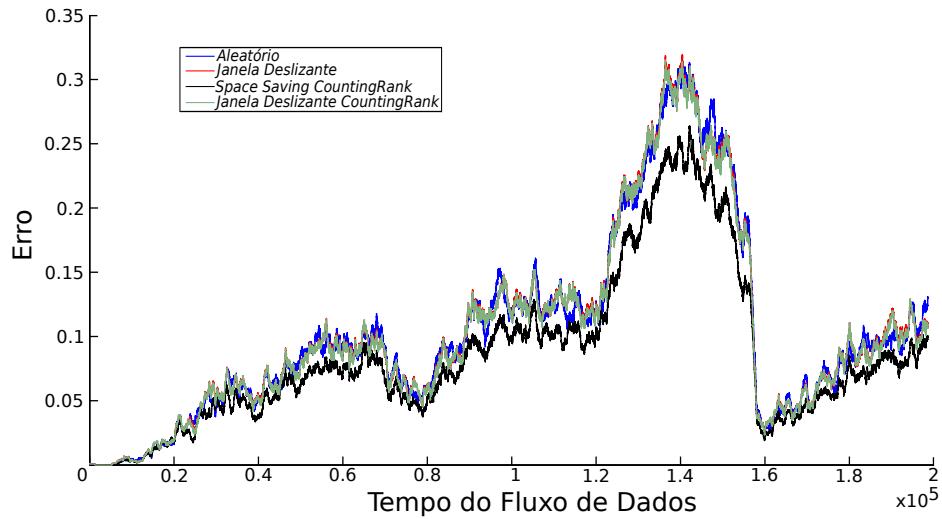


Figura 14 – Variação da taxa de erro ao longo do fluxo de dados no conjunto de dados *UG_2C_5D* para a configuração $m = 200$; $\alpha = 0.10$; $\beta = 0.75$

exemplos de treino para disponibilizar a resposta em uma interrupção. A Tabela 8 apresenta as áreas sobre a curva do erro obtidas pelos métodos analisados neste trabalho.

Tabela 8 – Área sobre a curva da taxa de erro obtida pelos métodos para a *Configuração*: [$m = 200$; $\alpha = 0.90$; $\beta = 0.20$]. O tamanho da janela deslizante para o cálculo do erro é de 7 vezes o tamanho da base (m).

Base de Dado	Aleatório	Janela Deslizante	Space Saving CountingRank	Janela Deslizante CountingRank
1CDT	0.001403	0.000886	0.001955	0.000427
1CHT	0.005112	0.003996	0.007994	0.003489
1CSurr	0.035024	0.028918	0.158703	0.024940
2CDT	0.139595	0.074029	0.194228	0.047483
2CHT	0.193717	0.157144	0.209416	0.122006
4CE1CF	0.039599	0.039888	0.019913	0.034212
4CR	0.000325	0.000275	0.182247	0.000208
4CRE-V1	0.038621	0.028661	0.209364	0.020604
4CRE-V2	0.119181	0.118209	0.221076	0.099706
5CVT	0.209173	0.154730	0.206603	0.111529
FG_2C_2D	0.069032	0.069353	0.097774	0.057787
GEARS_2C_2D	0.018150	0.017534	0.039211	0.016610
MG_2C_2D	0.101347	0.102850	0.154060	0.086671
UG_2C_2D	0.061399	0.060098	0.185926	0.050555
UG_2C_3D	0.075950	0.077716	0.122944	0.064837
UG_2C_5D	0.116954	0.116078	0.144163	0.095905
keystroke	0.201724	0.151520	0.111276	0.115579
Vitórias	0	0	2	15

Nessa configuração o método *Janela Deslizante CountingRank* apresentou melhor desempenho na maioria das bases analisadas. Foram 15 do total de 17 bases em que o método apresentou desempenho superior. Esse método apresentou 4% a menos do valor da área sobre a curva do erro que o segundo melhor método na base de dados *5CVT*. Isso ocorreu pela boa com-

binação do ranqueamento feito pela equação proposta neste trabalho e o critério de desempate utilizado por este método. O método *Space Saving CountingRank* não apresentou bons resultados nesta configuração. Este método obteve melhor desempenho em apenas 2 bases (*4CE1CF* e *keystroke*). O grande número de interrupções e o baixo número de exemplos de treino verificados em uma interrupção podem ter contribuído para o seu baixo desempenho. Em um fluxo com estas características, este método atualiza apenas os exemplos presentes no topo da lista. Assim os novos exemplos de treino, que melhor representam os conceitos atuais no fluxo de dados, podem não ganhar o benefício de realizar a classificação de exemplos desconhecidos.

A Figura 15 apresenta os gráficos de espalhamento que mostram a relação de desempenho entre os métodos. Os três primeiros gráficos mostram a relação do método *Space Saving CountingRank* com os demais métodos. Em todos estes gráficos, as maioria dos pontos estão presentes no triângulo superior. Isso representa que o desempenho deste método foi muito inferior comparado com os outros métodos. A grande distância dos pontos para a linha diagonal clara a baixa performance deste método. Os dois últimos gráficos apresentam a relação do método *Janela Deslizante CountingRank* com os outros métodos. Em ambos os gráficos a maioria dos pontos está presente no triângulo inferior, indicando que este método obteve melhor desempenho que os outros. Os pontos estão bem afastados da linha diagonal, o que confirma que este método obteve melhor desempenho.

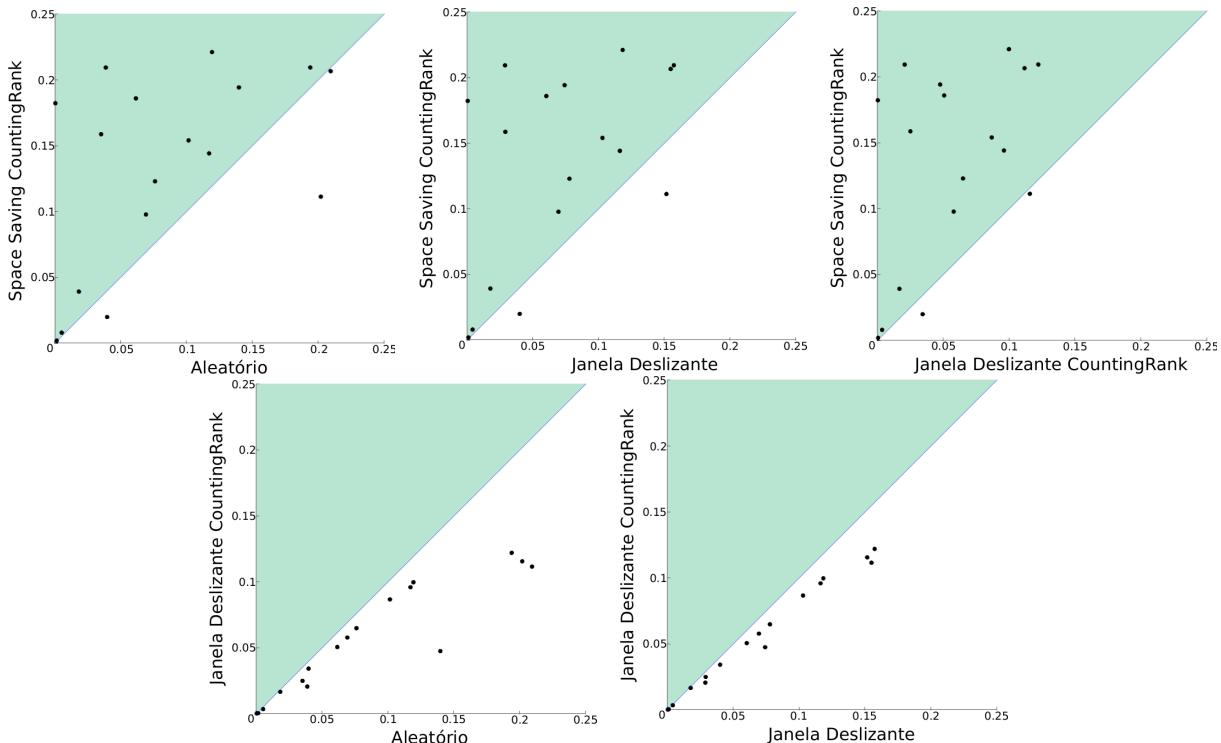


Figura 15 – Gráfico de espalhamento da relação entre o erro entre dois métodos para a configuração: $m = 200$; $\alpha = 0.90$; $\beta = 0.20$. Os três primeiros gráficos mostram a relação do método *Space Saving CountingRank* com os demais. Os dois restantes mostram a relação entre o método *Janela Deslizante CountingRank* e os demais métodos.

A Figura 16 mostra a variação da taxa de erro para a base de dados *5CVT*. O método

Space Saving CountingRank obteve uma alta variação da taxa de erro ao longo do fluxo, mantendo o valor da taxa de erro acima do valor obtido pelo método *Janela Deslizante*. O método *Janela Deslizante CountingRank* apresentou baixa taxa de erro, sem muita variação e com um valor bem abaixo do método *Janela Deslizante*. Em alguns momentos do fluxo, a diferença entre estes métodos chegou a quase 7%.

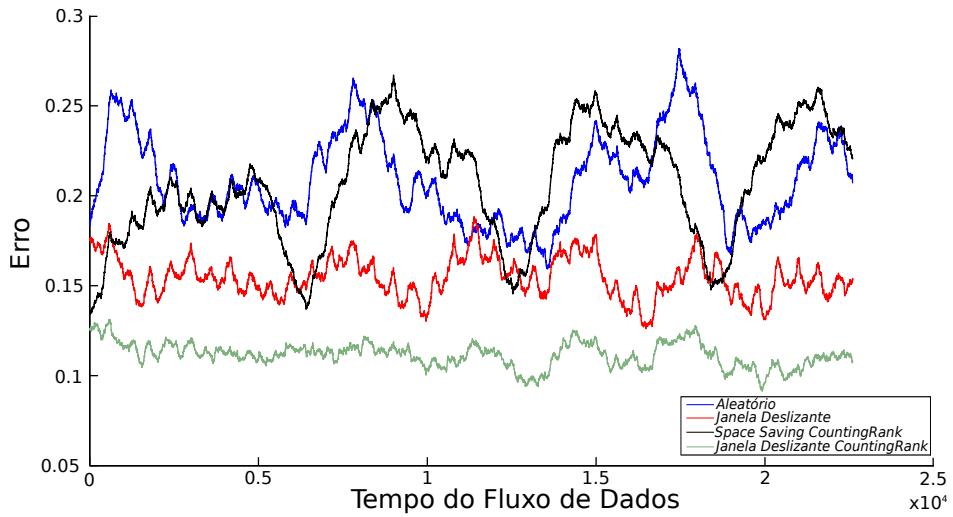


Figura 16 – Variação da taxa de erro ao longo do fluxo de dados no conjunto de dados 5CVT para a configuração $m = 200$; $\alpha = 0.90$; $\beta = 0.20$

A Figura 17 apresenta a variação da taxa de erro para a base de dados *keystroke*. Esta foi uma das poucas bases que o método *Space Saving CountingRank* obteve melhor desempenho. Apesar de que a variação da taxa de erro deste método ficou bem próxima da taxa de erro obtida pelo método *Janela Deslizante CountingRank*. A diferença entre esses métodos não ultrapassou o valor de 0.4% do valor da taxa de erro.

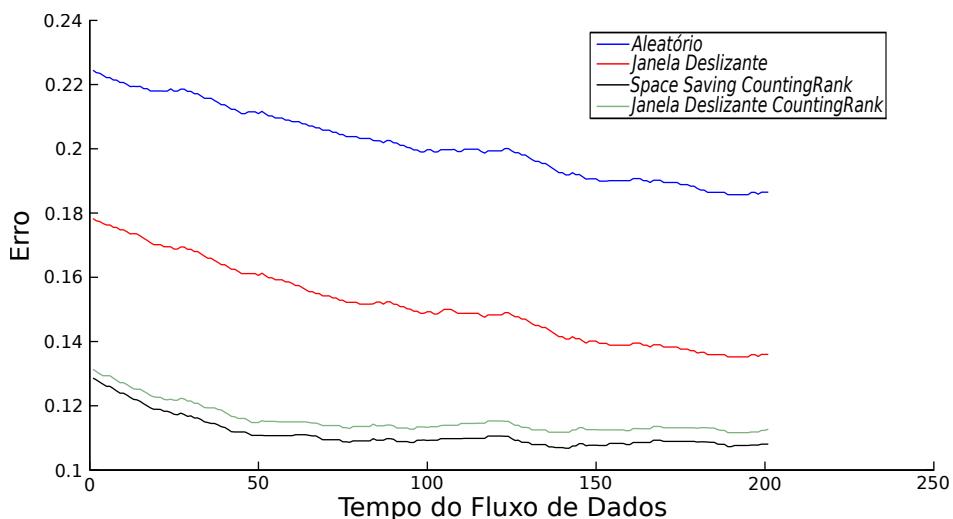


Figura 17 – Variação da taxa de erro ao longo do fluxo de dados no conjunto de dados *keystroke* para a configuração $m = 200$; $\alpha = 0.90$; $\beta = 0.20$

De uma forma geral, o método *Space Saving CountingRank* não apresentou melhor desempenho. Em um fluxo de dados com as características desta configuração, este método não mostrou ser o mais indicado. Já o método *Janela Deslizante CountingRank* mostrou resultados competitivos, e de acordo com os resultados pode ser o mais indicado a ser utilizado num fluxo com as características desta configuração.

4.5.3.4 Configuração 4: $m = 200$; $\alpha = 0.90$; $\beta = 0.75$

Um fluxo de dados com estas características apresenta 90% de probabilidade de ocorrência de interrupção da execução do algoritmo *anytime*. Continua representando uma grande quantidade de interrupções do algoritmo. Porém, nessa configuração o algoritmo verifica uma quantidade maior de exemplos de treino, 75% do tamanho da base de treino. Isso representa que o algoritmo verifica grande parte exemplos de treino para disponibilizar a resposta em uma interrupção. A Tabela 9 apresenta as áreas sobre a curva do erro obtidas pelos métodos analisados neste trabalho para a configuração 4.

Tabela 9 – Área sobre a curva da taxa de erro obtida pelos métodos para a *Configuração*: [$m = 200$; $\alpha = 0.90$; $\beta = 0.75$]. O tamanho da janela deslizante para o cálculo do erro é de 7 vezes o tamanho da base (m).

Base de Dado	Aleatório	Janela Deslizante	Space Saving CountingRank	Janela Deslizante CountingRank
1CDT	0.000831	0.000731	0.000394	0.000600
1CHT	0.005065	0.003916	0.003072	0.003420
1CSurr	0.020162	0.017731	0.017305	0.017480
2CDT	0.084268	0.064586	0.049858	0.052527
2CHT	0.164355	0.144854	0.121167	0.135165
4CE1CF	0.029948	0.030435	0.024736	0.029468
4CR	0.000142	0.000143	0.000175	0.000191
4CRE-V1	0.027683	0.024578	0.019596	0.021631
4CRE-V2	0.108821	0.108877	0.089546	0.103689
5CVT	0.147094	0.131170	0.105708	0.117491
FG_2C_2D	0.062178	0.063298	0.051009	0.059259
GEARS_2C_2D	0.002463	0.002097	0.001994	0.002594
MG_2C_2D	0.097118	0.098125	0.079751	0.093329
UG_2C_2D	0.057588	0.057989	0.047609	0.055057
UG_2C_3D	0.072052	0.072820	0.058860	0.068890
UG_2C_5D	0.105611	0.104891	0.083853	0.098187
keystroke	0.103274	0.095938	0.097196	0.099629
Vitórias	1	1	15	0

Nessa configuração o método *Space Saving CountingRank* voltou a obter melhor desempenho que os demais. O método apresentou melhor desempenho em 15 das 17 bases utilizadas neste trabalho. Ele chegou a atingir uma diferença de performance de pouco mais de 2% nas bases *2CHT*, *5CVT* e *UG_2C_5D* para o método *Janela Deslizante*. Devido à alta quantidade de exemplos verificados em uma interrupção, este método foi capaz de utilizar os novos exemplos de treino na classificação dos exemplos desconhecidos, mesmo quando o algoritmo é interrom-

rido. Por esta razão este método voltou a apresentar melhor desempenho nessa configuração. Diferente da configuração anterior, o método *Janela Deslizante CountingRank* não obteve melhor resultado. Apesar deste método ter apresentado uma taxa de erro menor que os métodos *Janela Deslizante* e *Aleatório*, ele não conseguiu superar o desempenho obtido pelo método *Space Saving CountingRank*.

A Figura 18 apresenta os gráficos de espalhamento que mostram a relação de desempenho entre os métodos. Os três primeiros gráficos mostram a relação de desempenho do método *Space Saving CountingRank* e os outros métodos. A maioria dos pontos dos três gráficos está presente no triângulo inferior, evidenciando que o método em destaque apresentou melhor desempenho que os demais. Os pontos estão bem distantes da linha diagonal, o que indica melhor performance deste método. Os dois últimos gráficos mostram a relação de desempenho do método *Janela Deslizante CountingRank* e os outros métodos. A maioria dos pontos continua presente no triangulo inferior, o que indica que este método obteve melhor desempenho que os métodos analisados. Diferente do que ocorreu na configuração 2, o método *Janela Deslizante CountingRank* obteve desempenho diferente do método *Janela Deslizante* nesta configuração . Acredita-se que devido à alta probabilidade de interrupção, foi possível que o método proposto neste trabalho apresentasse melhor desempenho.

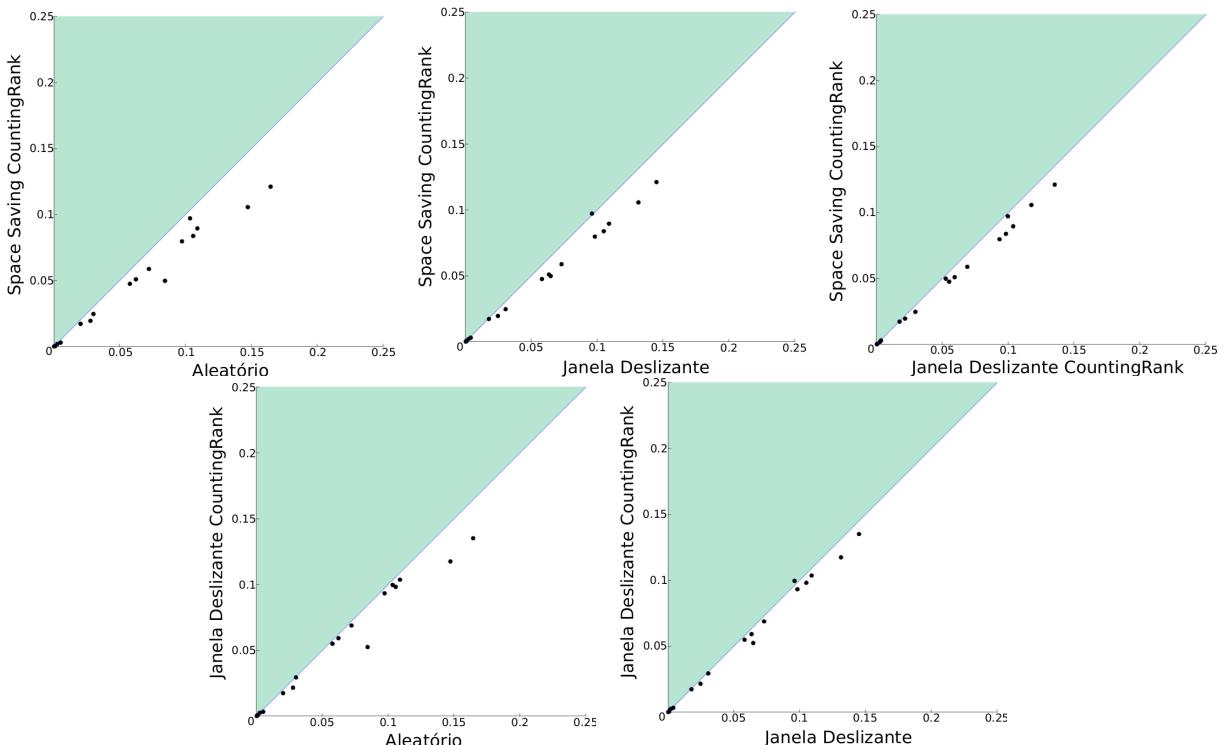


Figura 18 – Gráfico de espalhamento da relação entre o erro entre dois métodos para a configuração: $m = 200$; $\alpha = 0.90$; $\beta = 0.75$. Os três primeiros gráficos mostram a relação do método *Space Saving CountingRank* com os demais métodos. Os dois restantes mostram a relação entre o método *Janela Deslizante CountingRank* e os demais métodos.

A Figura 19 mostra a variação da taxa de erro para o conjunto de dados 5CVT. O método *Space Saving CountingRank* apresentou uma certa variabilidade da taxa de erro ao longo do

fluxo. Mas nada que prejudicasse o seu bom desempenho, mantendo sua curva do erro abaixo da curva dos demais métodos durante praticamente todo o fluxo. A curva do erro do método *Janela Deslizante CountingRank* se mantém abaixo dos outros dois métodos durante quase todo o fluxo também. Em alguns poucos momentos a sua curva aproxima bem da curva do método *Space Saving CountingRank*.

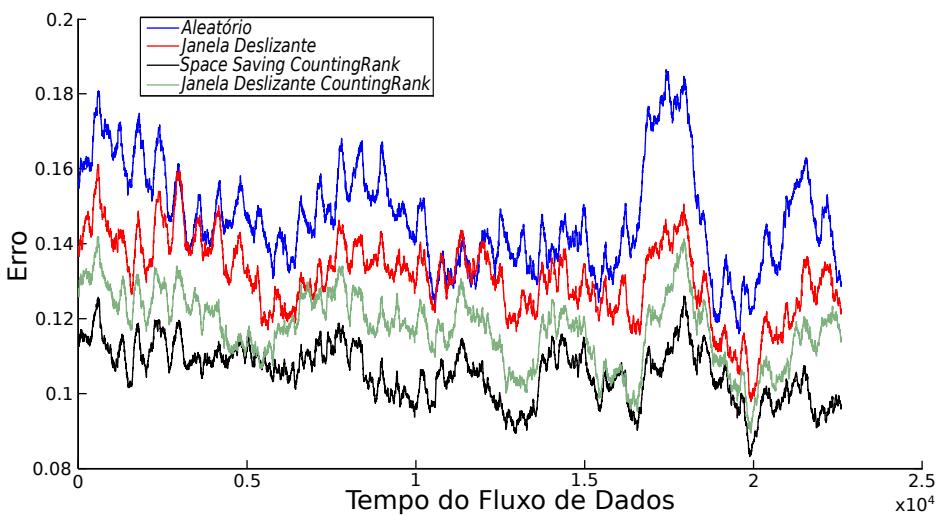


Figura 19 – Variação da taxa de erro ao longo do fluxo de dados no conjunto de dados *5CVT* para a configuração $m = 200$; $\alpha = 0.90$; $\beta = 0.75$

A Figura 20 mostra a variação da taxa de erro na base *UG_2C_5D*. Semelhante ao que foi exibido na configuração 2, nesta base de dados até aproximadamente 25000 exemplos gerados pelo fluxo, o desempenho dos métodos é bem próximo. Após esse número de exemplos gerados, o método *Space Saving CountingRank* apresenta melhor desempenho em quase todo o restante do fluxo. Apenas em alguns momentos que as curvas do erro voltam a se cruzarem. Nada que prejudique o desempenho melhor deste método. A curva do erro do método *Janela Deslizante CountingRank* fica próxima aos outros dois métodos durante todo o fluxo. Apresentando em alguns pontos do fluxo uma curva do erro mais baixa.

4.6 Considerações finais

Neste capítulo foram apresentadas algumas características dos algoritmos incrementais e como são definidos. Foram diferenciados os conceitos de tarefa de aprendizado incremental e algoritmo de aprendizado incremental. Foram apresentadas também algumas aplicações em que os algoritmos incrementais obtiveram bons resultados. E também algumas versões de algoritmos incrementais baseadas no algoritmo k -vizinhos mais próximos.

Em seguida foi apresentado dois métodos baseados no algoritmo k -vizinhos mais próximos e que apresentam propriedades incrementais e *anytime*. Os experimentos mostraram que em um fluxo de dados que apresenta poucas interrupções o método *Space Saving CountingRank*

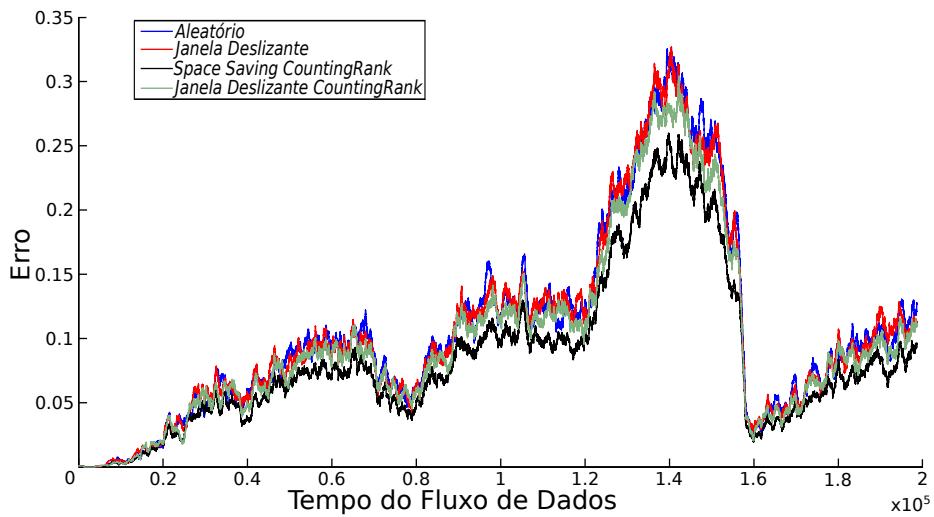


Figura 20 – Variação da taxa de erro ao longo do fluxo de dados no conjunto de dados *UG_2C_5D* para a configuração $m = 200$; $\alpha = 0.90$; $\beta = 0.75$

pode ser o mais indicado a ser utilizado e obter bons resultados. Isso independente da quantidade de exemplos de treino verificados pelo algoritmo antes da interrupção. Já em um fluxo que apresenta alta probabilidade de interrupção, este método apresentou bons resultados quando a quantidade de exemplos de treino verificados antes da interrupção foi alta. Em um fluxo com alta probabilidade de interrupção e baixa quantidade de exemplos de treino verificados pelo algoritmo antes da interrupção, o método *Janela Deslizante CountingRank* mostrou ser mais capaz de obter melhores resultados.

No capítulo a seguir será realizado uma conclusão sobre o trabalho que foi desenvolvido neste projeto e os resultados alcançados.



CONCLUSÃO

Realizar uma tarefa de aprendizado de máquina sobre um fluxo de dados não é uma tarefa simples. Os sistemas inteligentes devem estar preparados para operar sobre restrições impostas por este tipo de aplicação. Os principais desafios neste aprendizado é tratar o tempo de processamento variado e o espaço em memória limitado. Outra restrição que os sistemas inteligentes devem estar aptos a contornar é a mudança de conceito.

Algoritmos de classificação em lote não são adequados para serem diretamente utilizados em uma aplicação com estas características. Estes algoritmos geralmente possuem tempo de resposta fixa e constante, o que o impossibilita tratar todos os eventos gerados pelo fluxo de dados.

Por outro lado, os algoritmos de classificação *anytime* tem demonstrado bons resultados quando utilizados nestas aplicações. Estes algoritmos são capazes de utilizar, de uma melhor maneira, o tempo de processamento disponível entre cada evento do fluxo. Neste trabalho foi investigado uma versão do algoritmo *k*-vizinhos mais próximos *anytime* estado-da-arte (UENO *et al.*, 2006). O método *SimpleRank* ordena os exemplos de treino de acordo com sua contribuição para a classificação correta de instâncias desconhecidas. Este algoritmo apresentou bons resultados quando aplicado a vários conjuntos de dados de *benchmark*. Porém o critério de desempate utilizado pelo algoritmo apresenta várias desvantagens, como foi mostrado nos experimentos.

Então foi proposto um método de desempate para ser utilizado em conjunto com este algoritmo (LEMES *et al.*, 2014). O método *DiversityTieBreak* gera uma lista de exemplos ordenada pela posição no espaço d dimensional dos exemplos e utiliza esta lista para desempatar os exemplos que obtiveram o mesmo *score* calculado pelo fórmula utilizada pelo método *SimpleRank*. Os experimentos mostraram que este método de desempate aumentou consistentemente a acurácia alcançada pelo algoritmo em diversos conjuntos de dados de *benchmark*.

O algoritmo *k*-vizinhos mais próximos *anytime* investigado não possui habilidade de

tratar mudanças de conceito. Para atualizar o modelo de aprendizado utilizado pelo algoritmo é necessário realizar o cálculo do *rank* de várias instâncias do conjunto de treinamento. Em um fluxo de dados cujos dados são gerados constantemente e com alta variabilidade, o algoritmo pode não possuir tempo suficiente para realizar esta operação. Portanto, esse algoritmo é inadequado para ser utilizado em muitas aplicações de fluxo de dados com mudança de conceito.

Os algoritmos *incrementais* possuem a habilidade de atualizar modelos de aprendizado na medida que os exemplos são gerados pelo fluxo. Porém, estes algoritmos não possuem a habilidade de utilizarem da melhor forma possível o tempo de processamento disponível entre cada evento gerado pelo fluxo. Um algoritmo que possua as características dos algoritmos *anytime* e *incremental* é uma boa solução para esta situação.

Então foi proposto neste trabalho uma versão do algoritmo *k*-vizinhos mais próximo *anytime incremental*. Esta proposta também utiliza a ideia de que os exemplos de treino que melhor contribuem para a classificação correta de instâncias desconhecidas devem ser utilizados primeiro pelo algoritmo. Porém, o conjunto de treinamento é construído na medida em que os dados são gerados pelo fluxo. E quando o conjunto atinge o tamanho limite, um exemplo de treino é removido e um novo exemplo é adicionado.

Foram propostas duas abordagens para remover o exemplo de treino. A primeira é baseada no algoritmo *Space Saving*, um algoritmo para realizar contagem de elementos em um fluxo de dados. Nesta abordagem, *Space Saving CountingRank*, o intuito é de realizar a contagem de quantos vizinhos mais próximos o exemplo de treino possui. O exemplo de treino que possuir o menor valor de contador é removido e o novo exemplo é adicionado. A segunda proposta é baseada em uma *Janela Deslizante* sobre o fluxo de dados. O exemplo de treino mais antigo é removido e o novo exemplo é adicionado. Em ambas as propostas, os exemplos de treino são ordenados de acordo com a quantidade de vizinhos mais próximos de mesma classe que possuem.

Os resultados dos experimentos mostraram que em um fluxo de dados que apresenta baixa probabilidade de interrupção o método *Space Saving CountingRank* é o mais indicado a ser utilizado, independente da quantidade de exemplos verificados antes da interrupção. Em um fluxo que possua alta probabilidade de interrupção e a quantidade de exemplos verificados antes da interrupção seja alto, este método continuou apresentando os melhores resultados. Porém, se a quantidade de exemplos verificados antes da interrupção for baixa, este método apresentou o mais baixo desempenho. Neste fluxo, o mais indicado é utilizar o método *Janela Deslizante CountingRank*, que mostrou excelente desempenho em um fluxo com estas características. Este método também apresentou melhor desempenho quando comparado com os métodos *Aleatório* e *Janela Deslizante* nos demais fluxos de dados analisados.

5.1 Perpectivas Futuras

Os resultados deste trabalho mostram que o algoritmo *k*-vizinhos mais próximo *anytime* incremental pode ser aplicado a um problema de classificação em fluxo de dados e obter bons resultados. Seria interessante explorar no futuro outros classificadores incrementais mais sofisticados que o *k*-vizinhos mais próximos e tentar transformá-los em classificadores *anytime*. Uma possibilidade seria o *Hoeffding trees* (HULTEN; SPENCER; DOMINGOS, 2001), um algoritmo estado da arte capaz de processar fluxos de dados de alta velocidade.

REFERÊNCIAS

- AGGARWAL, C. C.; HAN, J.; WANG, J.; YU, P. S. On demand classification of data streams. In: ACM. **Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.], 2004. p. 503–508. Citado na página 51.
- AHA, D. W.; KIBLER, D.; ALBERT, M. K. Instance-based learning algorithms. **Machine learning**, Springer, v. 6, n. 1, p. 37–66, 1991. Citado 3 vezes nas páginas 44, 45 e 74.
- ARASU, A.; BABU, S.; WIDOM, J. Cql: A language for continuous queries over streams and relations. In: SPRINGER. **Database Programming Languages**. [S.l.], 2004. p. 1–19. Citado na página 37.
- BABCOCK, B.; BABU, S.; DATAR, M.; MOTWANI, R.; WIDOM, J. Models and issues in data stream systems. In: ACM. **Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems**. [S.l.], 2002. p. 1–16. Citado 3 vezes nas páginas 35, 36 e 38.
- BATISTA, G. E. A. P. A. Pré-processamento de dados em aprendizado de máquina supervisado. **Instituto de Ciências Matemáticas e de Computação, ICMC, São Carlos, SP**, 2003. Citado na página 46.
- BERINGER, J.; HÜLLERMEIER, E. An efficient algorithm for instance-based learning on data streams. In: **Proceedings of the 7th industrial conference on Advances in data mining: theoretical aspects and applications**. Berlin, Heidelberg: Springer-Verlag, 2007. (ICDM'07), p. 34–48. ISBN 978-3-540-73434-5. Disponível em: <<http://dl.acm.org/citation.cfm?id=1770770.1770776>>. Citado 2 vezes nas páginas 31 e 70.
- _____. Efficient instance-based learning on data streams. **Intelligent Data Analysis**, New York, NY: North-Holland, 1997-, v. 11, n. 6, p. 627–650, 2007. Citado na página 73.
- BØHLING, L.; BAILEY, N. P.; SCHRØDER, T. B.; DYRE, J. C. Estimating the density-scaling exponent of a monatomic liquid from its pair potential. **arXiv preprint arXiv:1401.2606**, 2014. Citado 2 vezes nas páginas 30 e 44.
- BOJINSKI, S.; VERSTRAETE, M.; PETERSON, T. C.; RICHTER, C.; SIMMONS, A.; ZEMP, M. The concept of essential climate variables in support of climate research, applications, and policy. **Bulletin of the American Meteorological Society**, 2014. Citado na página 41.
- BUTT, R.; JOHANSSON, S. J. Where do we go now?: anytime algorithms for path planning. In: ACM. **Proceedings of the 4th International Conference on Foundations of Digital Games**. [S.l.], 2009. p. 248–255. Citado na página 53.
- CHEN, H.; LU, J.; LI, Q.; LOU, C.; PAN, D.; YU, Z. A new evolutionary fuzzy instance-based learning approach: Application for detection of parkinson's disease. In: **Advances in Swarm and Computational Intelligence**. [S.l.]: Springer, 2015. p. 42–50. Citado na página 44.

- CHENG, M.-Y.; HOANG, N.-D. A swarm-optimized fuzzy instance-based learning approach for predicting slope collapses in mountain roads. **Knowledge-Based Systems**, Elsevier, v. 76, p. 256–263, 2015. Citado na página 44.
- CLARK, P.; NIBLETT, T. The cn2 induction algorithm. **Machine learning**, Springer, v. 3, n. 4, p. 261–283, 1989. Citado na página 72.
- CLI, D. T.; HARVEY, I.; HUSBANDS, P. Incremental evolution of neural network architectures for adaptive behaviour. In: **Proceedings of the European Symposium on Artificial Neural Networks (ESANN'93)**. [S.l.: s.n.], 1992. p. 39–44. Citado na página 73.
- COVER, T. M.; HART, P. E. Nearest neighbor pattern classification. **Information Theory, IEEE Transactions on**, IEEE, v. 13, n. 1, p. 21–27, 1967. Citado 2 vezes nas páginas 44 e 74.
- DASARATHY, B. V. Nosing around the neighborhood: A new system structure and classification rule for recognition in partially exposed environments. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, IEEE, n. 1, p. 67–71, 1980. Citado na página 45.
- DEAN, T. L.; BODDY, M. S. An analysis of time-dependent planning. In: **AAAI**. [S.l.: s.n.], 1988. v. 88, p. 49–54. Citado na página 52.
- DECOSTE, D. Anytime interval-valued outputs for kernel machines: Fast support vector machine classification via distance geometry. In: **ICML**. [S.l.: s.n.], 2002. p. 99–106. Citado na página 52.
- DELATTRE, M.; IMBERT, B. **Method for management of data stream exchanges in an autonomic telecommunications network**. [S.l.]: Google Patents, 2015. US Patent 8,949,412. Citado na página 36.
- DIEHL, C. P.; CAUWENBERGHS, G. Svm incremental learning, adaptation and optimization. In: **IEEE. Neural Networks, 2003. Proceedings of the International Joint Conference on**. [S.l.], 2003. v. 4, p. 2685–2690. Citado na página 73.
- DOMINGOS, P.; HULTEN, G. A general method for scaling up machine learning algorithms and its application to clustering. In: **ICML**. [S.l.: s.n.], 2001. p. 106–113. Citado na página 110.
- _____. A general framework for mining massive data streams. **Journal of Computational and Graphical Statistics**, Taylor & Francis, v. 12, n. 4, p. 945–949, 2003. Citado 2 vezes nas páginas 35 e 37.
- DONGRE, P. B.; MALIK, L. G. A review on real time data stream classification and adapting to various concept drift scenarios. In: **IEEE. Advance Computing Conference (IACC), 2014 IEEE International**. [S.l.], 2014. p. 533–537. Citado 2 vezes nas páginas 17 e 42.
- ESMEIR, S.; MARKOVITCH, S. Interruptible anytime algorithms for iterative improvement of decision trees. In: **ACM. Proceedings of the 1st international workshop on Utility-based data mining**. [S.l.], 2005. p. 78–85. Citado na página 52.
- GABER, M.; ZASLAVSKY, A.; KRISHNASWAMY, S. Mining data streams: a review. **ACM Sigmod Record**, ACM, v. 34, n. 2, p. 18–26, 2005. Citado na página 39.
- GAMA, J.; RODRIGUES, P. P.; SEBASTIÃO, R. Evaluating algorithms that learn from data streams. In: **ACM. Proceedings of the 2009 ACM symposium on Applied Computing**. [S.l.], 2009. p. 1496–1500. Citado 3 vezes nas páginas 76, 82 e 112.

- GAMA, J.; SEBASTIÃO, R.; RODRIGUES, P. P. On evaluating stream learning algorithms. **Machine Learning**, Springer, v. 90, n. 3, p. 317–346, 2013. Citado 2 vezes nas páginas 35 e 38.
- GANTER, B. **Two basic algorithms in concept analysis**. [S.l.]: Springer, 2010. Citado na página 70.
- GATES, G. W. The reduced nearest neighbor rule. Citeseer, 1972. Citado na página 45.
- GEISLER, S.; QUIX, C. Evaluation of real-time traffic applications based on data stream mining. In: **Data Mining for Geoinformatics**. [S.l.]: Springer, 2014. p. 83–103. Citado na página 40.
- GIRAUD-CARRIER, C. A note on the utility of incremental learning. **Aicomunications**, IOS Press, v. 13, n. 4, p. 215–223, 2000. Citado 3 vezes nas páginas 31, 71 e 72.
- GOBEILL, J.; GAUDINAT, A.; RUCH, P. Instance-based learning for tweet categorization in clef replab 2014. In: **Proceedings of Conference and Labs of the Evaluation Forum (CLEF)**. [S.l.: s.n.], 2014. p. 1491–1499. Citado na página 44.
- GOLAB, L.; ÖZSU, M. T. Issues in data stream management. **ACM Sigmod Record**, ACM, v. 32, n. 2, p. 5–14, 2003. Citado 2 vezes nas páginas 36 e 37.
- GRASS, J.; ZILBERSTEIN, S. Anytime algorithm development tools. **ACM SIGART Bulletin**, ACM, v. 7, n. 2, p. 20–27, 1996. Citado na página 52.
- GRUMBERG, O.; LIVNE, S.; MARKOVITCH, S. Learning to order bdd variables in verification. **Journal of Artificial Intelligence Research (JAIR)**, v. 18, p. 83–116, 2003. Citado na página 54.
- HART, P. E. **The condensed nearest neighbor rule**. [S.l.]: Institute of Electrical and Electronics Engineers and Transactions on Information Theory, 1968. v. 14. Citado na página 45.
- HOY, M.; MATVEEV, A. S.; SAVKIN, A. V. Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey. **Robotica**, Cambridge Univ Press, v. 33, n. 03, p. 463–497, 2015. Citado na página 71.
- HU, B.; CHEN, Y.; KEOGH, E. Classification of streaming time series under more realistic assumptions. **Data Mining and Knowledge Discovery**, Springer, p. 1–35, 2015. Citado na página 30.
- HUI, B.; YANG, Y.; WEBB, G. I. Anytime classification for a pool of instances. **Machine learning**, Springer, v. 77, n. 1, p. 61–102, 2009. Citado na página 52.
- HULTEN, G.; DOMINGOS, P. Mining complex models from arbitrarily large databases in constant time. In: ACM. **Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. [S.l.], 2002. p. 525–531. Citado na página 52.
- HULTEN, G.; SPENCER, L.; DOMINGOS, P. Mining time-changing data streams. In: **ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining**. [S.l.]: ACM Press, 2001. p. 97–106. Citado na página 99.
- JANANI, J.; GOMATHI, S. “formulating and implementing competency modelling, profiling and mapping” at private limited, ranipet, vellore. **Mediterranean Journal of Social Sciences**, v. 6, n. 1, p. 23, 2015. Citado na página 71.

- JOU, C. Spam e-mail classification based on the ifwb algorithm. In: **Intelligent Information and Database Systems**. [S.l.]: Springer, 2013. p. 314–324. Citado na página 41.
- KENKRE, P. S.; PAI, A.; COLACO, L. Real time intrusion detection and prevention system. In: SPRINGER. **Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014**. [S.l.], 2015. p. 405–411. Citado 2 vezes nas páginas 29 e 36.
- KOHAVI, R.; PROVOST, F. Glossary of terms. **Machine Learning**, v. 30, n. 2-3, p. 271–274, 1998. Citado na página 69.
- KOURIE, D. G.; OBIEDKOV, S.; WATSON, B. W.; MERWE, D. van der. An incremental algorithm to construct a lattice of set intersections. **Science of Computer Programming**, Elsevier, v. 74, n. 3, p. 128–142, 2009. Citado na página 70.
- KRANEN, P.; SEIDL, T. Harnessing the strengths of anytime algorithms for constant data streams. **Data Mining and Knowledge Discovery**, Springer, v. 19, n. 2, p. 245–260, 2009. Citado na página 52.
- KRANJC, J.; SMAILOVIĆ, J.; PODPEČAN, V.; GRČAR, M.; ŽNIDARŠIĆ, M.; LAVRAČ, N. Active learning for sentiment analysis on data streams: Methodology and workflow implementation in the crowdflows platform. **Information Processing & Management**, Elsevier, v. 51, n. 2, p. 187–203, 2015. Citado 2 vezes nas páginas 29 e 36.
- KUMAR, V.; DIXIT, A.; JAVALGI, R. R. G.; DASS, M. Research framework, strategies, and applications of intelligent agent technologies (iats) in marketing. **Journal of the Academy of Marketing Science**, Springer, p. 1–22, 2015. Citado na página 71.
- LAW, Y.-N.; ZANIOLLO, C. An adaptive nearest neighbor classification algorithm for data streams. In: **Knowledge Discovery in Databases: PKDD 2005**. [S.l.]: Springer, 2005. p. 108–120. Citado na página 70.
- LEMES, C. I.; SILVA, D. F.; BATISTA, G. E. *et al.* Adding diversity to rank examples in anytime nearest neighbor classification. In: IEEE. **Machine Learning and Applications (ICMLA), 2014 13th International Conference on**. [S.l.], 2014. p. 129–134. Citado 5 vezes nas páginas 33, 58, 59, 67 e 97.
- LICHMAN, M. **UCI Machine Learning Repository**. 2013. Disponível em: <<http://archive.ics.uci.edu/ml>>. Citado na página 61.
- LIN, J.; VLACHOS, M.; KEOGH, E.; GUNOPULOS, D. Iterative incremental clustering of time series. In: **Advances in Database Technology-EDBT 2004**. [S.l.]: Springer, 2004. p. 106–122. Citado 2 vezes nas páginas 52 e 53.
- LIU, C.-L.; WELLMAN, M. P. On state-space abstraction for anytime evaluation of bayesian networks. **ACM SIGART Bulletin**, ACM, v. 7, n. 2, p. 50–57, 1996. Citado na página 52.
- MALEKIAN, D.; HASHEMI, M. R. An adaptive profile based fraud detection framework for handling concept drift. In: IEEE. **Information Security and Cryptology (ISCISC), 2013 10th International ISC Conference on**. [S.l.], 2013. p. 1–6. Citado na página 41.
- MAO, C.; HU, B.; MOORE, P.; SU, Y.; WANG, M. Nearest neighbor method based on local distribution for classification. In: **Advances in Knowledge Discovery and Data Mining**. [S.l.]: Springer, 2015. p. 239–250. Citado 2 vezes nas páginas 30 e 44.

- MERWE, D. V. D.; OBIEDKOV, S.; KOURIE, D. Addintent: A new incremental algorithm for constructing concept lattices. In: **Concept Lattices**. [S.l.]: Springer, 2004. p. 372–385. Citado na página 70.
- METWALLY, A.; AGRAWAL, D.; ABBADI, A. E. Efficient computation of frequent and top-k elements in data streams. In: **Database Theory-ICDT 2005**. [S.l.]: Springer, 2005. p. 398–412. Citado 4 vezes nas páginas 46, 47, 48 e 49.
- MINKU, L.; WHITE, A.; YAO, X. The impact of diversity on online ensemble learning in the presence of concept drift. **Knowledge and Data Engineering, IEEE Transactions on**, IEEE, v. 22, n. 5, p. 730–742, 2010. Citado 3 vezes nas páginas 42, 43 e 49.
- MYERS, K.; KEARNS, M.; SINGH, S.; WALKER, M. A. A boosting approach to topic spotting on subdialogues. **Family Life**, Citeseer, v. 27, n. 3, p. 1, 2000. Citado na página 52.
- OUTRATA, J.; VYCHODIL, V. Fast algorithm for computing fixpoints of galois connections induced by object-attribute relational data. **Information Sciences**, Elsevier, v. 185, n. 1, p. 114–127, 2012. Citado na página 70.
- QUINLAN, J. R. Induction of decision trees. **Machine learning**, Springer, v. 1, n. 1, p. 81–106, 1986. Citado na página 72.
- RODRIGUES, P. P.; CORREIA, R. C. Streaming virtual patient records. In: **Proceedings of the European conference on machine learning and principles and practice of knowledge discovery in databases**. [S.l.: s.n.], 2013. Citado 2 vezes nas páginas 29 e 36.
- RUMELHART, D. E.; MCCLELLAND, J. L.; GROUP, P. R. *et al.* **Parallel distributed processing**. [S.l.]: IEEE, 1988. v. 1. Citado na página 72.
- SALGANICOFF, M. Tolerating concept and sampling shift in lazy learning using prediction error context switching. **Artificial Intelligence Review**, Springer, v. 11, n. 1-5, p. 133–155, 1997. Citado 2 vezes nas páginas 40 e 74.
- SAMANTHULA, B. K.; ELMEHDWI, Y.; JIANG, W. k-nearest neighbor classification over semantically secure encrypted relational data. **arXiv preprint arXiv:1403.5001**, 2014. Citado na página 30.
- SERRÀ, J.; ARCOS, J. L. Particle swarm optimization for time series motif discovery. **arXiv preprint arXiv:1501.07399**, 2015. Citado na página 30.
- SHAH, R.; KRISHNASWAMY, S.; GABER, M. Resource-aware very fast k-means for ubiquitous data stream mining. 2005. Citado na página 51.
- SHIEH, J.; KEOGH, E. Polishing the right apple: Anytime classification also benefits data streams with constant arrival times. In: **Data Mining (ICDM), 2010 IEEE 10th International Conference on**. [S.l.: s.n.], 2010. p. 461–470. ISSN 1550-4786. Citado 3 vezes nas páginas 51, 54 e 61.
- SOUZA, V. M. A.; SILVA, D. F.; GAMA, J.; BATISTA, G. E. A. P. A. Data stream classification guided by clustering on nonstationary environments and extreme verification latency. In: **Proceedings of SIAM International Conference on Data Mining (SDM)**. [S.l.: s.n.], 2015. p. 873 –881. Citado na página 80.

SOUZA, V. M. A. de; SILVA, D. F.; BATISTA, G. E. A. P. A. Classification of data streams applied to insect recognition: Initial results. In: **Proceedings of the 2nd Brazilian Conference on Intelligent Systems (BRACIS)**. [S.l.: s.n.], 2013. p. 1–6. Citado 3 vezes nas páginas 38, 44 e 51.

TERRY, D.; GOLDBERG, D.; NICHOLS, D.; OKI, B. **Continuous queries over append-only databases**. [S.l.]: ACM, 1992. v. 21. Citado na página 37.

TRAWIŃSKI, B.; SMETEK, M.; TELEC, Z.; LASOTA, T. Nonparametric statistical analysis for multiple comparison of machine learning regression algorithms. **International Journal of Applied Mathematics and Computer Science**, v. 22, n. 4, p. 867–881, 2012. Citado na página 65.

TSYMBAL, A. **The problem of concept drift: Definitions and related work**. [S.l.], 2004. Citado 2 vezes nas páginas 40 e 41.

UENO, K.; XI, X.; KEOGH, E.; LEE, D.-J. Anytime classification using the nearest neighbor algorithm with applications to stream mining. In: **Data Mining, 2006. ICDM '06. Sixth International Conference on**. [S.l.: s.n.], 2006. p. 623–632. ISSN 1550-4786. Citado 14 vezes nas páginas 30, 31, 32, 33, 49, 52, 53, 54, 56, 57, 64, 67, 70 e 97.

UTGOFF, P. E. Id5: An incremental id3. **Proceedings of the Fifth International Workshop on Machine Learning**, Morgan Kaufmann, p. 107–120, 1988. Citado na página 73.

WANG, F.; HU, L.; ZHOU, D.; SUN, R.; HU, J.; ZHAO, K. Estimating online vacancies in real-time road traffic monitoring with traffic sensor data stream. **Ad Hoc Networks**, Elsevier, 2015. Citado na página 36.

WEBB, G. I.; YANG, Y.; BOUGHTON, J.; KORB, K.; TING, K. M. **Classifying under computational resource constraints: Anytime classification using probabilistic estimators**. [S.l.], 2005. Citado na página 54.

WIDMER, G.; KUBAT, M. Learning in the presence of concept drift and hidden contexts. **Machine learning**, Springer, v. 23, n. 1, p. 69–101, 1996. Citado 2 vezes nas páginas 40 e 76.

XI, X.; KEOGH, E.; SHELTON, C.; WEI, L.; RATANAMAHATANA, C. A. Fast time series classification using numerosity reduction. In: ACM. **Proceedings of the 23rd international conference on Machine learning**. [S.l.], 2006. p. 1033–1040. Citado na página 56.

YANG, Y.; WEBB, G.; KORB, K.; TING, K. M. Classifying under computational resource constraints: anytime classification using probabilistic estimators. **Machine Learning**, Springer, v. 69, n. 1, p. 35–53, 2007. Citado na página 52.

ZHOU, L.; JIANG, J.; LIAO, R.; YANG, T.; WANG, C. Fpga based low-latency market data feed handler. In: **Computer Engineering and Technology**. [S.l.]: Springer, 2015. p. 69–77. Citado na página 36.

ZILBERSTEIN, S. Using anytime algorithms in intelligent systems. **AI magazine**, v. 17, n. 3, p. 73, 1996. Citado na página 53.

ZILBERSTEIN, S.; RUSSELL, S. Approximate reasoning using anytime algorithms. **Inprecise and Approximate Computation**, Springer, p. 43–62, 1995. Citado na página 52.

ZLIOBAITE, I. **Learning under concept drift: an overview**. Lithuania, 2009. Citado 3 vezes nas páginas 41, 42 e 49.

ZOU, L.; ZHANG, Z.; LONG, J. A fast incremental algorithm for constructing concept lattices. **Expert Systems with Applications**, Elsevier, v. 42, n. 9, p. 4474–4481, 2015. Citado na página 71.

ZUO, Y.-C.; SU, W.-X.; ZHANG, S.-H.; WANG, S.-S.; WU, C.-Y.; YANG, L.; LI, G.-P. Discrimination of membrane transporter protein types using k-nearest neighbor method derived from the similarity distance of total diversity measure. **Molecular BioSystems**, Royal Society of Chemistry, v. 11, n. 3, p. 950–957, 2015. Citado 2 vezes nas páginas 30 e 44.

ANÁLISE EXPERIMENTAL *SPACE SAVING COUNTINGRANK*

Esta análise experimental foi realizada em três etapas. Na primeira etapa foi utilizado conjuntos de dados sintéticos que não apresentam mudanças de conceito. Esta etapa foi importante para verificar se método *Space Saving CountingRank* apresentava bons resultados na classificação em fluxo de dados estacionários. Na segunda etapa experimental foram utilizados conjuntos de dados sintéticos e um conjunto de dado real que apresentam mudança de conceito. Assim é possível avaliar a eficácia do método em fluxos de dados não estacionários. Na terceira etapa foram utilizados os mesmos conjuntos de dados com mudança de conceito da etapa anterior, porém realizando uma pequena alteração na fórmula utilizada pelo método proposto.

A seguir são descritas as análises experimentais realizadas em cada etapa. Na Seção A.1 são descritos os experimentos com fluxo de dados sem mudança de conceito. Os experimentos com fluxo de dados com mudança de conceito estão descritos na Seção A.2. Na Seção A.3 são descritos os experimentos com fluxos de dados com mudança de conceito e a fórmula utilizada pelo método alterada. Por fim, é realizado uma breve conclusão sobre este apêndice na Seção A.4.

A.1 Análise experimental – Sem mudança de conceito

A primeira versão do algoritmo *k*-vizinhos mais próximos *anytime* incremental utilizava apenas uma estratégia de atualização do modelo de aprendizado. Nesta versão foi utilizada apenas a estratégia baseada no algoritmo *Space Saving*. Este algoritmo foi adaptado para ser utilizado em problemas de classificação *anytime* incremental. A descrição destas alterações foram exibidas no Capítulo 4.

Nesta etapa experimental foi utilizada apenas a versão já descrita do algoritmo *Space Saving com CountingRank*, com exceção da fórmula utilizada pelo algoritmo. A equação utilizada

por este algoritmo só foi proposta após toda esta primeira análise experimental ser finalizada. A Equação A.1 representa a equação utilizada nesta primeira etapa de análise experimental.

$$counter_t(x) = counter_{t-1}(x) + \begin{cases} 1 & , \text{if } classe(x) = classe(x_j); \\ 0 & , \text{caso contrário.} \end{cases} \quad (\text{A.1})$$

onde x é a instância a ter o seu contador atualizado e x_j é a instância gerada pelo fluxo de dados que tem a instância x como o seu vizinho mais próximo. $counter_{t-1}(x)$ é o contador da instância x antes de x_j ser adicionado ao conjunto de treino e $counter_t(x)$ é o contador após a adição de x_j .

Com esta fórmula os valores dos *ranks* dos exemplos de treino podem ser facilmente recalculados. Basta identificar o vizinho mais próximo do novo exemplo de treino e incrementar o seu contador. Porém, esta equação não penaliza os exemplos de treino que não contribuem para a classificação correta de novas instâncias. Esta fase da análise foi realizada com o propósito de verificar se o método proposto era capaz de apresentar bons resultados comparado a outros métodos em um problema simples de classificação em fluxo de dados.

Para avaliar a eficácia deste método, ele foi comparado com os métodos *Aleatório* e *Janela Deslizante*. Ambos os métodos tem como objetivo principal selecionar as m instâncias do fluxo de dados que irão compor o conjunto de treinamento utilizado pelo algoritmo k -vizinhos mais próximos *anytime* e atualizar o modelo de aprendizado do algoritmo. Estes dois métodos foram descritos na análise experimental realizada na Seção 4.5.

A.1.1 Configuração experimental

A primeira parte da avaliação utiliza os dados sintéticos gerados conforme o método proposto por Domingos e Hulten (2001). Neste método todos os dados são gerados por misturas de gaussianas. Os dados foram gerados de acordo com os seguintes parâmetros:

- Médias (μ_k): são as médias das distribuições gaussianas utilizadas para cada uma das dimensões. Cada dimensão é independente das demais;
- Desvio padrão (σ): também utilizado na distribuição gaussiana para controlar a sobreposição entre os exemplos de diferentes classes. Quanto maior o valor deste parâmetro, maior será a área sobreposta entre os exemplos. O valor deste parâmetro varia entre $\{0.01, 0.05, 0.1\}$ nos experimentos realizados neste trabalho;
- Dimensionalidade (d): número de dimensões dos exemplos, que é considerado o número de atributos de cada exemplo. O valor deste parâmetro varia entre $\{2, 4, 6, 8, 10\}$;
- Número de classes (c): número de componentes gaussianas. Pode ser considerado também como sendo o número de classes diferentes a serem geradas nos experimentos. Os número de classes variam entre $\{2, 3, 4, 5\}$.

As médias μ_k são escolhidas de forma independente amostrando um valor uniformemente no intervalo $(2\sigma, 1 - 2\sigma)$. Esses valores foram escolhidos de maneira que os dados gerados ficassem dentro de um hipercubo unitário, mas os valores gerados pelas distribuições gaussianas que ficaram fora do hipercubo não foram removidos. As médias de uma componente gaussiana μ_k que tiver o valor de sua distância para as demais médias geradas anteriormente menor que $(\sqrt{d}/c)\sigma$ são rejeitadas e geradas novamente. Isso garante que as médias não são demasiadamente próximas, o que poderia tornar o problema de difícil aprendizado. Os exemplos x foram gerados escolhendo uma das médias μ_k , com probabilidade uniforme. Para cada dimensão do exemplo x_d é gerado um valor por uma distribuição gaussiana com média μ_{kd} e desvio padrão σ . Na Figura 21 é possível verificar dois exemplos de conjuntos de dados gerados pelo método descrito aqui. Foram gerados apenas alguns pontos para visualizar a distribuição dos dados gerados.

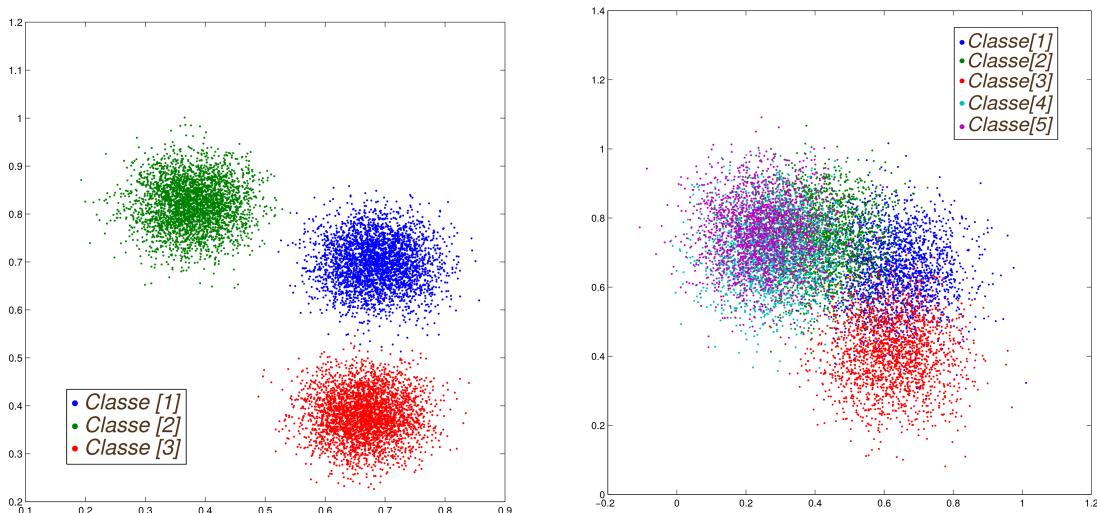


Figura 21 – Gráficos de um conjunto de dados gerados pelo método utilizado na primeira etapa dos experimentos. Na primeira figura tem alguns pontos gerados para a **Configuração**: $[d: 2; c: 3; \sigma: 0.05]$; e na segunda tem alguns pontos gerados para a **Configuração**: $[d: 2; c: 5; \sigma: 0.1]$.

Para simular uma avaliação de um algoritmo *anytime*, é necessário decidir se uma determinada classificação deve ou não ser interrompida e quando deve ser realizada a interrupção. Para isso, a avaliação também inclui os parâmetros α e β . O primeiro define se o fluxo de dados deve gerar um novo exemplo antes do anterior ser completamente processado, ou seja, este parâmetro define se o algoritmo *anytime* deve ou não ser interrompido no meio do seu processamento. Já o segundo define quantos exemplos de treino devem ser processados pelo algoritmo *anytime* antes dele ser interrompido.

Estes dois parâmetros foram utilizados nos experimentos com as seguintes configurações:

- α : indica a probabilidade de interrupção do algoritmo *anytime* para um determinado exemplo de teste. Ou seja, este parâmetro determina se o algoritmo *anytime* será executado

por completo ou interrompido em algum momento. As probabilidades de interrupções utilizadas foram $\{0.0, 0.01, 0.05, 0.1, 0.3, 0.5, 0.7, 0.9\}$.

- β : indica a quantidade de exemplos processados antes de interromper o algoritmo *anytime*. Quanto maior seu valor, maior será a quantidade de instâncias verificadas antes da interrupção. Seu valor varia entre $\{0.1, 0.2, 0.5, 0.75, 0.95\}$. O valor de β é multiplicado pelo valor de m (número de exemplos em memória) para calcular o número de exemplos a serem processados. Para este parâmetro foi utilizado também uma distribuição normal. Assim é possível simular um fluxo de dados real, onde não se conhece a priori a quantidade de instâncias que o algoritmo deve verificar antes da interrupção.

O tamanho do conjunto de treinamento (m) foi escolhido no intervalo $\{100, 250, 500, 1000, 2500, 5000\}$. O fluxo de dados foi gerado com 1.000.000 de exemplos. Cada experimento foi repetido 10 vezes para diminuir a variância dos resultados.

Nesta etapa inicial de experimentos, não foi incluída nenhuma mudança de conceito na geração dos dados. Dessa maneira, é possível analisar o comportamento do algoritmo *Space Saving com CountingRank* em um cenário estacionário.

A.1.2 Avaliação

A avaliação realizada neste experimento utiliza o *prequential error* (GAMA; RODRIGUES; SEBASTIÃO, 2009). O erro é calculado pela divisão do número total de erros que o algoritmo cometeu pelo total de instâncias geradas pelo fluxo de dados até o momento, de acordo com a Equação A.2. Os valores de erro são atualizados para cada nova instância de maneira que é possível visualizar as variações das taxas de erro durante todo o fluxo de dados.

$$erro_t = \frac{total_erros_t}{total_instancias_geradas_t} \quad (\text{A.2})$$

Na primeira avaliação foram gerados dois gráficos para cada configuração. O primeiro mostra apenas a variação da taxa de erro durante o fluxo de dados. Já o segundo mostra a relação logarítmica da taxa de erro entre duas técnicas. Por exemplo, a taxa de erro obtida pelo método *Space Saving com CountingRank* ($taxa_erro_a$) e Aleatório ($taxa_erro_b$), $\ln(taxa_erro_a)/\ln(taxa_erro_b)$. Neste último gráfico, se a curva permanecer próxima do valor 1 indica que ambos os métodos apresentam desempenho semelhante. Caso a curva permaneça acima de 1 indica que o método *a* apresenta melhor desempenho que o método *b*. Caso contrário ocorre o inverso, o método *b* apresenta melhor desempenho que o método *a*.

As taxas de erro obtidas com esta fórmula utilizam todos os erros obtidos durante o fluxo de dados. Assim, os erros obtidos no início do fluxo podem influenciar o cálculo do erro no final do fluxo. Dessa maneira, esta avaliação não seria indicada para medir o desempenho dos métodos em uma mudança de conceito. Mesmo que nessa etapa dos experimentos os dados não

sofram nenhuma alteração, é importante utilizar a mesma forma de avaliação que seria utilizada em experimentos com mudança de conceito. Foi realizada uma segunda análise dos resultados, também utilizando a Equação A.2, porém aplicando a fórmula em uma janela deslizante sobre o fluxo de dados. Com isso, o cálculo do erro é realizado considerando apenas os erros presentes nesta janela deslizante. O tamanho da janela deslizante para o cálculo do erro varia de acordo com os valores do parâmetro m .

Esta segunda maneira de avaliar utiliza um gráfico que relaciona o erro obtido pelos métodos variando um determinado parâmetro, por exemplo a probabilidade de interrupção do algoritmo (α). Para cada valor do parâmetro selecionado é calculado a sua média e intervalo de confiança do erro obtido por cada método. Considera-se o valor do erro obtido pela última janela deslizante para este cálculo. Com este gráfico é possível visualizar a diferença de desempenho entre os métodos variando um parâmetro de configuração. Foi gerado um gráfico diferente para cada valor do parâmetro σ .

A.1.3 Resultados

A seguir serão apresentados alguns gráficos gerados na primeira forma de avaliação. A Figura 22 mostra o desempenho alcançado pelos métodos analisados em uma configuração que apresenta baixa probabilidade de interrupção. No primeiro gráfico pode ser visto que o método proposto apresentou menor taxa de erro durante praticamente todo o fluxo de dados. A relação de desempenho entre o método *Space Saving com CountingRank* e os demais permaneceu acima de 1 durante todo o fluxo. Isso demonstra a eficácia do método proposto. Vale notar que a relação da taxa de erro entre os métodos *Aleatório* e *Janela Deslizante* permaneceu próximo do valor 1 durante todo o fluxo, o que indica que estes métodos apresentaram desempenhos semelhantes.

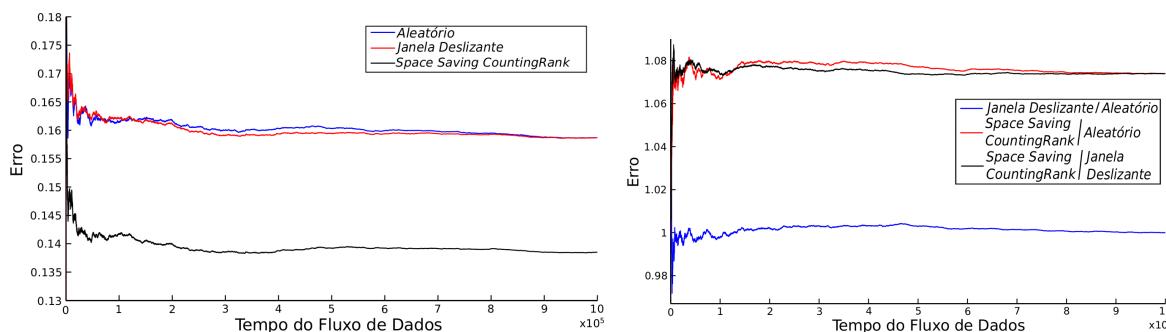


Figura 22 – **Configuração:** [$\sigma: 0.1$; $d: 2$; $c: 2$; $m: 1000$; $\alpha: 0.05$; $\beta: -1.0$]. O primeiro gráfico mostra a taxa de erro total obtida pelos métodos *Aleatório*, *Janela Deslizante* e *Space Saving com CountingRank*; O segundo gráfico mostra a relação logarítmica da taxa de erro [$\ln(\text{taxa_erro}_1)/\ln(\text{taxa_erro}_2)$] entre os métodos.

A Figura 23 apresenta o resultado para uma configuração que possui alta probabilidade de interrupção. No primeiro gráfico pode ser visto que o método *Space Saving com CountingRank* continua apresentando a menor taxa de erro. Isso se confirma na segunda figura, onde a curva com a relação deste método com os demais ficou acima do valor 1. A relação de desempenho entre os métodos *Aleatório* e *Janela Deslizante* permaneceu próxima do valor 1.

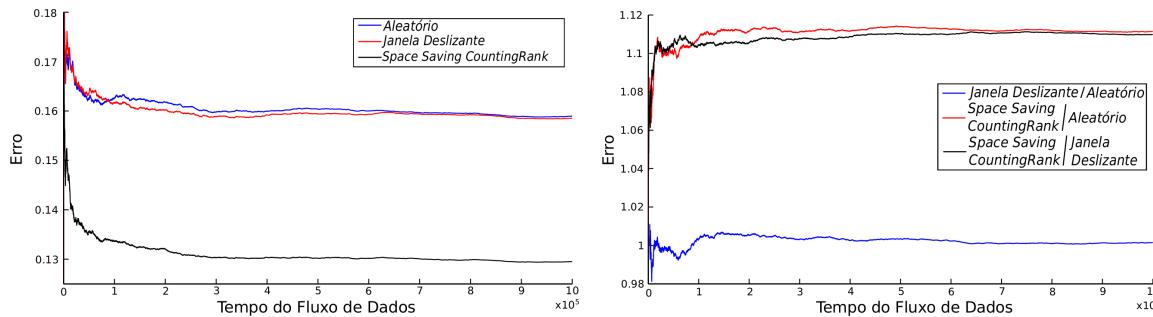


Figura 23 – **Configuração:** $[\sigma: 0.1; d: 2; c: 2; m: 2500; \alpha: 0.30; \beta: 0.20]$. O primeiro gráfico mostra a taxa de erro total obtida pelos métodos Aleatório, Janela Deslizante e Space Saving com CountingRank; O segundo gráfico mostra a relação logarítmica da taxa de erro $[\ln(\text{taxa_erro}_1)/\ln(\text{taxa_erro}_2)]$ entre os métodos.

A Figura 24 apresenta o resultado para uma configuração que apresenta alta probabilidade de interrupção. Além disso, a quantidade de exemplos de treino verificados antes da interrupção é baixa. Mesmo em um fluxo com estas características, o método Space Saving CountingRank apresentou a menor taxa de erro, como pode ser visto na primeira figura. Na segunda figura a curva da relação deste método com os demais ficou bem acima de 1, o que confirma a sua superioridade.

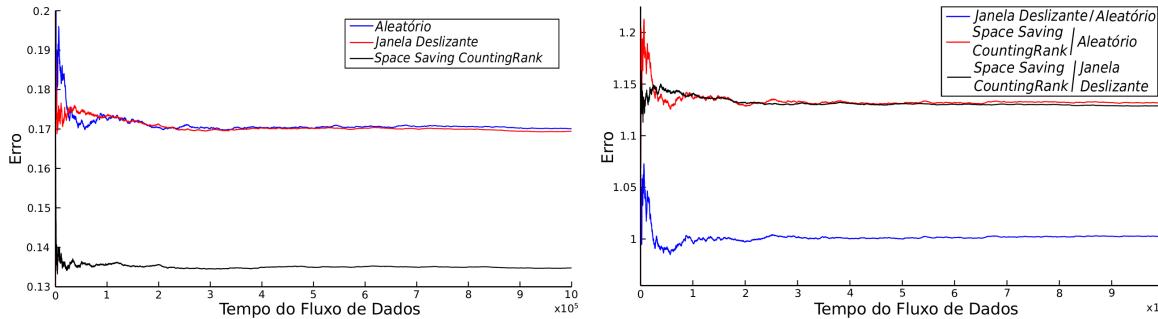


Figura 24 – **Configuração:** $[\sigma: 0.1; d: 2; c: 2; m: 250; \alpha: 0.90; \beta: 0.10]$. O primeiro gráfico mostra taxa de erro total obtida pelos métodos Aleatório, Janela Deslizante e Space Saving com CountingRank; O segundo gráfico mostra a relação logarítmica da taxa de erro $[\ln(\text{taxa_erro}_1)/\ln(\text{taxa_erro}_2)]$ entre os métodos.

A partir de agora são apresentados alguns gráficos gerados na segunda forma de avaliação. A Figura 25 apresenta dois gráficos onde a janela para o cálculo do erro tem $1/4$ do tamanho do conjunto de treinamento. O primeiro gráfico apresenta o desempenho dos métodos variando o parâmetro m para o valor $\sigma = 0.05$. O valor da média da taxa de erro obtida pelo método Space Saving com CountingRank ficou abaixo do valor mínimo do intervalo de confiança dos demais métodos em todos os valores do parâmetro. O segundo gráfico apresenta as médias das taxas de erro variando o parâmetro β para o valor $\sigma = 0.1$. O valor da média da taxa de erro obtida pelo método proposto permanece abaixo dos demais métodos, apresentando em poucos valores do parâmetro uma área sobreposta dos intervalos de confiança.

A Figura 26 apresenta dois gráficos onde a janela para o cálculo do erro tem o mesmo tamanho do conjunto de treinamento. O parâmetro utilizado no primeiro gráfico foi o α para o valor $\sigma = 0.05$. A média da taxa de erro obtida pelo método proposto aqui ficou abaixo dos demais métodos, apresentando pouquíssimas interseções entre os intervalos de confiança. Estas

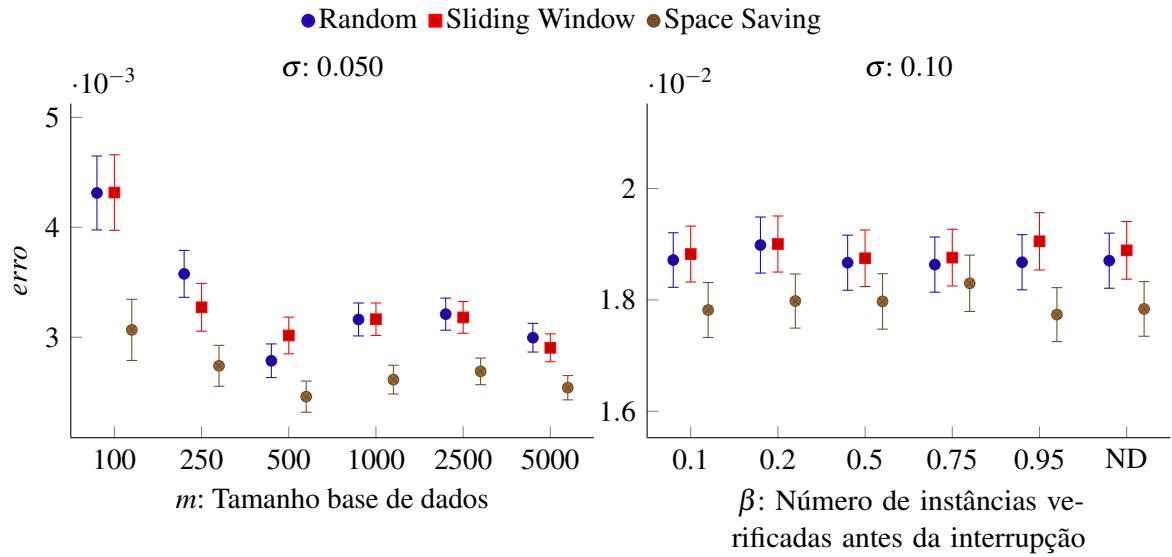


Figura 25 – Média e intervalo de confiança da taxa de erro obtida pelos métodos. No primeiro gráfico tem-se a variação do parâmetro m – Tamanho da base de treinamento; no segundo tem-se a variação do parâmetro β – número de instâncias verificadas antes da interrupção. A janela para o cálculo do erro tem $1/4$ do tamanho do conjunto de treinamento.

interseções ocorreram nos primeiros valores do parâmetro, onde há pouca interrupção e os métodos operam de forma semelhante avaliando todo o conjunto de treinamento. O segundo gráfico varia o parâmetro c para o valor $\sigma = 0.1$. Em um dos valores do parâmetro o método *Space Saving com CountingRank* não apresentou melhor desempenho que os outros métodos. Porém nos demais valores deste parâmetro este método apresentou melhor desempenho. O intervalo de confiança neste gráfico é pequeno, o que indica que a taxa de erro não varia tanto.

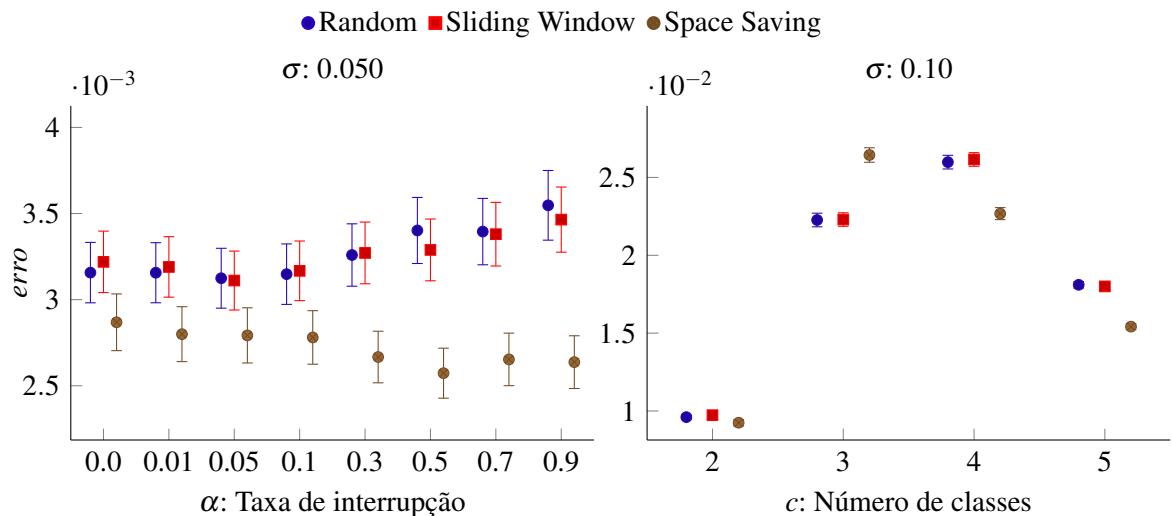


Figura 26 – Média e intervalo de confiança da taxa de erro obtida pelos métodos. No primeiro gráfico tem-se a variação do parâmetro α – Taxa de interrupção; no segundo tem-se a variação do parâmetro c – número de classes. A janela para o cálculo do erro é do tamanho do conjunto de treinamento.

A Figura 27 apresenta dois gráficos onde a janela para o cálculo do erro é igual a 3 vezes o tamanho do conjunto de treinamento. O primeiro gráfico foi utilizado o parâmetro β com o

valor $\sigma = 0.05$. A média da taxa de erro obtida pelo método proposto se manteve abaixo dos outros métodos em todos os valores deste parâmetro e não houve interseção entre os intervalos de confiança. O parâmetro d foi utilizado no segundo gráfico com o valor $\sigma = 0.1$. O método *Space Saving com CountingRank* apresentou pior desempenho apenas em um dos valores deste parâmetro. Este método apresentou melhor desempenho nos demais valores do parâmetro. O intervalo de confiança neste gráfico também é pequeno, ou seja a taxa de erro não varia tanto.

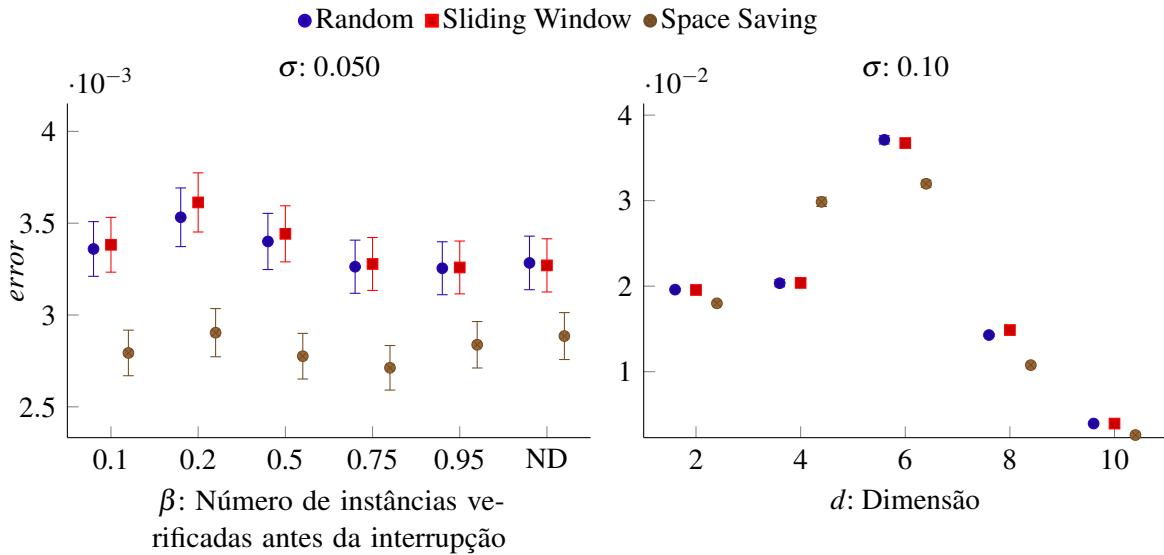


Figura 27 – Média e intervalo de confiança da taxa de erro obtida pelos métodos. No primeiro gráfico tem-se a variação do parâmetro β – número de instâncias verificadas antes da interrupção; no segundo tem-se a variação do parâmetro d – dimensionalidade. A janela para o cálculo do erro tem 3 vezes o tamanho da base de treinamento

Embora não seja possível mostrar os resultados para todas as possíveis configurações, o método proposto teve um desempenho superior aos demais para a grande maioria dos experimentos. Os resultados positivos desta primeira etapa experimental motivaram o pesquisador a executar a segunda etapa do experimento, desta vez utilizando bases de dados que apresentassem mudança de conceito.

A.2 Análise experimental – Com mudança de conceito

Nesta etapa experimental também foi utilizada apenas uma das estratégias de atualização do modelo de aprendizado, o *Space Saving com CountingRank*. O método utilizado nesta análise não sofreu nenhuma alteração. A fórmula utilizada pelo algoritmo também se manteve a mesma descrita na análise anterior. Com exceção de que, nesta etapa experimental foram utilizadas bases de dados sintéticas com mudanças de conceito, além de uma base de dados real com mudanças de conceito. Foram utilizados os mesmos métodos *Aleatório* e *Janela Deslizante* para comparar o desempenho obtido pelo método proposto.

Esta análise experimental é idêntica a realizada no Capítulo 4. Devido a sua semelhança, será utilizada a mesma configuração experimental utilizada neste capítulo (Seção 4.5.1). In-

cluindo os parâmetros descritos nesta configuração. E também será utilizada a mesma avaliação realizada neste capítulo nesta etapa experimental (Seção 4.5.2).

Não foi realizada uma análise muito detalhada, pois esta é uma etapa experimental parcial que foi responsável por determinar os métodos propostos neste trabalho. Então nestes experimentos foi realizada a análise de duas configurações, ambas apresentando alta probabilidade de ocorrência de interrupção. A *Configuração 1* apresenta baixo número de exemplos de treino verificados em uma interrupção. A *Configuração 2* verifica um alto número de exemplos de treino em uma interrupção. A seguir será apresentado os resultados para estas configurações.

A.2.1 Resultado Configuração 1: $m = 200$; $\alpha = 0.90$; $\beta = 0.20$

A Tabela 10 apresenta as áreas sobre a curva do erro em todas as bases de dados analisadas. Estes resultados mostram que o método proposto aqui não obteve bom desempenho para essa configuração. O método *Space Saving com CountingRank* apresentou melhor resultado em apenas 2 das 17 bases de dados. O método *Janela Deslizante* apresentou melhores resultados em 12 bases. Para a base *4CR*, a diferença entre as áreas sobre a curva obtida pelo método proposto aqui e pelo método que obteve melhor desempenho foi de mais de 56%.

Tabela 10 – Área sob a curva da taxa de erro obtida pelos métodos para a *Configuração*: [m: 200; alpha: 0.90; beta: 0.20]. A janela para o cálculo do erro tem 7 vezes o tamanho do conjunto de treinamento (m).

Base de Dado	Aleatório	Janela Deslizante	Space Saving CountingRank
1CDT	0.00140298	0.00088614	0.00342421
1CHT	0.00511176	0.00399611	0.01195559
1CSurr	0.03502386	0.02891787	0.31040621
2CDT	0.13959458	0.07402909	0.39692222
2CHT	0.19371722	0.15714412	0.41625184
4CE1CF	0.03959899	0.03988788	0.02442720
4CR	0.00032467	0.00027519	0.56807663
4CRE-V1	0.03862063	0.02866065	0.54494718
4CRE-V2	0.11918087	0.11820863	0.60323674
5CVT	0.20917258	0.15473029	0.37462017
FG_2C_2D	0.06903160	0.06935325	0.16086960
GEARS_2C_2D	0.01814961	0.01753408	0.04933596
MG_2C_2D	0.10134720	0.10284975	0.32850885
UG_2C_2D	0.06139920	0.06009795	0.36607335
UG_2C_3D	0.07595027	0.07771584	0.17996595
UG_2C_5D	0.11695373	0.11607820	0.22986766
keystroke	0.20172388	0.15152026	0.14017093
Vitórias	3	12	2

A Figura 28 apresenta o gráfico de espalhamento que mostra a relação de desempenho entre os métodos. Eles confirmam que o método proposto não apresentou melhor desempenho quando comparado com os demais métodos. Os pontos presentes no triângulo superior e a grande

distância dos pontos para a linha diagonal que divide o gráfico confirmam o seu mau desempenho nessa configuração.

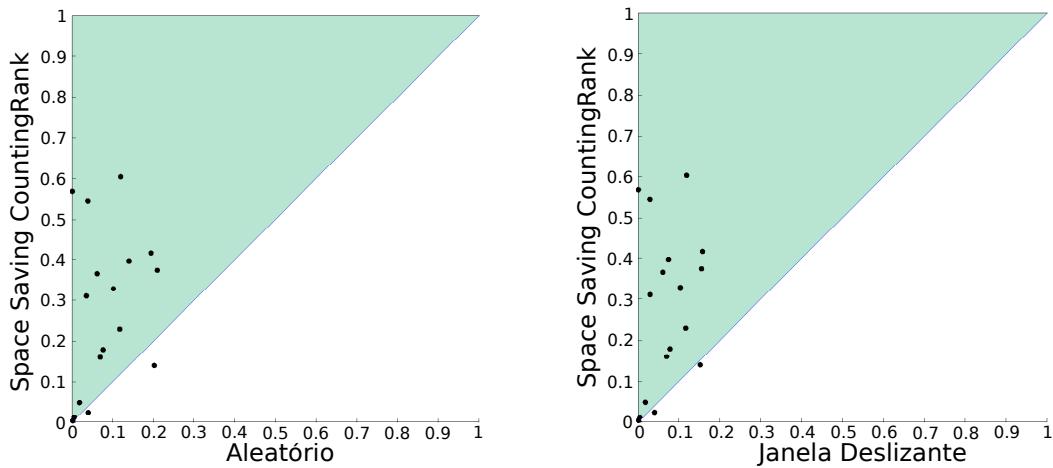


Figura 28 – Gráfico de espalhamento da relação do erro entre os métodos para a configuração: $m = 200$; $\alpha = 0.90$; $\beta = 0.20$. Os gráficos mostram a relação do método *Space Saving CountingRank* com os demais métodos.

A Figura 29 apresenta as curvas do erro para a base *4CR*. O método proposto aqui não obteve bom resultado nesta base para esta configuração. Nela este método obteve a maior diferença para o método com melhor resultado nesta base. A diferença de desempenho é notável, quase não se identifica as curvas dos erros dos métodos *Aleatório* e *Janela Deslizante*. A taxa de erro obtida pelo método chegou a atingir aproximadamente 80%, um valor muito alto diante o valor obtido pelos outros métodos.

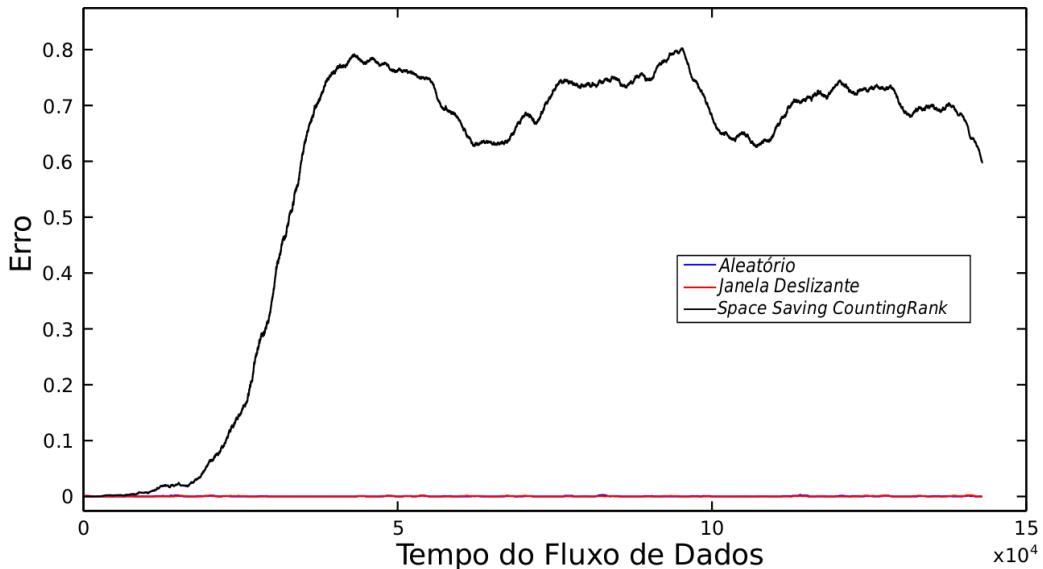


Figura 29 – Variação da taxa de erro durante o fluxo de dados no conjunto de dados *4CR* para a configuração $m = 200$; $\alpha = 0.90$; $\beta = 0.20$.

A Figura 30 apresenta as curvas do erro para a base *UG_2C_5D*. O método *Space Saving com CountingRank* também não obteve bons resultados nessa base e configuração. Apesar da

curva obtida por este método apresentar algumas interseções com as curvas obtidas pelos outros métodos, durante grande parte do fluxo ela se mantém acima das outras curvas.

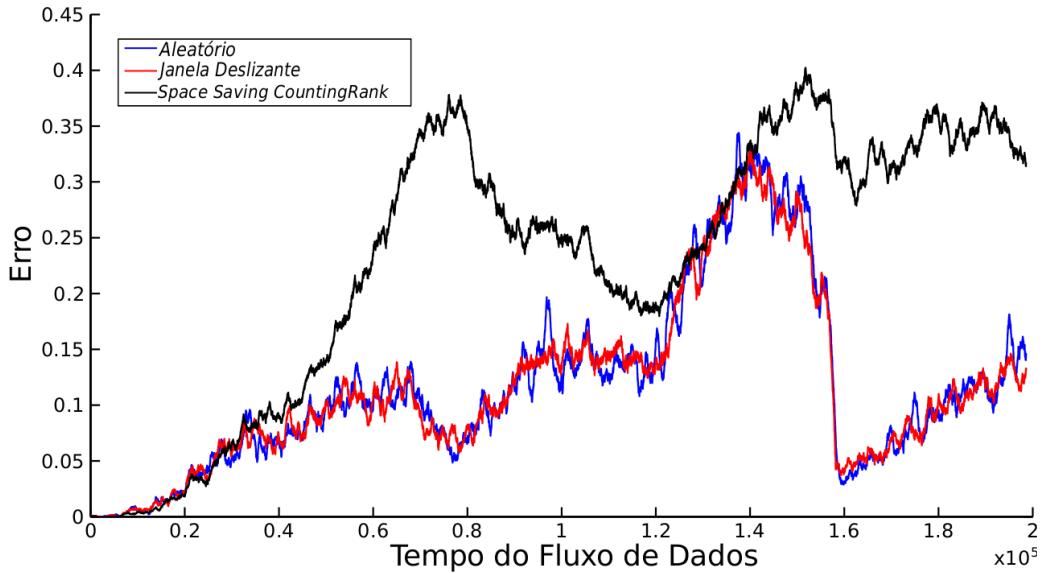


Figura 30 – Variação da taxa de erro durante o fluxo de dados no conjunto de dados *UG_2C_5D* para a configuração $m = 200$; $\alpha = 0.90$; $\beta = 0.20$.

A.2.2 Resultado Configuração 2: $m = 200$; $\alpha = 0.90$; $\beta = 0.75$

A Tabela 11 apresenta as áreas sobre a curva da taxa de erro para todas as bases de dados analisadas. Os resultados mostram que o método proposto obteve melhor desempenho que os demais métodos em quase todas as bases de dados para essa configuração. De 17 bases de dados, o método apresentou melhor desempenho que os demais em 14 bases. A diferença de desempenho do método foi baixa nas bases que não apresentou melhor resultado, e nas bases que apresentou melhor resultado, essa diferença chegou a alcançar quase 2%.

A Figura 31 apresenta o gráfico de espalhamento que mostra a relação de desempenho entre os métodos. Os gráficos mostram que o método proposto apresentou melhores resultados nesta configuração. Os pontos presentes no triângulo inferior do gráfico e com uma distância notável da linha diagonal confirmam isso.

A Figura 32 apresenta as curvas da taxa de erro para a base de dados *UG_2D_5D*. O método proposto aqui obteve melhor desempenho nesta base para esta configuração. Apesar das curvas estarem próximas, é possível notar que o método *Space Saving com CountingRank* alcançou melhores resultados que os demais. E o mais importante, após a ocorrência de mudança de conceito o método continuou apresentando taxas de erros mais baixas.

A Figura 33 apresenta as curvas da taxa de erro na base de dados *MG_2D_2D*. O método proposto também obteve melhor desempenho nesta base para esta configuração. Mesmo após a mudança de conceito, o método apresentou baixa taxa de erro. Dentre as bases testadas, esta é uma das mais difíceis de realizar o aprendizado, pois os dados se sobrepõem em grande parte do

Tabela 11 – Área sob a curva da taxa de erro obtida pelos métodos para a *Configuração*: [$m = 200$; $\alpha = 0.90$; $\beta = 0.75$]. A janela para o cálculo do erro tem 7 vezes o tamanho da base (m).

Base de Dado	Aleatório	Janela Deslizante	Space Saving CountingRank
1CDT	0.00083068	0.00073103	0.00040183
1CHT	0.00506523	0.00391628	0.00321006
1CSurr	0.02016245	0.01773131	0.01723527
2CDT	0.08426841	0.06458641	0.06671865
2CHT	0.16435530	0.14485401	0.13851180
4CE1CF	0.02994801	0.03043534	0.02401495
4CR	0.00014190	0.00014260	0.00017823
4CRE-V1	0.02768317	0.02457757	0.02137691
4CRE-V2	0.10882111	0.10887735	0.09249318
5CVT	0.14709375	0.13117024	0.11656567
FG_2C_2D	0.06217809	0.06329792	0.05161645
GEARS_2C_2D	0.00246285	0.00209748	0.00174569
MG_2C_2D	0.09711829	0.09812522	0.08230850
UG_2C_2D	0.05758756	0.05798877	0.04858799
UG_2C_3D	0.07205247	0.07281985	0.06092406
UG_2C_5D	0.10561112	0.10489097	0.08647226
keystroke	0.10327363	0.09593816	0.10034008
Vitórias	1	2	14

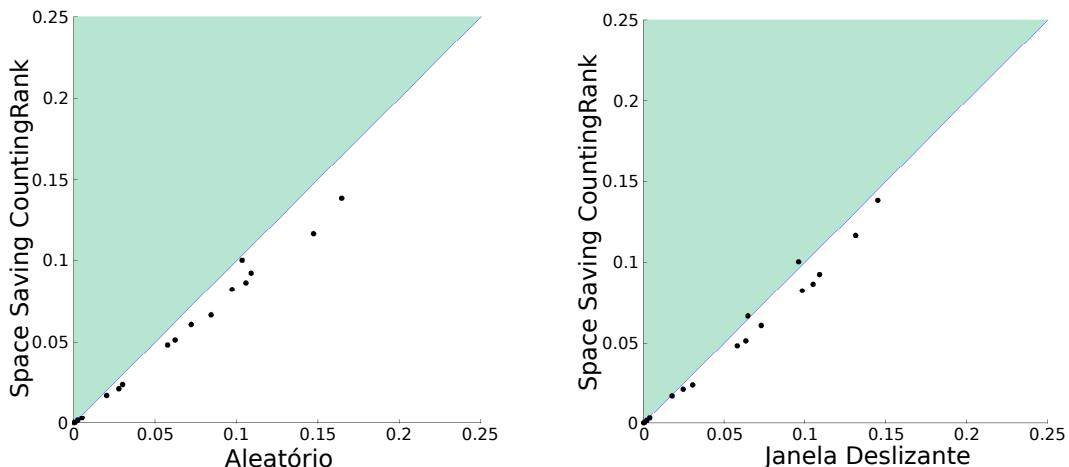


Figura 31 – Gráfico de espalhamento da relação do erro entre os métodos para a configuração: $m = 200$; $\alpha = 0.90$; $\beta = 0.75$. Os gráficos mostram a relação do método *Space Saving CountingRank* com os demais métodos.

tempo do fluxo de dados. Mesmo assim a taxa de erro obtida pelo método proposto se manteve inferior aos demais.

A.3 Análise experimental – Nova Fórmula

Os resultados inferiores obtidos pelo método proposto na etapa anterior, motivaram a realização de mais uma etapa experimental. Nela continua a ser utilizada apenas uma das

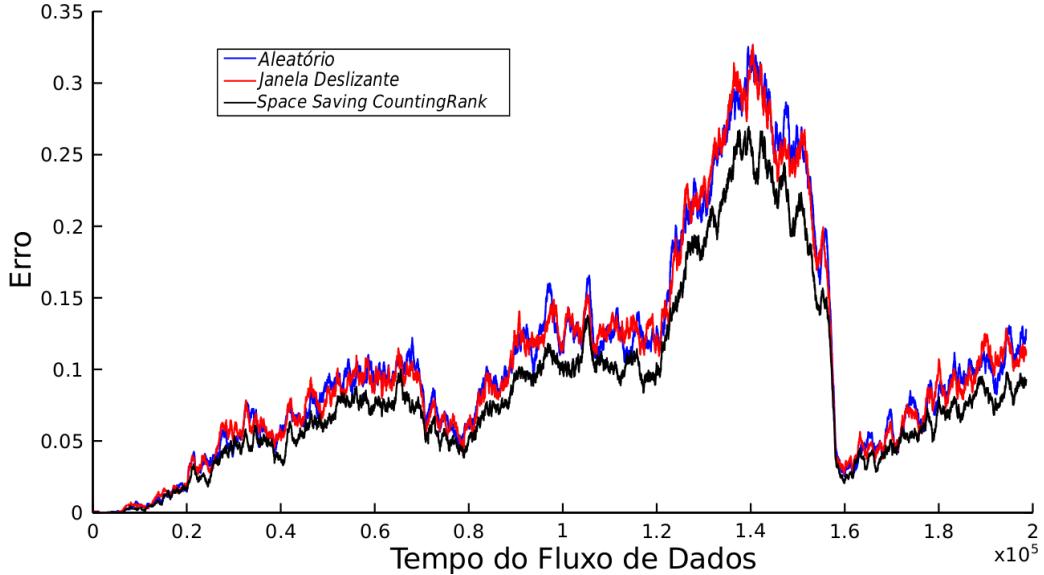


Figura 32 – Variação da taxa de erro durante o fluxo de dados no conjunto de dados UG_2C_5D para a configuração $m = 200; \alpha = 0.90; \beta = 0.75$.

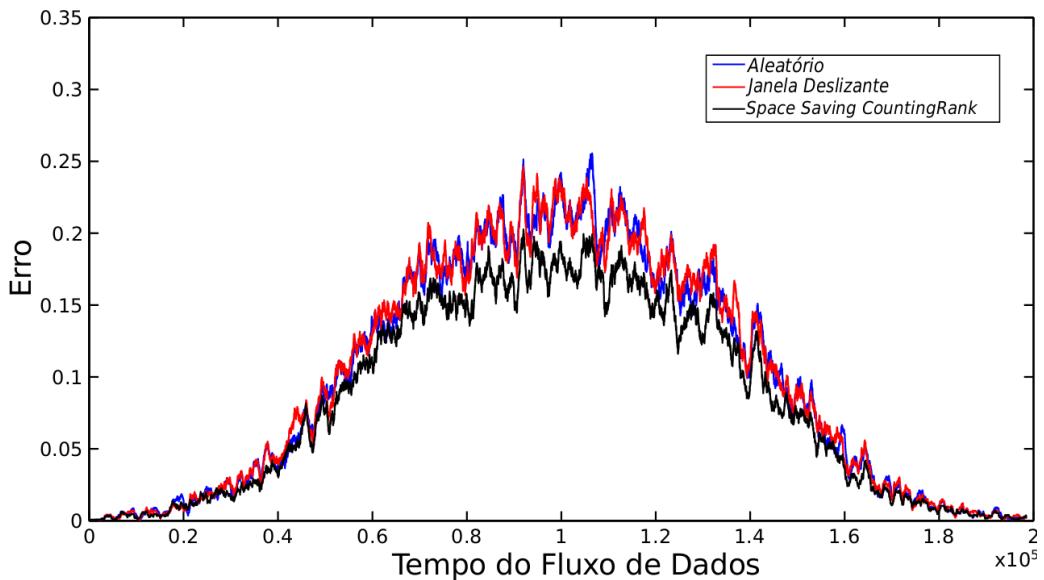


Figura 33 – Variação da taxa de erro durante o fluxo de dados no conjunto de dados MG_2C_2D para a configuração $m = 200; \alpha = 0.90; \beta = 0.75$.

estratégias de atualização do modelo de aprendizado, o *Space Saving com CountingRank*. Nesta análise somente a fórmula utilizada pelo algoritmo *anytime* incremental foi alterada. O algoritmo utiliza a Equação A.3.

$$counter_t(x) = counter_{t-1}(x) + \begin{cases} 1 & , \text{if } classe(x) = classe(x_j); \\ -2 & , \text{caso contrário.} \end{cases} \quad (\text{A.3})$$

onde x é a instância a ser atualizado o seu contador e x_j é a instância gerada pelo fluxo de dados que tem a instância x como o seu vizinho mais próximo. $counter_{t-1}(x)$ é o contador da instância x antes de x_j ser adicionado ao conjunto de treino e $counter_t(x)$ é o contador após a adição de x_j .

Esta fórmula ainda possibilita recalcular o valor do *rank* dos exemplos de treino. Porém ela penaliza os exemplos de treino que não contribuirem para a classificação correta de novas instâncias. Assim, quando um exemplo de treino realiza uma classificação errada, o seu contador de vizinhos mais próximos será decrementado em duas unidades. Desta forma, este exemplo pode deixar de ocupar as primeiras posições da lista ordenada utilizada pelo algoritmo *anytime*. Isso permite que os novos exemplos de treino possam estar presentes no topo da lista.

Os experimentos realizados nesta etapa apresentam as mesmas propriedades dos realizados na etapa anterior. Então, nesta análise foi utilizada a mesma configuração experimental e a mesma forma de avaliação dos métodos. Também foram utilizadas duas configurações semelhantes à etapa anterior, ambas apresentando alta probabilidade de ocorrência de interrupção. A *Configuração 1* apresenta baixo número de exemplos de treino verificados em uma interrupção. A *Configuração 2* verifica um alto número de exemplos de treino em uma interrupção. A seguir serão exibidos os resultados.

A.3.1 Resultado Configuração 1: $m = 200$; $\alpha = 0.90$; $\beta = 0.20$

A Tabela 12 apresenta as áreas sobre a curva do erro para esta configuração. O método proposto aqui continuou não apresentando bons resultados para esta configuração. Mesmo que as áreas sobre a curva para este método tenham diminuído, este ganho não foi suficiente para superar o desempenho dos demais métodos. A diferença entre as áreas sobre a curva dos métodos na base *4CR* diminuiu para 18%.

A Figura 34 apresenta o gráfico de espalhamento que mostra a relação de desempenho entre os métodos. Os pontos estão menos distantes da linha diagonal, comparados com a análise da etapa anterior. Porém eles continuam presentes no triângulo superior, que indica que o método proposto aqui não obteve bom desempenho. Em algumas bases o valor da área da curva do erro do método *Space Saving com CountingRank* diminuiu aproximadamente 38 pontos percentuais.

A Figura 35 mostra as curvas da taxa de erro na base de dados *4CR*. Com a nova fórmula o método *Space Saving com CountingRank* não ultrapassou a taxa de erro de 31%. Um grande ganho de desempenho comparado com a análise realizada na etapa anterior. Porém o método continua apresentando uma taxa de erro altíssima com relação aos métodos *Aleatório* e *Janela Deslizante*. As curvas destes métodos continuam pouco notáveis no gráfico.

A Figura 36 mostra as curvas da taxa de erro na base de dados *4CRE-V2*. Durante alguns instantes do fluxo as curvas se sobrepõem, em poucas delas o método proposto aqui apresenta melhor desempenho que os demais métodos. Porém em grande parte do fluxo o método apresentou taxas de erro acima dos outros métodos. Mesmo que a sua curva do erro tenha diminuído com relação a curva obtida na etapa experimental anterior, não foi suficiente para superar a performance dos outros métodos.

Tabela 12 – Área sob a curva da taxa de erro obtida pelos métodos para a *Configuração*: [m: 200; alpha: 0.90; beta: 0.20], sendo que a fórmula utilizada pelo método *Space Saving Anytime* foi alterada. A janela para o cálculo do erro tem 7 vezes o tamanho da base (m).

Base de Dado	Aleatório	Janela Deslizante	Space Saving CountingRank
1CDT	0.00140298	0.00088614	0.00195492
1CHT	0.00511176	0.00399611	0.00799448
1CSurr	0.03502386	0.02891787	0.15870343
2CDT	0.13959458	0.07402909	0.19422784
2CHT	0.19371722	0.15714412	0.20941629
4CE1CF	0.03959899	0.03988788	0.01991265
4CR	0.00032467	0.00027519	0.18224720
4CRE-V1	0.03862063	0.02866065	0.20936447
4CRE-V2	0.11918087	0.11820863	0.22107618
5CVT	0.20917258	0.15473029	0.20660290
FG_2C_2D	0.06903160	0.06935325	0.09777442
GEARS_2C_2D	0.01814961	0.01753408	0.03921149
MG_2C_2D	0.10134720	0.10284975	0.15406028
UG_2C_2D	0.06139920	0.06009795	0.18592619
UG_2C_3D	0.07595027	0.07771584	0.12294375
UG_2C_5D	0.11695373	0.11607820	0.14416263
keystroke	0.20172388	0.15152026	0.11127647
Vitórias	3	12	2

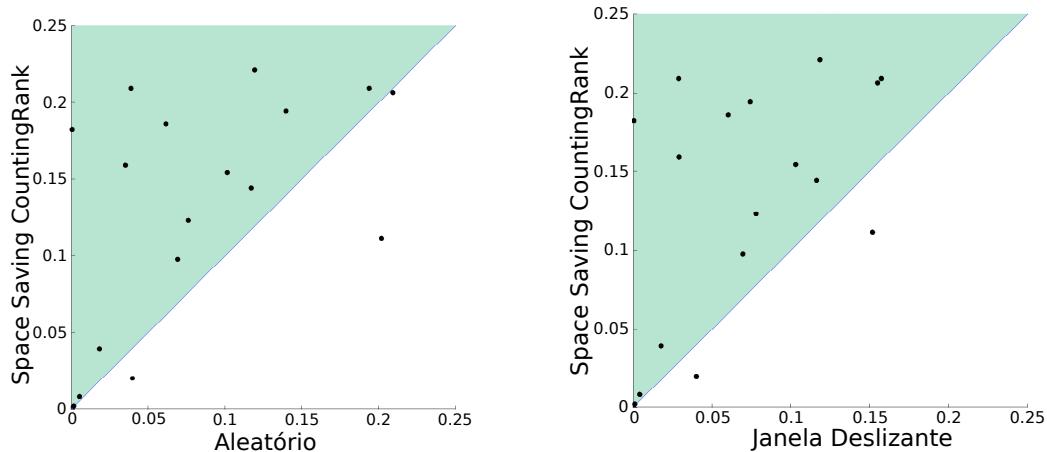


Figura 34 – Gráfico de espalhamento da relação do erro entre os métodos para a configuração: $m = 200$; $\alpha = 0.90$; $\beta = 0.20$. Os gráficos mostram a relação do método *Space Saving CountingRank* com os demais métodos.

A.3.2 Resultado Configuração 2: $m = 200$; $\alpha = 0.90$; $\beta = 0.75$

A Tabela 13 apresenta as áreas sobre a curva do erro para esta configuração. O desempenho alcançado pelo método aumentou com esta alteração da fórmula. Os valores das áreas sobre a curva do erro diminuíram em todas as bases de dados, de acordo com a análise realizada na etapa experimental anterior. Além disso, o método proposto aqui obteve melhor desempenho em 15 das 17 bases de dados. A maior diferença do valor da área obtida nesta etapa de experimento

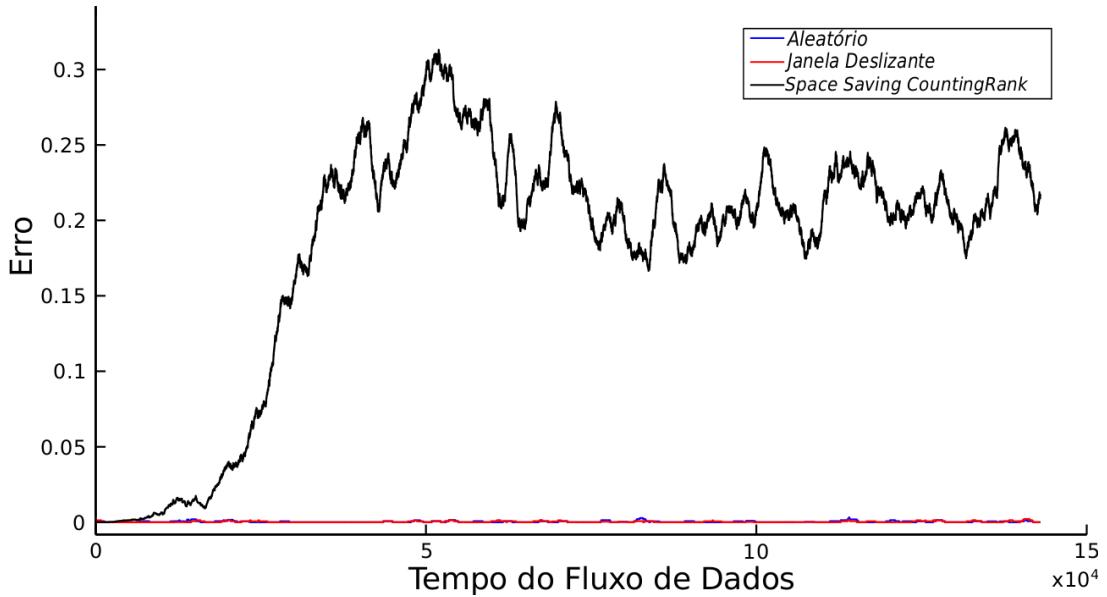


Figura 35 – Variação da taxa de erro durante o fluxo de dados no conjunto de dados 4CR para a configuração $m = 200$; $\alpha = 0.90$; $\beta = 0.20$

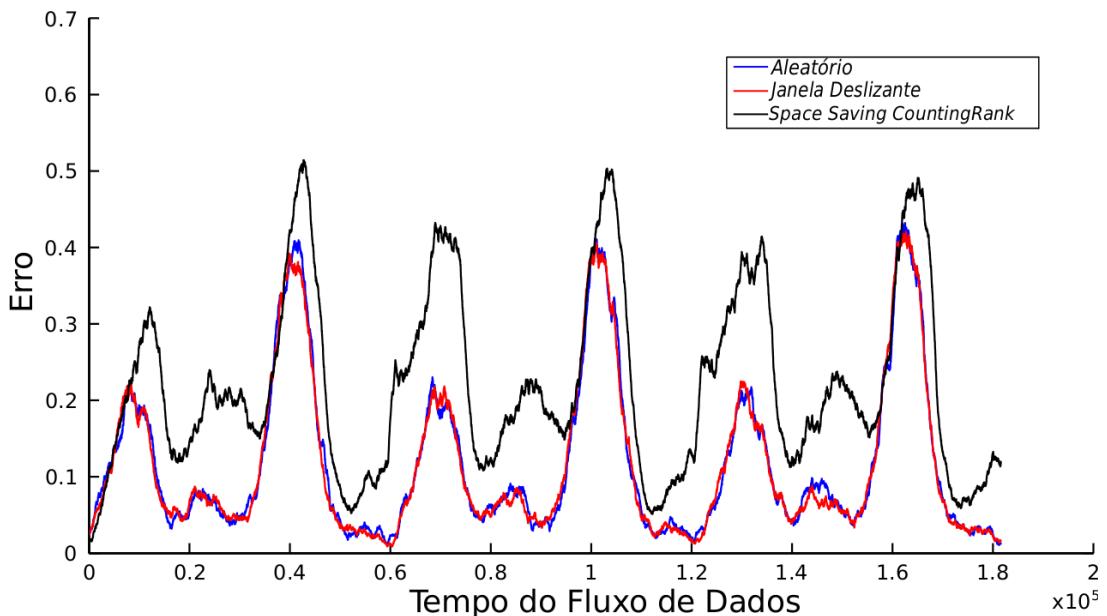


Figura 36 – Variação da taxa de erro durante o fluxo de dados no conjunto de dados 4CRE-V2 para a configuração $m = 200$; $\alpha = 0.90$; $\beta = 0.20$

foi de 2.5% para a base 5CVT.

A Figura 37 apresenta o gráfico de espalhamento que mostra a relação de desempenho entre os métodos. Pode-se notar neles que os pontos estão ligeiramente mais distantes da linha diagonal do gráfico, com relação à mesma análise da etapa experimental anterior. Isso evidencia que o método proposto aqui continua apresentando melhor desempenho que os demais métodos nesta configuração.

A Figura 38 exibe as curvas da taxa de erro para a base UG_2C_5D. É possível notar alguns pontos do fluxo onde as curvas se sobrepõem. Porém, em grande parte do tempo o método

Tabela 13 – Área sob a curva da taxa de erro obtida pelos métodos para a Configuração: [$m = 200$; $\alpha = 0.90$; $\beta = 0.75$], sendo que a fórmula utilizada pelo método *Space Saving Anytime* foi alterada. A janela para o cálculo do erro tem 7 vezes o tamanho do conjunto de treinamento (m).

Base de Dado	Aleatório	Janela Deslizante	Space Saving CountingRank
1CDT	0.00083068	0.00073103	0.00039352
1CHT	0.00506523	0.00391628	0.00307152
1CSurr	0.02016245	0.01773131	0.01730465
2CDT	0.08426841	0.06458641	0.04985760
2CHT	0.16435530	0.14485401	0.12116738
4CE1CF	0.02994801	0.03043534	0.02473588
4CR	0.00014190	0.00014260	0.00017517
4CRE-V1	0.02768317	0.02457757	0.01959584
4CRE-V2	0.10882111	0.10887735	0.08954625
5CVT	0.14709375	0.13117024	0.10570848
FG_2C_2D	0.06217809	0.06329792	0.05100927
GEARS_2C_2D	0.00246285	0.00209748	0.00199356
MG_2C_2D	0.09711829	0.09812522	0.07975093
UG_2C_2D	0.05758756	0.05798877	0.04760852
UG_2C_3D	0.07205247	0.07281985	0.05885992
UG_2C_5D	0.10561112	0.10489097	0.08385305
keystroke	0.10327363	0.09593816	0.09719616
Vitórias	1	1	15

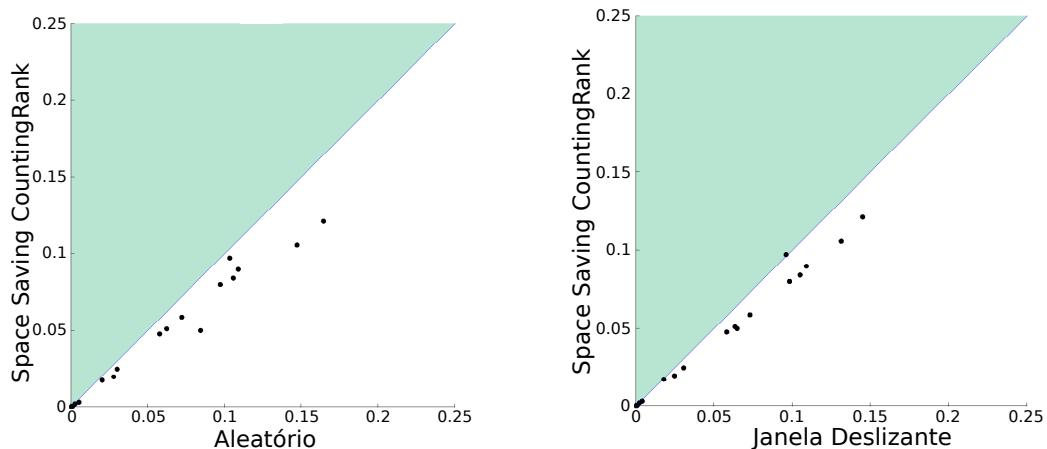


Figura 37 – Gráfico de espalhamento da relação do erro entre os métodos para a configuração: $m = 200$; $\alpha = 0.90$; $\beta = 0.75$. Os gráficos mostram a relação do método *Space Saving CountingRank* com os demais métodos.

proposto aqui apresenta taxa de erro inferior aos demais. Principalmente nos instantes em que ocorre mudança de conceito.

A Figura 39 exibe as curvas da taxa de erro para a base *5CVT*. A curva do erro do método proposto aqui se manteve abaixo das outras curvas durante todo o fluxo de dados. Isso vem a confirmar o bom desempenho deste método para esta configuração analisada.

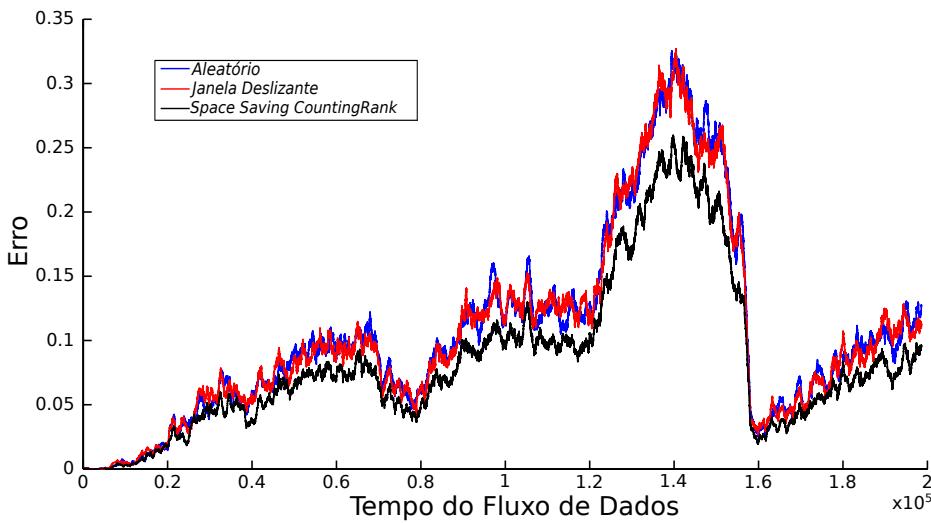


Figura 38 – Variação da taxa de erro durante o fluxo de dados no conjunto de dados *UG_2C_5D* para a configuração $m = 200; \alpha = 0.90; \beta = 0.75$

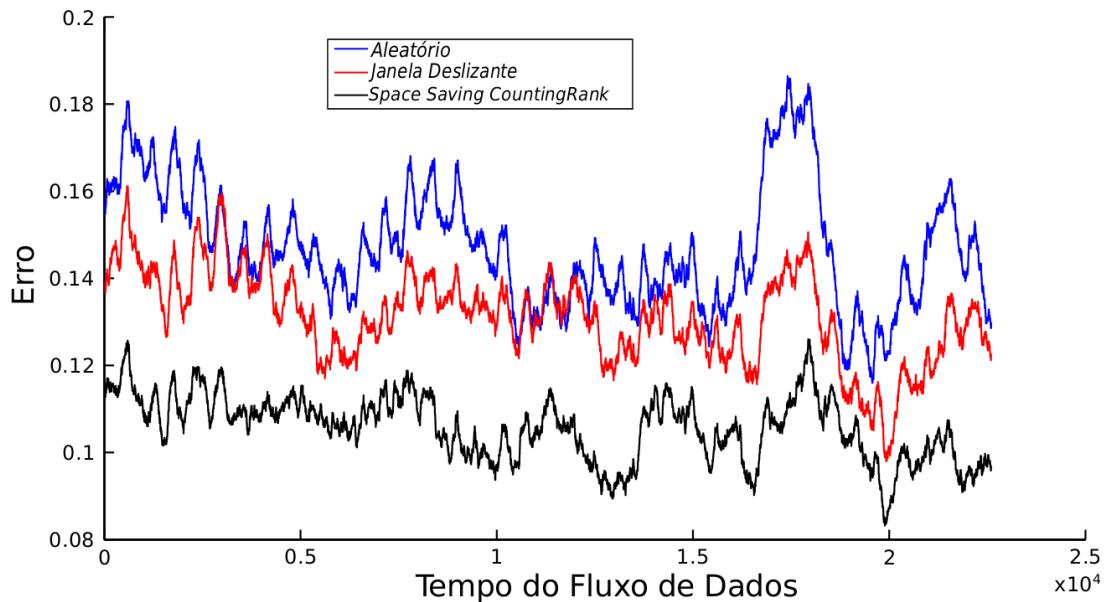


Figura 39 – Variação da taxa de erro durante o fluxo de dados no conjunto de dados *5CVT* para a configuração $m = 200; \alpha = 0.90; \beta = 0.75$

A.4 Considerações Finais

A busca pelo algoritmo *anytime* incremental iniciou com o estudo do algoritmo *Space Saving*. Após entender como funciona este algoritmo, foi proposto uma fórmula capaz de ranquear os exemplos de treino e que, ao mesmo tempo, possibilitasse a atualização do modelo de aprendizado de forma simples. A fórmula apenas realizava a contagem dos vizinhos mais próximos de mesma classe dos exemplos de treino. Esta fórmula traz um pouco da essência do *Space Saving*, que é um algoritmo para contagem de palavras em um fluxo de dados. Baseado nesta contagem, o método ranqueava os exemplos de treino.

Foi realizada então a primeira etapa de experimentos para avaliar se o método era capaz de

apresentar bons resultados em um problema de classificação. Foram utilizados conjuntos de dados sintéticos estáticos, gerados através da combinação de vários parâmetros. Cada combinação dos parâmetros representa diferentes tipos de fluxos de dados. Os resultados mostraram que o método apresentou melhor desempenho que os demais métodos analisados em diversas configurações. Mesmo em fluxos de dados que apresentam alta probabilidade de interrupção e baixa quantidade de exemplos de treino verificados antes da interrupção, o método mostrou melhor desempenho. Estes bons resultados motivaram a realização de novos experimentos com o método.

Na segunda etapa experimental, foram utilizadas bases de dados sintéticas e uma base de dados real que apresentam mudanças de conceito e o mesmo método definido na primeira etapa experimental. Também foram utilizados alguns parâmetros para simular diferentes tipos de fluxo de dados. Nessa etapa experimental o método apresentou bons resultados para alguns tipos de fluxo de dados, enquanto para outros não apresentou bons resultados. Em fluxos com alta probabilidade de interrupção e alta quantidade de exemplos de treino verificados antes da interrupção, o método apresentou ótimos resultados. Já em fluxos com alta probabilidade de interrupção e baixa quantidade de exemplos de treino verificados antes da interrupção, o método alcançou elevadas taxas de erro.

Acredita-se que isso ocorreu pelo fato de que, na Equação A.1 que atualiza a contagem de vizinhos mais próximos dos exemplo, quando um exemplo de treino classifica errado nenhuma penalidade é aplicada sobre ele. Mesmo este exemplo deixando de contribuir para a classificação correta de instâncias desconhecidas, este exemplo de treino continua presente ao topo da lista ordenada. Sendo que o algoritmo *anytime* utilizará estes exemplos nos estágios iniciais do algoritmo. A probabilidade do algoritmo *anytime* retornar uma resposta errada é alta em um fluxo de dados que possibilita verificar apenas estes primeiros exemplos de treino. Isso motivou a realização de uma terceira etapa experimental, agora realizando uma alteração no método.

Nesta nova etapa experimental foi alterada a fórmula inicial, agora penalizando os exemplos de treino que classificam errado as novas instâncias. Foram utilizados os mesmo conjuntos de dados da etapa experimental anterior. O método *Space Saving com CountingRank* apresentou menores taxas de erro comparado com a etapa experimental anterior. Isso confirma a importância de penalizar os exemplos de treino que não contribuem para a classificação correta das instâncias desconhecidos.

Porém ele mostrou o mesmo comportamento encontrado na etapa experimental anterior. Em um fluxo com alta probabilidade de interrupção e baixa quantidade de exemplos de treino verificados antes da interrupção, o método continuou apresentando altas taxas de erro. Mesmo que ela tenha sido menor que a etapa experimental anterior, seu desempenho não superou os outros métodos.

Em uma breve análise da ordenação dos exemplos de treino, notou-se que mesmo com esta alteração, os novos exemplos de treino não alcançavam o topo da lista ordenada facilmente. Especialmente em um fluxo onde poucos exemplos de treino são verificados em uma

interrupção, estes exemplos nem chegaram a ser utilizados pelo algoritmo *anytime*. Desta forma seus contadores não eram incrementados e estes exemplos permaneciam no final da lista. Após uma mudança de conceito os exemplos de treino presentes no topo da lista ordenada podem não representar bem os conceitos na qual pertencem. Isso explica o mau desempenho do método em um fluxo com esta característica.

Já em um fluxo onde muitos exemplos de treino são verificados antes da interrupção, os novos exemplos de treino foram utilizados pelo algoritmo *anytime*. Os novos exemplos de treino representam bem o conceito na qual pertencem. Isso explica o seu bom desempenho em um fluxo com estas características.

Após uma mudança de conceito o ideal seria que os exemplos de treino presentes no topo da lista ordenada fossem substituídos pelos novos exemplos de treino. Daí, surgiu a proposta de aplicar o mesmo método de ordenação do *Space Saving com CountingRank* em uma janela deslizante sobre o fluxo de dados. Este método foi chamado de *Janela Deslizante com CountingRank*. Estes dois métodos foram descritos em detalhes no Capítulo 4. Onde também foi realizada uma análise experimental detalhada de ambos os métodos.