UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ANDRÉ KENJI KAGAWA NUNES

Análise do impacto de *feature selection* e detecção de *concept drift* na classificação de fluxos de dados em evolução

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Orientadora: Prof.ª Dra. Mariana R. Mendoza
Co-orientadora: Prof.ª Dra. Ana Lúcia C. Bazzan

Porto Alegre
2019

AGRADECIMENTOS

Agradeço à minha orientadora Prof.ª Dra. Mariana Recamonde Mendoza e ao futuro doutor, Jorge Chamby-Diaz, por todo o apoio durante a realização deste trabalho, e à Prof.ª Dra. Ana Bazzan pela disponibilidade como co-orientadora.

Também agradeço aos meus amigos e colegas da graduação, por todos esses semestres que estudamos juntos, e aos excelentes professores do instituto de informática com quem tive aula.

RESUMO

O aprendizado a partir de fluxos de dados sofre de diversos problemas que não existem em outros cenários de aprendizado de máquina convencional. Os principais problemas tratados nesse trabalho são a sua natureza *online* e não estacionária, além de suas consequências: as possíveis mudanças na função geradora dos dados, que caracterizam um *concept drift*. Esse cenário cria a necessidade de novos algoritmos desevolvidos para se adaptar à essas possíveis mudanças, que podem ocorrer de maneira gradual ou abrupta. Mas não limitado a essas anomalias, também existe a necessidade de lidar com uma possível alta dimensionalidade dos dados somado à necessidade de realizar o aprendizado incrementalmente, e em tempo real. Como resposta a esses desafios, existem diferentes soluções no estado da arte, envolvendo variados mecanismos como *feature selection* e métodos de detecção de *concept drift*. Portanto, nesse trabalho é analisado o impacto e viabilidade da adição desses mecanismos à algoritmos de classificação, em relação ao custo computacional e poder preditivo. Para isso, é proposto um algoritmo de controle definido como um comitê de árvores de decisão equipado com adaptação passiva externa aos classificadores base, e três modificações distintas originadas dele, onde uma utiliza um seletor de *features* relevantes, outra um detector de *concept drift* e adaptação ativa, e a última modificação utiliza os dois simultaneamente. Para comparar estas variações, são apresentados os resultados dos testes em múltiplos cenários representados por *datasets* sintéticos e reais. Deste modo, o algoritmo utilizando apenas *feature selection* e adaptação passiva apresentou ganho de acurácia e diminuição do custo computacional, enquanto os algoritmos utilizando métodos de detecção de *concept drift* apresentaram um desempenho inferior ao demais algoritmos, na maioria dos casos. Esses resultados foram verificados utilizando o teste de Friedman com procedimento post-hoc, onde a modificação com apenas *feature selection* foi a única que mostrou diferença estatisticamente significativa sobre os outros algoritmo com grau de confiança de 0.10, enquanto os testes foram inconclusivos para os algoritmos que utilizaram adaptação ativa.

Palavras-chave: Fluxos de dados. Aprendizado de máquina. Classificação. *Concept drift*. *Feature drift*. *Feature selection. Drift detection.*

Online Subspace Method for evolving data streams classification

ABSTRACT

Learning from data streams poses many challenges that are not present in conventional machine learning. The main problems addressed in this work are the datastream's online nature and non stationarity, and its consequences: the possible changes in the generating function, that is also called a concept drift. This scenario creates the need for new algorithms that are able to adapt to these changes, which can happen in a gradual, or abrupt manner. But not limited to these anomalies, there is also the need to deal with the possibility of high dimensional data, added to the need for real time incremental learning. As an answer for these challenges, many solutions have been suggested in the state of the art, involving many mechanisms like feature selection and concept drift detection methods. Therefore, in this work we analyze the impact and viability of the adition of these mechanisms to classification algorithms, in relation to prediction power and computing cost. For this purpose, an ensemble classifier made of decision trees, equiped with a passive adapting method extern to the base learners, is proposed as a control algorithm, while three modifications originated by it are used as comparison, one using a relevant features selector, a second one using concept drift detector and active adapting, and the last one using both simultaneously. In order to compare these variations, the results of testing on multiple scenarios are presented, represented by testing on real and synthetic datasets. This way, the algorithm using only feature selection and passive adapting showed increases in accuracy and lower computing cost, while those equiped with active adaptation performed worse in most cases. These results were verified using a Friedman test with a post-hoc procedure, where the modification using only feature selection was the only one that showed statistically significant difference to the other algorithms with confidence level of 0.10, while the tests were inconclusive for those using active adapting.

Keywords: Datastreams. Machine learning. Classification. Concept drift. Feature drift. Feature selection. Drift detection.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AM	Aprendizado de Máquina
RSM	*Random Subspace Method*
ARF	*Adaptive Random Forest*
HEFT	*Heterogeneous Ensemble for Feature Drifts in Data Streams*
FCBF	*Fast Correlation Based Filter*
OSM	*Online Subspace Method*
EDDM	*Early Drift Detection Method*
DDM	*Drift Detection Method*

SUMÁRIO

1 INTRODUÇÃO

Um dos principais algoritmos utilizados para classificação de dados é o algoritmo de Florestas Aleatórias, já tendo sido demonstrada a sua eficiência em cenários estacionários (GOMES et al., 2017a). Devido à sua construção, ele se baseia na premissa implícita de que os conceitos a serem aprendidos são estáticos e não evoluem significativamente com o tempo.

Porém, um número crescente de problemas reais como a detecção de spam e a detecção de fraudes não seguem esse modelo, e os dados são recebidos gradativamente a partir de um fluxo de dados, que pode apresentar mudanças de conceito ao longo de sua duração. Devido a essas anomalias, aprender conceitos provenientes de fluxos de dados é uma tarefa significativamente diferente do aprendizado de máquina tradicional, como apresentado no estudo de Gomes et al. (2017b).

Atualmente, no contexto de fluxos de dados em evolução, o algoritmo de Florestas Aleatórias é representado pelo algoritmo *Adaptive Random Forests* proposto por Gomes et al. (2017a). Todavia, os seus resultados apontam alto custo computacional, e a necessidade de processamento paralelo para a obtenção de resultados competitivos com o estado da arte.

Embora a maioria dos problemas causados pela natureza evolutiva dos fluxos de dados possua diversas propostas de soluções atuais em aprendizado de máquina, pouca atenção foi dada a alterações de conceito onde a relevância das *features* dos dados muda ao longo do tempo, o chamado *feature drift* (BARDDAL et al., 2016).

Em problemas reais, os dados podem possuir muitos atributos, como no caso do conjunto de dados *Spam Corpus*, onde cada instância possui 40.000 atributos. Como consequência, algoritmos que não utilizam mecanismos para a divisão do espaço de *features* em sub-espaços precisam processar um elevado número de atributos irrelevantes.

De acordo com matéria apresentada na Forbes[1], os prejuízos causados por fraudes em cartões de crédito no ano de 2015 ultrapassam os 21 bilhões de doláres. Portanto, percebe-se que problemas como a detecção de fraudes em cartões de crédito, precisam de uma rápida adaptação quando ocorrida uma mudança de conceito, seja ela qual for, e por isso dependem de mecanismos que detectem essa alteração o quanto antes.

[1] AITKEN, R. U.S. Card Fraud Losses Could Exceed $12B By 2020. **Forbes**, 2016. Disponível em <https://www.forbes.com/sites/rogeraitken/2016/10/26/us-card-fraud-losses-could-exceed-12bn-by-2020/>.

Nesse contexto, é aparente a necessidade de alternativas que se proponham a solucionar os diferentes problemas causados pelo *concept drift* (termo em inglês utilizado para indicar alterações de conceito) em fluxos de dados em evolução, de maneira rapida e eficiente.

Sendo assim, o objetivo geral deste estudo é analisar o impacto e viabilidade do uso de mecanismos de seleção de atributos relevantes e de detecção de *concept drift*, com a finalidade de diminuir o custo computacional enquanto se mantém a eficácia na classificação de diferentes *datasets*, sintéticos e reais. Para isso, pretende-se propor diferentes modificações do algoritmo *Heterogeneous Ensemble for Feature Drifts in Data Streams* (HEFT), como também avaliá-las em comparação com outros algoritmos do estado da arte e seus resultados para diferentes métricas de avaliação como: tempo de execução, RAM-Hora, acurácia e *kappa-m*.

Este trabalho está organizado da seguinte maneira: No capítulo 2 está a base teórica necessária para o entendimento dos capítulos seguintes; No capítulo 3 são apresentadas propostas existentes para resolução dos problemas relacionados à proposta do trabalho; No capítulo 4 apresenta-se a abordagem seguida para a obtenção dos resultados, e as métricas utilizadas para análise; No capítulo 5 os resultados são apresentados e analisados e no capítulo 6 é apresentada a conclusão do trabalho.

2 REFERENCIAL TEÓRICO

Neste capítulo são apresentados os principais conceitos relacionados ao trabalho, formando o embasamento teórico necessário para compreensão dos capítulos seguintes.

2.1 Aprendizado de Máquina

Uma das vertentes da inteligência artificial é o Aprendizado de Máquina(AM), ou do inglês, *machine learning.* Esta linha de estudos se baseia na ideia de que fornecidos os dados corretos, máquinas podem aprender a se modificar ou adaptar suas ações para se tornarem mais precisas, sendo estas ações das mais diversas: desde realizar predições, até controlar um robô, e sendo a noção de precisão dada pela diferença entre a ação escolhida e a ação correta (MARSLAND, 2015).

Neste trabalho, uma ação pode ser representada como a classificação de novos dados a partir da observação de dados passados. No caso, as áreas do AM que tratam de problemas de classificação são principalmente Aprendizado Supervisionado e derivações como Aprendizado Incremental, que se assemelham pela sua capacidade de realizar previsões sobre dados não analisados anteriormente, a partir de um conjunto de dados fornecidos para treinamento.

A seguir são explicados alguns conceitos importantes e essenciais, utilizados em AM no geral.

2.1.1 Instância

Uma instância caracteriza uma unidade em um conjunto de dados, geralmente padronizados, utilizados para treinamento e testes (MOHRI; ROSTAMIZADEH; TALWALKAR, 2012). É subdividida em campos chamados atributos, que indicam características referentes à instância em questão. Em um conjunto de dados hipotético D que representa uma população, seja $X_i(1 \leq i \leq n) = (A_i, Y_i)$ uma instância de D, X_i indica uma pessoa com um conjunto de atributos $A_i = (A_i^1, A_i^2,..., A_i^n)$ onde cada $A_x^y(y = 1,...,m)$ representa uma caracteristica referente à pessoa X_i, com um valor atribuído podendo ser categórico, ordinal ou numérico. Além disso, para o aprendizado supervisionado de problemas de classificação é disponibilizado um rótulo referente a instância através de um campo Y_i representando a sua classe. A figura 2.1.1.1 representa um exemplo de instância X_i do conjunto de dados D.

Figura 2.1.1.1 – Exemplo de instância

A_i^1	A_i^2	A_i^3	A_i^4	Y_i
Adam	20	1.82	Married	M

Fonte: O Autor.

2.1.2 Classificador

Um classificador em AM representa um modelo treinado com o objetivo de corretamente classificar novas instâncias, ou seja, predizer a classe de uma dada instância gerada e rotulada através de um função desconhecida $g(x) = Y_{real}$, onde Y_{real} é o rótulo real. Utilizando uma nova função $f(x) = Y_{pred}$, induzida através do treinamento a partir de um conjunto de instâncias, com base em algum algoritmo, o classificador é capaz de realizar uma predição Y_{pred} sobre novas instâncias não vistas antes.

Existem diferentes métricas para avaliar a eficiência de um classificador e embora uma métrica comum seja o erro entre as saídas da função induzida $f(x)$ e da função geradora $g(x)$, em determinados contextos ela pode dar uma falsa sensação de eficácia, sendo assim sugerida a utilização em conjunto com outras técnicas de avaliação. (BIFET et al., 2015)

2.1.3 Aprendizado Supervisionado

O principal ramo de AM utilizado em classificação de dados é o aprendizado supervisionado. Para essa classe de problemas, um conjunto de dados de treinamento onde os rótulos das instâncias estão disponíveis é entregue ao algoritmo responsável pelo treinamento do classificador e, baseado nesse conjunto de treino, o classificador é generalizado para tentar classificar corretamente à todas possíveis entradas (MARSLAND, 2015).

2.1.3.1 Batch Learning

A técnica convencional para treinamento de classificadores onde o conceito a ser aprendido é estático, é a utilização de treinamento em lotes (CARBORNA; BORROWMAN, 1998). Nessa abordagem, utiliza-se um conjunto limitado de instâncias de entrada, e o classificador é treinado processando esse lote uma ou múltiplas vezes. Esse treinamento é válido principalmente para situações onde o lote consegue ser suficientemente expressivo para

representar o conceito a ser aprendido, é viável de ser armazenado e processado em relação a recursos computacionais, e o conceito a ser aprendido não muda com o tempo.

2.1.3.2 Árvore de Decisão

Segundo Quinlan (1986), árvores de decisão são criadas a partir de um nó raiz, e um conjunto inicial de dados de treinamento é dividido de acordo com os valores dos seus atributos, de modo que haja o maior ganho de informação. A divisão de um grupo para quando todas as instâncias desse grupo possuem a mesma classe, sendo então criado um nó folha no lugar representando essa classe.

Como base para o cálculo do ganho de informação referente a um atributo, é necessário calcular a entropia dos dados antes da divisão utilizando a equação (1):

$$EntropiaAntes(S) = \sum_{i=1}^{c} -p_i log_2 p_i \quad (1)$$

Neste caso a entropia simboliza a homogeneidade dos dados antes da repartição, onde p_i é a fração de instâncias que pertencem à classe i, em relação ao total de instâncias. Caso o conjunto seja totalmente homogêneo, a entropia equivale a 0. Já se as divisão de instâncias entre as classes for simétrica, a entropia equivale a 1.

Em seguida, calcula-se a entropia resultante após a divisão para cada atributo utilizando a equação (2),

$$EntropiaDepois(S, X) = \sum_{c \in X} P(c)E(c) \quad (2)$$

sendo o ganho de informação, equação (3), a diferença entre a entropia antes da divisão e a entropia depois da divisão:

$$Ganho(S, X) = EntropiaAntes(S) - EntropiaDepois(S, X) \quad (3)$$

Para percorrermos uma árvore de decisão já treinada, com o intuito de classificar uma instância arbitrária, devemos utilizar as divisões definidas no treinamento da árvore. Comparamos o valor do atributo correspondente ao nó, na instância, com os possíveis caminhos a partir daquele nó. Ao chegar em um nó folha, basta verificarmos o valor do nó que ele indicará a classe predita.

2.1.4 Aprendizado Incremental e *Hoeffding Trees*

O aprendizado incremental propõe uma solução alternativa, para problemas que não se encaixam nas necessidades do *batch learning*. Algoritmos desse tipo, apresentam uma abordagem onde o treinamento é realizado de maneira gradual conforme novos dados são apresentados, e o conhecimento sobre o conceito a ser aprendido é refinado de acordo.

Um algoritmo pode ser classificado como incremental caso obedeça os seguintes critérios: estar apto a treinar e atualizar o classificador a cada nova instância apresentada, preservar conhecimento adquirido anteriormente, não necessitar acesso aos dados originais, gerar uma nova classe ou *cluster* quando necessário, e ser dinâmico de acordo com as mudanças do ambiente (ADE; DESHMUKH, 2013).

A principal alternativa para as árvores de decisão convencionais, em cenários de fluxos de dados, onde é necessário o aprendizado incremental, são as Árvores de *Hoeffding*. Esse tipo de árvore de decisão se destaca por se basear em algumas premissas que garantem que sua eficiência seja comparável à de árvores de decisão treinadas utilizando *Batch Learning.* Em outras palavras, a natureza incremental desse algoritmo não afeta a qualidade das árvores que ele produz (DOMINGOS; HULTEN, 2000).

A construção deste modelo de árvores de decisão ocorre a partir de substituições recursivas dos nós folha por nós de decisão. A medida em que os dados chegam, eles são processados em tempo constante e individualmente, e de acordo com diferentes heurísticas como o ganho de informação ou o índice Gini, é feita a decisão sobre a necessidade de repartição de algum nó, ou não (BARDDAL, 2016).

2.1.5 Comitê de Classificadores e *Random Forest*

Criar um comitê ou do inglês, um *ensemble*, de classificadores é uma técnica utilizada em AM, para aumentar o poder de generalização ao utilizar múltiplos classificadores e combinar suas predições através de alguma heurística. Entre os modos mais populares para combinação dos votos estão a utilização do voto majoritário simples, e a atribuição de diferentes pesos ao voto de diferentes classificadores.

De acordo com Zhou (2009), existem diversos motivos para um *ensemble* ser mais eficiente do que um classificador sozinho, sendo dois deles as seguintes: o conjunto de treino

pode não prover informação suficiente para a escolha de um único melhor classificador e o algoritmo pode não conseguir generalizar o aprendizado devido à função objetivo não estar presente no espaço de hipóteses buscado. Portanto, podem existir situações onde múltiplos classificadores diferentes possuem performances igualmente satisfatórias, ou o algoritmo utilizado não é capaz de aprender a função objetivo por limitações de formulação, e juntando diferentes classificadores é possível obter uma melhor aproximação da função objetivo.

No entanto, Krogh e Vedelsby (1995) mostram que para se obter um comitê eficiente, é necessário que os classificadores sejam individualmente melhores que um classificador aleatório, ou seja, sejam individualmente eficazes, e também que suas predições discordem entre si o máximo possível, a fim de aumentar a diversidade do *ensemble*. Embora a definição de diversidade no conceito de classificadores não esteja bem definida na literatura, quanto mais os classificadores concordarem entre si, mais a sua performance se aproximará da performance de um único classificador, assim como um classificador com uma eficácia ruim realizará, em sua maioria, predições que prejudicarão a decisão do comitê.

Um exemplo de comitê muito utilizado para resolver problemas de classificação, utilizando *batch learning*, é o método de criação de Florestas Aleatórias (do inglês, *Random Forests*) proposto por Ho (1998).

Neste método são criadas múltiplas árvores de decisão utilizando *Random Subspace Method*, que é descrito nas próximas seções, para adicionar diferença no espaço de *features* dos classificadores base. Como resultado, é criado um comitê formado por diferentes classificadores, que no fim tem os seu votos individuais unido através de votação majoritária simples.

2.2 Fluxos de Dados

Fluxos de dados podem ser caracterizados como uma sequência continua sem um fim delimitado dos dados, onde estes ficam disponíveis gradualmente (GAMA; RODRIGUES; AGUILAR-RUIZ, 2006). O AM voltado à essa fonte de informação requer a atenção para características que não estavam presentes no treinamento através de lotes, e necessita assim algoritmos capazes de se adaptar à essas adversidades de acordo. A tabela 2.2 indica as diferenças que devem ser consideradas no aprendizado a partir de ambas fontes de dados:

Tabela 2.2 – Premissas utilizadas no aprendizado em fluxos de dados

	Tradicional	*Fluxo de dados*
Número de passadas	Múltiplas	Única
Tempo de processamento	Ilimitado	Restrito
Uso de memória	Ilimitado	Restrito
Tipo de resultado	Preciso	Aproximado
Distruibuido?	Não	Sim

Fonte: Gama (2010, p. 9).

Devido à natureza dinâmica dos fluxos de dados, Gama (2010) sugere que para se extrair informações destes, o algoritmo de AM precisa estar adaptado para: receber novos dados gradualmente, não ter controle sobre a ordem com que os dados chegam, se adaptar ao tamanho variável do fluxo e lidar com recursos computacionais limitados. Entre as soluções propícias para tratar dessas necessidades, estão os algoritmos incrementais, pela sua capacidade de treinar e atualizar o classificador a cada nova instância, dispensando o seu armazenamento e múltiplas passadas sobre a mesma.

2.3 *Concept Drift*

Durante a tarefa de classificação, um classificador tenta predizer a classe y_i ($i = 1,...,c$) da instância $\boldsymbol{x}$, sendo essa predição baseada em estimar a distribuição D que representa a probabilidade conjunta $P(\boldsymbol{x},y_i)$. Definimos um conceito como uma distruibuição D_t em um tempo t, como representado na Equação (5) (KHAMASSI et al., 2016).

$$D_t = \{P_t(\boldsymbol{x},y_1), P_t(\boldsymbol{x},y_2),..., P_t(\boldsymbol{x},y_c)\} \tag{5}$$

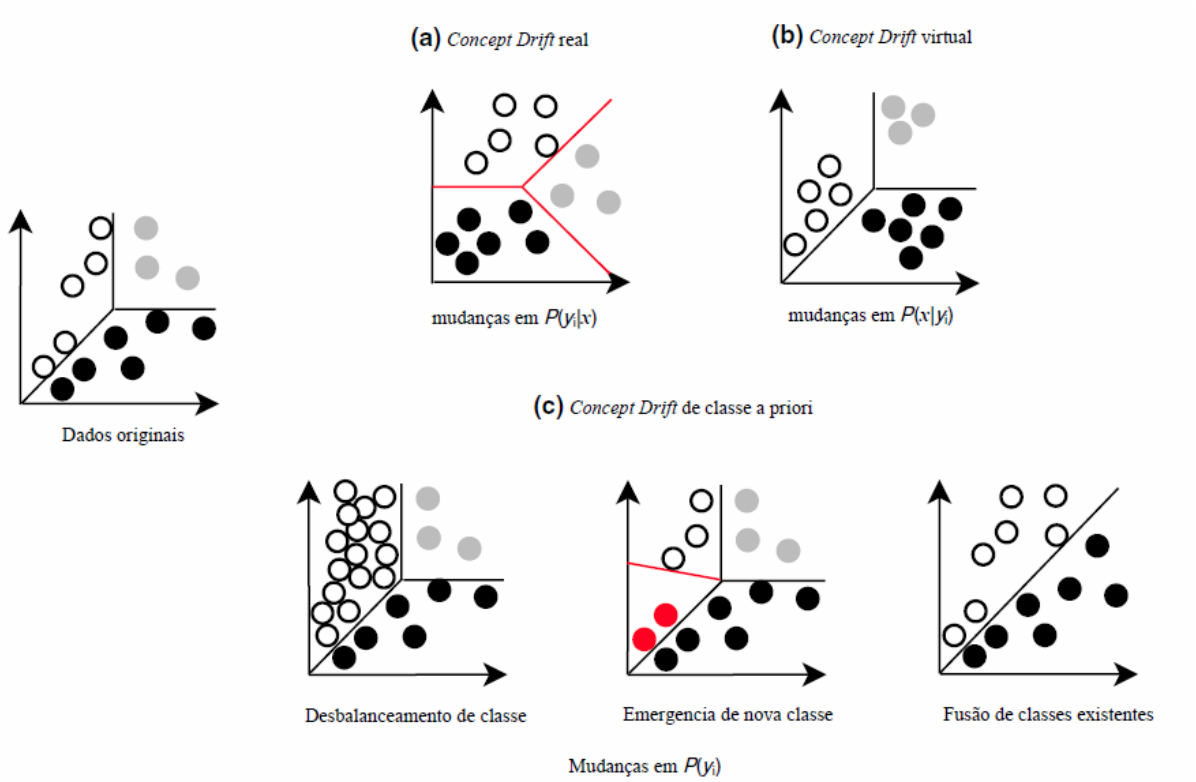
Um dos fenômenos que tornam os fluxos de dados um ambiente em evolução é o *concept drift*, termo do inglês utilizado para representar a ocorrência de uma alteração no conceito ao longo do tempo, ou seja, quando dados dois tempos t_1 e t_2, $D_{t_1} \neq D_{t_2}$. Essa anomalia pode se manifestar em partes distintas do cálculo de $P(\boldsymbol{x},y_i)$, resultando nos diferentes tipos de desvios detalhados abaixo:

• *Concept Drift* real: neste tipo de desvio, existe uma alteração na probabilidade *a posteriori* $P(y_i|\boldsymbol{x})$, ou seja, a probabilidade de a instância com o vetor de atributos $\boldsymbol{x}$ pertencer à determinada classe y_i . Considerado o tipo de desvio de maior impacto, pois afeta diretamente a eficiência do classificador resultando em mais predições incorretas devido à movimentar as fronteiras de decisão como mostrado na Figura 2.3a.

• *Concept Drift* virtual: quando existe uma alteração na probabilidade de dado que uma instância pertence à classe y_i, ela possuir um conjunto de atributos iguais a $\boldsymbol{x}$, ou seja, $P(\boldsymbol{x}|y_i)$. Afeta a distruibuição de instâncias dentro das fronteiras de decisão como mostrado na Figura 2.3b.

• *Concept Drift* de classe *a priori*: relacionado à probabilidade *a priori* das classes, podendo ser causado por um desbalanceamento no número de instâncias por classe, emergência de uma nova classe ou pela fusão de classes existentes, como apresentado na Figura 2.3c.

Figura 2.3 – *Concept Drift* sobre um conjunto de dados



Fonte: Adaptação de Khamassi et al. (2016, p. 4).

Também podem ser classificados em relação à velocidade com que a transição entre conceitos ocorre:

• Abrupto: se refere à mudança brusca de conceito. Ocorre quando em até dado instante t_1 as instâncias pertencem ao conceito D_{t_1} e a partir de um instante t_2, onde $t_1 < t_2$, as instâncias passam a ser geradas em sua plenitude de um conceito D_{t_2} e $D_{t_1} \neq D_{t_2}$.

• Gradual: diferentemente do abrupto, no gradual existe um período onde instâncias podem pertencer à conceitos diferentes, até que haja a transição completa de um conceito A para um conceito B.

2.4 Adaptação ao *Concept Drift*

Detectar e adaptar-se à alterações de conceito em fluxos de dados é um dos problemas do aprendizado em fluxos de dados em evolução, e os métodos para se adaptar à essas alterações podem ser divididos em duas subcategorias: adaptação cega e adaptação informada(ativa).

2.4.1 Adaptação Cega

A adaptação cega ao *Concept Drift* ocorre quando o algoritmo utilizado, lida de forma passiva com os problemas causadas por alterações de conceito. Esse resultado pode ser obtido pois algoritmos dessa categoria realizam o descarte de conceitos antigos em uma velocidade constante, ocorrendo alterações de conceito ou não (KHAMASSI et al., 2016).

Um exemplo de abordagem que utiliza adaptação cega para lidar com *Concept Drift* são comitês classificadores onde o critério para a atualização ou substituição de um membro do *ensemble* é o preenchimento de uma janela de tamanho fixo para treinamento. Nesse cenário, ocorrendo ou não uma troca de conceito o comitê será atualizado em função das últimas instâncias de dados.

Esse tipo de adaptação é especialmente eficiente em classificadores *ensemble* para situações de troca de conceito gradual, pois ao passo em que ocorre a troca de conceito também ocorre a atualização do modelo. No entanto, em situações de troca de conceito abruptas, a velocidade de adaptação será a mesma, resultando em um potencial atraso na resposta ao incidente.

2.4.2 Adaptação Informada

No caso de algoritmos que utilizam adaptação informada, também conhecida como adaptação ativa, é necessário o uso de algum mecanismo de detecção aliado de uma resposta ativa às notificações lançadas por ele, para se lidar com as consequências de uma situação de *Concept Drift.* Quando atingidas essas duas condições, é possível alcançar uma resposta mais rapida e uma melhor adaptação ao conceito atual do que quando utilizados mecanismos de adaptação cega (GOMES et al., 2017a).

De modo geral, os métodos de detecção de desvio de conceito tratados nesse trabalho funcionam adicionando etapas no processo de treinamento, que analisando estatísticas do desempenho do modelo conseguem emitir sinalizações de alerta de possível mudança de conceito ou de mudança identificada. Dois desses mecanismos serão abordados nas seções a seguir: o DDM e o EDDM.

2.4.2.1 Drift Detection Method

O *Drift Detection Method* (DDM) funciona criando um classificador base, e o treinando em paralelo com o classificador principal. Ele também mantém atualizada uma medida chamada *error-rate* que é calculada em função do número de erros de predição do classificador, e se pode ser interpretada como a probabilidade p_i de uma nova instância ser incorretamente classificada, com desvio padrão s_i calculado através da Equação (6) (GAMA et al., 2004).

$$s_i = \sqrt{p_i(1 - p_i) \,/\, i} \tag{6}$$

O método DDM considera que em um cenário estacionário, a *error-rate* deveria diminuir ou se manter estável, a medida em que o classificador se adapta ao conceito a ser aprendido. Contudo, em um cenário de mudança de conceito, haveria um aumento anormal na *error-rate* devido ao classificador não representar corretamente instâncias do novo conceito, ocasionando repetidos erros de predição.

São utilizadas duas sinalizações distintas:

• Caso $p_i + s_i \geq p_{min} + 2 \cdot s_{min}$, a situação é interpretada como um possível desvio, e um aviso é lançado.

• Caso $p_i + s_i \geq p_{min} + 3 \cdot s_{min}$, é considerado como a confirmação da ocorrência de uma alteração de conceito e neste caso, um aviso de alteração detectada é lançado.

É eficiente na detecção de *concept drift* abruptos, onde é esperado que repentinamente haja uma queda na acurácia do modelo no momento em que o novo conceito entre em vigor. Porém, o mesmo não é válido para cenários de *concept drift* gradual, onde a transição pode ser lenta e suave, de maneira que não seja refletida em um aumento brusco da *error-rate*.

2.4.2.2 Early Drift Detection Method

Baseado no DDM, o *Early Drift Detection Method*(EDDM) foi proposto para lidar eficientemente com os cenários de *concept drift* gradual, enquanto mantém a boa performance para os cenários de *concept drift* abrupto (BAENA-GARCÍA et al., 2006). Para isso é utilizada uma nova medida referente à distância entre dois erros de classificação.

Considera-se que conforme o modelo vai aprendendo e melhorando sua habilidade de predição, a distância média entre dois erros aumenta. Portanto, são calculados os valores p'_i representando essa distância média e seu desvio padrão s'_i, e armazenados os valores p'_{max} e s'_{max}.

Similar ao DDM, o EDDM também utiliza dois tipos de sinalização:

• Caso $(p_i + s_i)/(p_{min} + 2 \cdot s_{min}) < \alpha$, a situação é interpretada como um possível desvio, e um aviso é lançado.

• Caso $(p_i + s_i)/(p_{min} + 2 \cdot s_{min}) < \beta$, é considerado como a confirmação da ocorrência de uma alteração de conceito e neste caso, um aviso de alteração detectada é lançado.

Como resultado de experimentos, Baena-García et al.(2016) recomenda o uso dos valores $\alpha = 0.95$ e $\beta = 0.90$.

2.5 *Feature Drift* e Relevância de *features*

Um dos problemas menos estudados na literatura referente à fluxos de dados em evolução, é o *Feature Drift*. Esta situação que pode ser caracterizada como um tipo de *concept drift* a parte, ocorre quando em dois momentos distintos, o sub-conjunto ótimo de *features* relevantes é diferente, ou seja, alguma *feature* antes irrelevante se tornou relevante, ou o contrário ocorreu (BARDDAL et al., 2016).

Para a análise de *feature drift* é necessária a definição de conceitos como a relevância de *features*, algo que embora não haja um consenso entre os autores, geralmente é divida em três classificações: relevante, irrelevante e redundante.

Segundo a definição de Barddal el al. (2016), uma *feature* D_i é classificada como revelante caso, assumindo $S_i = D \setminus \{D_i\}$ obedeça a regra (1):

$$\exists S'_i \subset S_i, tal\ que\ P[Y|D_i, S_i] \neq P[Y|S'_i] \text{ seja verdade} \quad (1)$$

Portanto, caso a remoção de uma *feature* D_i afete diretamente a capacidade preditiva de um classificador, ela é considerada relevante, caso contrário, é considerada irrelevante.

Existe também a definição para *features* redundantes, onde, assumindo a mesma situação, uma *feature* D_i é redundante caso obedeça a regra (2):

$$\exists S_i' \subset S_i, tal\ que\ P[Y|D_i, S_i] \neq P[Y|S_i]\ e\ P[Y|S_i] \neq P[Y|S_i'] \text{ seja verdade} \quad (2)$$

O problema causado por *features* redundantes, ou seja, aquelas que possuem alguma outra *feature* correspondente que influencie na predição da mesma maneira que ela, é a capacidade de influenciar negativamente no aprendizado, adicionando uma importância artificial à atributos redundantes por estarem em maior número. Isso pode ser refletido em forma de *overfitting* em relação aos dados de treinamento.

É considerado um tipo de *concept drift* a parte, pois pode ou não resultar em uma diferença na probabilidade *a posteriori* de uma instância pertencer à determinada classe, evoluindo de um cenário de *concept drift* virtual para um real. Em consequência disso, é um desafio existente se adaptar em tempo real às mudanças de relevância das *features.*

2.6 Redução do espaço de *features*

Em cenários onde os dados a serem processados possuem alta dimensionalidade do espaço de *features*, uma das consequências é justamente o processamento de informações pouco relevantes ou duplicadas. Como saída para esse problema, são utilizados diferentes tipos de técnicas para redução do espaço de features em sub-espaços que representem todo, ou uma parte do conjunto completo.

2.6.1 *Random Subspace Method*

O *Random Subspace Method* (RSM), é um método para diminuição da dimensionalidade do espaço de *features*, onde não é considerada a relevância de cada *feature*. Nele são selecionadas aleatoriamente *k* dimensões distinstas de um dado espaço de *features n*-dimensional *D*, de modo que $1 \leq k \leq n$, para formar um sub-espaço D_x que é atribuido à um classificador *x*, de um comitê classificador. Por fim, as instâncias de treinamento são projetadas para esse novo sub-espaço D_x quando utilizadas para o treinamento do classificador *x*.

Foi inicialmente proposto no artigo de Ho(1998), onde a autora utiliza o método para adicionar aleatoriedade na geração de árvores de decisão para o algoritmo de Florestas Aleatórias. No caso, ainda é proposta a utilização de um sub-espaço diferente a cada repartição da árvore, no intuito de aumentar ainda mais a diversidade entre os classificadores do comitê.

Neste mesmo estudo é comparada a eficiência de modelos treinados a partir do método de sub-espaços aleatórios, e daqueles treinados utilizando apenas outras técnicas de criação de independência entre os classificadores, como *boosting* e *bootstrapping*. Aqueles treinados usando RSM obtiveram os melhores resultados quando escolhido valor de *k* (dimensionalidade dos sub-espaços) igual a metade do conjunto total de *features*.

Algoritmo 2.6.1: RSM

Algorithm 1 Random Subspace Method

```
1: procedure SELECIONASUBESPACOSALEATORIOS(D, k)
2:     subspaceSize = k
3:     featureSpace = D
4:     for each classificador x in ensemble do
5:         D_x = selecionaDimensoesAleatorias(k, featureSpace)      ▷ seleciona k dimensoes
6:     end for
7: end procedure
```

Fonte: Adaptado de Ho (1998).

2.6.2 *Feature Selection* e *Fast Correlation Based Filter*

Selecionar um sub-conjunto de *features*, utilizando como critério a sua relevância, é um método de redução da dimensionalidade do espaço de *features* que vêm sendo utilizado de maneira crescente como solução para problemas atuais. Cenários de processamento de texto, por exemplo, podem possuir milhares(ou até centenas de milhares) de atributos, representando a presença ou não de uma palavra no texto, onde nesses casos, apenas uma ordem de grandeza muito inferior é realmente relevante para o conceito a ser aprendido.

Possui duas categorias principais, com relação ao modo como as *features* são selecionadas. O primeiro modelo é o de filtros, onde são utilizadas heurísticas baseadas nas características dos dados para selecionar um sub-conjunto de *features* relevantes, sem necessitar nenhum treinamento externo. Já o segundo modelo, os *wrappers*, utilizam algum mecanismo externo de aprendizado de máquina com a finalidade de aprender a selecionar esse mesmo sub-conjunto (YU; LIU, 2003).

Embora os *wrappers* possuam maior fidelidade ao cenário real de treinamento, são também computacionalmente mais custosos, dificultando ou até inviabilizando seu uso em

aplicações que necessitam resposta em tempo real, como é o caso do aprendizado a partir de fluxos de dados (MOLINA et al., 2002).

Algoritmo 2.6.2: FCBF

Algorithm 2 Fast Correlation Based Filter

```
1:  procedure SELECIONAFEATURESBOAS(dados, threshold)
2:      S_boas = {}
3:      for each feature F_i do
4:          SU_i,c = calculaSU_i,c()          ▷ calcula o SU entre a feature e a classe
5:          if SU_i,c >= threshold then
6:              insere(F_i, S_boas)           ▷ insere a feature no conjunto de features boas
7:          end if
8:      end for
9:      ordena(S_boas)                        ▷ ordena em ordem descrescente as features relevantes
10:     F_p = selecionaPrimeiro(S_boas)
11:     while F_p ≠ NULL do
12:         F_q = selecionaProximo(S_boas, F_p)
13:         while F_q ≠ NULL do
14:             F_r = F_q
15:             if SU_p,q >= SU_q,c then      ▷ verifica redundancia entre as features
16:                 remove(F_q, S_boas)       ▷ remove a feature q se for redundante
17:                 F_q = selecionaProximo(S_boas, F_r)
18:             else
19:                 F_q = selecionaProximo(S_boas, F_q)
20:             end if
21:         end while
22:         F_p = selecionaProximo(S_boas, F_p)
23:     end while
24:     return S_boas
```

Fonte: Adaptado de Yu e Liu (2003)

Sendo assim, neste estudo serão abordados os seletores do tipo filtro, mais especificamente o *Fast Correlation Based Filter* (FCBF) descrito no algoritmo 2.6.2. No seu artigo, Yu e Liu(2003) testaram a eficiência deste filtro, onde apresentou um baixo custo computacional e se mostrou efetivo em cenários das mais abrangentes dimensionalidades quando comparado a outros filtros.

O algoritmo FCBF assume que *features* são boas, e devem ser mantidas, quando são altamente relacionadas e classe, portanto relevantes, mas não relacionadas à outras *features*, ou seja, são não redundantes. Para calcular os valores dessas relações, é utilizada a medida *symmetrical uncertainty* como definida em Press et al. (1988).

3 TRABALHOS RELACIONADOS

O aprendizado a partir de fluxos de dados é um tema bastante estudado na literatura, e a seguir são apresentadas algumas das soluções já propostas por outros autores. Os algoritmos aqui listados lidam com a natureza não estácionaria e em evolução de maneiras distintas, e servem de influência para a metodologia proposta neste trabalho.

Algoritmos como o *Adaptive Random Forest* (ARF), utilizam adaptação informada para lidar com alterações de conceito, já outros como o HEFT aproveitam de classificadores base adaptados para lidar de forma passiva com esse problema. Além disso, adicionar diversidade em classificadores *ensemble* em cenários *online* se torna um desafio pela incapacidade de se aplicar de maneira direta métodos do AM convencional, como *bagging* e *boosting*, que dependem da divisão do conjunto de treinamento. Por isso, trabalhos como o *Online Bagging* propõem alternativas de métodos utilizados em *batch learning* para o cenário *online* dos fluxos de dados.

3.1 *Online Bagging*

Com o surgimento dos classificadores *ensemble*, também surgiu a necessidade de criar classificadores heterogêneos, como analisado por Krogh e Vedelsby (1995). Sendo assim, um dos métodos de uso difuso dentro do AM para introduzir diversidade foi o *Bagging*, que teve sua eficiência demonstrada no trabalho de Domingos (1997).

No *Bagging* utillizado em *batch learning* os conjunto de treinamento são individuais para cada classificador do comitê, sendo compostos pela seleção de *n* elementos do conjunto de dados inicial, com reposição. Deste modo, é inserido um elemento de aleatoriedade, onde os classificadores darão mais importância a determinadas instâncias por serem treinados com duplicatas de certas instâncias como também nem chegarão a ser treinados com algumas outras.

O *Online Bagging* foi um método proposto por Oza e Russell (2001), com a finalidade de traduzir o funcionamento do *Bagging* em cenários estacionários, para os cenário *online* dos fluxos de dados. Para isso, utiliza uma distribuição de Poisson com variância $\lambda = 1$. O processo de treinamento de um comitê composto por M classificadores utilizando *Online Bagging* é demonstrado no algoritmo 3.1.

Algoritmo 3.1: *Online Bagging*

Algorithm: *Online Bagging* para M classificadores

1: Dado um conjunto de classificadores base $m \in \{1, 2, ..., M\}$
2: **for each** instância de treino x **do**
3: **for** $m = 1, 2, ..., M$ **do**
4: $w = Poisson(1)$
5: Treina o classificador m atribuindo peso w à instância x

Fonte: Traduzido de Bifet, Holmes e Pfahringer (2010).

3.2 *Adaptive Random Forests*

O *Adaptive Random* Forests (ARF) é baseado diretamente no algoritmo de Florestas Aleatórias proposto para *batch learning*, buscando trazer os benefícios já estudados extensivamente e comprovados teóricamente no ambiente estacionário, para o cenário não estacionário dos fluxos de dados (GOMES et al., 2017a). Com isso em mente, é um comitê de árvores de decisão que tenta manter as propriedades originais do método escolhendo como classificador base uma modificação das *Hoeffding Trees* que utiliza RSM a cada divisão de nós.

Como proposta para lidar com a natureza evolutiva dos fluxos de dados, utiliza um detector de *concept drift* acoplado à cada classificador do *ensemble*. Caso um aviso de possível alteração de conceito seja lançado, é iniciado o treinamento de um novo classificador, que irá substituir o classificador correspondente ao detector no caso da confirmação da ocorrência de uma mudança de conceito. Esta confirmação vem através de um aviso de mudança confirmada, pelo mesmo detector.

Uma das principais razões para essa abordagem, de utilizar o treinamento de classificadores em segundo plano para o caso de necessidade de substituição, é explicada na proposta inicial de Gomes et al. (2017a). O mesmo cita a rápida adaptação a diferentes alterações de contexto e melhor representação do contexto atual, visto que o modelo começa a ser treinado a partir da detecção de uma possível mudança, assim conhecendo informações do novo contexto.

Embora acrescente de fato a possibilidade para uma rápida adaptação, é adicionada também a chance de existirem o dobro de classificadores sendo treinados simultaneamente, do que inicialmente estabelecido. Essa situação pode levar à grandes impactos no tempo de processamento, dependo da complexidade individual de cada classificador.

Como contramedida à este problema, são propostas duas versões do mesmo algoritmo, sendo uma sequencial e outra com paralelismo integrado, possibilitando o treinamento em paralelo das árvores. No entanto, ocorre uma troca de recursos computacionais no segundo modelo, pois a quantidade de memória primária necessária aumenta consideravelmente, o que é refletido nos resultados de Gomes et al. (2017a) quando comparadas as duas versões.

Com a finalidade de aumentar ainda mais a diversidade entre os classificadores do ensemble, introduz *Online Bagging* no treinamento das árvores, porém, utiliza $\lambda = 6$ na distribuição de Poisson. Essa decisão de utilizar um valor diferente para λ vem do estudo realizado por Bifet, Holmes e Pfahringer (2010), onde é utilizado um valor de λ maior para diminuir a chance de não utilizar uma instância no treino de um classificador, e portanto aumentar a chance de usá-la uma ou mais vezes.

Por fim, como forma de combinar os votos do comitê são atribuídos pesos aos votos individuais de cada classificador, de acordo com à sua respectiva acurácia. Essa medida visa atribuir maior relevância à classificadores com melhor desempenho, enquanto os votos de classificadores com desempenho ruim tendem ter pouca influência na decisão final. Contudo, para viabilizar essa abordagem é necessário manter estatísticas individuais sobre cada classificador, o que requer testes durante os treinos e é diferente de métodos mais ingênuos como o voto majoritário simples, onde todos os classificadores tem a mesma influência.

A sua implementação possui código aberto e está inclusa no *framework* de simulação de fluxos de dados para mineração MOA[2] (Massive Online Analysis). Uma das consequências disso é a possibilidade de realizar testes fiéis à implementação original dos autores.

3.3 *Heterogeneous Ensemble for Feature Drifts in Data Streams*

No espectro de algoritmos que se propõe a resolver o problema de *feature drift* está o algoritmo proposto por Nguyen et al. (2012), *Heterogeneous Ensemble for Feature Drifts in Data Streams* (HEFT). Consiste em utilizar adaptações de classificadores, projetadas para lidar com *concept drift* passivamente, enquanto utiliza uma heurística para detectar e se adaptar expecificamente à *feature drifts.*

[2] **MOA *Machine Learning for Streams***. Disponível em: <https://moa.cms.waikato.ac.nz/>

O algoritmo HEFT pode ser dividido em duas partes: de treinamento, onde utiliza uma solução para seleção de *features* e classificadores em tempo constante, focando na criação de um *ensemble* heterogêneo, e de classificação, onde une os votos individuais dos classificadores em uma decisão final.

Como um dos recursos usados para adicionar maior diversidade ao comitê, ele disponibiliza a opção de utilizar uma lista composta de diferentes tipos de classificadores base, criando assim a possibilidade de ser gerado um comitê misto. Na proposta original são utilizadas adaptações dos algoritmos destes classificadores, modificadas para lidar com desvios de conceito ao longo do tempo.

Embora os classificadores base utilizados sejam capazes de lidar com *concept drift* de maneira passiva, eles sofrem das consequências causadas pela adaptação cega, onde a adaptação ao *concept drift* abrupto ocorre no mesmo ritmo que a adaptação ao *concept drift* gradual.

Algoritmo 3.2a: HEFT - Treinamento

Algorithm: HEFT - Treinamento

Input: Um fluxo de dados $\mathbf{X} = [\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_t}, \ldots]$, onde $\mathbf{x_t}$ é um vetor de features p dimensional $[f_t^1, f_t^2, \ldots, f_t^p]^T$ com rótulo y_t chegando no tempo t, $y_i \in \{y_1, y_2, \ldots, y_c\}$. Um conunto de l tipos de classificadores base diferentes, $\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_l\}$.
Output: Um ensemble heterogêneo $\mathcal{E}$.

1: Inicializa o ensemble $\mathcal{E}$ com k classificadores de cada tipo em $\mathcal{M}$, chamados $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{k*l}$.
2: **while** X ainda possuir instâncias **do**
3: **if** *chunk* não estiver cheia **then**
4: Adiciona x_i à *chunk*.
5: **else**
6: Aplica FCBF para conseguir o conjunto de features relevantes φ_i.
7: **if** $\varphi_i \neq \varphi_{i-1}$ **then**
8: Encontra o melhor classificador $\mathcal{C}_{best}$ que possuir o menor erro agregado no ensemble e copia o seu tipo.
9: Constrói um novo classificador $_{new}$ do mesmo tipo de $\mathcal{C}_{best}$, utilizando o sub-espaço de features φ_i.
10: Remove o classificador com pior acurácia de $\mathcal{E}$.
11: Adiciona o novo classificador $\mathcal{C}_{new}$ into $\mathcal{E}$.
12: **end if**
13: **for** each classificador C in ensemble **do**
14: **for** each instância x in *chunk* **do**
15: Seleciona m de acordo com distruibuição Poisson(1)
16: Treina m vezes o classificador C com a instância x.
17: **end for**
18: **end for**
19: **end if**
20: **end while**

Fonte: Traduzido de Nguyen et al. (2012).

Com o objetivo de lidar com *feature drifts*, e possíveis *concept drifts* abruptos, adiciona um mecanismo para detectar alterações na relevância de *features*. Utiliza um sistema de janela deslizante e uma alteração do FCBF para determinar o conjunto de *features* relevantes em tempo real, e considera a existência de um *feature drift* caso esse conjunto venha a mudar entre dois instantes consecutivos.

A cada vez que uma *chunk* (a janela deslizante de tamanho fixo) é preenchida com os dados originados do fluxo, é realizada uma seleção de atributos e a seleção passada é usada para comparação. Caso seja detectado um possível *feature drift*, o classificador com menor acurácia é removido e um novo classificador é adicionado no seu lugar utilizando o novo sub-espaço de atributos relevantes.

Sendo assim, existe a premissa implícita de que todo *concept drift* abrupto resulta em um *feature drift*, o que não necessariamente precisa acontecer.

Utiliza a versão original do algoritmo *Online Bagging*, onde $\lambda = 1$. O pseudo-código da metodologia usada para o treinamento no HEFT, como proposto originalmente em Nguyen et al. (2012), é apresentado no algoritmo 3.2a.

Algoritmo 3.2b: HEFT - Classificação

Algorithm: HEFT - Classificação

Input: Uma nova instância de teste x_t.
Um ensemble $\mathcal{E}$ de N classificadores, chamados $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_N$. Todo classificador $\mathcal{C}_i$ é associado com um sub-espaço de features φ_i.
Dada uma instância de testes, cada classificador $\mathcal{C}_i$ retorna um vetor de probabilidades $\rho_i = [\rho_{i1}, \rho_{i2}, \ldots, \rho_{ic}]$, $(\sum_{j=1}^{c} \rho_{ij} = 1,\ i \in \{1, 2, \ldots, N\})$.
Output: O vetor de distruibuições de probabilidade do ensemble para x_t.

1: for each classificador $\mathcal{C}_i$ no ensemble $\mathcal{E}$ do
2: Projeta a instância x_t no sub-espaço φ_i conseguindo x'_t.
3: Calcula o vetor de probabilidades $\rho_i = \mathcal{C}_i(\mathrm{x}'_t)$.
4: Calcula o erro agredado de $\mathcal{C}_i$, err_i, e atribui um peso ao classificador usando a formula $w_i = 1/(err_i + \alpha)$.
5: end for
6: Junta todos os vetor de probabilidade multiplicando pelos seus pesos:

$$\mathcal{E}(\rho) = \sum_{i=1}^{z} w_i * \rho_i$$

7: Normaliza e retorna o vetor $\mathcal{E}(\rho)$.

Fonte: Traduzido de Nguyen et al. (2012).

Existem diferentes maneiras de combinar os votos de cada classificador, a fim de gerar a decisão do comitê. No caso do HEFT, é utilizado o cálculo do erro agregado individual de cada classificador para definir a relevância do seu voto na decisão final.

Embora esse método de pesagem dos votos tenha se mostrado eficiente na tarefa de classificação, ele adiciona a necessidade de realizar testes de classificação durante o treinamento para a obtenção de estatísticas sobre o erro dos classificadores. O processo de classificação do HEFT é demonstrado mais detalhadamente no algoritmo 3.2b.

Infelizmente esse algoritmo não possui código aberto, impossibilitando a realização de *benchmarks* e comparações com outros algoritmos do estado da arte. Em função disso, algumas questões como o critério utilizado na criação dos classificadores iniciais do comitê ficam em aberto, visto que não é explicitado o sub-espaço de *features* atrelado a cada um desses classificadores de primeira geração, antes de serem substituidos pela novas gerações em decorrência da detecção de *feature drifts*.

4 ABORDAGEM PROPOSTA

Como visto até agora, devido às limitações impostas pelos fluxos de dados, a extração de conhecimento a partir destas fontes de informação requer cuidados especiais como a proposição de novas técnicas de AM incremental e a adaptação de técnicas de AM convencional para esse ambiente desafiador, não estácionario e em evolução.

Técnicas envolvendo comitês de classificadores têm sido uma opção crescente para lidar com esses problemas, tendo Gomes et al. (2017b) descrito diferentes estruturas de comitês e os cenários onde possuem maior relevância. Em Wolpert e Macready (1997) é proposto o "*No Free Lunch Theorem"*, onde é demonstrado que não existe algoritmo ótimo para todas as situações, incluindo os fluxos de dados e classificadores *ensemble*.

4.1 Algoritmos propostos

Portanto, com o objetivo de analisar o impacto e a viabilidade dos diferentes mecanismos utilizados nas soluções atuais, como a redução de dimensionalidade dos dados através da seleção de sub-espaços de *features* e o uso de adaptação ativa à alterações de conceito utilizando detectores, são propostas diferentes modificações de um mesmo algoritmo *ensemble* a serem comparadas.

4.1.1 Algoritmo do classificador controle

Para possibilitar a comparação das modificações com e sem esses mecanismos, é proposto um novo algoritmo a ser tomado como controle. Este algoritmo não utiliza técnicas de seleção de atributos relevantes ou adaptação ativa à alterações de conceito, em vez disso, utiliza apenas RSM e adaptação cega.

Consiste de um comitê de *M* Árvores de *Hoeffding*, inicializadas utilizando um sub-espaço aleatório individual, selecionado através de RSM com número de dimensões igual a metade do número total de dimensões do espaço completo, valor recomendado por Ho(1998).

A escolha pelas Árvores de *Hoeffding* se dá pela ausência de adaptação cega embutida nesse tipo de classificador. Sendo assim, é necessária a adição de meios externos para lidar ativa ou passivamente com possíveis alterações de contexto, o que possibilita a comparação desses mecanismos utilizando o mesmo classificador base.

É utilizada a abordagem de janela *landmark* para substituir classificadores. Neste caso, a cada vez que uma janela de tamanho *n* é preenchida, ocorre a substituição do classificador com pior acurácia, por um novo classificador treinado nas *n* instâncias da janela. O classificador com pior acurácia é determinado como aquele que classificou incorretamente o maior número de instâncias, desde a ultima substituição. Como consequência, o comitê tende a se atualizar ao contexto mais recente com o tempo, lidando de maneira passiva com *concept drift*, o que será utilizado como controle.

O critério utilizado na junção dos votos individuais no voto do *ensemble* é o de voto majoritário com pesos, sendo utilizado como peso o valor da acurácia do classificador desde a última substituição. Esse critério também é utilizado em classificadores do estado da arte, como HEFT e ARF para atribuir maior relevância aos votos dos classificadores que apresentam maior eficiência. No entanto, nesse algoritmo ele foi adaptado para usar apenas a acurácia desde a última substituição de classificadores, assim, classificadores que obtiveram um bom desempenho no passado não necessariamente terão alta influência na votação de instâncias recentes.

Algoritmo 4.1.1: Treinamento - OSM

Algorithm 1: Treinamento - OSM

```
   Entrada: Tamanho da janela n, Tamanho do comite M
   Dados: Um fluxo de dados F
 1 for i=0 ate M do
 2     Gera novo sub-espaco aleatorio S_i usando RSM
 3     Cria nova Hoeffding Tree C_i
 4     Atribui S_i a C_i
 5     Adiciona C_i ao comite
 6 for each instancia_x de F do
 7     if janela esta cheia then
 8         Remove o classificador com pior acuracia do comite
 9         Reseta a estatistica de acuracia dos classificadores
10         Gera novo sub-espaco aleatorio S_x usando RSM
11         Cria nova Hoeffding Tree C_x
12         Atribui S_x a C_x
13         Adiciona C_x ao comite
14         for each instancia_y de janela do
15             Treina C_x com instancia_y
16     else
17         Adiciona instancia_x a janela
18     for each classificador C_z de comite do
19         Seleciona m de acordo com a distribuicao de Poisson(1)
20         Treina C_z m vezes com instancia_x
21 return
```

Fonte: O Autor.

O treino é feito normalmente, de maneira incremental a cada instância, utilizando o método *Online Bagging* com λ = 1 para adicionar diversidade ao comitê. Esse algoritmo de controle será chamado *Online Subspace Method* (OSM) nos experimentos desse trabalho, e seu treinamento segue a descrição do pseudo-código ilustrado no algoritmo 4.1.1.

4.1.2 Algoritmo do classificador com *feature selection*

Como técnica para a seleção de *features* relevantes em tempo real é utilizado o filtro FCBF, com a modificação proposto no HEFT, onde o filtro é aplicado sobre as instâncias de uma janela deslizante assumindo a premissa de que estas são suficientemente representativas do conceito atual.

O novo algoritmo com *feature selection* será referenciado como "FS+OSM" e sua implementação segue a modificação evidenciada em vermelho, no pseudo-código apresentado a seguir, no algoritmo 4.1.2 .

Algoritmo 4.1.2: Treinamento – FS+OSM

Algorithm 2: Treinamento - FS+OSM

Entrada: Tamanho da janela n, Tamanho do comite M
Dados: Um fluxo de dados F

```
1  for i=0 ate M do
2      Gera novo sub-espaco aleatorio S_i usando RSM
3      Cria nova Hoeffding Tree C_i
4      Atribui S_i a C_i
5      Adiciona C_i ao comite
6  for each instancia_x de F do
7      if janela esta cheia then
8          Remove o classificador com pior acuracia do comite
9          Reseta a estatistica de acuracia dos classificadores
10         Seleciona o conjunto relevante de features S_x aplicando FCBF
             sobre as instancias da janela
11         Cria nova Hoeffding Tree C_x
12         Atribui S_x a C_x
13         Adiciona C_x ao comite
14         for each instancia_y de janela do
15             Treina C_x com instancia_y
16     else
17         Adiciona instancia_x a janela
18     for each classificador C_z de comite do
19         Seleciona m de acordo com a distribuicao de Poisson(1)
20         Treina C_z m vezes com instancia_x
21 return
```

Fonte: O Autor.

4.1.3 Algoritmo do classificador com *drift detector*

Para adicionar adaptação ativa à *concept drifts* ao algoritmo de controle, foi criada a modificação que adiciona um detector de *drifts* do tipo EDDM, e será chamada “DD+OSM” nas próximas seções. A escolha pelo método EDDM veio em decorrência da sua proposta de melhorar o desempenho do seu antecessor, o DDM, tanto nos ambientes de *concept drift* gradual como nos de *concept drift* abrupto.

Algoritmo 4.1.3: Treinamento – DD+OSM

Algorithm 3: Treinamento - DD+OSM

```
   Entrada: Tamanho minimo da janela n, Tamanho do comite M
   Dados: Um fluxo de dados F
 1 Cria novo drift detector D_eddm do tipo EDDM
 2 for i=0 ate M do
 3     Gera novo sub-espaco aleatorio S_i usando RSM
 4     Cria nova Hoeffding Tree C_i
 5     Atribui S_i a C_i
 6     Adiciona C_i ao comite
 7 for each instancia_x de F do
 8     Atualiza D_eddm
 9     sinal = Checa pelo status de D_eddm
10     if instancias na janela ≥ n and sinal ==mudanca detectada then
11         Remove o classificador com pior acuracia do comite
12         Reseta a estatistica de acuracia dos classificadores
13         Reseta o aprendizado de todos classificadores do comite
14         Gera novo sub-espaco aleatorio S_x usando RSM
15         Cria nova Hoeffding Tree C_x
16         Atribui S_x a C_x
17         Adiciona C_x ao comite
18         for each instancia_y de janela do
19             Treina C_x com instancia_y
20     else
21         if sinal == aviso detectado or janela nao vazia then
22             Adiciona instancia_x a janela
23     for each classificador C_z de comite do
24         Seleciona m de acordo com a distribuicao de Poisson(1)
25         Treina C_z m vezes com instancia_x
```

Fonte: O Autor.

O critério utilizado para lidar com as notificações de aviso e de mudança vindas desse detector foi a substituição do classificador com menor acurácia quando recebida uma notificação de mudança de conceito, além de resetar o aprendizado de todos os classificadores do comitê. Com isso, deseja-se testar o impacto da adição de um detector de *drifts* e adaptação ativa, em troca de uma menor frequência de atualização da estrutura do comitê quando

comparada a da abordagem de adaptação cega, implementada na versão de controle, por exemplo.

Além disso, é realizado o armazenamento de instâncias somente após uma notificação de aviso. Recebida esta notificação, as instâncias são armazenadas até a confirmação da ocorrência do *concept drift* vinda em forma de notificação de mudança. Caso essa notificação de mudança não venha a acontecer em até um número *m* máximo de instâncias, a janela é limpa e o algoritmo retorna ao estado de espera por uma nova notificação de aviso, para recomeçar a armezenar instâncias. Esse mecanismo de limpeza é adicionado para evitar o armazenamento infinito, em caso de alarme falso causado por ruídos, onde uma confirmação de alteração de contexto pode nunca chegar.

Quando ocorre uma notificação de mudança após a notificação de aviso, dentro do intervalo permitido, é realizada a verificação do número de instâncias dentro da janela. Caso o valor seja maior que o valor *n* mínimo estipulado, é realizada a substituição de um classificador do comitê, utilizando os mesmos critérios que o algoritmo de controle, por um treinado nas instâncias da janela. Também é resetado o aprendizado dos outros classificadores para que possam melhor se adaptar ao novo conceito, visto que não possuem mecanismos de esquecimento de conhecimento passado.

A opção de começar a armazenar instâncias desde a notificação de aviso, vem da abordagem utilizada no ARF, onde supõe-se que após um aviso existe grande chance de as instâncias corresponderem ao conceito mais recente, caso tenha ocorrido uma troca de conceito. Neste caso, existiria a possibilidade de uma adaptação mais rapida do que quando existe a espera pela confirmação da ocorrência para então começar o processo de adaptação.

O algoritmo utilizado na implementação da modificação DD+OSM é indicado no pseudo-código no algoritmo 4.1.3, onde as principais alterações estão evidenciadas em vermelho.

4.1.4 Algoritmo do classificador com *feature selection* e *drift detector*

Como ultima modificação do algoritmo de controle para comparação, é proposto o algoritmo DD+FS+OSM. Nele, são adicionados ambos os mecanismos, tanto de adaptação ativa à detecções de alteração de conceito como a utilização de um seletor de atributos relevantes em tempo real, para a criação de novos classificadores.

Essa versão do OSM tem como característica uma taxa de atualização do classificador significativamente menor do que a utilizada nos modelos FS+OSM e OSM, onde a atualização ocorre, independentemente, a cada *n* instâncias, sendo *n* o tamanho definido para a janela *landmark*. No DD+FS+OSM a substituição de classificadores ocorre somente quando detectada uma mudança de conceito, sendo também utilizado o conjunto de *features* relevantes, encontrado pelo filtro FCBF, para a criação do substituto.

A implementação do algoritmo DD+FS+OSM segue o pseudo-código apresentado no algoritmo 4.1.4, com as principais mudanças em relação ao algoritmo de controle OSM marcadas em vermelho.

Algoritmo 4.1.4: Treinamento – DD+FS+OSM

Algorithm 4: Treinamento - DD+FS+OSM

Entrada: Tamanho minimo da janela n, Tamanho do comite M
Dados: Um fluxo de dados F

```
Cria novo drift detector D_eddm do tipo EDDM
for i=0 ate M do
    Gera novo sub-espaco aleatorio S_i usando RSM
    Cria nova Hoeffding Tree C_i
    Atribui S_i a C_i
    Adiciona C_i ao comite
for each instancia_x de F do
    Atualiza D_eddm
    sinal = Checa pelo status de D_eddm
    if instancias na janela ≥ n and sinal ==mudanca detectada then
        Remove o classificador com pior acuracia do comite
        Reseta a estatistica de acuracia dos classificadores
        Reseta o aprendizado de todos classificadores do comite
        Seleciona o conjunto relevante de features S_x aplicando FCBF
         sobre as instancias da janela
        Cria nova Hoeffding Tree C_x
        Atribui S_x a C_x
        Adiciona C_x ao comite
        for each instancia_y de janela do
            Treina C_x com instancia_y
    else
        if sinal == aviso detectado or janela nao vazia then
            Adiciona instancia_x a janela
    for each classificador C_z de comite do
        Seleciona m de acordo com a distribuicao de Poisson(1)
        Treina C_z m vezes com instancia_x
```

Fonte: O Autor.

4.1.5 Classificação de novas instâncias

Com o objetivo de padronizar os critérios utilizados para a classificação de novas instâncias para uma melhor base de comparação, o algoritmo de controle OSM e suas modificações FS+OSM, DD+OSM e DD+FS+OSM, utilizam a mesma metodologia para definir o voto final do comitê.

O processo de agrupamento dos votos individuais consiste em projetar a nova instância *x* a ser classificada, individualmente no sub-espaço de *features* correspondente a cada um dos classificadores do *ensemble*. Com isso, essa a instância projetada é classificada por cada uma das *Hoeffding Trees*, retornando o vetor de probabilidades de cada possível classe, que é então escalado de acordo com a acurácia do classificador, através de uma multiplicação, e somado ao vetor de probabilidades final.

Sendo assim, o voto do comitê corresponde a classe com a maior probabilidade acumulada, de acordo com o algoritmo de classificação como apresentado no algoritmo 4.1.5 a seguir:

Algoritmo 4.1.5: Classificação – OSM e modificações

Algorithm 5: Classificacao - OSM e modificacoes

Entrada: Um comite E, Uma instancia x
Saida: Uma predicao de classe para a instancia x

```
1 for each classificador de E do
2     P_{x,c} =Projecao da instancia x no sub-espaco de features de C_i
3     voto_individual = acuracia(classificador) *
        classificacacao_individual(classificador,P_{x,c})
4     vetor_de_probabilidades += voto_individual
5 return max(vetor_de_probabilidades)
```

Fonte: O Autor.

4.1.6 Resumo dos algoritmos propostos

De modo geral, as principais diferenças entre os algoritmos propostos nas subseções anteriores podem ser resumidas na tabela 4.1.6 abaixo, que indica quais foram as modificações realizadas sobre o OSM.

Tabela 4.1.6: Principais diferenças entre os algoritmos

	Classificador			
Modificaçâo	**OSM**	**FS+OSM**	**DD+OSM**	**DD+FS+OSM**
Adaptação a concept drift	Passiva	Passiva	Ativa	Ativa
Seleção de features	Só RSM	RSM e FCBF	Só RSM	RSM e FCBF
Drift Detector	Nenhum	Nenhum	EDDM	EDDM

Fonte: O autor

4.2 *Datasets* utilizados

O conjunto de *datasets* escolhido para testar os algoritmos propostos anteriormente busca abrangir os mais diversos cenários existentes no contexto de classificação de dados. Portanto, é composto por fluxos de dados gerados sintéticamente e por conjuntos de dados coletados de situações reais, do cotidiano.

4.2.1 SEA

O SEA é um gerador de fluxos de dados sintético proposto por Street e Kim (2004) e utiliza um epaço de *features* tridimensional, onde os valores de cada atributo varia entre 0 e 10 e apenas 2 dos 3 influenciam na definição da classe. Nele, o espaço de *features* é dividido em quatro blocos e são definidos 4 valores para θ à serem usados como limitante na definição da classe da instância, utilizando o seguinte critério: caso $f1 + f2 \leq \theta$ a instância pertence à classe 1, se não, pertence a classe 2.

Na sua versão SEA_G ocorrem 3 *concept drifts* graduais, já na sua versão SEA_A ocorrem 3 *concept drifts* abruptos. O número de instâncias a serem geradas é configurável, sendo 10% delas correspondentes a ruído. Neste trabalho serão utilizadas as duas versões do algoritmo, ambas com 100.000 instâncias.

4.2.2 SEAFD

O SEAFD é um gerador de fluxos de dados sintético proposto como uma extensão do SEA por Barddal et al. (2015), herdando o mesmo contexto, onde apenas 2 de todas as *features* são relevantes para a definição da classe. Porém, agora nesta nova versão é simulado um fluxo de dados com ocorrência de *feature drifts* e com número de atributos configurável.

Como o SEA, ele também possui 2 versões: $SEAFD_G$ onde ocorrem 3 *feature drifts* graduais, e $SEAFD_A$ onde ocorrem 3 *feature drifts* abruptos. Serão utilizadas as duas versões, com 10% de ruído, 100.000 instâncias e 50 atributos diferentes em cada uma.

4.2.3 *Covertype*

O dataset *Covertype* representa o tipo de cobertura de floresta em blocos de 30 metros x 30 metros. É um conjunto de dados real coletado por instituições geológicas dos Estados Unidos, disponível no repositório UCI[3], sendo peculiar por apresentar uma dimensionalidade acima da média e uma grande quantidade de instâncias. No total apresenta 581.012 instâncias, 10 atributos numéricos e 44 binários, totalizando 54 atributos.

4.2.4 AGR

AGR é outro conjunto de dados sintético, desta vez construído através do gerador proposto em Agrawal et al. (1992). Utiliza dez funções diferentes para mapear as instâncias nas duas classificações existentes, utilizando os 6 atributos nominais e 3 contínuos de cada uma delas.

É adicionado ruído sobre os valores dos atributos usando um fator de perturbação de 10%. Assim como os outros datasets sintéticos vistos até agora, também possui sua versão AGR_G que apresenta 3 *concept drifts* graduais e sua versão AGR_A, que conta com 3 *concept drifts abruptos.*

4.2.5 HYPER

O dataset HYPER é gerado como apresentado por Hulten, Spencer e Domingos (2001), e simula a rotação de um hiperplano ao mudar seus pesos. Esse comportamento causa um tipo de *concept drift* entre o gradual e o abrupto, onde as instâncias não pertencem ao conceito inicial nem ao final durante o processo de transição, e sim a uma interpolação dois dois. Os parâmetros utilizados na geração deste conjunto de dados foram 10 atributos e magnitude 0,001, sendo geradas 100.000 instâncias.

4.2.6 *Electricity*

Este dataset foi coletado pelo *Australian New South Wales Electricity Market*, descrito por Harries (1999), e representa o preço variável da eletricidade. Nesse mercado os preços são atualizados a cada 5 minutos causando perturbações nos preços ao longo do tempo.

[3] **Covertype Data Set**. Diponível em <https://archive.ics.uci.edu/ml/datasets/covertype>

O conjunto de dados possui 45.312 instancias coletadas em intervalos de 30 minutos cada, e representa um cenário real de *concept drift.* As classes possíveis são apenas 2, que indicam se o preço está subindo ou descendo em comparação às últimas 24 horas, e nele, cada instância possui 8 atributos.

4.2.7 *Spam Corpus*

O dataset mais desafiador desta lista, pois além de apresentar dados reais possui dimensionalidade de 39.917 atributos binários. Cada instância representa um email e cada atributo representa a presença ou ausência de uma palavra no texto, sendo a classe o indicador se o e-mail é considerado SPAM ou NÃO SPAM. Esse conjunto de dados possui 9.324 instâncias e é conhecido por conter um *feature drift* gradual ao redor da instância 1.500 (BARDDAL et al., 2016).

4.3 Métricas utilizadas para avaliação do desempenho

Para a execução dos testes será utilizado o framework *Massive Online Analysis* (MOA) para simulação de fluxos de dados utilizando os datasets mencionados anteriormente, rodando em um processador i7 7700k de 8 threads e com 16 GB de RAM. Como base para análise do impacto das mudanças sobre o algoritmo controle, serão utilizadas as métricas a seguir.

4.3.1 *Prequential K-Fold Distributed Cross-Validation Evaluation*

O treinamento dos classificadores utilizará a abordagem *prequential*, ou como também chamada, teste-e-treino, onde primeiro uma instância é utilizada para teste do modelo, e em seguida disponibilizada para o treino do mesmo. Dessa maneira o classificador não possuí informações a respeito da instância a ser classificada antes do teste, evitando o cenário de *overfitting*, onde o modelo se adapta demais às instâncias de treino, ficando viciado e não tendo bom desempenho com dados desconhecidos.

Além disso, também é utilizado *K-Fold distributed cross-validation*, como sugerido em Bifet et al. (2015), onde são treinados *K* comitês simultâneos e a cada instância em um destes *K* comitês ela é utilizada apenas para teste, enquanto nos outros é utilizada para teste e treino.

Na versão do *K-fold cross-validation* utilizada em *batch learning* ocorre a divisão do conjunto de dados em *K* sub-conjuntos, sendo *K-1* são utilizados para treinamento de cada classificador, e apenas 1 utilizado para o seu teste. Esse comportamento é aproximado pela metodologia proposta por Bifet et al. (2015) para o cenário *online* dos fluxos de dados, sendo que a adição do treinamento *prequential* possibilita o melhor aproveitamento das instâncias no uso para testes do que na sua versão original.

4.3.2 Métricas de avaliação

São testadas tanto as estatísticas referentes ao desempenho do classificador na tarefa de classificação, como também as estatísticas referentes ao uso de recursos computacionais. Segue abaixo o conjunto completo de estatísticas a serem avaliadas:

• Acurácia *Prequential*: representa o desempenho geral do modelo, sendo a porcentagem de instâncias classificadas corretamente dentro de uma janela deslizante de tamanho fixo *n* , contendo as *n* instâncias mais recentes, como proposto em Gama et al. (2012). Porém, essa medida apresenta problemas em conjuntos de dados muito desbalanceados dando a falsa sensação de eficácia em algumas situações, como no caso do classificador majoritário que sempre prediz a classe com maior probabilidade *a priori* (a classe da maioria das instâncias).

• Kappa-m: em função dos problemas apresentados quando utilizado a acurácia isoladamente, Bifet et al. (2015) propõe uma métrica baseada na estatística Kappa de Cohen (1960). Essa nova métrica Kappa-m é baseada no classificador majoritário, onde este recebe valor Kappa-m = 0, e valores positivos indicam eficácia maior do que o classificador majoritário, e negativos significam uma eficácia inferior. O seu cálculo utiliza a equação (7), onde p_o indica a acurácia *prequential* do classificador e p_m indica a acurácia *prequential* do classificador majoritário:

$$k_m = \frac{p_o - p_m}{p_m} \tag{7}$$

• Tempo de execução: os testes serão realizados no mesmo ambiente, com exclusividade de recursos, e os valores serão tomados como a média de 10 execuções através da metodologia de 10-fold cross-validation.

• RAM-Hora: esta medida representa o uso de memória, considerando que uma unidade significa o uso de 1 GB de RAM durante 1 hora.

4.3.3 Teste de *Friedman* com ajuste *post-hoc*

Como maneira de verificar se houve diferença relevante na adição ou remoção de mecanismos de seleção de *features* e detecção de *concept drift*, serão realizados dois testes de Friedman, auxiliados por um procedimento *post-hoc* para comparação de algoritmos, como sugerido por Demsar (2006). Um destes testes será realizado sobre os resultados referentes à acurácia dos algoritmos, e o outro sobre os resultados de tempo de execução.

5 EXPERIMENTOS E RESULTADOS

Os experimentos relatados neste capítulo foram realizados de acordo com a metologia proposta no capítulo anterior. Sendo a acurácia prequential e estatísticas derivadas dela, calculadas sobre uma janela deslizante de tamanho igual a 5 mil instâncias.

A fim de padronizar os testes, os parâmetros dos algoritmos foram mantidos constantes durante todos os *datasets* e entre os diferentes algoritmos, quando possível. Portanto, o número de classificadores base em cada comitê foi definido como 10, e a janela utilizada no método *landmark*, quando existente, possuí capacidade para 500 instâncias.

5.1 Resultados Gerais

Os resultados gerais são a compilação de testes extensivos realizados sobre os diferentes *datasets* utilizando a abordagem de *Prequential K-Fold Distributed Cross-Validation Evaluation*, com K = 10, onde os resultados foram agrupados em tabelas para melhor visualização. Na Tabela 5.1a e na Tabela 5.1b são apresentadas as estatísticas médias de acurácia *prequential* e *kappa-m* do OSM e suas três modificações (FS+OSM, DD+OSM e DD+FS+OSM), além dos classificadores *ensemble* ARF e *Online Bagging* (OzaBag) representando um algoritmo do estado da arte que utiliza adaptação ativa e um que utiliza adaptação passiva, respectivamente, para comparação. Para isso, no OzaBag foram utilizadas *Hoeffding Adaptive Trees*, disponíveis no MOA e propostas por Bifet e Gavaldà (2009). As tabelas 5.1c e 5.1d são referentes às estatísticas de tempo de execução, em segundos, e uso de RAM-Hora, em GB por hora, respectivamente, representando o gasto computacional total da execução de todos os 10-folds da validação cruzada. Em todas as tabelas, os melhores resultados são sinalizados em negrito.

Com base na análise da Tabela 5.1a, é possível perceber que o algoritmo com maior desempenho na maioria dos casos é o que utiliza somente *feature selection*, FS+OSM, em adição ao mecanismo de adaptação passiva utilizado na versão de controle, OSM. Esse desempenho é refletido na tabela 5.1b, onde a estatística *kappa-m* se mantém positiva, representando desempenho bastante superior ao do classificador majoritário na mesma situação.

Em *datasets* como o $SEAFD_G$ e *Spam Corpus*, esse comportamento é esperado pela capacidade de se adaptar melhor ao conjunto de *features* realmente relevantes para o

aprendizado, quando grande parte do conjunto total de *features* é irrelevante. Em cenários de *concept drift* abrupto como AGR_A e $SEAFD_A$ a frequência de atualização do comitê utilizando janela *landmark* demonstrou ser suficientemente rápida para lidar melhor com alterações bruscas de contexto, quando comparado ao método utilizado nos classificadores com adaptação ativa.

Tabela 5.1a: Acurácia Prequential com desvio padrão dos algoritmos

ACURÁCIA PREQUENTIAL						
Dataset	OSM	FS.OSM	DD.OSM	DD.FS.OSM	ARF	OzaBag
$SEAFD_A$	84.80 ± 1.52	**87.46 ± 2.92**	84.68 ± 2.11	82.72 ± 4.29	85.75 ± 2.27	87.13 ± 1.25
$SEAFD_G$	85.02 ± 1.39	**87.15 ± 2.56**	85.35 ± 1.55	82.45 ± 4.11	85.51 ± 2.32	86.99 ± 1.07
SEA_A	85.86 ± 1.22	87.92 ± 1.75	86.60 ± 2.35	86.71 ± 2.48	**88.19 ± 1.55**	86.87 ± 1.70
SEA_G	86.07 ± 1.44	87.66 ± 1.93	86.54 ± 2.32	86.70 ± 2.18	**87.87 ± 1.78**	86.76 ± 1.85
Covertype	90.72 ± 4.72	**91.78 ± 4.23**	88.06 ± 5.10	87.90 ± 4.83	91.25 ± 4.40	85.84 ± 7.22
AGR_A	81.03 ± 8.52	**85.79 ± 9.42**	84.49 ± 7.69	82.11 ± 9.04	76.96 ± 7.32	85.63 ± 8.37
AGR_G	79.68 ± 9.27	85.66 ± 8.97	84.22 ± 9.14	**86.55 ± 7.89**	74.64 ± 7.43	79.46 ± 12.33
HYPER	82.37 ± 1.26	84.83 ± 1.96	82.04 ± 1.86	82.74 ± 2.41	82.01 ± 2.42	**87.09 ± 1.14**
Electricity	84.99 ± 2.42	87.06 ± 2.36	87.90 ± 1.68	**87.92 ± 1.82**	87.69 ± 1.42	85.69 ± 3.12
SPAM	94.14 ± 2.04	**94.86 ± 2.10**	93.21 ± 1.98	93.65 ± 1.95	91.57 ± 3.33	81.80 ± 9.99
Média	85.54	**87.9**	86.67	86.3	85.08	85.33

Fonte: O Autor.

Porém, no dataset real *Electricity* e no AGR_G os classificadores que utilizavam métodos de detecção de *concept drift* e adaptação ativa conseguiram desempenho superior ao FS+OSM. Estes dois datasets tem como características a importância de praticamente todas suas *features* na definição da classe, além da alta volatilidade de conceitos, o que pode ter influenciado nesses resultados.

De maneira geral, todas modificações obtiveram resultados melhores do que o algoritmo de controle em questão de eficiência na tarefa de classificação, quando analisadas as medidas de acurácia *prequential* e *kappa-m* referentes ao fluxo de dado completo.

Tabela 5.1b: Kappa-m dos algoritmos

KAPPA-M						
Dataset	OSM	FS.OSM	DD.OSM	DD.FS.OSM	ARF	OzaBag
$SEAFD_A$	69.11	**74.49**	68.87	64.9	70.95	73.82
$SEAFD_G$	69.4	**73.72**	70.08	64.19	70.33	73.26
SEA_A	71.41	75.56	72.89	73.11	**76.1**	73.43
SEA_G	71.83	75.05	72.78	73.1	**75.47**	73.23
Covertype	77.11	**79.25**	70.3	69.58	77.45	65.33
AGR_A	54.21	**57.92**	38.9	34.77	37.58	57.9
AGR_G	34.76	54.08	49.28	**56.52**	18.92	35.33
HYPER	64.22	69.21	63.55	64.97	63.47	**73.79**
Electricity	64.16	69.03	71.06	**71.13**	70.59	65.72
SPAM	77.52	**80.31**	73.76	75.47	68.72	39.3
Média	65.37	**70.86**	65.15	64.77	62.96	63.11

Fonte: O Autor.

Quando comparadas as estatísticas referentes ao uso de recursos computacionais, presentes nas tabelas 5.1c e 5.1d, é possível perceber que as modificações que utilizaram o método de adaptação ativa foram mais custosas do que as que usaram adaptação passiva, em todos os *datasets* testados. Portanto, o impacto causado pelo mecanismo de seleção de *features* em tempo real utilizado, no caso o FCBF, e a maior taxa de substituição de membros do comitê devido à adaptação passiva, não demonstrou ser maior do que o custo de analisar estatísticas do desempenho do classificador, em tempo real, utilizando detectores de drift.

Tabela 5.1c: Tempo de execução dos algoritmos em segundos

TEMPO DE EXECUÇÃO						
Dataset	OSM	FS.OSM	DD.OSM	DD.FS.OSM	ARF	OzaBag
$SEAFD_A$	36.95	**26.3**	44.66	29.98	88.78	49.5
$SEAFD_G$	38.27	**26.17**	49.44	33.97	91.81	63.89
SEA_A	**18.12**	21.88	22.89	23.83	63.7	24.97
SEA_G	**17.33**	22.34	23.06	23.33	64.56	26.16
Covertype	483.3	**276.83**	542.88	517.67	949.98	788.8
AGR_A	42.39	**26.12**	43.3	37.66	151.92	38.02
AGR_G	44.09	**30.34**	46.42	36.75	155.81	40.27
HYPER	**37.7**	57.23	43.17	55.92	97.95	64.28
Electricity	16.59	**13.59**	21.58	15.02	43.7	17.98
SPAM	5926.17	**4241.91**	7048.75	6023.08	12507.42	5848.17
Média	666.09	**474.27**	788.62	679.72	1421.56	696.2

Fonte: O Autor.

Tabela 5.1d: RAM-hora dos algoritmos em GB/h

RAM-HORA						
Dataset	OSM	FS.OSM	DD.OSM	DD.FS.OSM	ARF	OzaBag
$SEAFD_A$	0.96	**0.68**	1.16	0.78	2.3	1.28
$SEAFD_G$	0.99	**0.68**	1.28	0.88	2.38	1.65
SEA_A	**0.47**	0.57	0.59	0.62	1.65	0.65
SEA_G	**0.45**	0.58	0.6	0.6	1.67	0.68
Covertype	12.5	**7.16**	14.21	13.39	24.58	20.41
AGR_A	1.1	**0.68**	1.12	0.97	3.93	0.98
AGR_G	1.14	**0.78**	1.2	0.95	4.03	1.04
HYPER	**0.98**	1.48	1.12	1.45	2.53	1.66
Electricity	0.43	**0.35**	0.56	0.39	1.13	0.47
SPAM	1.53	**1.1**	1.82	1.56	3.23	1.51
Média	2.06	**1.41**	2.37	2.16	4.74	3.03

Fonte: O Autor.

No entanto, é relevante ressaltar o critério utilizado na criação da primeira geração de classificadores do comitê, onde todos são gerados usando RSM e metade das dimensões do espaço de *features* completo, inclusive nas modificações com mecanismo de *feature selection*. Essa escolha pode influenciar no custo computacional quando o número de *features* relevantes é menor do que o número de *features* utilizadas no RSM inicial, e quando dada essa situação, os classificadores iniciais não são escolhidos para serem substituidos pelo critério de

substituição (aquele com a menor acurácia). Sendo assim, existe a possibilidade das modificações que utilizam *feature selection* ainda possuírem classificadores gerados através de RSM, e que dependendo da situação, impactem negativamente no seu desempenho.

O classificador controle (OSM), apresentou menor custo nos datasets SEA e no HYPER. Esse desempenho pode ser explicado pela possível utilização de um conjunto perto do completo de *features* na criação dos classificadores com *feature* selection, o que não é forçado pelos que utilizam RSM.

Além disso, é notável o custo elevado referente ao uso do classificador *ensemble* ARF, que utiliza mecanismos de detecção de *concept drift* individuais para cada classificador do comitê, além de realizar o treinamento de classificadores substitutos simultaneamente ao treinamento dos classificadores oficiais. Em contrapartida, a abordagem de adaptação ativa adotada se mostrou pouco eficiente quando comparada ao mecanismo de adaptação cega utilizado como controle, tanto no ARF, como nas modificações do OSM.

5.2 Resultados Individuais

A análise da acurácia *prequential* em relação ao número de instâncias analisadas, em diferentes *datasets* individualmente, possibilita a análise das oscilações de desempenho causadas pelas alterações de conceito, e uma melhor visualização do tempo de recuperação à essas anomalias. Nos gráficos apresentados nas sub-seções a seguir, a linha tracejada vertical marca o local aproximado da ocorrência da alteração de conceito, quando conhecido.

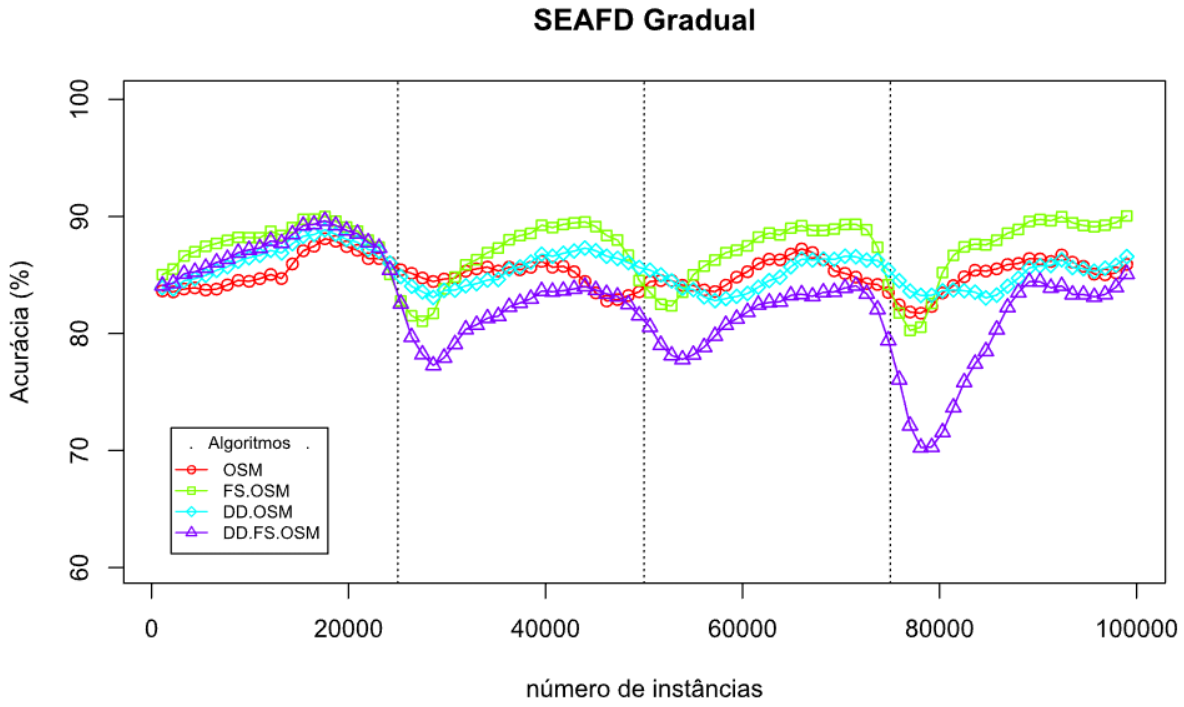
5.2.1 Resultados no SEAFD *dataset*

Os *datasets* SEAFD apresentam o ambiente propício para a utilização de algoritmos de seleção de *features*, por possuir muitas *features* irrelevantes e poucas relevantes. O algoritmo FS+OSM demonstrou bons resultados em ambas as variações do *dataset*, com mudança gradual e abrupta, no entanto, o algoritmo utilizando *feature selection* e adaptação ativa (DD+FS+OSM) não apresentou os mesmo resultados.

O método de detecção de *drifts* utilizado (EDDM) depende da distância entre erros, e é possivel que pare de atualizar o comitê com novos classificadores utilizando sub-espaços de *features* que melhor representam o conceito atual, caso venha a se “acomodar” em ter um

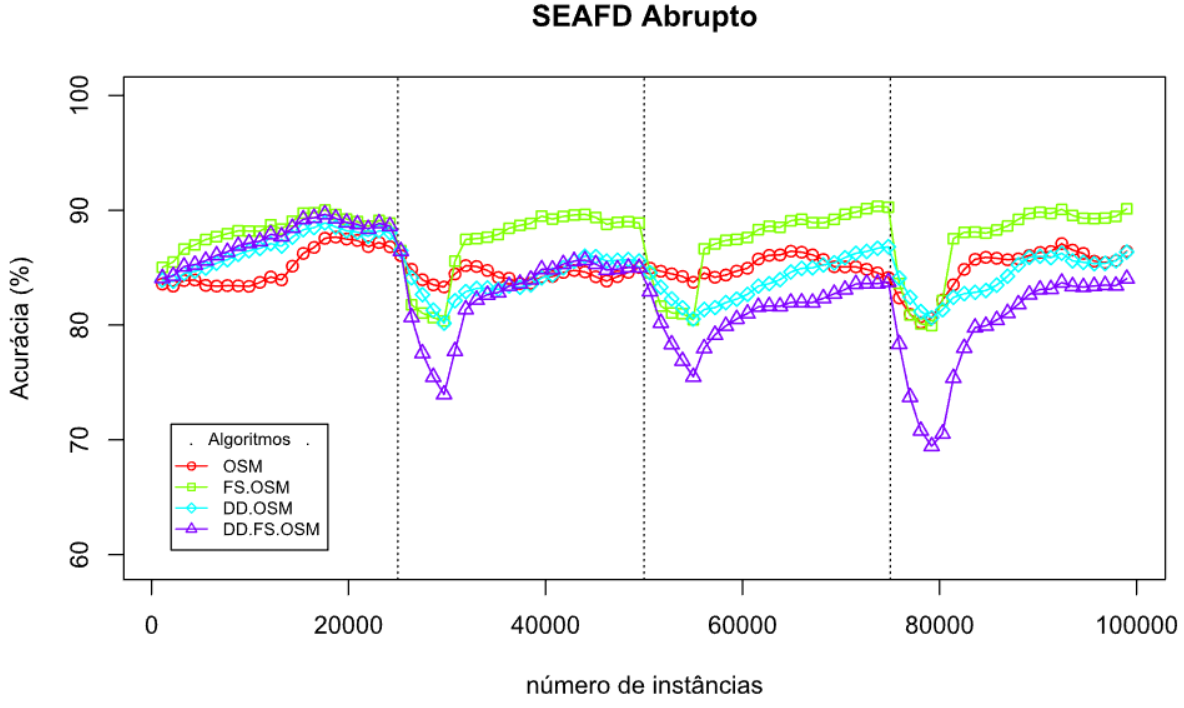
desempenho mediocre, assim considerando uma distancia entre erros baixa como normal. Além disso, caso não ocorram detecções de *concept drift* suficientes, é possível o classificador não substituir todos os classificadores criados sobre um espaço de *features* defasado, que irão impactar negativamente no voto final.

Figura 5.2.1a: SEAFD Gradual – Acurácia prequential ao longo do tempo



Fonte: O Autor.

Figura 5.2.1b: SEAFD Abrupto – Acurácia prequential ao longo do tempo



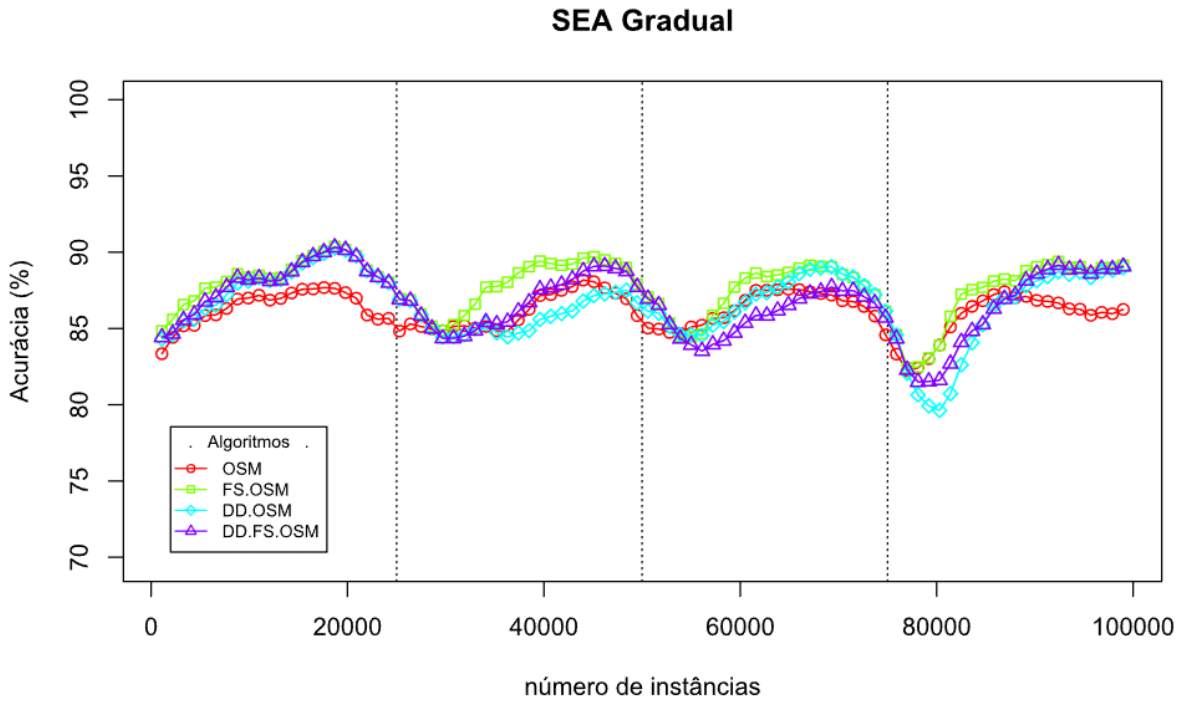
Fonte: O Autor.

É possível perceber também o impacto causado pela seleção de *features*, na ocorrência de um *feature drift*. Nas figuras 5.2.1a e 5.2.1b, os algoritmos equipados com mecanismos de *feature selection* foram os mais impactados logo após a mudança de contexto. Por outro lado, a recuperação de acurácia ocorre à uma velocidade mais rapida nesses mesmos algoritmos, quando comparados aos que estão utilizando somente RSM.

5.2.2 Resultados no SEA *dataset*

Os *datasets* SEA são compostos por instâncias com um número muito baixo de atributos (três), sendo o número de *features relevantes* coincidentemente igual ao número de *features* selecionadas através do critério de criação utilizando RSM (duas). O que contribui para o cenário ilustrado nas figuras 5.2.2a e 5.2.2b, que apresentam desempenho bastante similar entre os algoritmos, principalmente no cenário gradual.

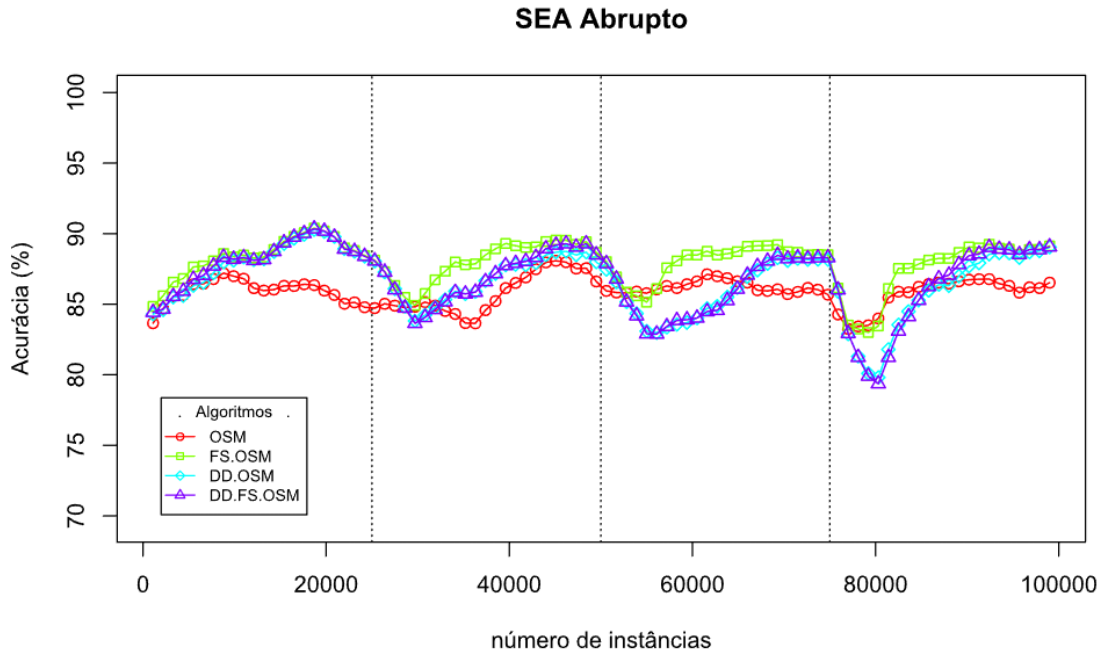
Figura 5.2.2a: SEA Gradual – Acurácia prequential ao longo do tempo



Fonte: O Autor.

Além disso, é possível novamente analisar a rápida recuperação do FS+OSM após a ocorrência da mudança de conceito. Por outro lado, quando na mesma situação, classificadores que utilizaram a proposta de adaptação ativa demonstraram perda de acurácia. Desta vez menos intensa do que no dataset anterior, porém ainda presente.

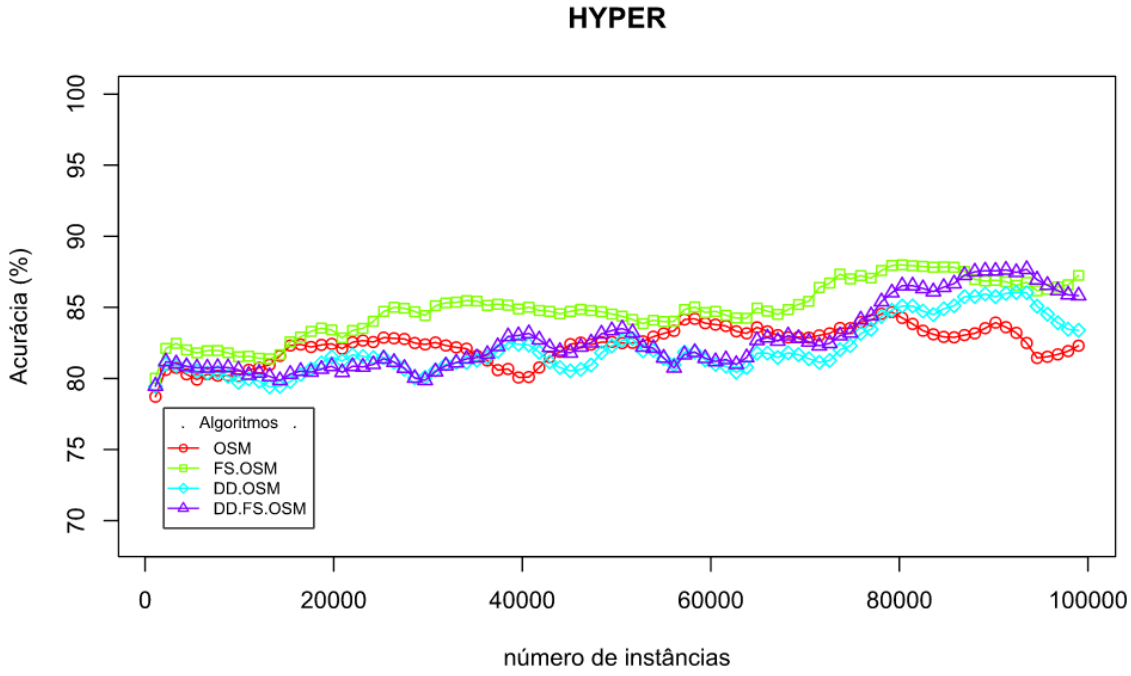
Figura 5.2.2b: SEA Abrupto – Acurácia prequential ao longo do tempo



Fonte: O Autor.

5.2.3 Resultados no HYPER *dataset*

Figura 5.2.3: HYPER – Acurácia prequential ao longo do tempo



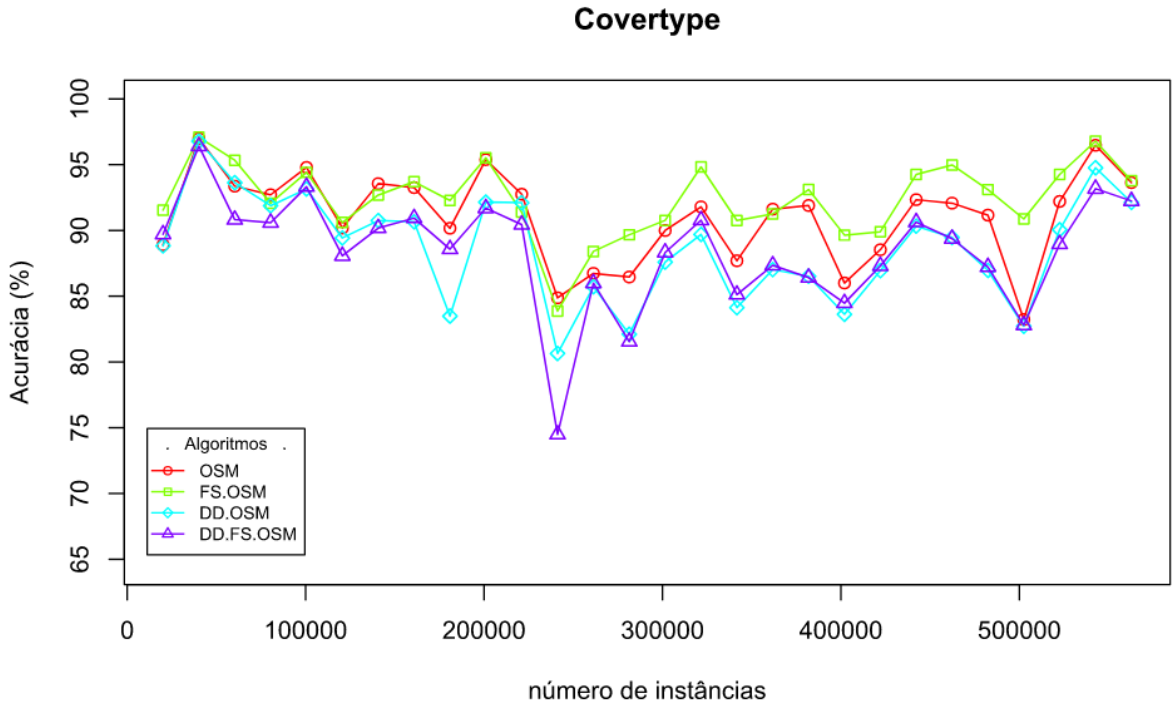
Fonte: O Autor.

O *dataset* HYPER contém uma alteração de contexto gradual, que dura do início ao fim do fluxo de dados. Nesse cenário, os classificadores apresentaram resultados similares, exceto o

FS+OSM, que apresentou resultados superiores durante todo o *dataset*, como visto na figura 5.2.3.

5.2.4 Resultados no *Covertype dataset*

Figura 5.2.4: Covertype – Acurácia prequential ao longo do tempo



Fonte: O Autor.

Os resultados obtidos no conjunto de dados *Covertype* foram similar aos apresentados anteriormente, onde os algoritmos que utilizaram adaptação passiva obtiveram, no geral, resultados melhores do que os equipados com detectores de *drift*.

5.2.5 Resultados no *Electricity dataset*

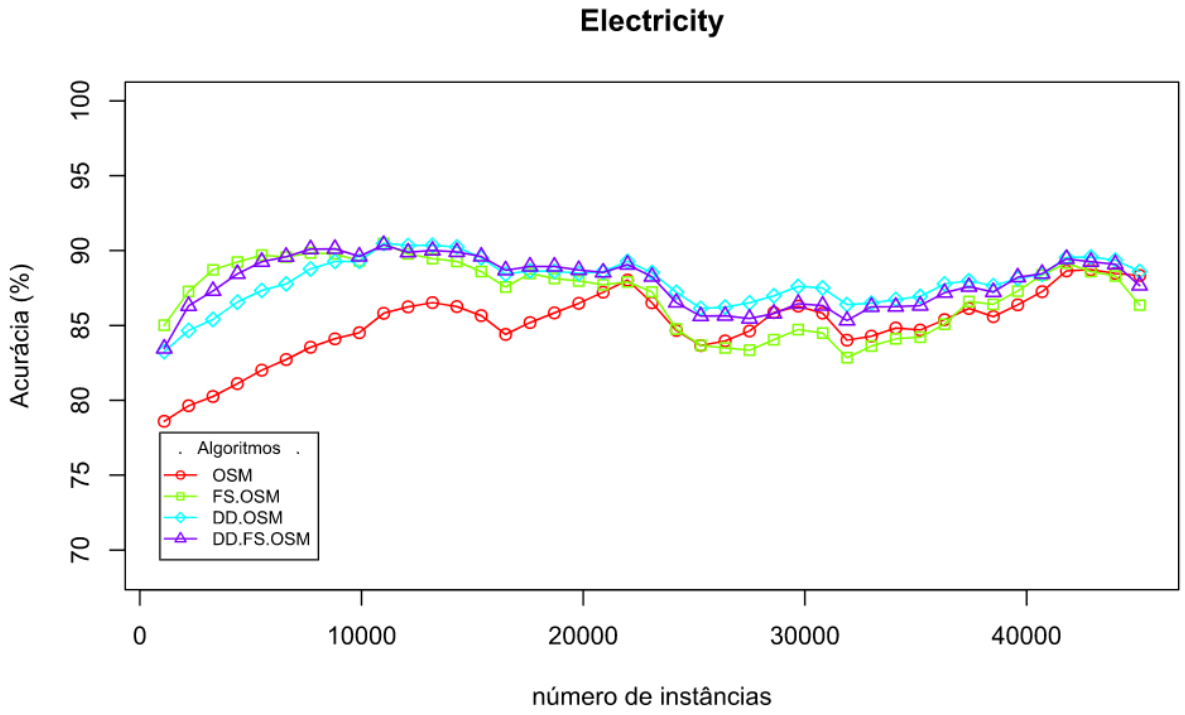
Os resultados referentes ao *dataset Electricity* são bastante curiosos, pois fogem um pouco do que estava sendo apresentado nos resultados anteriores. Nesse *dataset*, os algoritmos que se saíram melhor foram os que utilizaram adaptação ativa. Ademais, esse fenômeno já foi estudado em outros trabalhos como em Zliobate (2013) e em Bifet et al. (2013).

No primeiro trabalho, Zliobate demonstra a existência de alta dependência temporal nesse conjunto de dados, ou seja, a existência de pequenas sequências para qual a classificação

da instância é a mesma, já no segundo, Bifet et al. chega a resultados semelhantes aos encontrados aqui, onde o classificador equipado com métodos de adaptação ativa obtém o melhor resultado.

De acordo com a metodologia utilizada na construção dos algoritmos aqui comparados e que utilizam de adaptação ativa, acredita-se que esse cenário seja especialmente propício para esse tipo de abordagem, haja vista que na ocorrência de uma alteração de contexto seguida de resete de todos os classificadores, seria trivial o reaprendizado em um contexto de única classe.

Figura 5.2.5: Electricity – Acurácia prequential ao longo do tempo

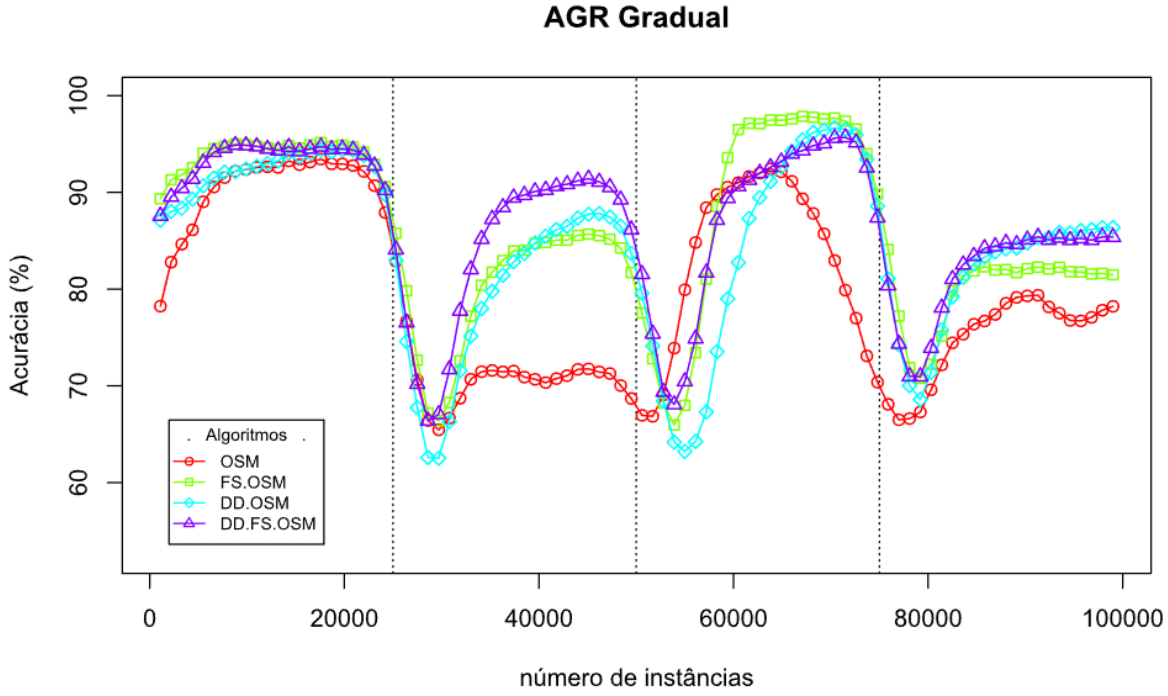


Fonte: O Autor.

5.2.6 Resultados no AGR *Dataset*

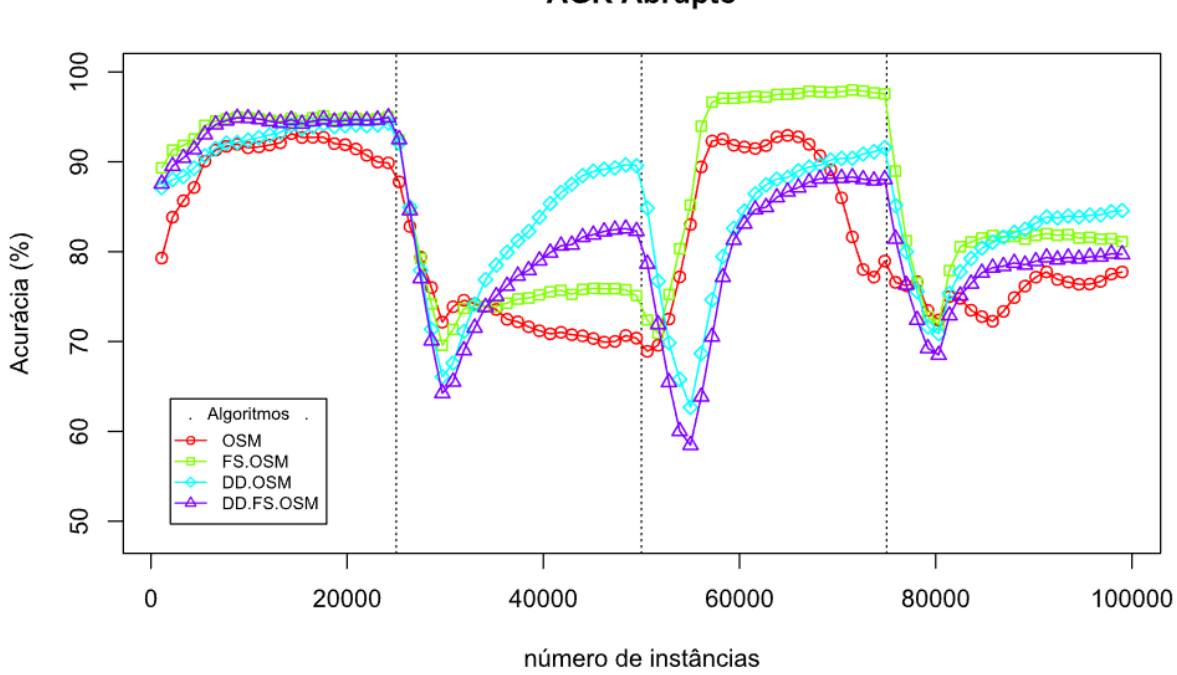
Os resultados referentes aos *datasets* criados através do gerador AGR também são bastante intrigantes. O método de geração deste conjunto de dados segue um procedimento complexo, que utiliza de 10 funções geradoras para mapear uma instância entre duas opções de classe. Como consequência disto, os fluxos de dados resultantes podem possuir padrões inusitados, como no caso das figuras 5.2.6a e 5.2.6b.

Figura 5.2.6a: AGR Gradual – Acurácia prequential ao longo do tempo



Fonte: O Autor.

Figura 5.2.6b: AGR Abrupto – Acurácia prequential ao longo do tempo



Fonte: O Autor.

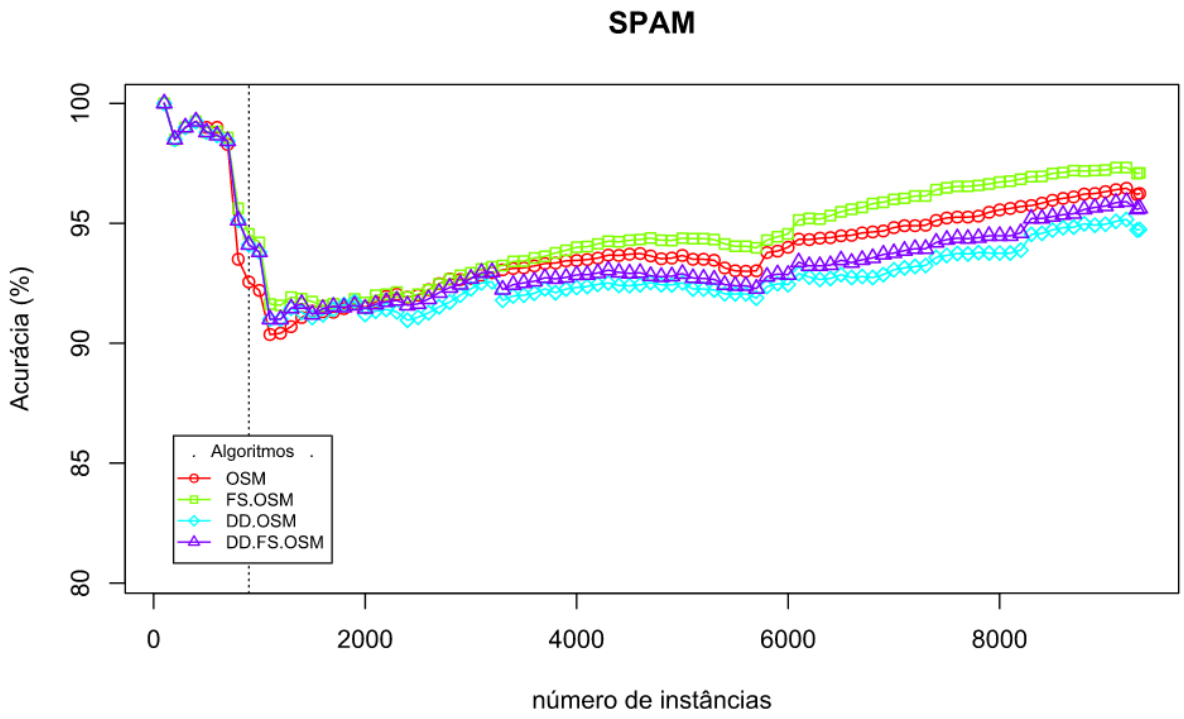
Nestas duas situações, de *concept drift* abrupto e gradual, ocorreu a troca entre conceitos bastante distintos, onde os algoritmos obtiveram resultados diferentes para estes diferentes conceitos. No caso, durante todo o fluxo de dados aqueles equipados com adaptação ativa

conseguiram se recuperar relativamente rapido, quando comparado com aqueles usando adaptação cega, principalmente no segundo e quarto conceitos.

5.2.7 Resultados no *Spam Corpus dataset*

Como o tamanho da janela utilizada no cálculo da acurácia foi de 5 mil instâncias, o aumento de desempenho é refletido em uma subida lenta nesse conjunto de dados, que conta com apenas 9 mil instâncias. Porém, ainda assim é possível ver a recuperação crescente dos algoritmos com e sem *feature selection*, após a ocorrência do *feature drift* em aproximadamente 1000 instâncias, sendo que o impacto da seleção de atributos nesse conjunto de dados não foi refletido em um aumento drástico de acurácia (embora o FS+OSM tenha obtido o melhor resultado)

Figura 5.2.7: Spam Corpus – Acurácia prequential ao longo do tempo



Fonte: O Autor.

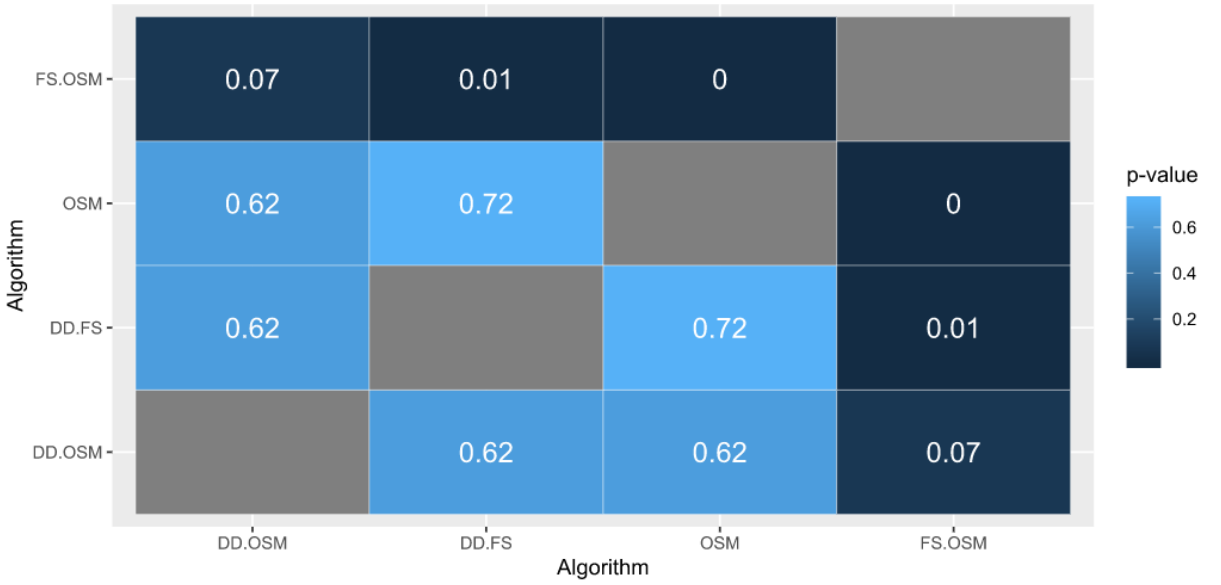
5.3 Análise estatística

Como mencionado na abordagem proposta, foram realizados dois testes de Friedman a fim de medir a diferença estatística entre as modificações propostas. Foi utilizado o

procedimento *post-hoc* Bergmann e Hommel para correção dos *p*-valores, e $\alpha = 0.1$, tomando como base a hipótese nula de que dois algoritmos são iguais.

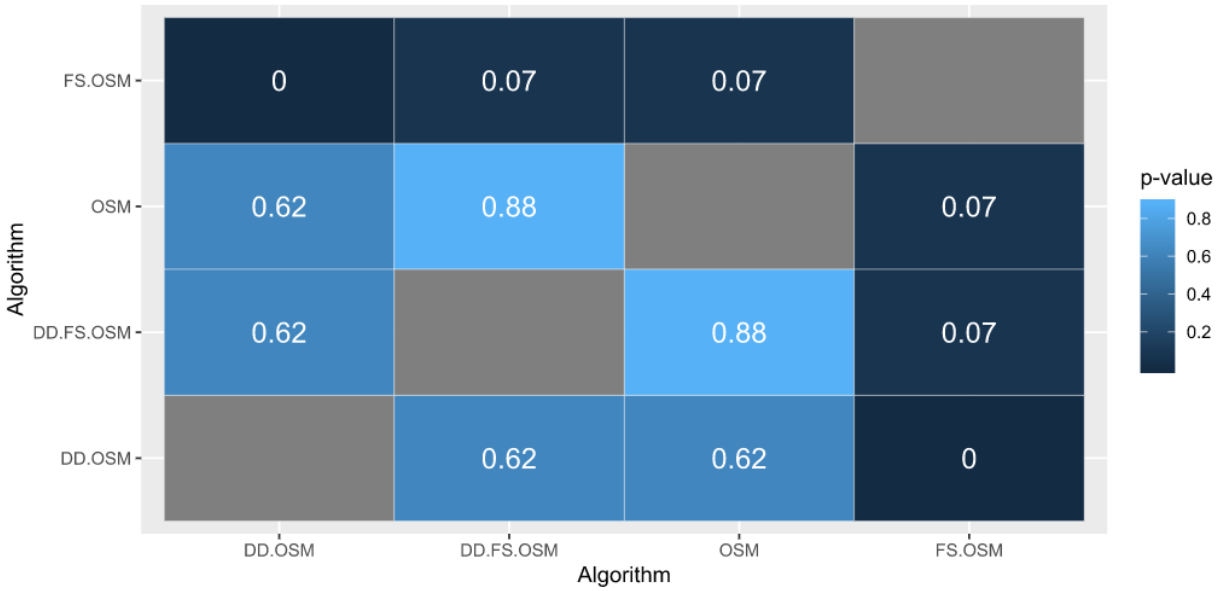
Sendo assim, o primeiro teste foi realizado sobre os valores obtidos de acurácia *prequential* de cada algoritmo, sobre o conjunto de *datasets*, o segundo foi sobre o tempo de execução. A figura 5.3a ilustra os resultados do primeiro teste, indicando os *p*-valores correspondentes a cada par de algoritmos, já a figura 5.3b indica os resultados do segundo teste.

Figura 5.3a: Teste de Friedman utilizando Bergmann-Hommel sobre a acurácia



Fonte: O Autor.

Figura 5.3b: Teste de Friedman utilizando Bergmann-Hommel sobre o tempo de execução



Fonte: O Autor.

De acordo com os resultados dos testes, é rejeitada a hipótese nula de que o algoritmo FS+OSM é estatísticamente equivalente aos outros com grau de confiança de $\alpha = 0.1$, tanto em questão de custo computacional como em relação ao desempenho na tarefa de classificação. Por outro lado, os testes sobre as modificações que utilizaram métodos de detecção de *concept drift* são inconclusivos, pois a hipótese nula não pode ser rejeitada.

6 CONCLUSÃO

O cenário de fluxos de dados não estácionários apresenta muitos desafios devido a sua natureza *online* e em evolução, e por ser um ambiente relativamente novo de pesquisa. Enquanto novos problemas de classificação que não se encaixam no formato do aprendizado de máquina convencional surgem a cada dia, é necessária a criação de novas técnicas ou a adaptação de técnicas já existentes para lidar com eles.

Muitos trabalhos foram propostos, com ideias para trazer soluções de *batch learning* para o cenário *online*, como o ARF e o *Online Bagging.* Porém, um dos principais problemas que torna esse ambiente desafiador são as alterações de conceito durante o aprendizado.

Foram vistas as diferentes maneiras em que essa anomalia pode acontecer, suas consequências, além de como algumas das soluções atuais lidam essa situação. Portanto, neste trabalho foram propostos 4 algoritmos utilizando diferentes mecanismos baseados no estado da arte, com a finalidade de comparar e analisar o impacto de cada ferramenta usada para adaptação ao novo conceito.

Deste modo, foram realizados testes no *framework* de simulação de fluxos de dados MOA, utilizando diferentes conjuntos de dados para representar boa parte dos cenários possíveis, e coletar dados sobre a eficiência dos classificadores na tarefa de classificação e uso de recursos computacionais. A análise desses dados indicou que a adição de um seletor de atributos relevantes, além de diminuir o custo computacional do algoritmo também resultou em um aumento de acurácia, suportado pela estatística *kappa-m.*

Por outro lado, os algoritmos que utilizaram EDDM para detectar e se adaptar a mudanças de conceito, não conseguiram resultados positivos quando comparados ao classificador controle. Esses resultados foram confirmados pelo teste de Friedman com Bergmann-Hommel com grau de confiança $\alpha = 0.1$, onde apenas o classificador FS+OSM, equipado com um seletor de *features* FCBF e adaptação passiva por janelas *landmark*, conseguiu rejeitar a hipótese nula e demonstrar diferença estatística aos outros algoritmos.

Embora não tenha sido confirmada a diferença dos algoritmos equipados com EDDM em relação aos outros, foi feita a análise detalhada da acurácia *prequential* em relação ao número de instâncias, onde foi possível analisar diferentes padrões de recuperação dos algoritmos, após uma mudança de conceito.

Sugere-se como trabalhos futuros, o uso de alguma abordagem alternativa para a análise do impacto dos detectores de *concept drift*, no desempenho dos classificadores. Neste trabalho optou-se por uma abordagem mais agressiva, onde a cada detecção, existe o reinicio do

aprendizado do comitê como um todo. Talvez o uso de uma abordagem mais individual, como a utilizada no ARF, onde existe um detector para cada classificador do *ensemble*, consiga responder de maneira mais eficiente a eventuais mudanças de conceito.

REFERÊNCIAS

ADE, R. R.; DESHMUKH, P. R. Methods for Incremental Learning : A Survey. **International Journal of Data Mining & Knowledge Management Process**, 2013. v. 3, p 119-125.

AGRAWAL, R. et al. An interval classifier for database mining applications. **In Proceedings of the International Conference on Very Large Data Bases**, 1992.

BAENA-GARCÍA, M. et al. **Early Drift Detection Method**, 2006.

BARDDAL, J. et al. A Survey on Feature Drift Adaptation: Definition, Benchmark, Challenges and Future Directions. **Journal of Systems and Software**, 2016. v.127, p. 278-294.

BARDDAL, J. et al. Analyzing the Impact of Feature Drifts in Streaming Learning. **Neural Information Processing. ICONIP 2015**, 2015. v. 1, p. 21-28.

BIFET, A. et al. Efficient Online Evaluation of Big Data Stream Classifiers. **Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '15**, 2015. v.1, p. 59–68.

BIFET, A. et al. Lecture Notes in Computer Science. **LNCS**, 2013. v.8188. p. 465-479.

BIFET, A.; GAVALDÀ, R. Adaptive Learning from Evolving Data Streams. **Advances in Intelligent Data Analysis VIII**, 2009. v.1, p.249-260.

BIFET, A.; HOLMES, G.; PFAHRINGER, B. Leveraging bagging for evolving data streams. **Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, 2010. v. 6321, p. 135–150.

CARBORANA, L; BORROWMAN, A. A comparison of batch and incremental supervised learning algorithms. **Principles of Data Mining and Knowledge Discovery**, 1998. v. 1510, p. 264-272.

COHEN, J. A Coefficient of Agreement for Nominal Scales. **Educational and Psychological Measurement**, 1960.

DEMSAR, J. Statistical comparisons of classifiers over multiple data sets. **Journal of Machine Learning Research**, 2006. v. 7, p. 1–30.

DOMINGOS, P.; HULTEN, G. Mining high-speed data streams. **Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '00)**, 2000. v. 1, p. 71-80.

DOMINGOS, P. Why does bagging work? a Bayesian account and its implications. **In Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD'97)**, 1997. p. 155-158.

GAMA, J.; RODRIGUES, P.; AGUILAR-RUIZ, J. An Overview on Learning from Data Streams. **New Generation Computing**, 2006. v. 25, p. 1-4.

GAMA, J. **Knowledge Discovery from Data Streams.** 1st ed. [S.l.]: Chapman & Hall/CRC, 2010.

GAMA, J. et al. Learning with Drift Detection. **Intelligent Data Analysis**, 2004. v. 8, p. 286-295.

GAMA, J et al. On evaluating stream learning algorithms. **Machine Learning**, 2012. v. 90, p. 317-346.

GOMES, H. et al. **Adaptive random forests for evolving data stream classification. Machine Learning**, 2017a.

GOMES, H. et al. A Survey on Ensemble Learning for Data Stream Classification. **ACM Computing Surveys**, 2017b. v. 50,p. 1–36.

GONÇALVES-JR, P.M. et al. A comparative study on concept drift detectors. **Expert Systems with Applications**, 2014. v. 41.

HARRIES, M. SPLICE-2 Comparative Evaluation: Electricity Pricing. **Technical report, The University of South Wales**, 1999.

HO, T. K. The random subspace method for constructing decision forests. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, 1998. v. 20, p. 832–844.

HULTEN, G.; SPENCER, L.; DOMINGOS, P. Mining time-changing data streams. **Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining**, 2001. p. 97–106.

KHAMASSI, I. et al. Discussion and review on evolving data streams and concept drift adapting. **Evolving Systems**, 2016. v. 9.

KROGH, A,; VEDELSBY, J. Neural Network Ensembles, Cross Validation, and Active Learning. **Advances in neural information processing systems**, 1995. v. 7.

MARSLAND, S. **Machine Learning: An Algorithmic Perspective**. 2nd ed. Boca Raton, FL: CRC press, 2015.

MOHRI, M.; ROSTAMIZADEH, A.; TALWALKAR, A. **Foundations of Machine Learning**. 1st ed. Cambridge, MA: The MIT Press. 2012.

MOLINA, L. C. et al. Feature Selection Algorithms : A Survey and Experimental Evaluation. **Proceedings of the 2002 IEEE International Conference on Data Mining**, 2002. v. 1, p. 306–313.

NGUYEN, H. et al. **Heterogeneous Ensemble for Feature Drifts in Data Streams**, 2012.

OZA, N.; RUSSELL, S. Online Bagging and Boosting. **Proceedings of Artificial Intelligence and Statistics**, 2001.

PRESS, W. H. et al. **Numerical recipes in C**. 1st ed. Cambridge: Cambridge University Press. 1988.

QUINLAN, J. R. Induction of Decision Trees. **Machine Learning,** 1986. v.1, p 81-106.

STREET, W. N.; KIM, Y. **A streaming ensemble algorithm (SEA) for large-scale classification**, 2004.

WOLPERT, D. H., MACREADY, W. G. No free lunch theorems for optimization. **IEEE Transactions on Evolutionary Computation**, 1997.

YU, L.; LIU, H. Feature selection for high-dimensional data: a fast correlation-based filter solution. **Proceedings of the twentieth international conference on machine learning**, 2003. v. 2, p. 856-863.

ZLIOBAITE, I. **How good is the Electricity benchmark for evaluating concept drift Adaptation**, 2013.

ZHOU, Z. H. Ensemble Learning. **Encyclopedia of Biometrics,** 2009. v. 1, p 270-273.