Múltiplos MLPs com Detecção Quantitativa de Concept Drift

1st Ramon Gomes Durães de Oliveira
Aluno do Programa de Pós Graduação em Engenharia Elétrica
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
ramongduraes@gmail.com

***Resumo*—Ao treinar modelos de aprendizado de máquina e colocá-los em produção, percebe-se em diversos casos que a performance do modelo decai ao longo do tempo ou se mostra instável a não-confiável. Isso ocorre devido ao concept drift, que caracteriza uma mudança na distribuição geradora dos dados de entrada em relação aos dados de saída. Formas de lidar com o problema e gerar modelos que se adaptem ao longo do tempo são amplamente estudadas. Neste trabalho, é proposta uma forma de lidar com o problema que combina um índice quantitativo para medir o concept drift e o uso de múltiplos modelos, gerados de acordo com a a variação neste índice. Os resultados mostram que a performance do sistema desenvolvido é consistentemente superior à performance de um modelo único que não considera o drift.**

***Index Terms*—Model Pool, Concept Drift, Concept Drift Map, MLPs**

I. Introdução

Redes neurais artificiais são ferramentas poderosas de aproximação de funções. De fato, o teorema de aproximação universal [1] diz que uma rede neural feed-forward com uma única camada escondida contendo um número finito de neurônios é capaz de aproximar qualquer função contínua em subespaços compactos de $\mathbb{R}^n$.

Entretanto, ao treinar redes neurais e outros modelos de aprendizado de máquina em um conjunto de dados e colocá-los em produção, é possível que observe-se uma piora na performance dos modelos. Isso ocorre devido a diferenças intrínsecas ao processo que se deseja modelar. Para uma rede neural que classifica cachorros e gatos, por exemplo, não espera-se grandes mudanças ao longo do tempo. Entretanto, para uma rede que tenta prever o tempo em um determinado local tenta modelar um processo muito mais dinâmico. Espera-se para este problema, por exemplo, uma lenta variação devido ao aquecimento global. Espera-se também grandes variações eventuais causadas pela chegada de massas de ar com características diferentes.

Às mudanças na distribuição de um conjunto dados no contexto de aprendizado de máquina é dado o nome de **concept drift**. Essas mudanças podem ocorrer de diversas formas (gradual, abrupta, recorrente...), como nos exemplos citados acima e é comum que isso aconteça ao colocar em produção modelos de aprendizado de máquina pré-treinados.

O objetivo deste trabalho é desenvolver uma abordagem que lide com os diversos tipos de concept drift. Esta abordagem será dividida em 4 módulos que, combinados, irão lidar com cada tipo de concept drift: módulos de memória, detecção de mudanças, aprendizado e estimativa de erro. Neste trabalho serão utilizados apenas ensembles de redes neurais de múltiplas camadas. Utilizando essa abordagem, espera-se poder lidar com mudanças recorrentes e abruptas no conjunto de dados pois haverá modelos diferentes para estes casos. Além disso, será utilizado o treinamento online por mini-bateladas, que espera-se poder adaptar aos casos de mudanças lentas e graduais.

Na Seção II será feita uma revisão de literatura abordando o concept drift, MLPs (definição e treinamento) e ensembles de modelos. A Seção III aborda a metodologia desenvolvida no trabalho. Na Seção IV são apresentados os resultados obtidos em diversas bases de dados. A Seção V apresenta as conclusões do trabalho.

II. Revisão de Literatura

A. Concept Drift

Em ambiente dinâmicos, não estacionários, a distribuição dos dados pode mudar com o passar do tempo. Este fenômeno é conhecido como concept drift. Neste trabalho, modelos de aprendizado de máquina que se adaptam de forma a reagir a isso são chamados de modelos evolutivos. A Figura 1 sumariza os quatro módulos principais de um modelo evolutivo, que serão discutidos a seguir.

1) Memória: O aprendizado sob a iminência da ocorrência de concept drift faz com que seja necessário não só aprender com os novos dados mas também esquecer os dados antigos.

O gerenciamento de dados diz respeito à escolha de quais dados serão utilizados para treinar o modelo. Em geral, utiliza-se ou o modelo mais recente, ou uma janela (um conjunto) de amostras mais recentes. Quando a mudança na distribuição dos dados é abrupta, o tamanho da janela é um parâmetro importante: quanto maior a janela, mais lentamente o sistema reage às mudanças, tornando-se menos reativo a mudanças rápidas.

Quanto ao mecanismo de esquecimento dos dados antigos, ele pode ser ou **abrupto**, ou **gradual**. O esquecimento abrupto é recomendável principalmente em casos de mudanças abruptas na distribuição dos dados, nos quais os dados antigos deixam de ser válidos para a nova versão do problema.

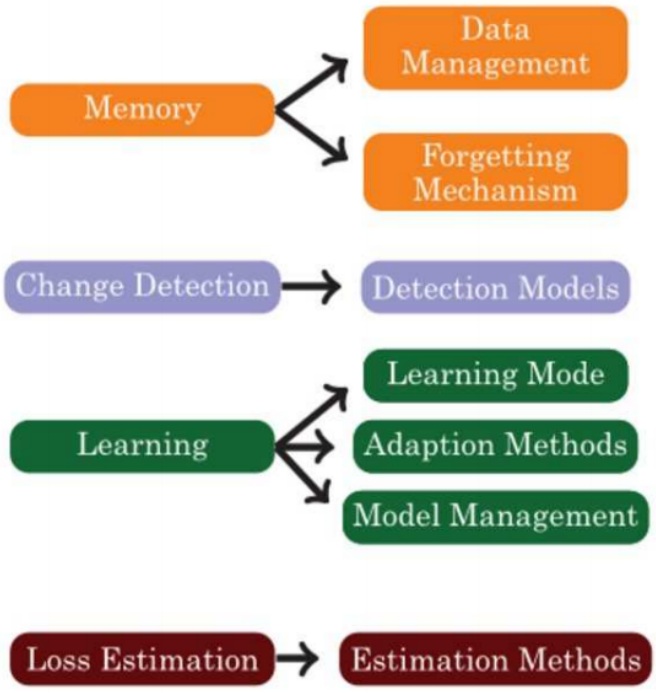

Figura 1. Esquema dos 4 módulos principais de um sistema de aprendizado de máquina evolutivo. Fonte: [2].

2) Detecção de Mudanças: A detecção de mudanças na distribuição dos dados de entrada é um importante passo para informar ao modelo quando ele deve utilizar estratégias de aprendizado que o adaptem às mudanças ocorridas. Quando a intenção do modelo é ser robusto a falhas de sensores, por exemplo, espera-se que ocorram mudanças abruptas no momento em que o sensor falha. Neste momento, é importante detectar essas mudanças e informar ao algoritmo de aprendizado para que ele tome precauções.

3) Aprendizado: Os modos de aprendizado de modelos evolutivos podem ser divididos em três: aprendizado online, no qual utiliza-se apenas a amostra mais recente para atualizar o modelo; incremental, no qual atualiza-se o modelo com base em sua performance em um conjunto de amostras; e aprendizado por retreinamento, que consiste em descartar o modelo atual e retreiná-lo com os novos dados.

Quanto à forma de adaptação ao concept drift, pode-se classificar os modelos em adaptação informada, que se adapta em função dos mecanismos de detecção de mudanças, e adaptação cega, que adapta o modelo continuamente, sem nenhuma forma explicita de detecção de mudanças. Os modos de aprendizado online e incremental são casos especiais de adaptação cega.

Na Figura 1, o gerenciamento de modelos (do inglês model management) refere-se aos métodos que utilizam um ensembles de modelos para a mesma variável de resposta. Ensembles são necessários para que sejam criadas estratégias mais complexas de adaptação ao concept drift.

4) Estimativa de Erro: Em modelos estáticos (não-evolutivos) a avaliação dos modelos em dados ainda não vistos pode ser feita por meio da separação de um conjunto de teste no qual é reportado o erro de generalização do modelo. Para modelos evolutivos, que obtêm os dados do processo de maneira online, isto não é possível.

A forma de estimativa de erro utilizada neste trabalho é o prequential error, ou erro preditivo sequencial. Nesta abordagem, a cada nova janela de amostras recebida, elas primeiro são utilizadas para teste e, em seguida, para treinamento do modelo. O erro preditivo sequencial consiste no acúmulo do erro de teste do modelo nessas novas amostras. Essa abordagem permite que todos os dados sejam utilizados para estimativa de erro.

B. Multi-Layer Perceptrons (MLPs)

A motivação inicial para a criação de perceptrons de múltiplas camadas era simular o cérebro humano e seu processo de aprendizado. Dessa forma, MLPs são compostas de múltiplas unidades, chamadas de neurônios. Esses neurônios são divididos em diferentes camadas: a camada de entrada, as camadas escondidas e a camada de saída.

Neste trabalho, iremos lidar apenas com MLPs com uma única camada escondida. Além disso, como trata-se de problemas de classificação binária, haverá sempre apenas um neurônio na camada de saída. A Figura 2 mostra um exemplo deste tipo de rede. As "features"X são colunas dos dados de entrada. As unidades denominadas "bias"são unidade de viés adicionadas à função que cada neurônio calcula. E a "output"$f(X)$ nesse caso é o vetor de classes $Y \in \{0, 1\}$.

Cada neurônio da camada escondida, mostrado na figura como $a_{1...k}$, calcula uma função linear dos neurônios de entrada $x_{0...n}$ (em que o neurônio x_0 representa o termo de viés $+1$), de acordo com (1):

$$a_k = \sum_{n=0}^{n} x_n w_{n,k} \tag{1}$$

Em que $w_{n,k}$ se refere ao peso da conexão entre o neurônio de entrada x_n e o da camada escondida a_k. Após o cálculo da função linear das entradas, é aplicada uma função não linear de forma que a saída real do neurônio a_1, por exemplo, seria $u_1 = tanh(a_1)$. Assim, são gerados novos valores que são propagados pela rede.

No caso da rede da Figura 2, a propagação acontece apenas até o neurônio de saída já que há apenas uma camada intermediária. No caso do neurônio de saída, como deseja-se obter um valor binário que represente a classe de cada observação, a função de ativação utilizada é a função degrau. Dessa forma, aplicando a função degrau sobre a função linear calculada, as saídas da rede serão sempre binárias.

C. Abordagens de Treinamento

Uma vez definida a forma dos parâmetros, é necessário que eles sejam estimados por meio da minimização da função de uma custo. Uma das formas fazê-lo é utilizando o algoritmo do **gradiente descendente** [3]. Este funciona por meio do cálculo

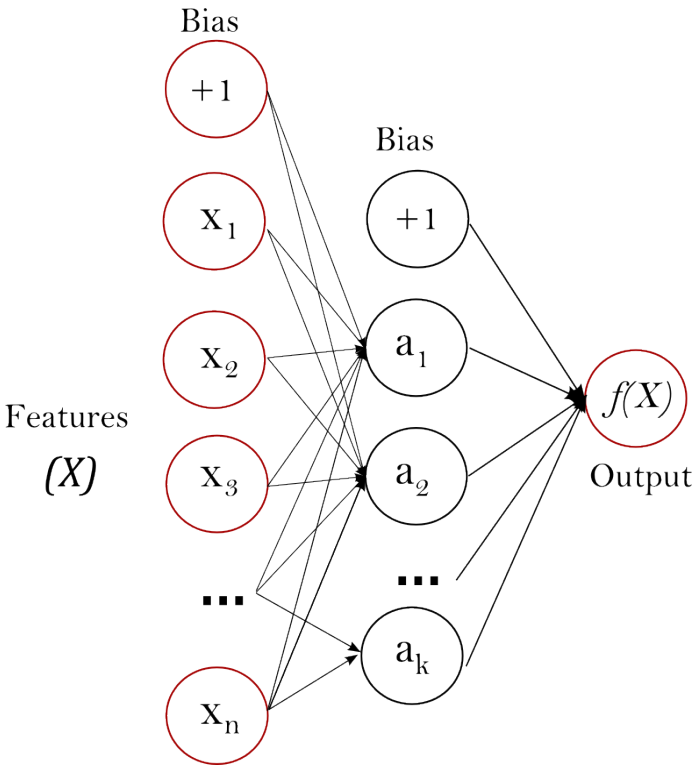

Figura 2. Exemplo de um Perceptron de Múltiplas Camadas.

do vetor gradiente da função de custo, que indica a direção na qual ela cresce mais rapidamente. Em seguida, este atualiza os parâmetros do modelo na direção contrária à do vetor gradiente. Esse procedimento de atualização dos parâmetros pode ser definido como na Equação 2, em que $\overrightarrow{\beta}$ indica o vetor de parâmetros estimados e $(\overrightarrow{\beta})$ indica o vetor gradiente da função de custo $J(\overrightarrow{\beta})$. O j-ésimo elemento do vetor gradiente corresponde à derivada parcial da função de custo em relação ao parâmetro $\overrightarrow{\beta}_j$. Na Equação 2, λ representa a **taxa de aprendizado** do algoritmo, um hiper-parâmetro que define o tamanho do passo a ser tomado na direção de minimização da função. A escolha deste de parâmetro é crucial para a convergência da solução.

$$\overrightarrow{\beta}_{t+1} = \overrightarrow{\beta}_t - \lambda \Delta J(\overrightarrow{\beta}_t) \tag{2}$$

Há três variações principais para o algoritmo do gradiente descendente. A primeira delas é o **gradiente descendente em batelada**, na qual a atualização dos parâmetros é feita depois que todas as amostras a serem utilizadas no ajuste do modelo já foram levadas em consideração. Ou seja, os parâmetros são atualizados na direção média do vetor gradiente calculado para todas as amostras. Cada passada pelo conjunto de dados é chamado de época. Em geral, são necessárias várias épocas para que o método convirja e, no caso do gradiente descendente em batelada, para cada época todo o conjunto de dados é avaliado. O número de épocas é mais um hiper-parâmetro do processo de otimização.

A utilização deste algoritmo para conjuntos de dados muito grandes torna-se inviável devido ao tempo e custo computacional. Este também não pode ser utilizado para problemas nos quais apenas parte dos dados estão disponíveis durante o treinamento. O método do **gradiente descendente estocástico** resolve este problema ao atualizar as estimativas dos parâmetros do modelo após cada amostra, ao invés de esperar até que todos os exemplos sejam computados. Nessa abordagem, a estimativa do gradiente da função de custo é mais rápida, mas menos precisa por ser feita com apenas uma amostra. Isso faz com que o algoritmo nem sempre caminhe na direção correta, mas na média os passos o levam à direção do mínimo global.

A terceira variação do gradiente descendente é o **gradiente descendente em mini-batelada**, que pode ser visto como um método intermediário entre o baseado em batelada e o estocástico. Ou seja, para cada época, este algoritmo calcula o gradiente da função de custo utilizando b observações e o utiliza para atualizar os parâmetros estimados. A vantagem dessa abordagem é fazer uma estimativa da direção de minimização melhor que a do método estocástico, mantendo a propriedade da alta velocidade de convergência. Em particular, implementações vetorizadas deste método permitem a paralelização do cálculo do gradiente, acelerando o processo [4].

III. Metodologia

O sistema proposto se trata de um conjunto de modelos MLPs que visam lidar com os quatro principais tipos de concept drift, mostrados na Figura 3, da seguinte forma:

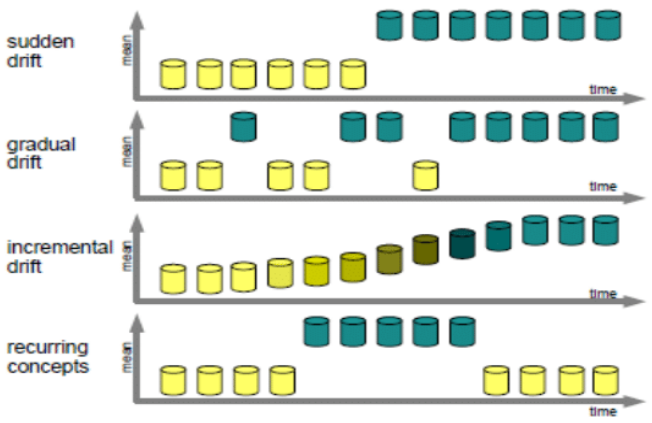

Figura 3. Os 4 principais tipos de concept drift.

- Mudanças súbitas: mudanças súbitas serão detectadas por meio de uma comparação da distribuição dos dados de entrada de novas bateladas com os das mais recentes. Caso detectadas, um novo modelo será treinado para lidar com elas.
- Mudanças graduais: a estratégia para lidar com mudanças graduais será também utilizar modelos diferentes. Neste caso, a distribuição de uma nova janela de dados será

comparada com as janelas de dados utilizadas para treinar todos os modelos disponíveis. Caso ela seja próxima o suficiente de algum padrão já visto, o modelo já treinado será utilizado. Senão, um novo modelo será treinado.
- Mudanças incrementais: para o caso de mudanças incrementais, o índice de concept drift calculado aumentará gradualmente com o tempo. Ao passar do threshold estabelecido um novo modelo será treinado para se adequar aos novos dados.
- Mudanças recorrentes: a estratégia neste caso é semelhante ao caso das mudanças graduais e haverá diferentes modelos disponíveis para cada mudança abrupta na distribuição de dados.

Para implementar uma estratégia como a descrita acima, define-se que, para os casos nos quais houver uma mudança abrupta, um novo modelo será treinado potencialmente em apenas uma batelada de dados. Portanto, os modelos serão todos MLPs simples, com poucos neurônios, logo poucos parâmetros a serem estimados. Além disso todos os modelos têm a mesma estrutura para simplificar e reduzir o número de parâmetros do sistema. Por fim, há uma duração máxima para os modelos, em número de bateladas. Isso faz com que não haja modelos inutilizados caso os padrões vistos por eles não apareçam mais

A. Algoritmo

1) Inicialmente, é treinado apenas um modelo na batelada de dados inicial.
2) Para cada nova batelada de dados, calcula-se o índice de concept drift em relação aos dados anteriores. Isto é feito calculando, para cada atributo e cada classe, o mínimo, máximo e média da batelada de dados. Em seguida, soma-se o valor absoluto da diferença entre as estatísticas da nova batelada e da anterior. Este índice foi desenvolvido tendo por base o concept drift map [5] [6], mas calcula o mínimo e máximo ao invés do desvio padrão (original) para não assumir que a distribuição dos dados de entrada é gaussiana.
3) Caso este índice de concept drift seja maior que um parâmetro max_drift, um novo modelo será treinado para esta batelada de dados.
4) Caso contrário, um modelo já treinado será utilizado. Ele é escolhido verificando qual o modelo de índice de concept drift mais próximo ao da nova batelada de dados.

O sistema desenvolvido tem alguns pontos negativos por design:

- Por serem treinados apenas em uma batelada de dados, os modelos utilizados são relativamente simples: todos perceptrons de múltiplas camadas com 5 neurônios na camada escondida. Entretanto modelos simples demais têm performance limitada.
- O módulo de identificação do concept drift lida apenas com atributos contínuos, potencialmente perde-se, portanto, muita informação ao não tratar atributos discretos.
- O tamanho da janela de dados influencia bastante na performance dos modelos e do sistema como um todo.

Os resultados são mostrados na próxima seção.

IV. Resultados

A. Bases de Dados

Neste trabalho foram utilizadas 3 bases de dados amplamente utilizadas pela comunidade:

1) Hyperplane: uma versão da famosa base de dados Hyperplane alterada para um problema de classificação binária, contendo 10.000 observações. Esta base de dados foi fornecida.
2) Electricity: um conjunto de dados coletados da Australian New South Wales Electricity Market. No mercado, preços não são fixos e são afetados pela oferta e demanda. As observações foram coletadas a cada 5 minutos, totalizando 45.312 instancias. Também trata-se de um problema de classificação binária.
3) SEA: uma base de dados proposta em [7] com 3 atributos numéricos entre 0 e 10. Existem 4 conceitos (concepts) com 15.000 observações cada, totalizando 60.000 observações. Neste caso, há 3 classes no problema.

B. Módulo de Identificação do Concept Drift

A Figura 4 mostra a variação do índice quantitativo de drift desenvolvido à medida em que novos dados são observados. No eixo X está o número de bateladas de dados já tratadas, cada uma com 150 observações.

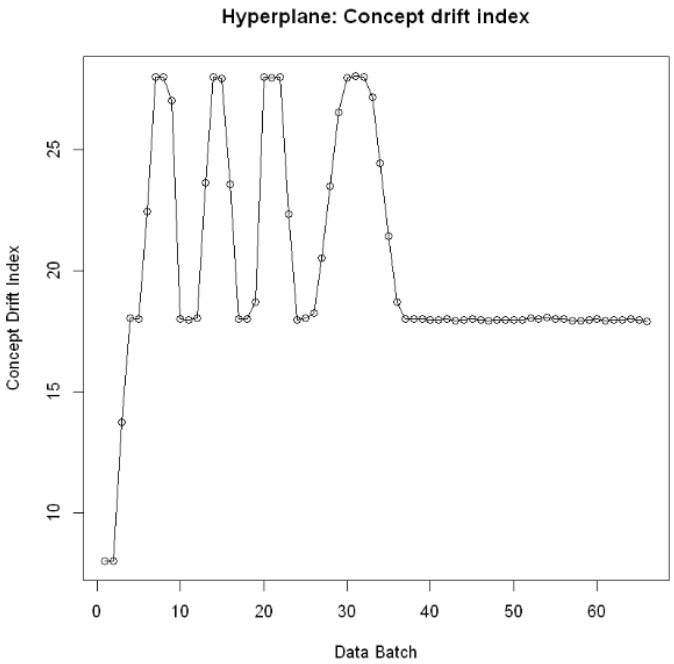

Figura 4. Índice quantitativo de drift por bateladas para a base de dados Hyperplane.

Percebe-se que o valor calculado de fato indica uma forte ocorrência de drift na base de dados: um drift recorrente que se estabiliza por volta da 40ª batelada.

Figuras semelhantes foram geradas para as outras duas bases de dados. Por se tratarem de bases de dados reais, a curva do índice de concept drift oscila mais mas ainda detecta com sucesso os padrões de ocorrência de drift.

C. Acurácia do Sistema

Os modelos foram treinados utilizando o pacote RSNNS do R. Para todos os modelos, o número de neurônios na camada escondida foi sempre 5. Isso faz com o que os modelos sejam simples o suficiente para serem treinados em apenas uma batelada de dados.

A Tabela I mostra os parâmetros e a performance obtida para cada conjunto de dados. Nela, n_models representa o número de modelos treinados e mantidos pelo sistema, $SMMA$ (single model mean accuracy) representa a acurácia média para o caso de apenas um modelo, desconsiderando o concept drift. $MMMA$ (multiple models mean accuracy) representa a acurácia média para o caso de múltiplos modelos, de acordo com o sistema desenvolvido.

Tabela I
PARÂMETROS, NÚMERO DE MODELOS E RESULTADOS PARA CADA BASE DE DADOS

	Hyperplane	Electricity	SEA
max_drift	2.0	0.3	0.02
batch_size	300	200	200
n_models	4	8	14
SMMA	0.4973	0.4234	0.8499
MMMA	**0.5517**	**0.6151**	**0.8558**

A Tabela I mostra que em todos os experimentos e conjuntos de dados a performance média do sistema desenvolvido foi superior ao caso em que um modelo é treinado previamente desconsiderando o concept drift. A Figura 5 mostra a performance do sistema de múltiplos em relação a um modelo estático ao longo do tempo com a chegada de novas bateladas de dados. Percebe-se que a acurácia do sistema desenvolvido é quase sempre superior à de um modelo único, o que é confirmado pela acurácia média superior mostrada na tabela I.

V. CONCLUSÃO

Os sistema desenvolvido se mostrou capaz de lidar melhor com problemas de concept drift do que modelos treinados estaticamente, resultado confirmado pela performance média do pool de modelos sempre acima da performance média de um modelo único.

Era também um objetivo deste trabalho que o sistema desenvolvido não introduzisse muitos parâmetros, já que a escolha de parâmetros ótimos requer muitos dados e iterações, o que não é ideal em cenários de aprendizado online. Considera-se atingido este objetivo pois apenas um parâmetro foi adicionado, com o adendo de que mesmo que seja mal escolhido, este parâmetro não deve piorar consideravelmente a performance do modelo em relação a um modelo único: caso a tolerância de drift seja muito grande, apenas um modelo será treinado. Caso seja muito pequeno, o número de modelos é ilimitado. Caso o número de modelos seja grande, as

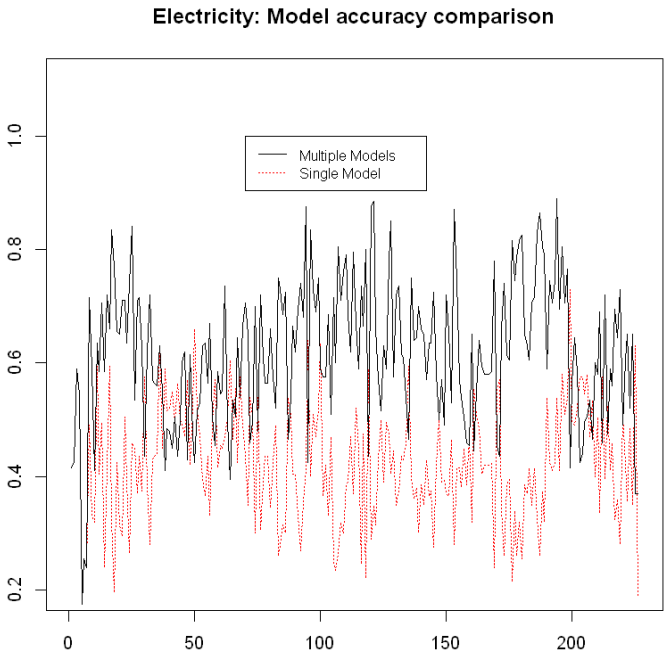

Figura 5. Performance de um único modelo versus do sistema de modelos desenvolvido para a base de dados Electricity.

predições serão potencialmente melhores por terem sido feitas com modelos "especializados". O ponto negativo disso seria principalmente espaço em disco, mas consideramos este um ponto secundário.

Para trabalhos futuros, é interessante expandir o algoritmo para que funcione com atributos discretos. Um outro ponto de melhoria interessante para o sistema desenvolvido seria a alteração do índice de concept drift calculado para um valor entre 0 e 1, facilitando a seleção do parâmetro de tolerância máxima do drift. Por fim, pode-se também adicionar um fator de esquecimento para remover modelos que já não são utilizados por muitas bateladas de dados.

REFERÊNCIAS

[1] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, Jul. 1989. [Online]. Available: http://dx.doi.org/10.1016/0893-6080(89)90020-8

[2] J. a. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 44:1–44:37, Mar. 2014. [Online]. Available: http://doi.acm.org/10.1145/2523813

[3] S. S. Petrova and A. D. Solov'ev, "The origin of the method of steepest descent," *Historia Mathematica*, vol. 24, no. 4, pp. 361 – 375, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0315086096921461

[4] M. Li, T. Zhang, Y. Chen, and A. J. Smola, "Efficient mini-batch training for stochastic optimization," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '14. New York, NY, USA: ACM, 2014, pp. 661–670. [Online]. Available: http://doi.acm.org/10.1145/2623330.2623612

[5] G. I. Webb, R. Hyde, H. Cao, H. Nguyen, and F. Petitjean, "Characterizing concept drift," *CoRR*, vol. abs/1511.03816, 2015. [Online]. Available: http://arxiv.org/abs/1511.03816

[6] H. Yu and G. I. Webb, "Adaptive online extreme learning machine by regulating forgetting factor by concept drift map," *Neurocomputing*, vol. 343, pp. 141 – 153, 2019, learning in the Presence of Class Imbalance and Concept Drift. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231219301572

[7] W. N. Street and Y. Kim, "A streaming ensemble algorithm (sea) for large-scale classification," in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '01. New York, NY, USA: ACM, 2001, pp. 377–382. [Online]. Available: http://doi.acm.org/10.1145/502512.502568