

LEILA MARIA VRIESMANN

**BOOSTING E ESTRATÉGIAS EVOLUCIONÁRIAS NA TAREFA DE
REGRESSÃO PARA A MINERAÇÃO DE DADOS TEMPORAIS**

CURITIBA

MARÇO DE 2006

LEILA MARIA VRIESMANN

**BOOSTING E ESTRATÉGIAS EVOLUCIONÁRIAS NA TAREFA DE
REGRESSÃO PARA A MINERAÇÃO DE DADOS TEMPORAIS**

**Dissertação de Mestrado em Informática
apresentada ao Programa de Pós-
Graduação em Informática, Setor de
Ciências Exatas, Universidade Federal
do Paraná.**

**Orientadora: Prof^a. Dra. Aurora
Trinidad Ramírez Pozo**

**Co-orientadora: Prof^a. Dra. Alaine
Margarete Guimarães**

CURITIBA

MARÇO DE 2006

AGRADECIMENTOS

A todos que, direta ou indiretamente, contribuíram para a realização deste trabalho.

Um agradecimento especial a Deus, que sempre me auxilia nesta pequena caminhada pela vida.

Aos meus pais, aos meus avós, aos meus tios, aos meus irmãos, enfim, aos meus familiares, por tudo o que fazem por mim.

À CAPES (pela bolsa de Mestrado), aos colegas de laboratório, de linha de pesquisa e de Mestrado.

Ao pessoal da coordenação, do apoio, do ensino e da pesquisa do Mestrado em Informática da UFPR, que com dedicação nos instruem e nos auxiliam.

Aos colegas da UEPG, pela compreensão e pela amizade.

Agradeço à professora e orientadora Aurora Trinidad Ramírez Pozo, pelo acompanhamento, auxílio e revisão do estudo, e à co-orientadora Alaine Margarete Guimarães, pelo apoio, pela assistência no inglês e pelos conselhos que, sem dúvida nenhuma, colaboraram na elaboração do trabalho.

SUMÁRIO

LISTA DE FIGURAS	IV
LISTA DE TABELAS.....	V
LISTA DE GRÁFICOS	VI
RESUMO	VII
ABSTRACT	VIII
1 INTRODUÇÃO.....	1
2 REVISÃO BIBLIOGRÁFICA.....	5
2.1 MINERAÇÃO DE DADOS TEMPORAIS	5
2.2 A TAREFA DE REGRESSÃO	8
2.2.1 Mapeamento de Regressão em Classificação.....	9
2.2.2 Modelos Lineares	10
2.2.3 Modelos Não Lineares	12
2.2.4 Mineração de Dados Temporais e Modelos de Regressão	12
2.3 COMPUTAÇÃO EVOLUCIONÁRIA E ESTRATÉGIAS EVOLUCIONÁRIAS.....	13
2.3.1 Estratégias Evolucionárias na Tarefa de Regressão	23
2.4 BOOSTING	26
2.4.1 AdaBoost e Regressão.....	30
2.5 TRABALHOS NA LITERATURA.....	34
3 BOOSTING EM ESTRATÉGIAS EVOLUCIONÁRIAS PARA A TAREFA DE REGRESSÃO... 42	
4 EXPERIMENTOS	47
4.1 BASES DE DADOS DE REGRESSÃO	49
4.2 SÉRIES TEMPORAIS UNIVARIADAS	51
4.2.1 Comparativo com Modelo ARMA.....	55
4.2.2 Resultados com Outra Função de <i>Fitness</i>	56
4.3 SÉRIES TEMPORAIS MULTIVARIADAS.....	57
5 CONCLUSÃO	59
6 REFERÊNCIAS BIBLIOGRÁFICAS	61

LISTA DE FIGURAS

FIGURA 1 – PERÍODO DE PREDIÇÃO	7
FIGURA 2 – DIAGRAMA PARA MAPEAR A REGRESSÃO EM CLASSIFICAÇÃO.....	10
FIGURA 3 – CICLO DE EVOLUÇÃO BÁSICO.....	14
FIGURA 4 – ALGORITMO (1+1)-ES.....	16
FIGURA 5 – NOTAÇÃO DE ES.....	17
FIGURA 6 – ALGORITMO GERAL DE ES	18
FIGURA 7 – CRIAÇÃO DE UM DESCENDENTE EM ES	20
FIGURA 8 – EXEMPLO DE MUTAÇÃO COM ÚNICO σ PARA O INDIVÍDUO	21
FIGURA 9 – EXEMPLO DE MUTAÇÃO COM UM VALOR DE σ PARA CADA GENE.....	22
FIGURA 10– EXEMPLO DE MUTAÇÃO COM UM VALOR DE σ E DE α PARA CADA GENE	23
FIGURA 11– PSEUDOCÓDIGO DO ADABOOST.....	29
FIGURA 12– PSEUDOCÓDIGO DO ADABOOST.R2.....	31
FIGURA 13– PSEUDOCÓDIGO DO GABOOST	38
FIGURA 14– PSEUDOCÓDIGO DO GPBOOST.....	40
FIGURA 15– ENTRADA E SAÍDA DO ESBOOST	43
FIGURA 16– REPRESENTAÇÃO DO ALGORITMO ESBOOST.....	45
FIGURA 17– PSEUDOCÓDIGO DO ESBOOST	46

LISTA DE TABELAS

TABELA 1 – CONJUNTO DE DADOS EXEMPLO PARA EVOLUÇÃO DE PERCEPTRON	24
TABELA 2 – HIPÓTESES E TAXAS DE CONFIANÇA DE UM EXEMPLO DE EXECUÇÃO DO GPBOOST.....	41
TABELA 3 – PARÂMETROS PARA O (10+50)-ES PADRÃO E PARA O ESBOOST COM T =100... 47	
TABELA 4 – BASES DE DADOS DE REGRESSÃO.....	49
TABELA 5 – MELHOR RMSE DE 10 EXECUÇÕES EM BASES DE DADOS DE REGRESSÃO.....	50
TABELA 6 – MÉDIA E DESVIO PADRÃO DO RMSE DE 10 EXECUÇÕES EM BASES DE REGRESSÃO	51
TABELA 7 – SÉRIES TEMPORAIS UNIVARIADAS	51
TABELA 8 – MELHOR RMSE DE 10 EXECUÇÕES EM SÉRIES TEMPORAIS UNIVARIADAS ...	52
TABELA 9 – MÉDIA E DESVIO PADRÃO DO RMSE DE 10 EXECUÇÕES EM SÉRIES TEMPORAIS UNIVARIADAS.....	52
TABELA 10 – MELHOR RMS DE 10 EXECUÇÕES EM SÉRIES TEMPORAIS UNIVARIADAS.....	56
TABELA 11 – MÉDIA E DESVIO PADRÃO DO RMS DE 10 EXECUÇÕES EM SÉRIES TEMPORAIS UNIVARIADAS.....	56
TABELA 12 – MELHOR RMSE DE 10 EXECUÇÕES NA SÉRIE SUNSPOTS COM OUTRA FUNÇÃO DE FITNESS.....	56
TABELA 13 – MÉDIA E DESVIO PADRÃO DO RMSE DE 10 EXECUÇÕES NA SÉRIE SUNSPOTS COM OUTRA FUNÇÃO DE FITNESS.....	57
TABELA 14 – MELHOR RMSE DE 10 EXECUÇÕES EM SÉRIES TEMPORAIS MULTIVARIADAS	58
TABELA 15 – MÉDIA E DESVIO PADRÃO DO RMSE DE 10 EXECUÇÕES EM SÉRIES TEMPORAIS MULTIVARIADAS	58

LISTA DE GRÁFICOS

GRÁFICO 1 – RESULTADOS DAS PREVISÕES OBTIDAS PARA O CONJUNTO DE TESTE (ATMOSFERA).....	53
GRÁFICO 2 – RESULTADOS DAS PREVISÕES OBTIDAS PARA O CONJUNTO DE TESTE (BEDIDA).....	54
GRÁFICO 3 – RESULTADOS DAS PREVISÕES OBTIDAS PARA O CONJUNTO DE TESTE (FORTALEZA).....	54

RESUMO

O aumento no volume de dados armazenados contribui para o crescimento da expectativa de obter informações implícitas que possam auxiliar em decisões futuras. A extração de padrão nos dados que possuem alguma dimensão de tempo é tarefa da Mineração de Dados Temporais. Quando os atributos têm características numéricas, os padrões podem ser descobertos por meio da regressão, a qual constrói modelos que mapeiam variáveis de entrada $x = \{x_1 \dots x_n\}$ para uma variável de saída y . Para determinar modelos (equações) lineares na regressão, convém que a técnica utilizada seja capaz de manipular variáveis contínuas. Diferentes técnicas podem ser utilizadas na Mineração de Dados, como Redes Neurais e algoritmos de Computação Evolucionária (Programação Genética, Algoritmos Genéticos, etc). A Computação Evolucionária (EC – *Evolutionary Computation*) é uma das áreas do aprendizado de máquina onde a solução de problemas baseia-se na teoria da evolução natural das espécies, proposta por Darwin. Entre as técnicas de Computação Evolucionária, as Estratégias Evolucionárias (ESs – *Evolution Strategies*) são apropriadas para a manipulação de variáveis contínuas. ESs codificam indivíduos (soluções) na forma de variáveis reais (contínuas) e evoluem essas soluções ao longo das gerações. Assim sendo, Estratégia Evolucionária pode ser utilizada na tarefa de regressão. Para aperfeiçoar as hipóteses obtidas por meio de ES, pode-se utilizar o Boosting, um método de aprendizado de máquina que visa gerar uma única solução precisa, combinando hipóteses fracas. Um algoritmo de Boosting é o AdaBoost, o qual fornece um conjunto de exemplos de treinamento com uma distribuição de pesos a um programa que gera hipóteses fracas. Isso força o programa a focar em exemplos de maior peso. O objetivo deste trabalho foi utilizar AdaBoost e ESs na tarefa de regressão para a Mineração de Dados Temporais. O algoritmo exposto nessa dissertação, denominado ESboost, foi implementado em C++ e utilizou a biblioteca EO. Realizaram-se diferentes experimentos com séries temporais univariadas, com séries temporais multivariadas e com bases de dados de regressão. Em relação às execuções da ES sem Boosting, observou-se que na maioria dos casos houve uma melhora pelo menos com uma das funções de perdas. Em muitos casos, houve melhora em relação à regressão linear e às *model trees* do software Weka, o qual é uma ferramenta para Mineração de Dados. Os resultados do ESboost nas séries temporais univariadas superaram o modelo ARMA em quase todas as séries. Trabalhos futuros poderão constituir-se da aplicação de outras técnicas de aprendizado de máquina no lugar de ES, de experimentos com outras funções de *fitness* e de estudos comparativos com outras metodologias para a predição de eventos.

Palavras chaves: Estratégias Evolucionárias; Boosting; dados temporais; regressão.

ABSTRACT

The increase in the volume of stored data contributes to the growth of the expectation of obtaining implicit information that can aid in future decisions. The pattern discovery in data presenting some time dimension represents a Temporal Data Mining task. The patterns can be discovered through the regression method when the attributes have numeric characteristics. Regression methods build models that map input variables $x = \{x_1 \dots x_n\}$ for an output variable y . In order to determine linear models (equations) in regression, we need a technique that is able to manipulate continuous variables. Different techniques can be used in Data Mining as, for example, Neural Networks and algorithms of Evolutionary Computation (Genetic Programming, Genetic Algorithms, etc). The Evolutionary Computation (EC) is one of the machine learning areas that defines the solution of problems based on the theory of the natural evolution of the species, proposed by Darwin. Among Evolutionary Computation techniques, the Evolution Strategies (ESs) are appropriated for manipulation of continuous variables. ESs codify individuals (solutions) in the form of real variables (continuous) and those solutions develop along the generations. Then, Evolution Strategies can be used in the regression task. In order to improve the hypotheses obtained by means of Evolution Strategies, it is possible the use of Boosting, a method of machine learning that seeks to generate a more precision solution combining weak hypotheses. An example of Boosting algorithm is the AdaBoost, which supplies a group of training examples with a distribution of weights to a program that generates weak hypotheses. That procedure forces the program in the examples with larger weight. The goal of this work was to use AdaBoost and ESs in the regression task for Temporal Data Mining. The algorithm showed in this dissertation, denominated ESboost, was implemented in C++ and used the EO library. Different experiments were accomplished using univariate time series, multivariate time series and regression dataset. Related to the executions without Boosting, it was observed an improvement in most of the cases at least with one of the loss functions. In many cases, it gets better in relation to the methods implemented in Weka software, a Data Mining tool. The results of ESboost in the univariate time series outperform the ARMA model in almost all the series. Some future works can be constituted of the application of other machine learning techniques in the place of ES, of experiments with other fitness functions and of comparative studies with other methodologies for the events prediction.

Keywords: Evolution Strategies; Boosting; temporal data; regression.

1 INTRODUÇÃO

Atualmente, um grande volume de dados é gerado e armazenado. Cada vez mais, cresce a velocidade com que esse processo ocorre e também a expectativa de obter informações implícitas nos dados que possam auxiliar em decisões futuras. Conseqüentemente, segundo WEISS (1999), aumenta-se o interesse em como o aprendizado de máquina e as técnicas estatísticas podem ser empregados para extrair conhecimento útil desses dados.

A extração de padrões (ou conhecimento) de um conjunto de dados é tarefa da Mineração de Dados, a qual é a etapa principal do processo de descoberta de conhecimento em base de dados¹ (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996). Os conjuntos de dados minerados podem ser estáticos ou possuir alguma dimensão de tempo e/ou espaço. A dimensão de tempo é percebida quando o fator tempo é uma das características da base de dados. Isso pode ser representado por meio de um atributo temporal ou por meio de seqüências, no caso dos dados serem coletados em intervalos regulares. Quando o banco de dados possui ao menos uma dimensão de tempo, a qual será considerada no processo de descoberta de conhecimento, utiliza-se o termo Mineração de Dados Temporais.

A predição de eventos é um dos objetivos mais procurados com a Mineração de Dados Temporais. Sua função é prever fatos do futuro com base em informações do passado.

Os atributos de um conjunto de dados utilizados para predição de eventos são classificados em: atributos previsores e atributo meta. O atributo meta representa a informação que se deseja prever e os atributos previsores indicam as características que serão levadas em consideração para determinar o atributo meta.

Na predição de eventos da Mineração de Dados Temporais, muitos

¹ O processo de descoberta de conhecimento em base de dados é também denominado KDD (*Knowledge Discovery in Database*).

problemas importantes do mundo real possuem características numéricas, com atributos metas e/ou previsores de valores contínuos². Uma das tarefas possíveis para prever eventos, nesses casos, é a regressão, que consiste no mapeamento de uma função $f(x)$ que descreve a base de dados.

Diferentes técnicas podem ser utilizadas na Mineração de Dados como, por exemplo, Redes Neurais e algoritmos de Computação Evolucionária. A Computação Evolucionária (EC – *Evolutionary Computation*) é uma das áreas do aprendizado de máquina onde o paradigma para a solução de problemas baseia-se na teoria da evolução natural das espécies, proposta por Charles Darwin (DARWIN, 1859). Algoritmos Genéticos (HOLLAND, 1992), Estratégias Evolucionárias (BÄCK; RUDOLPH; SCHEWEL, 1993), Programação Genética (KOZA, 1992) e Programação Evolucionária (FOGEL, 1962; FOGEL; OWENS; WALSH, 1966; BÄCK; RUDOLPH; SCHEWEL, 1993) são as principais técnicas de EC.

Várias abordagens são encontradas na literatura para manipular seqüências temporais por meio de projetos de sistemas neurais (BARRETO; ARAÚJO, 2001; PEARLMUTTER, 1995). Por outro lado, a utilização de técnicas de EC na Mineração de Dados Temporais é um campo de recentes estudos. Poucos são os trabalhos que abordam essa área (PARIS; ROBILLIARD; FONLUPT, 2001), especialmente quando se trabalha com a tarefa de regressão, ou seja, com a descoberta de um padrão por meio de funções na base de dados estudada.

O mapeamento de funções que descrevem o comportamento de um determinado atributo em uma base de dados pode ser realizado por técnicas capazes de manipular valores contínuos para, por exemplo, especificar pesos em cada atributo previsor. Entre as técnicas de Computação Evolucionária, as Estratégias Evolucionárias (ESs – *Evolution Strategies*) são apropriadas para a manipulação desse

² Valores contínuos são aqueles que podem possuir infinitos valores num determinado intervalo. Como exemplos, têm-se a altura de uma pessoa, o peso de um objeto, e demais valores obtidos por meio de um instrumento de medida.

tipo de dado. Estratégias Evolucionárias (BÄCK; RUDOLPH; SCHEWEL, 1993) codificam indivíduos (soluções potenciais) na forma de variáveis reais, os quais são evoluídos ao longo das gerações. Portanto, Estratégia Evolucionária pode ser utilizada na tarefa de regressão.

As Estratégias Evolucionárias podem gerar uma função que descreve a base de dados, mas certamente, como em outras técnicas, um erro de precisão de predição estará embutido. Conseqüentemente, a função apresentada pode produzir erros relativamente grandes se comparada a outras possíveis soluções que não puderam ser geradas no processo evolutivo. Uma possibilidade de diminuição desse erro é por meio do uso de Boosting (SCHAPIRE, 2002), um método de aprendizado de máquina que objetiva aperfeiçoar diferentes hipóteses (nesse caso, funções) fracas, combinando-as para gerar uma única solução mais precisa.

Um algoritmo de Boosting é o AdaBoost (FREUND; SCHAPIRE, 1997), o qual chama repetidamente um programa que gera hipóteses fracas fornecendo um conjunto de exemplos de treinamento com uma determinada distribuição de pesos. Esse procedimento força o algoritmo de aprendizado fraco a focar nos exemplos com maior peso, fazendo com que os exemplos difíceis do conjunto de treinamento sejam explorados.

Na literatura, não foi encontrado, até o momento, nenhum experimento que relatasse o uso de ES com AdaBoost e nem de ES com Mineração de Dados Temporais. Logo, a aplicação do AdaBoost nas funções geradas pelo algoritmo de Estratégia Evolucionária para a tarefa de regressão de dados temporais pode preencher essa lacuna. Outras contribuições que podem ser visualizadas é a utilização de ES na tarefa de regressão e o comportamento do AdaBoost na Mineração de Dados Temporais contínuos.

O objetivo desse trabalho é, portanto, utilizar Estratégias Evolucionárias na tarefa de regressão para a Mineração de Dados Temporais. As funções geradas pela Estratégia Evolucionária são aperfeiçoadas por meio de um algoritmo de Boosting,

como o AdaBoost.

O conteúdo dessa dissertação está organizado como segue. A Seção 2 trata da revisão bibliográfica. A Seção 2.1 demonstra uma visão geral de Mineração de Dados Temporais e de séries temporais, com o seu embasamento teórico.

A Seção 2.2 introduz a tarefa de regressão. São discutidas algumas modelagens, como mapeamento de regressão em classificação, modelos lineares e modelos não lineares.

A Seção 2.3 apresenta definições de Computação Evolucionária e de Estratégias Evolucionárias (ESs). Na Seção 2.3.1 pode-se encontrar explicações sobre como ESs podem ser utilizadas para a tarefa de regressão.

A Seção 2.4 expõe o método de aprendizado de máquina chamado Boosting, bem como um de seus algoritmos, o AdaBoost.

Alguns trabalhos relacionados ao tema da dissertação são citados na Seção 2.5. Na Seção 3 localiza-se a descrição do algoritmo proposto, o qual utiliza Boosting e Estratégias Evolucionárias para a tarefa de regressão.

A Seção 4 relata os experimentos, os resultados obtidos e traz alguns comentários. Finalmente, a Seção 5 traz a conclusão, onde são ressaltados pontos importantes do estudo e algumas previsões de continuidade do trabalho.

2 REVISÃO BIBLIOGRÁFICA

2.1 MINERAÇÃO DE DADOS TEMPORAIS

A Mineração de Dados (*Data Mining*), segundo FAYYAD, PIATETSKY-SHAPIRO e SMYTH (1996), visa extrair padrões (estados) de um conjunto de dados. Sua extensão, a Mineração de Dados Temporais, tem a capacidade de minerar atividades ao invés de somente estados e, assim, inferir relacionamentos de proximidade temporal e contextual, indicando também uma associação causa-efeito (RODDICK; SPILIOPOULOU, 2002).

A Mineração de Dados Temporais tem a habilidade de minerar aspectos do comportamento de (comunidades de) objetos, ao invés de simplificar regras de mineração que descrevem seus estados em um ponto no tempo, ou seja, há a expectativa de entender “porquê” ao invés de simplesmente “o quê” (RODDICK; SPILIOPOULOU, 2002).

Quatro grandes categorias de temporalidade nos dados podem ser determinadas (RODDICK; SPILIOPOULOU, 2002). A primeira categoria é a estática, onde nenhum contexto temporal é incluído. A segunda é a categoria de seqüências, a qual inclui listas ordenadas de eventos recebidos em intervalos não regulares. Já na categoria *timestamped* os dados estáticos são obtidos em intervalos mais ou menos regulares. Exemplos da categoria *timestamped* incluem o censo e os dados meteorológicos de satélites. Finalmente, tem-se a categoria totalmente temporal, onde cada tupla na base de dados pode ter mais que uma dimensão de tempo armazenada.

Aprender a predizer eventos futuros baseando-se em seqüências de eventos obtidas no passado é um importante problema real que surge em vários contextos (WEISS, 1999). Nos parágrafos abaixo, é relatada uma definição formal do problema de predição de eventos, baseada em WEISS (1999).

Um evento E_t é uma observação *timestamped* ocorrendo em um tempo t

descrita por um conjunto de pares “características-valor”. O *target event* (evento objetivo) é um tipo específico de evento que se deseja aprender. No problema de previsão de evento deve-se aprender um procedimento que corretamente prediz se o evento objetivo irá ocorrer num futuro próximo, dada uma seqüência de eventos *timestamped*. Uma previsão P_t , ocorrendo num tempo t , é correta se um evento objetivo ocorre dentro do seu *prediction period* (período de previsão).

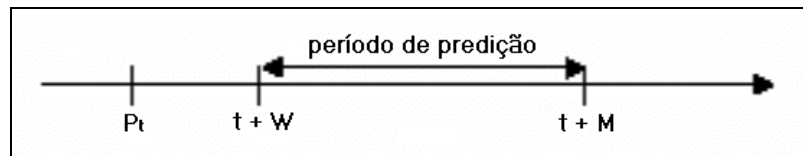
O período de previsão é definido por um tempo de alerta W (*warning time*) e um tempo de monitoramento M . O tempo de alerta é o “*lead time*” (tempo que serve de guia) necessário para uma previsão ser útil e o tempo de monitoramento determina o quanto a previsão se estende no futuro. Enquanto o valor do tempo de alerta é criticamente dependente do domínio do problema (por exemplo, quanto tempo levará para substituir uma peça de um equipamento), há geralmente muita flexibilidade na definição do tempo de monitoramento. Quanto maior o valor do tempo de monitoramento, mais fácil e menos significante será a previsão do problema.

Suponha-se um exemplo de aplicação onde a previsão da necessidade de substituição de uma peça de uma máquina é realizada com base nos ruídos emitidos. Em um instante t , observou-se que o ruído atingiu uma determinada freqüência que indica desgaste da peça. Sabe-se que, mesmo um pouco desgastada, essa peça ainda pode ser utilizada por um tempo de uma a duas semanas sem danos ao equipamento e a sua função. Antes de uma semana após a detecção do ruído a peça não precisa ser substituída e, após duas semanas, com certeza a peça desgastada deverá estar descartada. Depois da detecção do ruído, o tempo de alerta W corresponde a uma semana e o tempo de monitoramento M corresponde a duas semanas. Portanto, o período de previsão inicia-se em $t + W$ e termina em $t + M$. Dentro desse intervalo, a peça deverá ser substituída.

A Figura 1 demonstra uma previsão P_t , o seu tempo de alerta W e o seu tempo de monitoramento M .

Dessa maneira, observa-se que a acomodação do tempo em técnicas de

Mineração de Dados provê uma janela dentro do arranjo de eventos e, portanto, uma habilidade para sugerir causa e efeito que são negligenciados quando o componente temporal é ignorado ou tratado como um simples atributo numérico.



FONTE: TRADUÇÃO DE WEISS (1999).

FIGURA 1 – PERÍODO DE PREDIÇÃO

Os valores de um fenômeno ao longo do tempo são chamados, segundo KOUT, VLCEK e KLEMA (2005), séries temporais (*time series*). Séries temporais que consistem de observações individuais armazenadas seqüencialmente com incrementos iguais de tempo são chamadas “séries temporais univariadas (*univariate time series*)” (NIST/SEMATECH e-Handbook of Statistical Methods, 2006).

Uma série temporal univariada normalmente é apresentada em uma única coluna de números. O tempo, portanto, é uma variável implícita neste tipo de série temporal, uma vez que os dados são coletados em intervalos regulares de tempo e não existe a necessidade de fornecê-los explicitamente (NIST/SEMATECH e-Handbook of Statistical Methods, 2006).

Muitas vezes, a predição de valores futuros de um fenômeno não depende exclusivamente de dados do passado com uma única variável. Pode haver informações relevantes disponíveis em outras variáveis do ambiente, criando uma relação de dependência. Essas variáveis podem ser armazenadas na mesma base de dados, gerando uma série temporal multidimensional. Nesse contexto, segundo KLEMA, KOUT e VEJMEJKA (2004), tem-se uma “série temporal multivariada (*multivariate time series*)”.

Dados temporais numéricos podem ser mapeados por meio de uma função que descreve o comportamento das variáveis ao longo do tempo. O mapeamento pode

ser realizado por meio da tarefa de regressão, que é o assunto da próxima seção.

2.2 A TAREFA DE REGRESSÃO

O problema de aproximar os valores de uma variável contínua, segundo WEISS e INDURKHYA (1995), é descrito na literatura estatística como regressão. Fornecidos exemplos com variáveis de entrada $x = \{x_1 \dots x_n\}$ (previsoras) e variável de saída y (meta), a tarefa de regressão é encontrar um mapeamento $y = f(x)$. Os exemplos finitos, relativos ao espaço de possibilidades, são incompletos, e um modelo predefinido é necessário para concisamente mapear x para y . A regressão difere da classificação pelo fato de que a variável de saída y nos problemas de regressão é contínua, enquanto na classificação é estritamente categórica. Nesta perspectiva, classificação pode ser pensada como uma subcategoria de regressão. Alguns pesquisadores da área de aprendizado de máquina dão ênfase nessa conexão descrevendo regressão como “aprender a maneira de classificar classes contínuas” (QUINLAN, 1993).

No mundo real, problemas de classificação são mais frequentemente encontrados que problemas de regressão. Isso justifica a atenção maior dada à classificação do que à regressão. Mas muitos problemas importantes no mundo real são do tipo de regressão. Outra razão para o foco na regressão é que métodos de regressão podem ser utilizados para resolver problemas de classificação. Por exemplo, redes neurais são frequentemente aplicadas para problemas de classificação.

No contexto temporal, os problemas com variáveis numéricas podem ser tratados pela tarefa de regressão. No caso de séries temporais multivariadas, uma variável deve ser escolhida para ser o y (meta), e as demais serão as variáveis previsoras. A variável meta considerada pode variar no tempo em relação às variáveis previsoras, ou seja, pode haver um intervalo regular de variação no tempo (no futuro) entre as variáveis previsoras e meta.

As séries temporais univariadas são constituídas de valores obtidos ao longo

do tempo para uma mesma variável. Por esse motivo, tanto os valores das variáveis previsoras quanto o valor da variável meta correspondem, na realidade, a mesma variável. A diferença está no fato de que variáveis previsoras são constituídas por valores da variável alguns passos no passado em relação à variável meta. Por exemplo, almejando-se descobrir o valor da variável no tempo t , os valores da variável nos tempos $t-4$, $t-3$, $t-2$ e $t-1$ poderão constituir as variáveis previsoras.

A tarefa de regressão pode ser realizada por meio de alguns modelos, os quais são tratados nas três subseções a seguir.

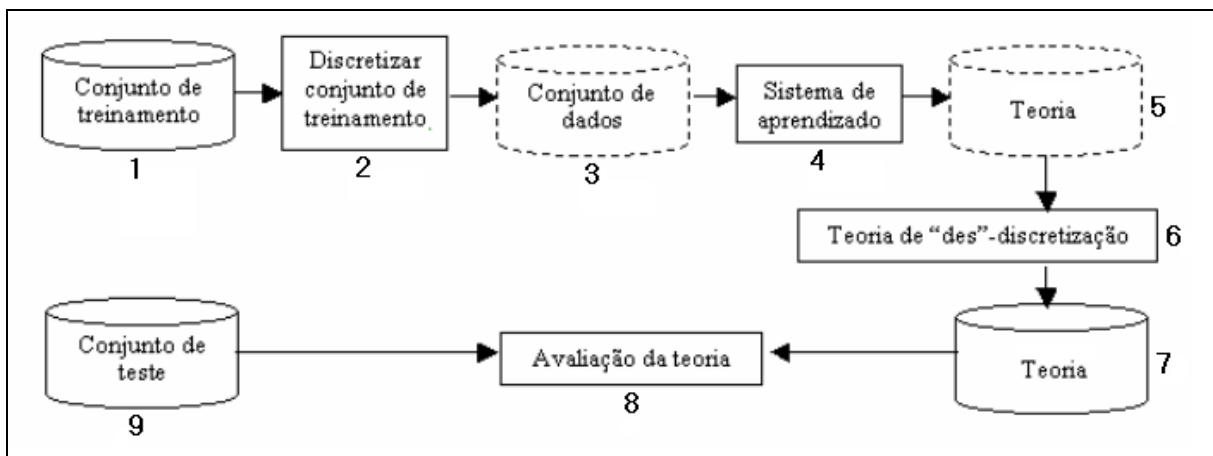
2.2.1 Mapeamento de Regressão em Classificação

Em problemas de regressão são fornecidos exemplos de um conjunto de variáveis independentes x_1, x_2, \dots, x_n (previsoras) e o valor da respectiva variável dependente y (meta). O objetivo é obter um modelo que capture o mapeamento $y = f(x_1, x_2, \dots, x_n)$ baseados nos exemplos fornecidos. A classificação difere dessa configuração no sentido de que a classe é categórica em vez de numérica (TORGO; GAMA, 1996).

Mapear a regressão em classificação, segundo TORGO e GAMA (1996), é um tipo de técnica de pré-processamento que torna possível usar algoritmos de classificação em problemas de regressão. Os usos desses algoritmos envolvem duas etapas principais. A primeira é a criação de um conjunto de dados com classes discretas (Passos 1, 2 e 3 da Figura 2), a qual realiza a busca nos valores das classes contínuas originais e a divisão em uma série de intervalos. Cada um desses intervalos será uma classe discreta. Cada exemplo cujo valor de variável de saída pertence a um intervalo será nomeado para a respectiva classe. Após essa etapa, é efetuado o aprendizado da teoria (Passos 4 e 5 da Figura 2). A segunda etapa consiste em reverter o processo de discretização (Passos 6 e 7 da Figura 2) depois da execução da fase de aprendizado. No Passo 8 da Figura 2 faz-se a avaliação da teoria, utilizando o conjunto de teste (Passo 9). Todo esse processo torna possível fazer previsões numéricas do

modelo de regressão aprendido.

A idéia de mapear a regressão em classificação, segundo TORGO e GAMA (1996), foi originalmente usada por WEISS e INDURKHYA (1993, 1995) com o sistema de regressão baseado em regras. Utilizou-se o algoritmo “*P-class*”³ para a discretização como parte do sistema de aprendizado. Esse trabalho mostra que é possível obter excelentes resultados preditivos transformando problemas de regressão em classificação e então usar um sistema de aprendizado para classificação. Baseado nesse trabalho, TORGO e GAMA (1996) orientaram sua pesquisa somente na fase de discretização para permitir o uso dessa metodologia com diferentes sistemas classificadores.



FONTE: BASEADO EM TORGO E GAMA (1996).

FIGURA 2 – DIAGRAMA PARA MAPEAR A REGRESSÃO EM CLASSIFICAÇÃO

2.2.2 Modelos Lineares

A regressão linear consiste no trabalho de determinar uma equação linear que contém uma ou mais variáveis independentes⁴ (x_1, \dots, x_n) para cada variável

³ Esse algoritmo é conhecido historicamente como método “*K-means*” na estatística e em reconhecimento de padrão.

⁴ Também denominadas variáveis predictoras ou de entrada.

dependente⁵ (y). Segundo RAMÍREZ, VELHO e FERREIRA (2004), quando a equação apresenta mais que uma variável independente (ou previsora), a tarefa é a regressão linear múltipla.

O tratamento da regressão linear pode ser realizado por diferentes técnicas. Um exemplo é o uso de redes neurais (FOG, 1995; LACHTERMACHER; FULLER, 1995; SARLE, 1995) de camada única. Uma rede de camada única é aquela que possui apenas uma camada de processamento, ou seja, os dados entram, sofrem o processamento e saem sem a passagem por camadas extras. ROSENBLATT (1958) concebeu um algoritmo de treinamento para um tipo de rede de camada única chamada de “*perceptron*”. Os valores de entrada e níveis de ativação do *perceptron* são ou -1 ou $+1$; os pesos têm valores reais. O nível de ativação do *perceptron* é dado pela soma dos valores ponderados das entradas $\sum (w_i x_i)$. Os *perceptrons* usam uma função de limiar abrupto simples, onde uma ativação acima de um limiar resulta num valor de saída 1, ou então, caso contrário, este valor é -1 (LUGER, 2004).

Uma abordagem mais tradicional para o problema de determinar uma função é a regressão linear clássica dos mínimos quadrados (SCHEFFE, 1959). Segundo WEISS e INDURKHYA (1995), o objetivo da regressão é minimizar a distância entre os valores de saída dos exemplos, y_i , e os valores preditos y_i' . Duas medidas de distância são freqüentemente usadas. A medida clássica de regressão é a Equação 1, que calcula a média da distância quadrada entre y_i e y_i' , isto é, a variância. Isso direciona para uma formulação para o modelo linear dos mínimos quadrados. A média da distância absoluta (desvio) na Equação 2 é usada em regressão de desvio absoluto mínimo (MAD – *Mean Absolute Distance*).

$$\text{Variância} = (\sum (y_i - y_i')^2) / n \quad (1)$$

⁵ Também denominada variável meta ou de saída.

$$\text{MAD} = (\sum |y_i - y_i'|) / n \quad (2)$$

2.2.3 Modelos Não Lineares

Desenvolvida e refinada ao longo de muitos anos, a regressão linear provou ser pouco efetiva para muitas aplicações do mundo real, as quais possuem características não lineares. Claramente, o modelo linear computacionalmente simples tem seus limites, e modelos mais complexos tratam de uma maneira melhor os dados. Um exemplo é a previsão de dados de chuva. Um modelo linear poderá apresentar erros relativamente grandes se comparado a um modelo não linear.

Com o aumento do poder computacional e com maiores volumes de dados, aumentou-se o interesse em métodos alternativos de regressão não linear. Modelos de regressão não lineares são explorados pela comunidade de pesquisa estatística e muitos novos métodos emergiram (EFRON, 1988). Métodos para regressão são também desenvolvidos fora da comunidade de pesquisa estatística. Um exemplo é a aplicação de redes neurais treinadas por *back-propagation* (MCCLELLAND; RUMELHART, 1988). Outros modelos podem ser encontrados na análise numérica (GIROSI; POGGIO, 1990). Uma revisão de diferentes modelos de regressão, com aplicações para classificação, está disponível na literatura em RIPLEY (1993). A maioria desses métodos produz soluções em termos de modelos com pesos.

2.2.4 Mineração de Dados Temporais e Modelos de Regressão

Na Mineração de Dados Temporais, quando os valores são numéricos, é possível utilizar qualquer um dos modelos apresentados nas subseções anteriores. O primeiro modelo, de mapeamento da regressão em classificação, não é muito interessante pela necessidade de discretização dos dados e de reversão do processo. Isso consome tempo e pode comprometer a precisão dos resultados, dependendo das

classes escolhidas.

Os modelos linear e não linear não possuem os mesmos problemas apresentados no mapeamento da regressão em classificação, uma vez que não necessitam da fase de discretização. O modelo não linear, apesar de melhor tratar os dados de muitas aplicações reais, tem a desvantagem de ser mais complexo que o modelo linear. Além disso, o desempenho de modelos lineares pode ser aperfeiçoado por meio de algoritmos específicos, como o Boosting (SCHAPIRE, 2002). Nessas condições, em termos de simplicidade, o modelo linear pode ser mais interessante para dados temporais do que o modelo não linear.

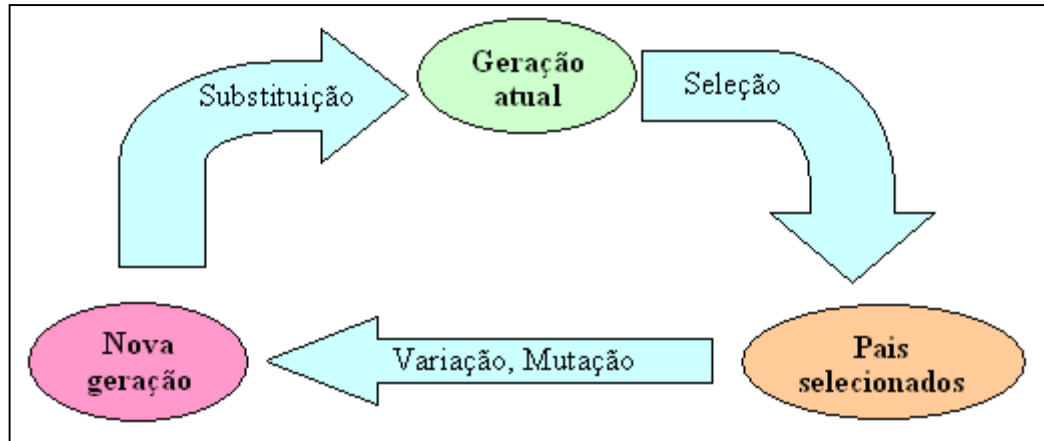
Técnicas de Computação Evolucionária (DARWIN, 1859) são capazes de realizar a regressão linear. Entre suas técnicas, têm-se Estratégias Evolucionárias (RECHENBERG, 1965), que são apropriadas para manipulação de valores numéricos contínuos e, portanto, são aptas para regressão linear. É justamente sobre a Computação Evolucionária e as Estratégias Evolucionárias que a próxima seção trata.

2.3 COMPUTAÇÃO EVOLUCIONÁRIA E ESTRATÉGIAS EVOLUCIONÁRIAS

A Computação Evolucionária (EC – *Evolutionary Computation*) é um dos ramos da Inteligência Computacional que propõe um paradigma para a solução de problemas inspirado na seleção natural (DARWIN, 1859). A EC imita a teoria da evolução proposta por Charles Darwin em 1858.

A EC compreende um conjunto de técnicas de busca e otimização onde indivíduos reproduzem-se e competem pela sobrevivência ao longo das gerações. Algumas das técnicas da EC são (WHIGHAM, 1995): Programação Evolucionária (FOGEL, 1962; FOGEL; OWENS; WALSH, 1966; BÄCK; RUDOLPH; SCHEWEFEL, 1993), Estratégias Evolucionárias (BÄCK; RUDOLPH; SCHEWEFEL, 1993), Algoritmos Genéticos (HOLLAND, 1992) e Programação Genética (KOZA, 1992). Esses algoritmos são feitos de várias iterações de um ciclo de evolução básico, como mostra a Figura 3. Esse ciclo termina quando o número

máximo de iterações é atingido, ou quando são satisfeitas as restrições do problema.



FONTE: TRADUÇÃO DE DIANATI, SONG E TREIBER (2002).

FIGURA 3 – CICLO DE EVOLUÇÃO BÁSICO

Diferentes técnicas da Computação Evolucionária incorporam o mesmo ciclo básico com diferentes modelos de representações ou combinações específicas de Variação, Mutação, Seleção e métodos de Substituição. O ponto interessante na implementação é o balanço entre duas operações opostas. Em um lado a operação de Seleção pretende reduzir a diversidade da população (conjunto de soluções possíveis) e no outro lado os operadores de Variação e Mutação tentam incrementar a diversidade da população. Esse fato direciona à convergência e à qualidade da solução (DIANATI; SONG; TREIBER, 2002).

Algoritmos Genéticos são algoritmos de busca baseados nos mecanismos de seleção natural e na genética. Nesta técnica, os indivíduos da população são utilizados como possíveis soluções do problema. Esses indivíduos são analisados segundo algum critério e novas soluções são criadas através da seleção de indivíduos bem adaptados e aplicação de operadores genéticos (GOLDBERG, 1989).

Programação Genética é uma extensão da técnica de Algoritmos Genéticos (KOZA, 1992; KOZA, 1994). Suas principais diferenças são a forma de representação e o significado das estruturas que manipulam. Em Programação Genética é comum

utilizar árvores para representar os programas de uma população (CAVALHEIRO, 2002).

Estratégia Evolucionária (ES – *Evolution Strategy*) é um algoritmo onde indivíduos (soluções potenciais) são codificados por um conjunto de variáveis de valores reais, o “genoma” (KUSIAK, 2000). Segundo DIANATI, SONG e TREIBER (2002), as ESs foram inicialmente desenvolvidas com o propósito de otimização de parâmetros. De acordo com RECHENBERG (1965), o primeiro algoritmo de ES foi apresentado em 1964 na Universidade Técnica de Berlim (TUB – *Technical University of Berlin*). A idéia era imitar os princípios da evolução orgânica em otimização de parâmetros experimental para aplicações tais como *pipe bending* ou controle de PID para um sistema não linear (DIANATI; SONG; TREIBER, 2002). Em suas palavras “o método de evolução orgânica representa uma estratégia ótima para a adaptação de seres vivos em seu ambiente... [e]... portanto seria conveniente utilizar os princípios da evolução biológica para a otimização de sistemas técnicos” (SCHWEFEL, 1981).

O primeiro algoritmo de Estratégia Evolucionária proposto utilizava um esquema mutação-seleção conhecido como *two-membered ES* ou (1+1)-ES (DIANATI; SONG; TREIBER, 2002). Segundo BÄCK, RUDOLPH e SCHEWEL (1993), o (1+1)-ES trabalhava com um indivíduo, o qual criava um descendente por meio de mutação, sendo que o melhor entre os dois era selecionado deterministicamente para integrar a próxima geração.

Para obter a taxa de mutação ótima desse esquema, RECHENBERG (1973) calculou as taxas de convergência de duas funções modelo e seus desvios padrão ótimos para mutações de sucesso (DIANATI; SONG; TREIBER, 2002). Disso, ele postulou sua regra de sucesso 1/5 (RECHENBERG, 1973):

“A proporção de mutações de sucesso para todas as mutações deve ser 1/5. Se for maior que 1/5, incremente a variância; se for menor, diminua a variância de mutação”.

O seguinte algoritmo para o (1+1)-ES foi adaptado de DIANATI, SONG e

TREIBER (2002):

1. Escolha um único vetor pai que contém m parâmetros $X = (x_1, x_2, \dots, x_m)$. Cada parâmetro é escolhido por um processo randômico e satisfaz as restrições do problema.
2. Crie um novo descendente por meio de mutação. Para obter a mutação nesse método, adicione um vetor randômico do tamanho de X com distribuição normal (média zero e variância σ):

$$X' = X + N(0, \sigma)$$
 Se for o caso, aplique a regra de sucesso 1/5.
3. Compare as soluções para X e X' . Escolha o melhor membro para a próxima geração.
4. Repita os passos 2 e 3 até encontrar uma solução satisfatória, ou até que tenha se esgotado o tempo computacional (ou número de gerações).

FIGURA 4 – ALGORITMO (1+1)-ES

Esse procedimento não é totalmente um método Monte Carlo e posteriormente foi aperfeiçoado, adicionando a noção de população. RECHENBERG (1973) propôs o *multimembered ES* onde $\mu > 1$ pais participam na geração de 1 descendente. A denotação utilizada foi $(\mu + 1)$ -ES. Nesse método, todos os pais têm as mesmas probabilidades de casamento e, como o *two-membered ES*, o membro de menor ajuste da população, incluindo todos os pais e um descendente, é eliminado em cada geração (DIANATI; SONG; TREIBER, 2002).

O $(\mu + 1)$ -ES não é uma estratégia muito utilizada, mas direcionou para avanços de habilitação da auto-adaptação de parâmetros como o desvio padrão das mutações (SCHWEFEL, 1975; SCHWEFEL, 1977; SCHWEFEL, 1981). Os algoritmos $(\mu + 1)$ -ES, segundo DIANATI, SONG e TREIBER (2002), implementaram auto-adaptação submetendo os parâmetros de evolução (desvio padrão de mutações) no processo de criação de novos indivíduos.

O $(\mu + \lambda)$ -ES especifica que μ pais produzem λ descendentes ($\lambda > \mu$). Os descendentes competem com seus pais na seleção dos μ melhores indivíduos para a criação da próxima geração. Esse procedimento apresenta problemas com ótimos

locais e, para solucioná-los, criou-se o (μ, λ) -ES, onde o tempo de vida de cada indivíduo é de somente uma geração. Enquanto Schwefel diz que o (μ, λ) -ES é melhor que o $(\mu + \lambda)$ -ES, evidências recentes sugerem que o último é tão bom quanto, ou melhor, que o primeiro em aplicações práticas (DIANATI; SONG; TREIBER, 2002).

Assim, uma ES é caracterizada pelo tamanho da população, o número de descendentes produzidos em cada geração e se a nova população é selecionada de pais e descendentes ou somente de descendentes (KUSIAK, 2000). Sua notação pode ser visualizada na Figura 5.

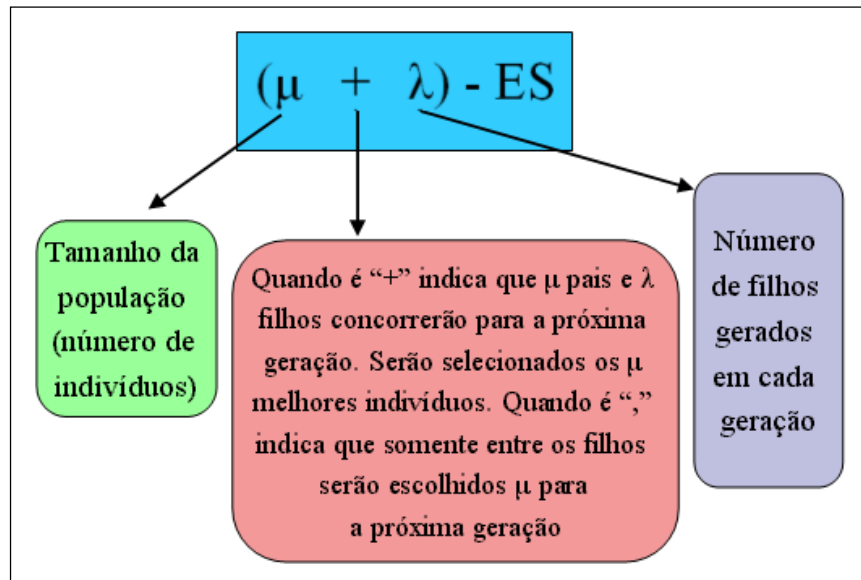


FIGURA 5 – NOTAÇÃO DE ES

Um algoritmo geral de ES pode ser, segundo DIANATI, SONG e TREIBER (2002), semelhante ao expresso na Figura 6. O número de gerações (ou as condições de parada do algoritmo), de pais (μ), de filhos (λ), de posições (m) em cada indivíduo, a probabilidade de recombinação, a taxa de mutação, o tipo de operador de recombinação (sem recombinação, discreto, intermediário, discreto global, intermediário global) e o tipo de ES ("+" ou ",") são valores de entrada para o algoritmo. Seu primeiro passo consiste na criação dos μ pais com m posições. Cada

1. Escolha μ vetores pai que contém m parâmetros $X = (x_1, x_2, \dots, x_m)$. Cada parâmetro é escolhido por um processo randômico e satisfaz as restrições do problema.
 2. Crie λ novos descendentes por meio de recombinação de μ pais e de mutação como no passo 2 do (1+1)-ES (Figura 4). Há 5 tipos de operadores de recombinação:
 - Sem recombinação: Selecione randomicamente um pai e deixe $x_t' = x_t$.
 - Discreto: Selecione randomicamente 2 pais a e b e deixe $x_t' = x_{t,a}$ ou $x_t' = x_{t,b}$ com probabilidade igual.
 - Intermediário: Selecione randomicamente 2 pais a e b e deixe $x_t' = \frac{1}{2} (x_{t,a} + x_{t,b})$.
 - Discreto global: Selecione um novo par de a_t e b_t pais para cada parâmetro x_t . Faça: $x_t' = (x_{at,1}$ ou $x_{bt,2})$ com probabilidade igual.
 - Intermediário global: Selecione um novo par de a_t e b_t pais para cada parâmetro x_t . Faça: $x_t' = \frac{1}{2} (x_{at,1} + x_{bt,2})$.
- Gere a população descendente com o seguinte algoritmo:
- População corrente: P^t
- Descendente intermediário: $X'(x_1', x_2', \dots, x_m')$
- Operador de mutação: M
- Meta-controle de tamanho de passo: $\Delta\sigma$
- Operador de recombinação: R
- $(X', \sigma') = R(P^t)$
- $(X'', \sigma'') = M[(X', \sigma')]$
- $\sigma'' = \sigma' \exp(N(0, \Delta\sigma))$
- $X'' = X' + N(0, \sigma')$
- Note que o operador de mutação é aplicado tanto para os parâmetros quanto para as variâncias.
3. Escolha as μ mais adaptadas soluções para a próxima geração:
 - Para $(\mu + \lambda)$ -ES: Selecione a próxima geração da população $(\mu + \lambda)$ de todos os pais e descendentes.
 - Para (μ, λ) -ES: Selecione a próxima geração da população λ de descendentes.
 4. Repita os passos 2 e 3 até encontrar uma solução satisfatória, ou até que tenha se esgotado o tempo computacional (ou número de gerações).

FONTE: TRADUÇÃO DE DIANATI, SONG e TREIBER (2002)

FIGURA 6 – ALGORITMO GERAL DE ES

posição recebe um valor aleatório, de acordo com o domínio do problema. É realizada a recombinação de pais para gerar os filhos, os quais depois passam por um processo de mutação. Posteriormente, de acordo com o tipo de ES, os melhores entre os pais e os filhos (tipo “+”), ou os melhores dos filhos (tipo “,”) são selecionados para a criação da próxima geração. O processo se repete até que a condição de parada seja atingida (Figura 6).

Nos meados dos anos sessenta, FOGEL, OWENS e WALSH (1966) descreveram a Programação Evolucionária para a evolução de máquinas de estado finito. Cada máquina na “população pai” gera um descendente por meio de mutação, e os melhores da metade do número de pais e filhos são selecionados para sobreviver. Esse processo poderia ser chamado de estratégia $(\mu+\mu)$, na terminologia de ES (BÄCK; SCHWEFEL, 1993).

A auto-adaptação de parâmetros de estratégia é uma das características mais importantes para o sucesso de Estratégias Evolucionárias e Programação Evolucionária, pois ambos utilizam princípios evolucionários para a busca no espaço de variáveis e parâmetros de estratégia (BÄCK; FOGEL; MICHALEWICZ, 2000).

O termo “parâmetros de estratégia”, segundo BÄCK, FOGEL e MICHALEWICZ (2000), refere-se a parâmetros que controlam o processo de busca evolucionário, tais como taxas de mutação, variâncias de mutação e probabilidades de recombinação. Normalmente, os parâmetros de estratégia são auto-adaptados ao nível dos indivíduos e incorporados na sua representação juntamente com o conjunto de variáveis, ou seja, o espaço individual I é dado por $I = A_x \times A_s$. Nessa representação, o A_x denota o conjunto de variáveis (representações de soluções) e o A_s denota o conjunto de parâmetros de estratégia (BÄCK; FOGEL; MICHALEWICZ, 2000).

Para um indivíduo $a = (x, s)$ consistindo de um vetor de variáveis x e um conjunto de parâmetros de estratégia s , o mecanismo de auto-adaptação é

normalmente implementado primeiro, recombina e mutando (de acordo com alguma função de probabilidade de densidade) o vetor de parâmetros de estratégia s , produzindo s' , e então utilizando os parâmetros de estratégia atualizados para (recombinar e) mutar o vetor de variáveis x , produzindo x' (BÄCK; FOGEL; MICHALEWICZ, 2000). Esse processo é ilustrado na Figura 7.

No caso mais geral, um indivíduo $a = (x, \sigma, \alpha)$ de uma Estratégia Evolucionária (μ, λ) consiste de até três componentes $x \in \mathbb{R}^n$, $\sigma \in \mathbb{R}^{n\sigma}$, e $\alpha \in [-\pi, \pi]^{n\alpha}$, onde $n_\sigma \in \{1, \dots, n\}$ e $n_\alpha \in \{0, (2n - n_\sigma)(n_\sigma - 1) / 2\}$. O operador de mutação trabalha adicionando uma variável randômica X de dimensão n , com distribuição normal de acordo com cada um dos parâmetros de estratégia σ' e α' do indivíduo (BÄCK; FOGEL; MICHALEWICZ, 2000).

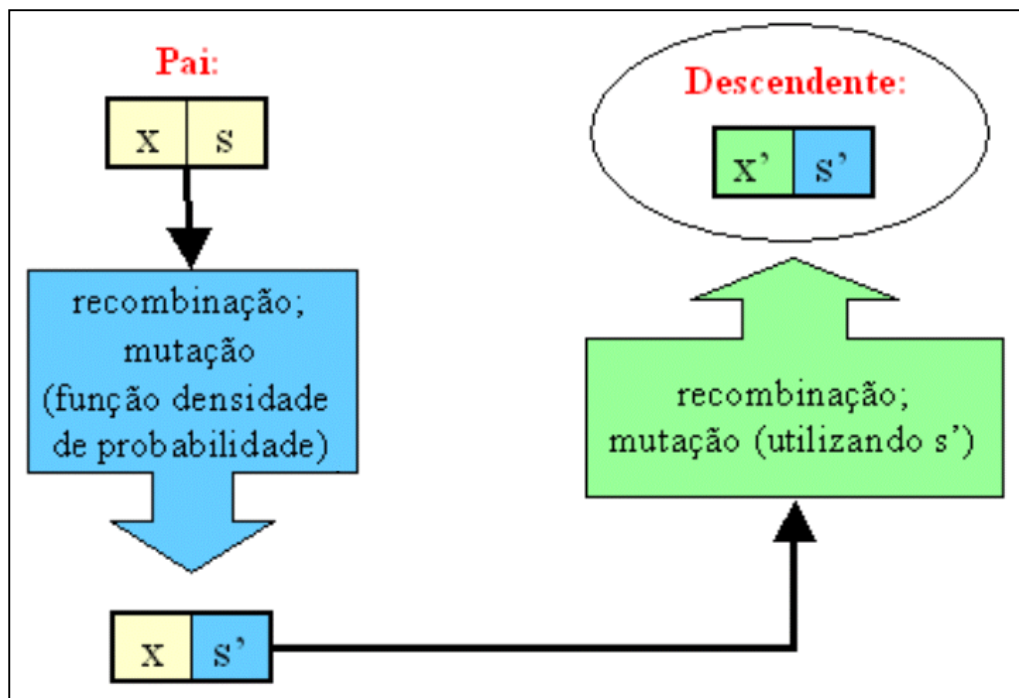


FIGURA 7 – CRIAÇÃO DE UM DESCENDENTE EM ES

Algumas das principais variações da auto-adaptação podem ser citadas, de

acordo com o número de parâmetros de estratégia incorporados na representação do indivíduo:

- a) Único σ' para todos os atributos x_i do indivíduo. Abaixo segue a fórmula da mutação (BÄCK; FOGEL; MICHALEWICZ, 2000):

$$\sigma' = \sigma \exp(\tau_0 N(0,1)), \text{ sendo } \tau_0 \propto n^{-1/2}$$

$$x_i' = x_i + \sigma' N_i(0,1)$$

onde n é o número de atributos e τ_0 é uma taxa de aprendizado configurável.

Um exemplo de uso de um σ' para todos os atributos x_i é representado na Figura 8.

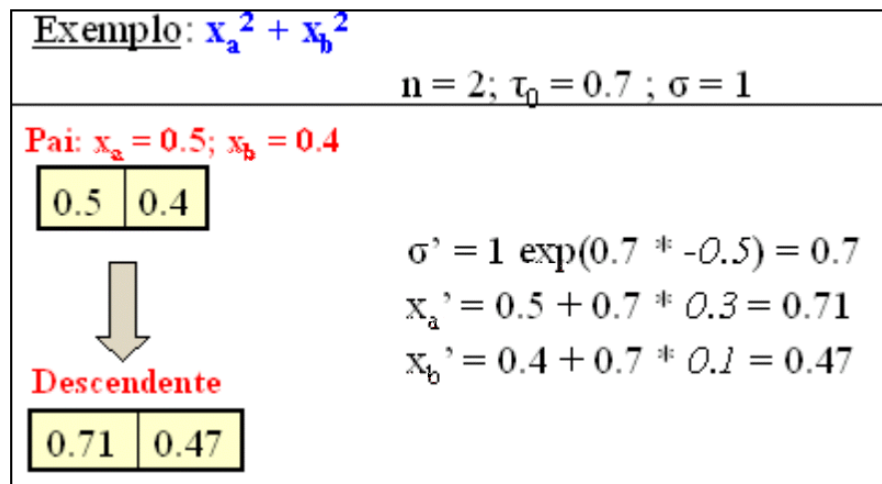


FIGURA 8 – EXEMPLO DE MUTAÇÃO COM ÚNICO σ PARA O INDIVÍDUO

- b) Cada atributo x_i do indivíduo tem seu σ_i . BÄCK, FOGEL e MICHALEWICZ (2000) expõem que as modificações ocorrem segundo:

$$\sigma_i' = \sigma_i \exp(\tau' N(0,1) + \tau N(0,1)), \tau' \propto (2n)^{-1/2} \text{ e } \tau \propto (2n^{1/2})^{-1/2}$$

$$x_i' = x_i + \sigma' N_i(0,1)$$

onde n é o número de atributos; τ' e τ são taxas de aprendizado.

Um exemplo está na Figura 9.

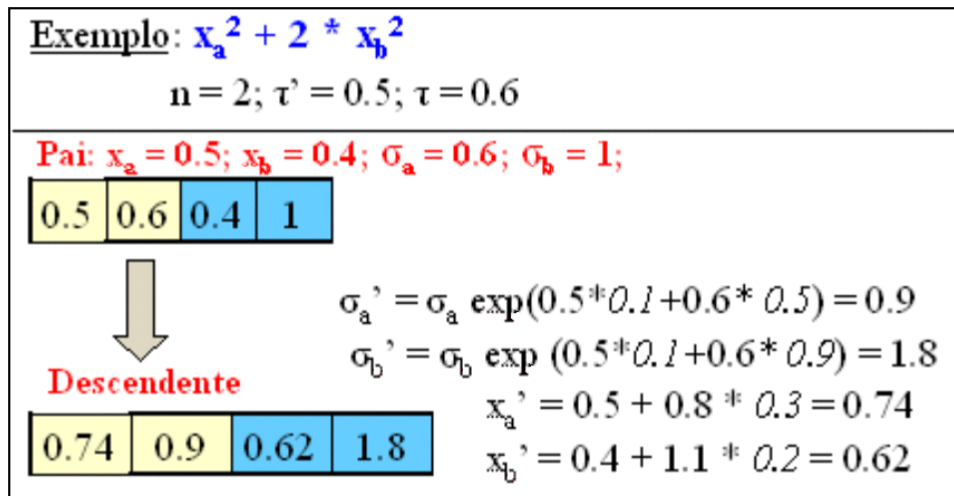


FIGURA 9 – EXEMPLO DE MUTAÇÃO COM UM VALOR DE σ PARA CADA GENE

- c) Cada atributo x_i do indivíduo tem seu σ_i e ângulos de rotação α_j descrevendo as rotações de coordenada necessárias para transformar um vetor de mutação não correlacionado em um correlacionado. A mutação é realizada, segundo BÄCK, FOGEL e MICHALEWICZ (2000), de acordo com:

$$\sigma_i' = \sigma_i \exp(\tau' N(0,1) + \tau N_i(0,1)), \tau' \propto (2n)^{-1/2} \text{ e } \tau \propto (2n^{1/2})^{-1/2}$$

$$\alpha_j' = \alpha_j + \beta N_j(0,1), \text{ sendo } \beta \approx 0.0873 (5^\circ)$$

$$x_i' = x_i + N_i(0, C(\sigma', \alpha'))$$

onde n é o número de atributos; τ' e τ são taxas de aprendizado.

Um exemplo da aplicação da mutação é mostrado na Figura 10.

Dessa maneira, observa-se que existem várias configurações que devem ser levadas em consideração na utilização de Estratégias Evolucionárias. Um exemplo de aplicação de ES na tarefa de regressão é apresentado na Seção 2.3.1.

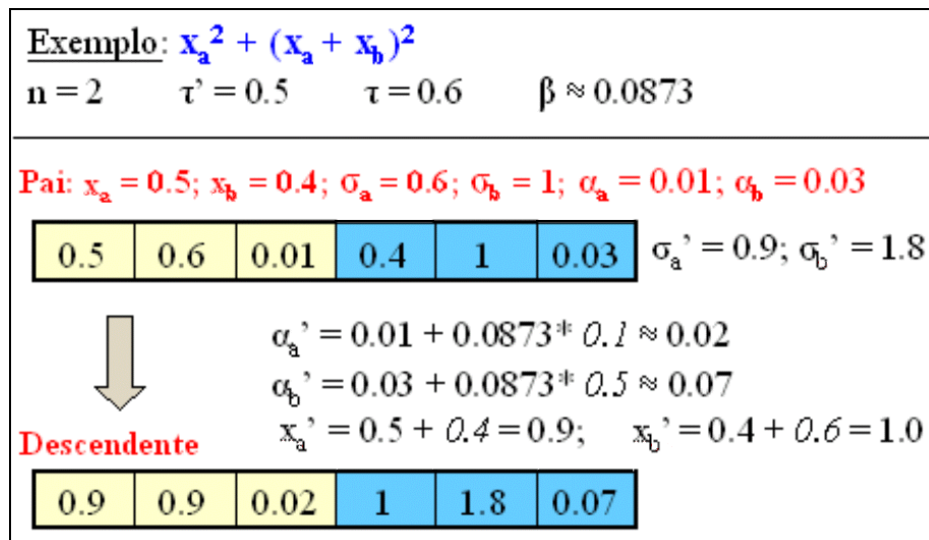


FIGURA 10 – EXEMPLO DE MUTAÇÃO COM UM VALOR DE σ E DE α PARA CADA GENE

2.3.1 Estratégias Evolucionárias na Tarefa de Regressão

Estratégias Evolucionárias podem ser utilizadas na regressão linear para a Mineração de Dados Temporais. Uma maneira de realizar essa tarefa é representar os pesos w_i da função $f(x) = (w_1x_1, \dots, w_nx_n)$ nos indivíduos da população, os quais serão evoluídos ao longo das gerações por meio de operadores de seleção e reprodução. Nessa seção, será apresentado um exemplo de como isso poderia ocorrer. Com o objetivo de simplificar o processo, relatar-se-á como o (1+1)-ES, utilizando uma determinada função de *fitness*⁶, executaria essa evolução.

O objetivo do algoritmo seria minimizar a média da distância quadrada entre y_i (real) e y_i' (estimado), representada como medida de aptidão do indivíduo (*fitness*). Abaixo segue um exemplo da execução.

Entradas:

⁶ *Fitness* é uma função que mede a aptidão dos indivíduos em técnicas de Computação Evolucionária.

a) Conjunto de dados

TABELA 1 – CONJUNTO DE DADOS EXEMPLO PARA EVOLUÇÃO DE PERCEPTRON

ID	x_1	x_2	y
1	0.2	0.3	0.5
2	0.21	0.6	0.7
3	0.6	0.7	0.3

b) $fitness = ((y_1 - y_1') + (y_2 - y_2') + (y_3 - y_3')) / 3$, onde

$$y' = w_1 x_1 + w_2 x_2, \text{ para cada linha da tabela 1.}$$

c) Desvio padrão inicial para mutação: 1

d) Aplicação da regra de sucesso 1/5 a cada 2 gerações

e) Taxa de incremento da regra de sucesso 1/5 (c_i) = 1.22

f) Taxa de decremento da regra de sucesso 1/5 (d_i) = 0.82

g) Número de gerações: 3

Execução:

a) 1ª Geração:

PAI	DESCENDENTE
Desvio padrão: $\sigma = 1$ Indivíduo pai: $X = (w_1, w_2)$ Supor $w_1 = 1$ e $w_2 = 1$. Então: $X = (1, 1)$ Para cada exemplo da tabela de entrada: $y'_1 = 0.5$ $y'_2 = 0.81$ $y'_3 = 1.3$ $fitness = 0.4107$	Supor $N = (0.5, -0.8)$ Indivíduo descendente: $X' = (1.5, 0.2)$ $y'_1 = 0.36$ $y'_2 = 0.435$ $y'_3 = 1.04$ $fitness = 0.037408$ Ainda não é o momento de aplicar a regra de sucesso 1/5
O descendente é melhor que o pai, pois possui o valor de $fitness$ menor	

b) 2ª Geração:

PAI	DESCENDENTE
Desvio padrão: $\sigma = 1$ Indivíduo pai: $X = (w_1, w_2) = (1.5, 0.2)$ Para cada exemplo da tabela de entrada: $y'_1 = 0.36$ $y'_2 = 0.435$ $y'_3 = 1.04$ $fitness = 0.037408$	Supor $N = (-0.2, 0.3)$ Indivíduo descendente: $X' = (1.3, 0.5)$ $y'_1 = 0.41$ $y'_2 = 0.573$ $y'_3 = 1.13$ $fitness = 0.125256$ Regra de sucesso 1/5: foram 2 descendentes gerados, um em cada geração, mas somente o indivíduo gerado na primeira geração era melhor que seu pai. 1 sucesso em 2 tentativas = $1/2 > 1/5$ $\sigma' = 1.22$
O pai é melhor que o descendente, pois possui o valor de <i>fitness</i> menor	

c) 3ª Geração:

PAI	DESCENDENTE
Desvio padrão: $\sigma = 1.22$ Indivíduo pai: $X = (w_1, w_2) = (1.5, 0.2)$ Para cada exemplo da tabela de entrada: $y'_1 = 0.36$ $y'_2 = 0.435$ $y'_3 = 1.04$ $fitness = 0.037408$	Supor $N = (-0.2, 1.2)$ Indivíduo descendente: $X' = (1.3, 1.4)$ $y'_1 = 0.68$ $y'_2 = 1.113$ $y'_3 = 1.76$ $fitness = 1.404936$ Ainda não é o momento de aplicar a regra de sucesso 1/5
O pai é melhor que o descendente, pois possui o valor de <i>fitness</i> menor	

d) Fim da execução:

A função que o indivíduo sobrevivente representa é $f(x) = 1.5 x_1 + 0.2 x_2$.

Sua média da distância quadrada entre y_i (real) e y_i' (desejado) é igual a 0.037408.

Saída:

$$f(x) = 1.5 x_1 + 0.2 x_2$$

As informações mostradas nessa seção são apenas a título de exemplificação do processo e, portanto, não indicam exatamente o algoritmo que será utilizado nos experimentos, já que existem outros algoritmos de ES mais elaborados, como o $(\mu + \lambda)$ -ES.

De acordo com o exposto, observa-se que a regressão linear utilizando ES permite a geração de uma equação linear geral que descreve o comportamento dos dados temporais. Certamente, como em outras técnicas, um erro de precisão de predição aparece embutido na equação obtida. Esses erros produzidos por uma única equação podem ser relativamente grandes se comparados a outras possíveis soluções (equações) específicas para alguns casos, as quais não são geradas no processo evolutivo. Uma possibilidade de encontrar essas equações específicas é por meio do uso de Boosting (SCHAPIRE, 2002), um método de aprendizado de máquina que objetiva aperfeiçoar hipóteses (nesse caso, equações/funções) fracas, gerando uma única solução mais precisa. O método de Boosting e alguns de seus algoritmos são apresentados na Seção 2.4.

2.4 BOOSTING

Estratégias Evolucionárias permitem a geração de uma equação linear que descreve o comportamento geral das variáveis predictoras em relação à variável meta para a Mineração de Dados Temporais. Sabendo-se que séries temporais podem exibir comportamentos específicos em alguns momentos, existe uma grande possibilidade de ocorrerem erros relativamente grandes em certos exemplos das bases de dados. Uma maneira de encontrar essas outras equações e aperfeiçoar os resultados obtidos é por meio de um algoritmo de Boosting (SCHAPIRE, 2002).

Inicialmente, o Boosting foi desenvolvido para o tratamento da tarefa de classificação. Portanto, a definição do método, exposta nos próximos parágrafos, fundamenta-se na classificação.

Boosting (SCHAPIRE, 2002), um método de aprendizado de máquina, é

baseado na observação de que encontrar muitas hipóteses fracas com um classificador pode ser mais fácil do que encontrar uma única hipótese de predição com alta precisão. Para aplicar a abordagem de Boosting, começa-se com um método ou algoritmo para encontrar as hipóteses fracas. O algoritmo de Boosting chama esse algoritmo de aprendizado repetidamente, cada vez alimentando-o com um diferente subconjunto dos exemplos de treinamento (ou, mais precisamente, uma diferente distribuição ou atribuição de pesos sobre os exemplos de treinamento). Cada vez que é chamado, o algoritmo de aprendizado gera uma nova hipótese de predição fraca, e depois de muitas execuções, no caso da tarefa de classificação, o algoritmo de Boosting deve combinar essas hipóteses fracas dentro de uma única hipótese de predição que, de acordo com o esperado, será mais precisa que qualquer outra gerada anteriormente (SCHAPIRE, 2002).

Um algoritmo de Boosting é o AdaBoost. Introduzido em 1995 por Freund e Schapire, o AdaBoost (FREUND; SCHAPIRE, 1997) recebe como entrada um conjunto de treinamento $(x_1, y_1), \dots, (x_m, y_m)$ onde cada x_i pertence a algum domínio ou espaço de instância X , e cada rótulo y_i está em algum conjunto de rótulo Y (FREUND; SCHAPIRE, 1999). Por simplicidade, assume-se $Y = \{-1, +1\}$, mas pode-se ter extensões para casos multiclases. AdaBoost chama um dado algoritmo de aprendizado base ou fraco repetidamente em uma série de execuções $t = 1, \dots, T$. Uma das idéias principais é manter uma distribuição ou conjunto de pesos sobre o conjunto de treinamento. O peso dessa distribuição no exemplo de treinamento i na execução t é denotado $D_t(i)$. Inicialmente, todos pesos são iguais, mas em cada execução, os pesos de exemplos classificados incorretamente são incrementados para que o algoritmo de aprendizado fraco seja forçado a focar nos exemplos difíceis do conjunto de treinamento (FREUND; SCHAPIRE, 1999). O pseudocódigo do algoritmo é fornecido na Figura 11.

O trabalho do algoritmo de aprendizado fraco é encontrar uma hipótese fraca $h_t : X \rightarrow \{-1, +1\}$ apropriada para a distribuição D_t (FREUND;

SCHAPIRE, 1999).

Após o recebimento da hipótese fraca h_t , é calculado o erro ϵ_t . Esse erro é medido com respeito à distribuição D_t , na qual o algoritmo de aprendizado fraco foi treinado:

$$\epsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i] = \sum_{i: h_t(x_i) \neq y_i} D_t(i). \quad (3)$$

Na Equação 3, se a classe obtida por meio da hipótese h_t for diferente do valor real da classe (y_i), o peso relacionado ao exemplo i é incrementado ao erro. Se $h_t(x_i) = y_i$, nada acontece.

Com o valor do erro ϵ_t , o AdaBoost calcula o valor do parâmetro α_t , conforme descrito na Figura 11. Intuitivamente, α_t mede a importância que é fornecida para h_t . Segundo FREUND e SCHAPIRE (1999), observa-se que $\alpha_t \geq 0$ se $\epsilon_t \leq \frac{1}{2}$ (assumindo não haver perda de generalidade), e que o valor de α_t aumenta quando o valor de ϵ_t diminui.

O próximo passo do algoritmo é a atualização da distribuição D_t utilizando a regra mostrada na Figura 11. O efeito produzido por essa regra é o aumento do peso de exemplos classificados de forma errada por h_t , e a diminuição do peso dos exemplos classificados corretamente. Assim, o peso tende a se concentrar em exemplos “difíceis” (FREUND; SCHAPIRE, 1999).

A hipótese final H é uma combinação das T hipóteses fracas, onde α_t é o peso de h_t (FREUND; SCHAPIRE, 1999). As hipóteses fracas que possuem um maior α_t têm um maior influência na geração de H .

A propriedade mais interessante do AdaBoost consiste na sua habilidade de reduzir o erro de treinamento, isto é, a fração de erros no conjunto de treinamento. A qualidade de uma hipótese fraca é, portanto, medida por seu erro (SCHAPIRE, 2002).

Dados: $(x_1, y_1), \dots, (x_m, y_m)$ onde $x_i \in X, y_i \in Y = \{-1, +1\}$

Inicialize $D_1(i) = 1/m$.

Para $t = 1, \dots, T$:

- Treine o algoritmo de aprendizado fraco utilizando a distribuição D_t .

- Receba a hipótese fraca $h_t : X \rightarrow \{-1, +1\}$ com erro

$$\epsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i].$$

- Selecione

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right).$$

- Atualize:

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{se } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{se } h_t(x_i) \neq y_i \end{cases} \\ &= \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \end{aligned}$$

onde Z_t é um fator de normalização (escolhido tal que D_t seja uma distribuição).

Forneça a hipótese final (onde *sign* é o sinal):

$$H(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right).$$

FONTE: TRADUÇÃO DE FREUND E SCHAPIRE (1999).

FIGURA 11 – PSEUDOCÓDIGO DO ADABOOST

Muitas variações do AdaBoost podem ser encontradas na literatura (FREUND; SCHAPIRE, 1997; SOLOMATINE; SHRESTHA, 2004), de acordo com o problema estudado. Todas as variações, no entanto, têm como base o algoritmo original apresentado na Figura 11.

O algoritmo exposto (Figura 11) é específico para a tarefa de classificação binária. Portanto, para trabalhar com a tarefa de regressão faz-se necessário o estudo de uma variação do AdaBoost. Esse assunto é tratado na próxima subseção.

2.4.1 AdaBoost e Regressão

Inicialmente, o algoritmo de Boosting foi desenvolvido para problemas binários de classificação. Posteriormente, FREUND e SCHAPIRE (1997) criaram as extensões AdaBoost.M1 e AdaBoost.M2, para casos multiclases, e AdaBoost.R para problemas de regressão.

Na regressão, ao invés de calcular o erro ϵ_t (Equação 3), utiliza-se uma função de perda L_t , a qual determina, para cada exemplo de treinamento, um valor associado ao erro de previsão. O tipo de fórmula (linear, quadrática, exponencial) utilizada para o cálculo do L_t pode variar, de acordo com o modelo do algoritmo. Outras variações que ocorrem na regressão são a função de atualização dos pesos D_t e a forma de calcular a hipótese final.

O AdaBoost.R (FREUND; SCHAPIRE, 1997) resolve problemas de regressão reduzindo-os a problemas de classificação. Segundo SOLOMATINE e SHRESTHA (2004), embora experimentos mostrem que o AdaBoost.R pode ser eficaz projetando os dados de regressão em classificação, existem duas desvantagens. A primeira é que o algoritmo expande cada exemplo na amostra de regressão em muitos exemplos de classificação. A segunda é que no AdaBoost.R a função de perda muda de iteração para iteração do Boosting e até mesmo entre exemplos na mesma iteração.

Para não necessitar reduzir o problema de regressão em um problema de classificação, DRUCKER (1997) desenvolveu o AdaBoost.R2, o qual é uma modificação *ad hoc* do AdaBoost.R. Seus experimentos com problemas de regressão indicaram resultados promissores. O algoritmo da Figura 12, adaptado de SOLOMATINE e SHRESTHA (2004), representa o AdaBoost.R2.

A primeira diferença em relação ao AdaBoost da Figura 11 são os dados de entrada. Observa-se que, ao invés de uma classe discreta (-1 ou +1), os valores do atributo meta (y) pertencem ao conjunto dos números reais. Quanto aos atributos previsores, o importante é que sejam numéricos.

Inicialmente, a distribuição de pesos D_t é igual para todos os exemplos. Esses

1. Entrada:

- Sequência de m exemplos $(x_1, y_1), \dots, (x_m, y_m)$ onde $y \in \mathbb{R}$
- Algoritmo de aprendizado fraco, denominado *WeakLearner*

2. Inicialize:

- Iteração $t = 1$
- Distribuição $D_t(i) = 1/m$ para todo i
- Função de perda comum $\bar{L}_t = 0$

3. Execute: enquanto a função de perda comum $\bar{L}_t < 0.5$

- Chame *WeakLearner*, proporcionando-o a distribuição D_t
- Construa o modelo de regressão: $f_t(x) \rightarrow y$
- Calcule a perda para cada exemplo de treinamento: $l_t(i) = |f_t(x_i) - y_i|$
- Calcule a função de perda $L_t(i)$ para cada exemplo de treinamento utilizando três fórmulas funcionais diferentes:

$$\text{Linear: } L_t(i) = l_t(i)/\text{Den}_t$$

$$\text{Quadrática: } L_t(i) = (l_t(i)/\text{Den}_t) (l_t(i)/\text{Den}_t)$$

$$\text{Exponencial: } L_t(i) = 1 - \exp(-l_t(i)/\text{Den}_t)$$

onde $\text{Den}_t = \max(l_t(i)), i=1 \dots m$

- Calcule a perda comum $\bar{L}_t = \sum_{i=1}^m (L_t(i) D_t(i))$

- Deixe $\beta_t = \bar{L}_t / (1 - \bar{L}_t)$

- Atualize a distribuição D_t :

$$D_{t+1}(i) = \frac{D_t(i) \beta_t^{(1-L_t(i))}}{Z_t}$$

onde Z_t é um fator de normalização escolhido tal que D_{t+1} será uma distribuição

- $t = t + 1$

4. Produza a hipótese final:

$$f_{fin}(x) = \inf \left[y \in Y : \sum_{t: f_t(x) \leq y} \log(1/\beta_t) \geq \frac{1}{2} \sum_t \log(1/\beta_t) \right]$$

FONTE: BASEADO EM SOLOMATINE E SHRESTHA (2004).

FIGURA 12 – PSEUDOCÓDIGO DO ADABOOST.R2

valores vão mudando ao longo das iterações do AdaBoost.R2, de acordo com a equação:

$$D_{t+1}(i) = \frac{D_t(i) \beta_t^{(1-L_t(i))}}{Z_t} \quad (4)$$

Na Equação 4, $D_t(i)$ é o valor do peso do exemplo i na atual iteração (t), Z_t é um fator de normalização escolhido tal que D_{t+1} seja uma distribuição, $L_t(i)$ representa o valor da função de perda para o exemplo i , e β_t é a taxa de confiança calculada com hipótese fraca gerada na execução t .

A função de perda $L_t(i)$, para cada exemplo, é obtida por meio de uma função linear (Equação 5), uma função quadrática (Equação 6) ou uma função exponencial (Equação 7). Em cada função, para cada exemplo, é calculado o erro $l_t(i)$ entre o valor predito $f_t(x_i)$ pela hipótese gerada pelo algoritmo de aprendizado fraco na iteração t e entre o valor real y_i do atributo meta, em módulo.

$$L_t(i) = |f_t(x_i) - y_i| / (\max (|f_t(x_i) - y_i|)) \quad (5)$$

$$L_t(i) = (|f_t(x_i) - y_i| / \max (|f_t(x_i) - y_i|)) (|f_t(x_i) - y_i| / \max (|f_t(x_i) - y_i|)) \quad (6)$$

$$L_t(i) = 1 - \exp(-|f_t(x_i) - y_i| / (\max (|f_t(x_i) - y_i|))) \quad (7)$$

No caso da função linear, esse erro (ou perda) é dividido pelo maior erro $l_t(i)$ encontrado. Na quadrática, o quadrado do erro é dividido pelo quadrado do maior erro encontrado. Já função de perda $L_t(i)$ exponencial, como o nome diz, utiliza uma equação exponencial para os cálculos.

A partir do momento em que a função de perda está calculada, pode se obter a equação de perda comum \bar{L}_t , que é a soma de cada uma das funções de perda multiplicada pelo peso do exemplo i :

$$\bar{L}_t = \sum_{i=1}^m (L_t(i) D_t(i)) \quad (8)$$

Com o valor da função de perda comum \bar{L}_t obtém-se a taxa de confiança β_t :

$$\beta_t = \bar{L}_t / (1 - \bar{L}_t) \quad (9)$$

Atualizados os pesos $D_t(i)$, o algoritmo de aprendizado fraco novamente é chamado. A hipótese fraca será gerada com base na nova distribuição de pesos, tentando privilegiar os exemplos com maior peso, ou seja, os que tiveram piores erros na iteração anterior.

A convergência para um erro de treinamento igual a zero no AdaBoost.R2 depende de obter uma máquina de aprendizado tal que a função de perda comum $\bar{L}_t \geq 0.5$. Assim sendo, essa é a condição de parada do algoritmo. No entanto, em dados reais e com máquinas de aprendizado reais, como o número de exemplos difíceis de treinamento aumenta na base de dados, fica mais difícil de alcançar esse limite (DRUCKER, 1997).

A hipótese final do AdaBoost.R2 é gerada por meio da média geométrica de todos os $f_t(x)$ (hipóteses fracas), ponderada por suas respectivas taxas de confiança β_t . Fornecido um valor de entrada x , é calculada cada uma das possíveis saídas com cada $f_t(x)$. Os valores de $f_t(x)$ são colocados em ordem crescente. Calcula-se a média geométrica por meio da equação:

$$\frac{1}{2} \sum_t (\log(1/\beta_t)) \quad (10)$$

Então, somam-se os termos $\log(1/\beta_t)$ na mesma ordenação de f_t até que o valor da soma seja maior ou igual ao valor da média geométrica. A saída será o valor

do $f_t(x)$ correspondente ao β_t da última soma.

O AdaBoost.R2, pelas suas características, pode melhorar resultados de ES na tarefa de regressão de dados temporais. Este assunto é tratado na Seção 3. Antes, porém, faz-se necessário um estudo de trabalhos existentes na literatura, os quais são apresentados na Seção 2.5.

2.5 TRABALHOS NA LITERATURA

Nessa seção, são discutidos alguns trabalhos relacionados, bem como alguns conceitos que autores utilizaram e que podem ser úteis no desenvolvimento do presente trabalho (Seção 3).

Na Mineração de Dados Temporais (Seção 2.1) podem ser encontradas várias pesquisas, para diferentes problemas. A maioria deles trata da tarefa de classificação (QUINLAN, 1993) e, portanto, serão os primeiros a receber comentários, uma vez que trazem conceitos importantes como, por exemplo, de função de caracterização de eventos (POVINELLI, 2000).

Certas implementações de Mineração de Dados Temporais exigem a apresentação de uma função de caracterização de evento, a qual objetiva conectar um padrão temporal (passado e presente) com um evento (futuro). Dessa maneira pode-se definir quantos passos adiante deve-se prever. Um exemplo é a tarefa de predição de ocorrência de um evento X numa base cujos dados são coletados mensalmente. Pode-se almejar descobrir o valor de um atributo meta em 1, 2, ... n meses no futuro.

No trabalho apresentado por POVINELLI (2000), uma função de caracterização de evento é $g(t) = x_{t+1}$, o qual captura a meta de caracterizar terremotos sintéticos um passo no futuro. Alternativamente, predizer um evento três passos adiante requer uma função de caracterização de evento como $g(t) = x_{t+3}$.

Para encontrar um modelo $f(x)$ na Mineração de Dados Temporais, faz-se necessário determinar quanto tempo (dias/semanas/meses) no futuro deseja-se prever. Assim, uma função de caracterização de evento, como em POVINELLI (2000), pode

ser útil quando trabalha-se com dados temporais.

Associando a tarefa de classificação na Mineração de Dados Temporais (Seção 2.1) com um algoritmo de Computação Evolucionária (Seção 2.3), WEISS e HIRSH (1998) desenvolveram uma técnica de aprendizado supervisionado para prever eventos raros em seqüências. O sistema de aprendizado de máquina, "timeweaver", utiliza um conjunto de seqüências de eventos para treinar um minerador baseado em Algoritmos Genéticos (HOLLAND, 1992). Nesse experimento, observa-se que métodos de classificação não temporais, ou seja, que são habitualmente utilizados em bases de dados não temporais, podem tranqüilamente ser utilizados na descoberta de padrões temporais. Os resultados promissores obtidos encorajam a utilizar também outras técnicas da área de Computação Evolucionária em outros experimentos.

Seguindo essa linha de investigação de métodos de classificação não temporais, DIETTERICH e MICHALSKI (1985) desenvolveram uma regra de geração de seqüências que caracterizasse eventos observados e pudesse ser utilizada para gerar próximos eventos possíveis e consistentes.

A Mineração de Dados Temporais, além da classificação, trabalha com outros tipos de problemas, como a regressão (Seção 2.2). Dentre as técnicas de Inteligência Artificial, Redes Neurais Artificiais⁷ (ANN) (FOG, 1995; LACHTERMACHER; FULLER, 1995; SARLE, 1995) têm sido bastante utilizadas para manipular seqüências temporais desse tipo.

⁷ Redes neurais artificiais (ANN – *Artificial Neural Networks*) (FOG, 1995; LACHTERMACHER; FULLER, 1995; SARLE, 1995) são modelos que incluem um número de elementos não lineares, os neurônios, trabalhando em paralelo e organizados em camadas tais como seus sócios biológicos. Eles podem aprender certos conhecimentos pela experiência (HAYKIN, 1999; EVANS; ALVIN, 1991; SIQUEIRA; SOARES FILHO, 2002). As ANNs podem ser de dois tipos: feedforward ou redes neurais recorrentes (CHIANG; CHANG; CHANG, 2004). As redes neurais sem feedback são estáticas, isto é, uma certa entrada somente produz um conjunto de saídas sem capacidade de memória. Redes neurais recorrentes são capazes de memorizar informações temporais.

Várias abordagens podem ser encontradas para o projeto de sistemas neurais no contexto temporal (BARRETO; ARAÚJO, 2001; PEARLMUTTER, 1995). Um exemplo está em RAMÍREZ, VELHO e FERREIRA (2004), onde uma rede neural *feedforward* (KADOUS, 1998) e um algoritmo de aprendizado de propagação são usados para construir um mapeamento não linear entre dados de um Modelo ETA de chuva regional no Centro para Previsões do Tempo e Estudos Climáticos/Instituto Nacional para Pesquisa Espacial (Brasil), e dados de chuva da superfície para a região do Estado de São Paulo. O objetivo foi gerar previsões quantitativas de chuva diária específicas do local. Variáveis meteorológicas do modelo ETA (umidade específica, temperatura do ar e outras) são utilizadas como dados de entrada para as redes treinadas, que geram previsão de chuva para o próximo período.

Quanto à Computação Evolucionária, observa-se que o uso de suas técnicas na tarefa de predizer uma função $f(x)$ não é algo novo, mesmo em modelagens que não tratam valores contínuos. Por exemplo, MINERVA e PATERLINI (2002) empregaram Algoritmos Genéticos para determinar as variáveis que devem ser incluídas num modelo $f(x)$. Cada gene do cromossomo representava uma constante binária $\in [0, 1]$ que deveria ser multiplicada por cada atributo x_i de $f(x)$. Os genes eram posicionais, ou seja, o primeiro gene representava a constante que deveria ser multiplicada por x_1 , o segundo gene representava a constante que deve ser multiplicada por x_2 , e assim por diante. Dessa maneira, se a constante fosse 1, a variável seria incluída no modelo, caso contrário, seria excluída. No entanto, o trabalho apresentado somente demonstrava as variáveis que seriam incluídas no modelo, sendo necessário o uso de outro modelo para determinar os pesos de cada variável, como, por exemplo, uma rede neural artificial para o caso de aproximação não linear. Por outro lado, se for uma aproximação linear, uma alternativa, dentro da Computação Evolucionária, seria Estratégia Evolucionária.

Uma determinada técnica, às vezes, não apresenta resultados satisfatórios para o problema, necessitando do uso de um algoritmo que melhore seu desempenho

como, por exemplo, o algoritmo de Boosting (Seção 2.4). Saindo do contexto temporal, mas continuando com a Computação Evolucionária, têm-se os trabalhos de LIU, MCKAY e ABBASS (2003) e de PARIS, ROBILLIARD e FONLUPT (2001) que utilizam Boosting.

LIU, MCKAY e ABBASS (2003) aplicaram Boosting em Algoritmos Genéticos (HOLLAND, 1992) para a descoberta de regras de classificação. O método foi testado em bases de dados públicas e, comparado com o Algoritmo Genético original, demonstrou ser capaz de classificar o conjunto de teste com maior precisão e menor avaliação de *fitness*. O algoritmo GAbost pode ser visualizado na Figura 13.

O GAbost recebe como entrada o conjunto de treinamento S (onde podem existir k classes, identificadas por números inteiros) e o número de iterações T do Boosting. Os valores dos pesos $w_1(i)$ das instâncias, inicialmente, são iguais para todos os registros.

O algoritmo GAbost é executado T vezes. Em cada iteração, uma variável, nomeada $D_t(i)$, recebe os valores dos pesos $w_1(i)$ normalizados. O Algoritmo Genético (GA – *Genetic Algorithm*) é executado, utilizando a distribuição $D_t(i)$ no conjunto de treinamento S. Sua função de *fitness* é expressa por:

$$fit = \frac{\sum_{k \in TPSet} D_t(k) + \sum_{k \in TNSet} D_t(k)}{\sum_{k \in S} D_t(k)} \quad (11)$$

onde

- S é o conjunto de treinamento;
- $D_t(k)$ é o peso da instância;
- TPSet inclui instâncias cobertas pela regra cujos valores da classe meta são preditos pela regra;
- TNSet inclui instâncias não cobertas pela regra cujos valores da classe

meta diferem da predição da regra.

Entradas: Um conjunto de treinamento $S = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$; $x_i \in X$; $y_i \in Y = \{1, 2, \dots, k\}$

GA(S,D): Um algoritmo de GA para classificação usando uma distribuição D em S;

T: execuções de boosting

Inicialize o peso da instância (x_i, y_i) : $w_t(i) = 1/m$ para $i = 1, \dots, m$.

Para $t = 1, 2, \dots, T$

$$D_t(i) = \frac{w_t(i)}{\sum_{i=1}^m w_t(i)}$$

Execute GA(S, D_t)

Receba um conjunto de regras denominadas como R_t : $X \rightarrow Y$

Calcule o erro de R_t : $\epsilon(R_t) = \sum_{i=1}^m D_t(i) \mathbb{1} \{R_t(x_i) \neq y_i\}$

Se $\epsilon(R_t) > 1/2$ e $k > 2$ então $T = t - 1$ e termine o laço

Senão

Deixe $\beta_t = \epsilon(R_t) / (1 - \epsilon(R_t))$

Atualize o peso de cada instância: $w_{t+1}(i) = w_t(i) \beta_t^{\mathbb{1} \{R_t(x_i) \neq y_i\}}$

Fim se

Fim para

Forneça a classe da instância x_k

$$R_f(x_k) = \arg \max_{c \in Y} \sum_{t=1}^T \log(1 / \beta_t) \mathbb{1} \{R_t(x_k) = c\}$$

FONTE: LIU, MCKAY e ABBASS (2003).

FIGURA 13 – PSEUDOCÓDIGO DO GABOOST

Terminada a execução do GA, para cada exemplo de treinamento, se a classe obtida com as regras $R_t(i)$ geradas pelo Algoritmo Genético for diferente da classe real y_i , o valor do $D_t(i)$ é acrescentado ao erro $\epsilon(R_t)$. Se $\epsilon(R_t)$ for maior que $1/2$ e o número de classes for maior que 2, a iteração é interrompida. Caso contrário, o valor de β_t é calculado e utilizado na geração dos valores dos pesos w para a próxima iteração.

Baseando-se nas regras obtidas e no valor de β_t pode ser gerada a classe para um exemplo de entrada fornecido.

PARIS, ROBILLIARD e FONLUPT (2001) propuseram um algoritmo chamado GPboost que utiliza Programação Genética (KOZA, 1992) para manipular problemas de regressão. O GPboost mantém-se próximo à idéia original de distribuição do AdaBoost e pode ser visto como um algoritmo de Boosting modelo, uma vez que permite ao usuário a mudança da função de *fitness*.

A Figura 14, traduzida de PARIS, ROBILLIARD e FONLUPT (2001), demonstra o algoritmo GPboost. O GP(S,D) é um algoritmo de Programação Genética (GP – *Genetic Programming*) que utiliza uma distribuição D em um conjunto de treinamento S. O autor fez uma multiplicação na função de *fitness* para obter valores dentro dos mesmos limites tanto para a Programação Genética padrão quanto para a Programação Genética com Boosting.

Inicialmente, os pesos $D_t(i)$ dos exemplos são iguais. Depois que o algoritmo de Programação Genética é chamado, de acordo com a função retornada, é calculada a perda para cada exemplo. Então, é obtida a média de perda multiplicando cada valor de perda por seu peso D_t . A média de perda é utilizada no cálculo da taxa de confiança β_t da função. Os valores dos pesos $D_{t+1}(i)$ são atualizados para a próxima iteração. Esse processo se repete até atingir o valor T de iterações. Depois disso, pode-se gerar a hipótese final.

A geração da hipótese final é realizada por meio da média geométrica de $f_i(x)$, ponderada por seus respectivos coeficientes de confiança. Por exemplo, dada a Tabela 2 de hipóteses e taxas de confiança obtidas em uma execução do GPboost, para $x = 0.5$ tem-se que $f_2(x) = 0.041667 \leq f_1(x) = 0.0625 \leq f_3(x) = 0.0625 \leq f_4(x) = 0.116305 \leq f_5(x) = 0.166667$. A média geométrica é 2.907245. Então, adiciona-se os termos $\log(1/\beta_t)$ na mesma ordenação de f_t :

$$\log(1/\beta_2) = 1.60544$$

$$1.60544 + \log(1/\beta_1) = 2.77862$$

Dados: um conjunto de aprendizado $S = \{(x_1, y_1), \dots, (x_m, y_m)\}$; $x_i \in X, y_i \in \mathbb{R}$

GP(S,D) um algoritmo de GP usando uma distribuição D em S.

Algoritmo GPboost

Deixe D_1 ser a distribuição para a iteração $t = 1$

$D_1(i)$ é o peso do exemplo (x_i, y_i)

Inicialize $D_1(i) := 1/m$ para todos $(x_i, y_i) \in S$

Para $t = 1..T$ **faça**

- Execute GP em D_t com a função de *fitness*:

$$fit = \sum_{i=1}^m (|f(x_i) - y_i| D_t(i)) / m$$

onde f é uma função da população do GP

- A melhor função da execução é denotada f_t

- Compute a perda para cada exemplo:

$$L_i = \frac{|f(x_i) - y_i|}{\max_{i=1..m} |f(x_i) - y_i|}$$

- Compute a média de perda:

$$\bar{L} = \sum_{i=1}^m L_i D_i$$

- Deixe

$$\beta_t = \frac{\bar{L}}{1 - \bar{L}}$$

ser a taxa de confiança dada à função f_t

- Atualize a distribuição:

$$D_{t+1}(i) = \frac{D_t(i)^{1 - L_i}}{Z_t}$$

com Z_t um fator de normalização tal que D_{t+1} seja uma distribuição

Fim para

Saída: Hipótese final:

$$F(x) = \min \{ y \in \mathbb{R} : \sum_{t: f_t(x) \leq y} \log(1/\beta_t) \geq 1/2 \sum_{t=1}^m \log(1/\beta_t) \}$$

FONTE: TRADUÇÃO DE PARIS, ROBILLIARD E FONLUPT (2001).

FIGURA 14 – PSEUDOCÓDIGO DO GPBOOST

$$2.77862 + \log(1/\beta_3) = 3.660312 \geq 2.907245$$

Como o terceiro valor é maior que a média geométrica, tem-se:

$$F(0.5) = f_3(0.5) = 0.0625$$

TABELA 2 – HIPÓTESES E TAXAS DE CONFIANÇA DE UM EXEMPLO DE EXECUÇÃO DO GPBOOST

HIPÓTESE	TAXA DE CONFIANÇA
$f_1(x) = x^4$	0.309380
$f_2(x) = x^4 / (1 + x)$	0.200801
$f_3(x) = x^4$	0.414084
$f_4(x) = x^4 (x + 2) / (3x^3 - x + 1) (x + 1)$	0.730217
$f_5(x) = x^2 / (1 + x)$	0.158855

FONTE: PARIS, ROBILLIARD E FONLUPT (2001).

O GPboost (PARIS; ROBILLIARD; FONLUPT, 2001) obtém funções não lineares de um par (x_i, y_i) , onde x_i representa somente um atributo x , ao invés de um vetor de n posições, de acordo com o número de atributos da base de dados. Mas o GPboost, como trabalha com a tarefa de regressão e com o Boosting, possui uma função de *fitness* que pode ser utilizada também em ES, no caso de se almejar aperfeiçoar funções.

BONÉ, ASSAD e CRUCIANU (2003) adaptaram um algoritmo de Boosting para o problema de prever valores futuros de séries temporais de valores reais, usando Redes Neurais Recorrentes (RNN – *Recurrent Neural Networks*) (CHIANG; CHANG; CHANG, 2004). Os experimentos mostraram que o Boosting realmente melhora os resultados e que a média geométrica é melhor para combinar as funções aprendidas do que a média obtida pela função $h_f(x) = \sum_{t=1}^T [\log \alpha_t h_t(x)]$. Porém, conforme citado, trabalham com Redes Neurais, e seria interessante observar o comportamento de uma técnica de EC nesse contexto.

Estratégias Evolucionárias podem realizar a regressão linear de séries temporais para prever valores futuros. Um algoritmo de Boosting pode auxiliar no aperfeiçoamento das funções geradas. A próxima seção trata desse assunto.

3 BOOSTING EM ESTRATÉGIAS EVOLUCIONÁRIAS PARA A TAREFA DE REGRESSÃO

O objetivo dessa dissertação é realizar a tarefa de regressão para a Mineração de Dados Temporais, com n variáveis independentes (previsoras) e uma variável dependente (meta).

Quando se trabalha com descoberta de padrões em dados temporais numéricos, suspeita-se que exista uma determinada tendência nos valores dos dados, a qual pode ser obtida por meio de funções lineares utilizando Estratégias Evolucionárias. No entanto, modelos lineares são relativamente simples e podem abranger somente alguns registros da base de dados, registros esses que, embora com bastante frequência, não são os mais significativos para a tarefa. Assim, faz-se necessário o uso de alguma técnica que melhore o resultado gerado, visando modelos mais complexos.

O problema de gerar funções complexas utilizando como base funções lineares pode ser solucionado por meio do uso de uma técnica de Boosting. Esse método de aprendizado de máquina tem a capacidade de combinar n funções fracas (que descrevem apenas uma pequena e não significativa parte do conjunto de dados) para gerar uma função mais complexa.

Observando os trabalhos com Boosting e técnicas de EC apresentados na Seção 2.5, têm-se o GAbost (LIU; MCKAY; ABBASS, 2003), que combina Algoritmos Genéticos com Boosting e o GPboost (PARIS; ROBILLIARD; FONLUPT, 2001), que utiliza Programação Genética. Nesse contexto, denominar-se-á ESboost o algoritmo que combina ES com Boosting.

O GPboost, apesar de trabalhar com Programação Genética (e não com ES), trata a regressão e, dessa maneira, sua função de *fitness* pode ser utilizada também pelo ESboost.

A função de *fitness* é de grande importância no GPboost (e, portanto, no

ESboost) pois é a responsável pela criação das hipóteses fracas, levando em consideração o erro da hipótese e o seu peso $D_t(i)$. Isso permite que o objetivo do Boosting seja atingido, ou seja, que os exemplos difíceis sejam focados.

O ESboost é baseado no algoritmo do AdaBoost.R2 (Figura 12), o qual é parecido com o GPboost (Figura 14). O algoritmo recebe uma base de dados temporal e alguns dados de entrada adicionais (configuração de parâmetros) e gera funções que descrevem os exemplos da base de dados (Figura 15).

Tratando-se de dados temporais, o y do x_i pode ser y_{i+p} , ao invés de y_i . O valor de p (quantos passos à frente deseja-se trabalhar) deve estar implícito na configuração da base de dados, de maneira que cada registro seja constituído por seus atributos previsores e pelo atributo meta.

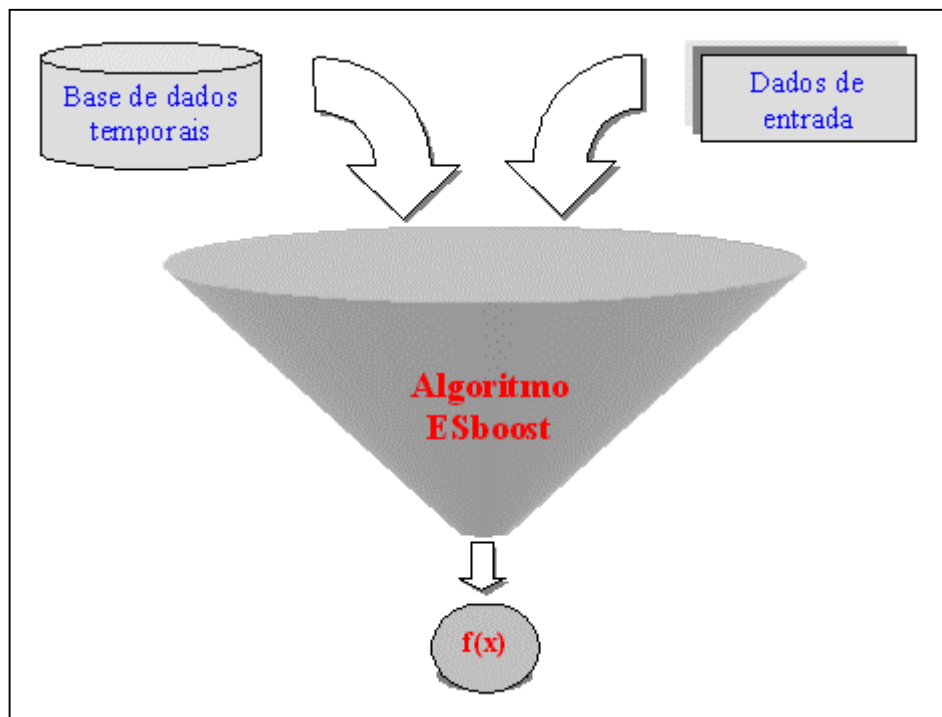


FIGURA 15 – ENTRADA E SAÍDA DO ESBOOST

As diferenças que o ESboost possui em relação ao GPboost são:

- cada x_i de um par (x_i, y_i) representa um vetor de n posições, de acordo

- com o número de atributos da base de dados temporal;
- o $f(x)$, ou hipótese fraca, é uma função linear obtida de um indivíduo da população de ES;
- é utilizado o AdaBoost.R2, com opção de uso das funções quadrática e exponencial, além da função linear. O algoritmo recebe a especificação do valor de Z_t como a soma dos pesos de todos os exemplos.

Conforme citado, o presente algoritmo (Figura 16) é constituído pelos algoritmos de AdaBoost.R2 e de ES. O AdaBoost.R2 é responsável pela distribuição dos pesos entre os exemplos de treinamento e pela chamada do ES. O ES fornece ao AdaBoost.R2 uma função (hipótese), denominada “função/hipótese fraca”. Essa função recebe um valor que indica a sua importância (β_t), de acordo com a sua taxa de erro. O AdaBoost.R2 faz novamente a distribuição dos pesos no conjunto de treinamento, dando um maior peso aos exemplos classificados incorretamente (maiores detalhes na Seção 2.4). Novamente o ES é executado, tentando privilegiar os exemplos com maiores pesos (Figura 17).

O algoritmo AdaBoost.R2 pára quando a função \bar{L}_t atinge um valor maior ou igual a 0.5, o que indica que o valor de β_t na última iteração foi maior ou igual a 1. Em dados do mundo real, como o número de exemplos difíceis aumenta no conjunto de treinamento, torna-se difícil a função \bar{L}_t atingir o valor 0.5. Por esse motivo, além da condição de parada do AdaBoost.R2, foi acrescentada mais uma condição de parada no ESboot, condição esta que é utilizada pelo AdaBoost da Figura 11 (FREUND; SCHAPIRE, 1999). Então, o ES é executado até que \bar{L}_t alcance esse limite, ou até que o número de iterações t atinja o valor T de entrada fornecido ao ESboot. Se o algoritmo não consegue o valor 0.5 para a variável \bar{L}_t , o algoritmo encerrará no número máximo de iterações T configurado.

No ES, o número de indivíduos na população, a seleção de componentes para a constituição da geração, os parâmetros de estratégia, o número de gerações, se o

algoritmo realiza recombinação, e demais configurações são decididos no momento da experimentação, ou seja, não existe uma configuração fixa para esses parâmetros.

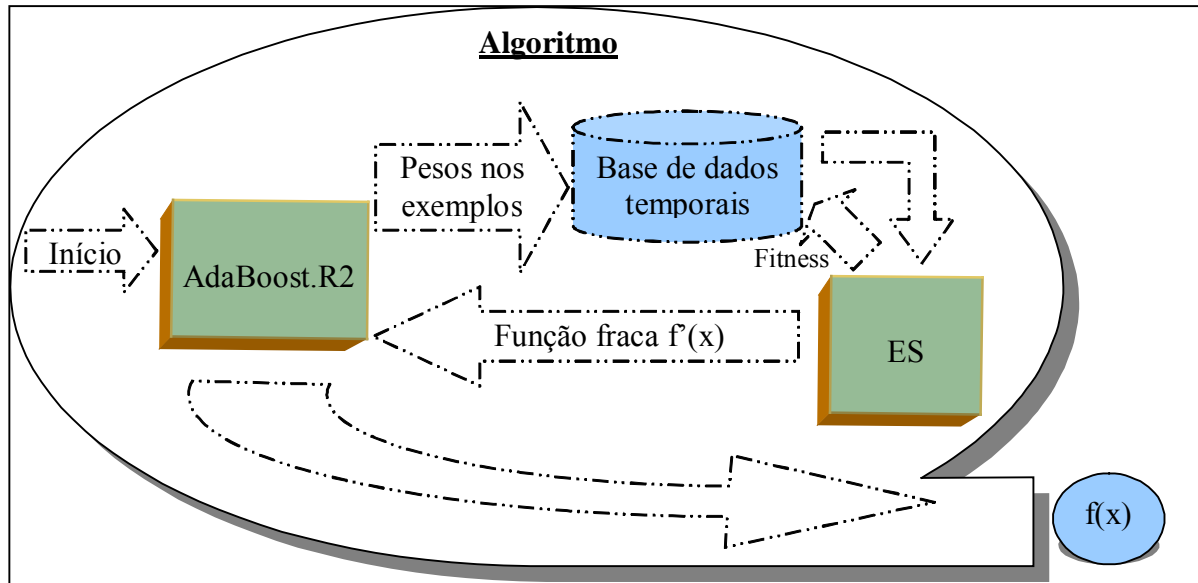


FIGURA 16 – REPRESENTAÇÃO DO ALGORITMO ESBOOST

A execução do projeto foi realizada em diversas etapas. A primeira delas foi a etapa de implementação. Para tanto, utilizou-se o código das Estratégias Evolucionárias presente na *Evolving Objects Library* (biblioteca EO), a qual está escrita em C++. Essa biblioteca (EO LIBRARY, 2005), com o acréscimo de algumas funções como, por exemplo, abertura e manipulação de arquivos de entrada e saída, gerou o programa de ES para a regressão linear. Posteriormente, o código do AdaBoost.R2 foi incorporado, gerando o ESboost (Figura 17). O trabalho foi desenvolvido no ambiente Linux.

Na segunda etapa foram realizados testes com o código obtido objetivando assegurar que o ESboot foi implementado corretamente. Depois disso, partiu-se para a etapa de experimentação, descrita na próxima seção.

1. Entrada:

- Sequência de m exemplos $(x_1, y_1), \dots, (x_m, y_m)$ onde $y \in \mathbb{R}$
- Valor de T (número máximo de execuções do Boosting)

2. Inicialize:

- Iteração $t = 1$
- Distribuição $D_t(i) = 1/m$ para todo i
- Função de perda comum $\bar{L}_t = 0$

3. Execute: enquanto a função de perda comum $\bar{L}_t < 0.5$ e $t < T$

- Execute ES em D_t com a função de *fitness*: $fit = \sum_{i=1}^m (|f(x_i) - y_i| D_t(i)) m$

onde

$f(x_i)$ = valor da função obtida com a execução de ES no exemplo i

- Construa o modelo de regressão: $f_t(x) \rightarrow y$
- Calcule a perda para cada exemplo de treinamento: $l_t(i) = |f_t(x_i) - y_i|$
- Calcule a função de perda $L_t(i)$ para cada exemplo de treinamento utilizando uma das três fórmulas funcionais:
 - Linear: $L_t(i) = l_t(i)/Den_t$
 - Quadrática: $L_t(i) = (l_t(i)/Den_t) (l_t(i)/Den_t)$
 - Exponencial: $L_t(i) = 1 - \exp(-l_t(i)/Den_t)$

onde $Den_t = \max (l_t(i)), i=1 \dots m$

- Calcule a perda comum $\bar{L}_t = \sum_{i=1}^m (L_t(i) D_t(i))$

- Deixe $\beta_t = \bar{L}_t / (1 - \bar{L}_t)$

- Atualize a distribuição D_t :

$$D_{t+1}(i) = \frac{D_t(i) \beta_t^{(1-L_t(i))}}{Z_t}$$

onde Z_t é um fator de normalização escolhido tal que D_{t+1} será uma distribuição

- $t = t + 1$

4. Produza a hipótese final:

$$f_{fin}(x) = \inf \left[y \in Y : \sum_{t: f_t(x) \leq y} \log(1/\beta_t) \geq \frac{1}{2} \sum_t \log(1/\beta_t) \right]$$

FIGURA 17 – PSEUDOCÓDIGO DO ESBOOST

4 EXPERIMENTOS

Alguns experimentos foram realizados no algoritmo de Estratégias Evolucionárias (ES) sem o uso do Boosting e no algoritmo ESboost. A primeira decisão que foi tomada nessa fase diz respeito às configurações de ES⁸. Optou-se pelo (10+50)-ES. No (10+50)-ES, 10 indivíduos, denominados pais, geram 50 filhos, os quais competem com seus pais na criação da próxima geração (Figura 6). As Estratégias Evolucionárias do ESboost tiveram os mesmos parâmetros de entrada que o (10+50)-ES padrão, objetivando a comparação de desempenho entre os algoritmos. O ESboost foi executado com cada uma de suas funções de perda: linear, quadrática e exponencial (Figura 17). Na Tabela 3 são apresentados os parâmetros de entrada do (10+50)-ES e do ESboost utilizados nos experimentos.

TABELA 3 – PARÂMETROS PARA O (10+50)-ES PADRÃO E PARA O ESBOOST COM T = 100

PARÂMETROS	(10+50)-ES PADRÃO	ESBOOST COM T = 100
Execuções do Boosting	/	100
Tamanho da população		10
Número de descendentes		50
Número máximo de gerações		500
Probabilidade de cruzamento		0.6
Tipo de recombinação do ES		Global
Recombinação das variáveis		Discreta
Probabilidade de mutação		1

Nas execuções com ES, a probabilidade de cruzamento foi de 60% e de mutação de 100%, o cruzamento dos objetos foi do tipo discreto global. Cada atributo x_i do indivíduo teve seu σ_i e seu ângulos de rotação α_j (Seção 2.3). Realizaram-se 10 execuções de ES e 10 de ESboost para cada uma das funções de perda, variando apenas o valor da semente, pois o desempenho de técnicas evolucionárias têm uma grande dependência da semente e do gerador utilizados. Os resultados obtidos foram comparados com a regressão linear (LR – *Linear Regression*) e com as *model trees*

⁸ Para maiores informações sobre parâmetros de ES, ver Seção 2.3.

(MT) do software Weka (FRANK; WITTEN, 1999; WEKA SOFTWARE, 2004), o qual é uma ferramenta específica para a Mineração de Dados. Utilizaram-se os valores *default* dos parâmetros em cada um dos métodos.

Para avaliar a medida de desempenho de um modelo de previsão, existe a necessidade de uma medida de erro. Nesse trabalho, a medida padrão foi a raiz do erro médio quadrático (RMSE – *Root Mean Squared Error*):

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (f(x_i) - y_i)^2}{n}} \quad (12)$$

onde

n é o número de exemplos de teste

$f(x_i)$ é o valor esperado para y_i

e y_i é o valor real do atributo meta

O ESboost (Figura 17) também foi executado com funções de *fitness* diferentes da apresentada. Um exemplo foi o uso de funções de *fitness* igual à função de perda comum \bar{L}_t apresentada na Figura 17. Assim, quando a função de perda $L_t(i)$ era exponencial, a função de *fitness* também era. O mesmo ocorria para as funções de perda linear e quadrática.

As comparações dos resultados obtidos por meio de ESboost, de ES sem Boosting, de LR e de MT do software Weka foram realizadas com base na diferença absoluta (da) por meio da fórmula:

$$da (As - Ap) = \frac{\text{média}(As) - \text{média}(Ap)}{\sqrt{\frac{dp (As)^2 + dp (Ap)^2}{2}}} \quad (13)$$

onde

As é o algoritmo padrão

Ap é o algoritmo proposto

Média(As) é a média obtida para o algoritmo padrão

Média(Ap) é a média obtida para o algoritmo proposto

dp(As) é o desvio padrão obtido para o algoritmo padrão

dp(Ap) é o desvio padrão obtido para o algoritmo proposto

A experimentação foi realizada em 3 tipos distintos de bases de dados: conjunto de dados de regressão, séries temporais univariadas e séries temporais multivariadas. Bases de dados de regressão foram testadas com o objetivo de verificar o comportamento da técnica apresentada em bases comuns na literatura de regressão.

Para cada base de dados, selecionou-se o melhor valor obtido em 10 execuções, cada uma com semente diferente. Também foi calculado a média e o desvio padrão de cada conjunto de execuções. As subseções que seguem tratam de cada uma delas.

4.1 BASES DE DADOS DE REGRESSÃO

Bases de dados de regressão (Tabela 4) foram obtidas do *UCI Repository* (BLAKE; MERZ, 1998) com a finalidade de verificar o comportamento de ES e de ESboost em bases de dados não temporais. Nas bases *autoprice*, *basketball*, *automp* realizaram-se *10-folds cross-validation*. Nas demais, o conjunto total foi dividido aleatoriamente em 90% para treinamento e 10% para teste.

TABELA 4 – BASES DE DADOS DE REGRESSÃO

BASES DE DADOS	CONJUNTO TOTAL (100%)	TREINAMENTO (90%)	TESTE (10%)
autoprice	159	-	-
basketball	96	-	-
bodyfat	252	-	-
automp	398	262	136
cpu	209	137	72

Os melhores resultados obtidos para cada base de dados de regressão testada juntamente com os valores obtidos com o software Weka (FRANK; WITTEN, 1999; WEKA SOFTWARE, 2004) são encontrados na Tabela 5. Nas bases *autoprice*, *basketball*, *bodyfat* e *cpu*, Estratégias Evolucionárias, sem o uso de Boosting, obteve resultado melhor que regressão linear do Weka. A função de perda que obteve o melhor desempenho, em relação à (10+50)-ES foi a linear.

O ESboost, na base *autompg*, obteve melhores resultados que a regressão linear e que *model trees*. O mesmo ocorre nas funções de perda linear e exponencial da base *cpu*. Na base *bodyfat*, apesar dos resultados do ESboost serem piores, o resultado do (10+50)-ES supera LR e MT do Weka.

TABELA 5 – MELHOR RMSE DE 10 EXECUÇÕES EM BASES DE DADOS DE REGRESSÃO

BASE DE DADOS	WEKA		(10+50)-ES	ESBOOST COM T =100		
	LR	MT		LINEAR	QUADRÁTICA	EXPONENCIAL
autoprice	2861.2434	2240.8748	2632.2870	2583.6760	2706.9600	2625.0290
basketball	0.0859	0.0859	0.0841	0.0823	0.0991	0.0873
bodyfat	1.3840	1.3267	0.9363	2.5036	2.9782	2.5600
autompg	5.1078	4.5509	5.67126	3.85049	3.92398	3.88634
cpu	50.2219	38.0717	49.7596	28.2626	45.4643	29.9461

A Tabela 6 apresenta as médias e desvios padrão obtidos para o (10+50)-ES e para o ESboost. As discussões abaixo referem-se à Tabela 6 e também à regressão linear e *model trees* com desvio padrão zero da Tabela 5. Na base *autoprice* as funções de perda linear e exponencial se comportaram melhores que o (10+50)-ES, enquanto que com a quadrática ocorreu o contrário. Na base *basketball* somente a função linear se comportou melhor. Na base *bodyfat* o ESboost, em todos os casos, teve desempenho pior em relação à ES.

Na base de dados *autompg*, com um grau de confiança de 95%, o ESboost obteve melhores resultados que ES, que as *model trees* e que a regressão linear do Weka. Também com um grau de confiança de 95%, na base *cpu*, as funções de perda linear e exponencial obtiveram melhor resultado que a ES e que a regressão linear do Weka.

TABELA 6 – MÉDIA E DESVIO PADRÃO DO RMSE DE 10 EXECUÇÕES EM BASES DE REGRESSÃO

BASE DE DADOS	ESBOOST COM T =100							
	(10+50)-ES		LINEAR		QUADRÁTICA		EXPONENCIAL	
	Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão
autoprice	2755.4003	88.3150	2736.1100	83.3007	2826.5373	58.0740	2669.4081	35.3826
basketball	0.0897	0.0067	0.0854	0.0023	0.1037	0.0028	0.0905	0.0017
bodyfat	0.9594	0.0101	2.6686	0.1018	3.0523	0.0604	2.6079	0.0510
autompg	9.3359	2.9954	4.0272	0.0986	4.0872	0.1286	3.9986	0.0987
cpu	80.8172	13.2065	34.9280	4.5791	61.9455	8.8666	36.7580	6.2615

Dessa maneira, observa-se que nas bases de dados de regressão, o ESboost utilizando as funções de perda linear e exponencial apresenta, na maioria dos casos, melhor comportamento que o (10+50)-ES sem Boosting.

4.2 SÉRIES TEMPORAIS UNIVARIADAS

As séries temporais univariadas, com exceção de *sunspots*, descritas na Tabela 7 e coletadas em MORRETIN e TOLOI (2004) foram divididas em 90% para treinamento e 10% para teste, de acordo com SOUZA, COSTA e POZO (2005). Considerou-se até os 4 últimos valores das séries.

TABELA 7 – SÉRIES TEMPORAIS UNIVARIADAS

BASES DE DADOS	CONJUNTO TOTAL (100%)	TREINAMENTO (90%)	TESTE (10%)
atmosfera	365	329	36
bebida	187	169	18
consumo	154	139	15
fortaleza	149	135	14
icv	114	102	12
ipi	187	169	18
lavras	384	346	38
manchas	176	159	17

A série *sunspots* (AKAIKE, 1978) contém o número anual de pontos escuros no sol de 1700 a 1979. Como é comum na literatura, os valores de 1700 a 1920 foram selecionados para a base de treinamento e o restante para teste.

Os melhores valores obtidos no (10+50)-ES e no ESboost para as séries

temporais univariadas, juntamente com os resultados do Weka, estão na Tabela 8. MT apresentou melhores resultados que ESboost nas bases *consumo* e *lavras*. Somente na base *manchas* o (10+50)-ES obteve-se um resultado melhor que as demais técnicas. Nas demais, os melhores valores foram apresentados pelo ESboost, principalmente com a função de perda quadrática.

TABELA 8 – MELHOR RMSE DE 10 EXECUÇÕES EM SÉRIES TEMPORAIS UNIVARIADAS

BASE DE DADOS	WEKA		(10+50)-ES	ESBOOST COM T =100		
	LR	MT		LINEAR	QUADRÁTICA	EXPONENCIAL
atmosfera	5.9835	5.9835	5.94083	5.91831	5.6913	5.77413
bebida	15.4606	15.4606	16.7905	15.4107	13.5091	14.2929
consumo	12.7171	11.0073	13.4258	11.1384	14.5242	14.1548
fortaleza	607.8307	607.8307	666.4270	651.8110	612.2010	589.9550
icv	21.9425	64.3882	22.5640	22.4244	20.4235	24.2025
ipi	11.1680	11.1680	11.2609	10.6709	10.7250	10.7382
lavras	87.5273	75.1177	92.6272	85.6767	135.9290	89.2674
manchas	17.7763	14.9274	14.0492	14.9995	35.2812	15.8667
sunspots	19.3874	19.6690	21.2325	19.9755	18.4974	18.6899

Na Tabela 9 estão as médias e os desvios padrão de 10 execuções. Com um grau de confiança de 95%, ESboost com a função de perda linear se comportou melhor que (10+50)-ES para todas as bases, exceto *manchas* e *icv*. Mas no caso do *icv*, também houve melhora.

TABELA 9 – MÉDIA E DESVIO PADRÃO DO RMSE DE 10 EXECUÇÕES EM SÉRIES TEMPORAIS UNIVARIADAS

BASE DE DADOS	ESBOOST COM T =100							
	(10+50)-ES		LINEAR		QUADRÁTICA		EXPONENCIAL	
	Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão
atmosfera	5.9428	0.0024	5.9232	0.0032	5.9894	0.2063	5.8304	0.0351
bebida	16.9412	0.0881	15.7926	0.2478	14.4427	0.7565	14.5292	0.1377
consumo	13.5087	0.1301	12.3696	0.5975	18.3985	1.9571	14.9579	0.6037
fortaleza	667.4844	0.3766	654.7697	1.9230	614.9244	1.1764	594.9591	2.9337
icv	28.5110	3.6368	25.7347	1.8061	23.8203	2.7204	26.2571	1.5486
ipi	11.3401	0.0539	10.694	0.0243	10.8634	0.2168	10.7869	0.0304
lavras	92.6740	0.0249	86.0646	0.3396	164.6376	10.1834	144.8937	32.7835
manchas	14.1412	0.0777	15.1276	0.1358	38.1893	1.5922	16.7650	0.7511
sunspots	21.2682	0.0210	19.9860	0.0065	20.7663	1.1260	18.8833	0.1719

Com 95% de confiança, as bases *bebida*, *fortaleza*, *icv* e *ipi* tiveram melhores resultados na função de perda quadrática e as bases *atmosfera*, *bebida*,

fortaleza, *ipi* e *sunspots* na função exponencial.

Comparando o ESboost e as *model trees* do software Weka, com desvio padrão igual a zero, houve melhora nas bases: *atmosfera* (linear, exponencial), *bebida* (quadrática, exponencial), *fortaleza* (exponencial), *icv* (linear, quadrática, exponencial), *ipi* (linear, quadrática, exponencial) e *sunspots* (exponencial).

Comparando o ESboost com a regressão linear do software Weka, com desvio padrão igual a zero, houve melhora nas bases: *atmosfera* (linear, exponencial), *bebida* (quadrática, exponencial), *fortaleza* (exponencial), *ipi* (linear, quadrática, exponencial), *manchas* (linear, exponencial) e *sunspots* (exponencial). Na maioria das bases de dados, então, houve melhoria nos resultados do ESboost em relação às *model trees* e à regressão linear do software Weka.

Objetivando visualizar os resultados de algumas séries temporais univariadas, foi feita a representação gráfica dos resultados obtidos na base de dados de teste. Os Gráficos 1, 2 e 3 demonstram os valores obtidos em uma execução (sorteada aleatoriamente) para a base de dados de teste de *atmosfera*, *bebida* e *fortaleza*, respectivamente.

GRÁFICO 1 – RESULTADOS DAS PREVISÕES OBTIDAS PARA O CONJUNTO DE TESTE (ATMOSFERA)

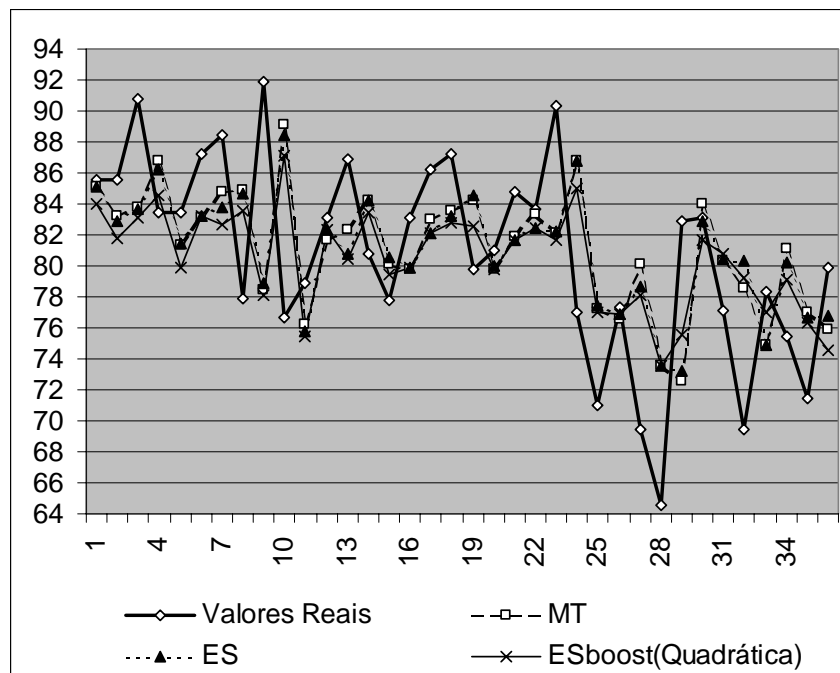


GRÁFICO 2 – RESULTADOS DAS PREVISÕES OBTIDAS PARA O CONJUNTO DE TESTE (BEDIDA)

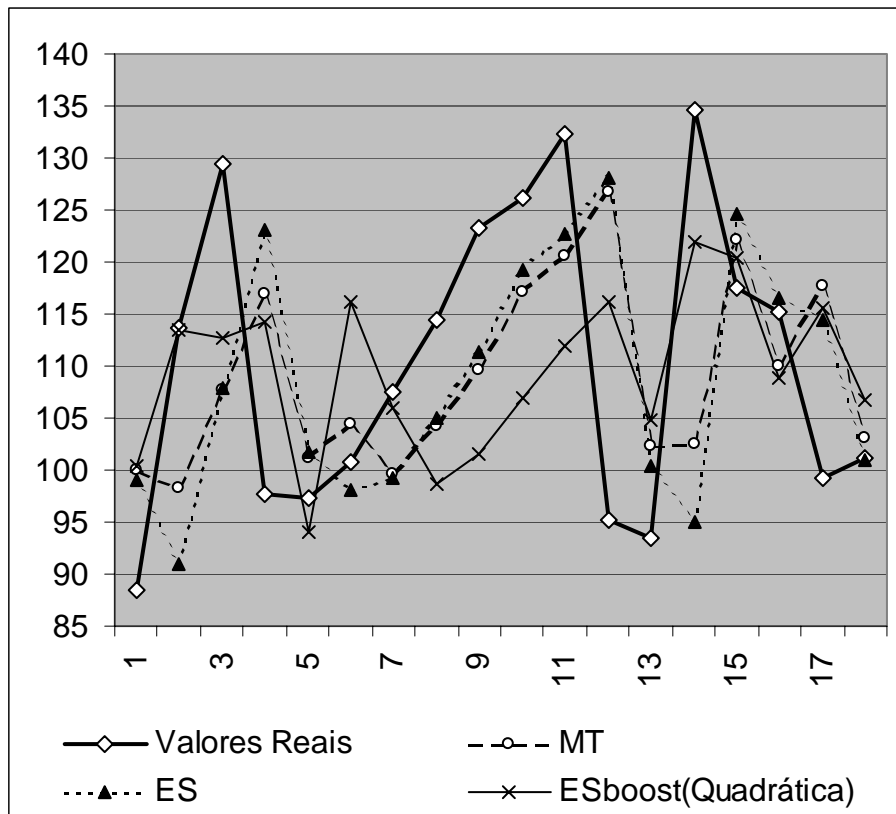
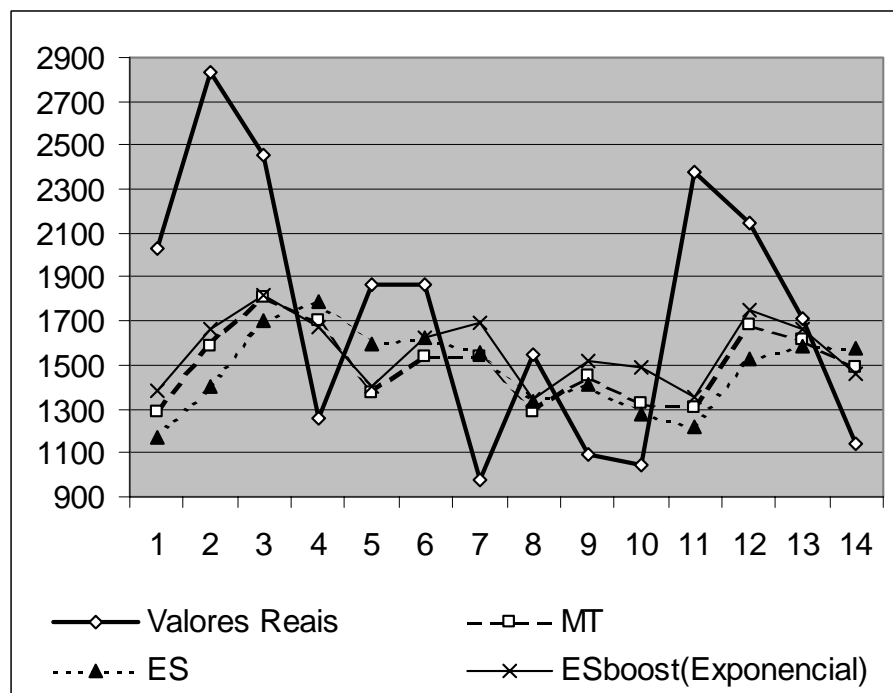


GRÁFICO 3 – RESULTADOS DAS PREVISÕES OBTIDAS PARA O CONJUNTO DE TESTE (FORTALEZA)



De acordo com o exposto, observa-se que a função de perda linear apresenta melhores resultados na maioria das bases de dados em relação ao (10+50)-ES. O mesmo ocorre em quase metade das bases analisadas para as funções de perda quadrática e exponencial.

4.2.1 Comparativo com Modelo ARMA

Os resultados do (10+50)-ES e do ESboost obtidos nas séries temporais univariadas da Tabela 7 também foram comparadas com o modelo ARMA (BOX; JENKINS, 1976), que é a combinação de modelos auto-regressivos (AR) e de médias móveis (MA). O modelo ARMA é comumente utilizado na tarefa de regressão, e serve como base para comparar com outros modelos.

Os resultados estão apresentados de acordo com a seguinte medida de erro:

$$RMS = \frac{\sqrt{\sum_{i=1}^n (f(x_i) - y_i)^2}}{n} \quad (14)$$

onde

n é o número de exemplos de teste

$f(x_i)$ é o valor esperado para y_i

e y_i é o valor real do atributo meta

Na Tabela 10 têm-se os melhores valores obtidos, e na Tabela 11 têm-se as médias e desvios padrão.

Em relação aos melhores valores obtidos, somente na base de dados *lavras* o modelo ARMA supera todos os algoritmos executados. Nas bases *atmosfera*, *icv* e *ipi* todos os algoritmos de ESboost e o (10+50)-ES são melhores que o modelo ARMA.

TABELA 10 – MELHOR RMS DE 10 EXECUÇÕES EM SÉRIES TEMPORAIS UNIVARIADAS

BASE DE DADOS	ARMA	ESBOOST COM T =100			
		(10+50)-ES	LINEAR	QUADRÁTICA	EXPONENCIAL
atmosfera	ARMA (4,3) = 1.06	0.990138	0.986385	0.948551	0.962355
bebida	ARMA (4,3) = 3.44	3.9576	3.6323	3.1841	3.3689
consumo	ARMA (4,3) = 2.99	3.4665	2.8759	3.7501	3.6548
fortaleza	ARMA (1,0) = 176.12	178.1100	174.2040	163.6180	157.6720
icv	ARMA (1,0) = 36.5	6.5137	6.4734	5.8958	6.9867
ipi	ARMA (3,2) = 4.04	2.6542	2.5152	2.5279	2.5310
lavras	ARMA (3,2) = 12.03	15.0261	13.8986	22.0506	14.4811
manchas	ARMA (2,1) = 6.27	3.4074	3.6379	8.5569	3.8482

TABELA 11 – MÉDIA E DESVIO PADRÃO DO RMS DE 10 EXECUÇÕES EM SÉRIES TEMPORAIS UNIVARIADAS

BASE DE DADOS	ESBOOST COM T =100							
	(10+50)-ES		LINEAR		QUADRÁTICA		EXPONENCIAL	
	Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão
atmosfera	0.9905	0.0004	0.9872	0.0005	0.9982	0.0344	0.9717	0.0058
bebida	3.9931	0.0208	3.7223	0.0584	3.4042	0.1783	3.4246	0.0325
consumo	3.4879	0.0336	3.1938	0.1543	4.7505	0.5053	3.8621	0.1559
fortaleza	178.3926	0.1007	174.9947	0.5140	164.3455	0.3143	159.0095	0.7840
icv	8.2304	1.0498	7.4290	0.5214	6.8763	0.7853	7.5798	0.4471
ipi	2.6729	0.0127	2.5207	0.0057	2.5605	0.0511	2.5425	0.0072
lavras	15.0337	0.0040	13.9615	0.0551	26.7077	1.6520	23.5049	5.3182
manchas	3.4297	0.0188	3.6690	0.0329	9.2623	0.3862	4.0661	0.1822

4.2.2 Resultados com Outra Função de *Fitness*

Execuções foram realizadas com outras funções de *fitness*. Em uma delas, a função de *fitness* era igual à função de perda comum. O melhor resultado obtido pela base de dados *sunspots* está exposto nas Tabela 12. Na Tabela 13 estão a média e o desvio padrão de cada conjunto de execução.

TABELA 12 – MELHOR RMSE DE 10 EXECUÇÕES NA SÉRIE SUNSPOTS COM OUTRA FUNÇÃO DE FITNESS

BASE DE DADOS	WEKA		(10+50)-ES	ESBOOST COM T =100		
	LR	MT		LINEAR	QUADRÁTICA	EXPONENCIAL
sunspots	19.3874	19.6690	49.3990	19.8849	21.3281	19.0906

Com o (10+50)-ES, o resultado fica longe do valor obtido para LR e MT pelo Weka. No entanto, com o aumento do valor do T, o RMSE diminui, mas não

como nas Tabelas 8 e 9. Isso demonstra que a função de *fitness* utilizada nas Tabelas 8 e 9 é melhor.

TABELA 13 – MÉDIA E DESVIO PADRÃO DO RMSE DE 10 EXECUÇÕES NA SÉRIE SUNSPOTS COM OUTRA FUNÇÃO DE FITNESS

BASE DE DADOS	ESBOOST COM T =100							
	(10+50)-ES		LINEAR		QUADRÁTICA		EXPONENCIAL	
	Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão
sunspots	52.7168	2.0626	21.9569	1.7483	23.8511	2.0286	20.1204	0.4495

4.3 SÉRIES TEMPORAIS MULTIVARIADAS

Utilizaram-se duas bases de dados temporais multivariadas. A base *precio* (KEEL DATASET, 2006) trata dos kWhs produzidos na Espanha em cada dia do ano de 2003, em 6 categorias. Portanto, tem-se 6 atributos previsores e 1 meta (saída). Os atributos previsores são: (1) *Hydraulics* (Hidráulica), (2) Nuclear, (3) *Coal* (*cob coal, importing coal, black and brown Lignito*), (4) *Fuel* (*fuel-oil and fuel/gas*), (5) *Gas* e (6) *Special regimen* (*eolithic, solar, hydraulics minicentrals, etc*). A saída é o custo em euros do KW/h. A base de dados possui 365 exemplos. Os atributos previsores foram utilizados para prever o custo em euros no mesmo dia em que foram coletados. Foi realizado *10-folds cross-validation*.

A base *esalq9899rad* foi retirada do *site* da ESALQ (DCE – ESALQ, 2006), e corresponde a dados diários climáticos do ano de 1998 e 1999. Os atributos previsores eram a temperatura média, a umidade relativa média, a velocidade do vento, a temperatura máxima, a umidade relativa máxima, a velocidade do vento, a temperatura mínima, a umidade relativa mínima e a quantidade de chuva. O atributo meta era o valor da radiação global no dia seguinte aos atributos previsores. Havia 730 registros na base de dados. Os registros correspondentes aos 2 últimos meses foram selecionados para teste e o restante para treinamento.

A Tabela 14 mostra os melhores resultados obtidos, juntamente com os valores de LR e de MT do software Weka. Na Tabela 15 estão a média e o desvio

padrão de cada conjunto de 10 execuções do (10+50)-ES e do ESboost. Na base de dados *precio*, (10+50)-ES e ESboost com função de perda linear se apresentou melhor que LR e MT. Na base *esalq9899rad*, somente um valor obtido com o (10+50)-ES foi melhor que a LR.

TABELA 14 – MELHOR RMSE DE 10 EXECUÇÕES EM SÉRIES TEMPORAIS MULTIVARIADAS

BASE DE DADOS	WEKA		(10+50)-ES	ESBOOST COM T =100		
	LR	MT		LINEAR	QUADRÁTICA	EXPONENCIAL
precio	0.4103	0.4157	0.4102	0.4102	0.5477	0.4176
esalq9899rad	4.6534	4.1835	4.6063	4.7111	5.7710	4.7027

TABELA 15 – MÉDIA E DESVIO PADRÃO DO RMSE DE 10 EXECUÇÕES EM SÉRIES TEMPORAIS MULTIVARIADAS

BASE DE DADOS	ESBOOST COM T =100							
	(10+50)-ES		LINEAR		QUADRÁTICA		EXPONENCIAL	
	Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão
precio	0.4108	0.0003	0.4106	0.0002	0.6110	0.0342	0.4190	0.0009
esalq9899rad	4.6844	0.0487	4.7682	0.0253	6.0774	0.2169	4.7616	0.0454

Comparando o ESboost com a MT do software Weka, com desvio padrão igual a zero, somente houve melhora com a função linear na base *precio*. Na base *esalq9899rad* não houve melhoras nem em relação à MT, nem em relação à LR. Na maioria das bases de dados, então, houve melhoria nos resultados do ESboost em relação à *model trees* e à regressão linear do software Weka. Dessa maneira, observa-se que nessas bases de dados não houve grandes avanços em relação aos métodos do software Weka.

5 CONCLUSÃO

Este trabalho propôs o uso de Estratégias Evolucionárias na tarefa de regressão para a Mineração de Dados Temporais. O algoritmo de ES foi aplicado em dados temporais (séries temporais univariadas e multivariadas). Na fase de busca de bases de dados, encontraram-se mais séries temporais univariadas que multivariadas. Isso demonstra a necessidade da disponibilidade de mais bases de dados e mais trabalhos sobre séries temporais multivariadas.

Além dos dados temporais, realizaram-se testes com algumas bases de dados de regressão não temporais. O (10+50)-ES permitiu que as melhores soluções sobrevivessem ao longo das gerações. Os resultados nas bases de dados provaram que, na maioria dos casos, ES (sem o uso do Boosting) teve comportamento similar à regressão linear do software Weka.

O uso da biblioteca EO (*Evolving Objects*) facilitou a implementação da regressão linear por meio de Estratégias Evolucionárias e assegurou que o algoritmo estivesse correto, uma vez que muitos estudos têm sido desenvolvidos com essa biblioteca.

Objetivando aperfeiçoar os resultados de ES, aplicou-se o algoritmo AdaBoost.R2 ao algoritmo de regressão linear com Estratégias Evolucionárias. O algoritmo obtido, denominado ESboost, tinha uma função de *fitness* na ES baseada nos pesos gerados pelo AdaBoost.R2, a qual levava em consideração todos os exemplos, juntamente com seus pesos. Isso permitiu que exemplos difíceis fossem focados.

Outro fato observado na função de *fitness* foi que ela tem uma grande importância na diminuição do erro de predição e que, alterando-a, os valores de erro sofrem grandes mudanças.

A união da condição de parada do algoritmo de AdaBoost original com a condição de parada do AdaBoost.R2 tornou possível trabalhar com dados reais sem a necessidade de identificar, para cada base de dados, um limite \bar{L}_t diferente do

especificado no AdaBoost.R2. Dessa maneira, o parâmetro T acabou com o risco de execuções infinitas, uma vez que determinava o número máximo de iterações do ESboost.

As séries temporais multivariadas não tiveram melhorias nos resultados com o ESboost como as séries temporais univariadas e como as bases de dados de regressão não temporais. Isso pode ter ocorrido devido à configuração dos períodos de tempo nas bases de dados. Para aperfeiçoar os resultados, talvez seja necessário unir os atributos previsores de dois ou mais períodos de tempo para prever o atributo meta.

Nas séries temporais univariadas e nas bases de dados não temporais, comparando o ESboost com as execuções do ES sem Boosting, observou-se que na maioria dos casos houve uma melhora pelo menos com uma das funções de perdas. Em muitos casos, houve melhora em relação aos métodos implementados em Weka.

Os resultados do algoritmo ESboost superaram os resultados do modelo ARMA em quase todas as séries temporais univariadas. Somente na base de dados *lavras*, o desempenho foi pior, provando que o ESboost obtém bons resultados em séries temporais univariadas.

Alguns trabalhos futuros poderão constituir-se da aplicação de outras técnicas de aprendizado de máquina no lugar de ES, de testes com outras funções de *fitness* e de estudos comparativos com outras metodologias para a predição de eventos.

6 REFERÊNCIAS BIBLIOGRÁFICAS

- AKAIKE, H. On the likelihood of time series model. **The statistician** 27: 217-235, 1978.
- BÄCK, T.; FOGEL, D.B.; MICHALEWICZ, Z. **Evolutionary computation 2: advanced algorithms and operators**. Bristol and Philadelphia: Institute of Physics Publishing, 2000.
- BÄCK, T.; RUDOLPH, G.; SCHEWEL, H.P. Evolutionary programming and evolution strategies: similarities and differences. In: Annual Conference on Evolutionary Programming, 2., San Diego, 1993. **Proceedings...** La Jolla, 1993. p. 11-22.
- BÄCK, T.; SCHWELFEL, H.-P. An overview of evolutionary algorithms for parameter optimization. **Evolutionary Computation**, v. 1, n. 1, p. 1-23, 1993.
- BARRETO, G. de A.; ARAÚJO, F. R. Time in self-organizing maps: an overview of models. **International Journal of Computer Research**, v. 1, n. 2, p. 235–259, 2001.
- BLAKE, C.L.; MERZ, C.J. UCI repository of machine learning databases. <http://www.ics.uci.edu/~mllearn/MLRepository.html>. 1998.
- BONÉ, R.; ASSAD, M.; CRUCIANU, M. Boosting Recurrent Neural Networks for Times Series Prediction. RFAI Publication, Artificial Neural Nets and Genetic Algorithms, Proceedings of the International Conference in Roanne (France), D. W. Pearson, N.C. Steele, R.F. Albrecht (eds), **Springer Computer Science**, pp. 18-22, April 2003.
- BOX, G.E.P.; JENKINS, G. M. **Time series analysis: forecasting and control**. Ed. Rev. San Francisco: Holden-Day, 1976.
- CAVALHEIRO, A.F. **GADBMS – Um algoritmo genético minerador de dados para base de dados relacionais**. Curitiba. Dissertação (Mestrado em Informática) - Setor de Ciências Exatas, Universidade Federal do Paraná. 2002.
- CHIANG, Y-M.; CHANG, L-C.; CHANG, F-J. Comparison of static-feedforward and dynamic-feedback neural networks for rainfall–runoff modeling. **Journal of Hydrology**, v. 290, ed. 3-4, p. 297-311, maio 2004.
- DARWIN, C. **On the origin of species by means of natural selection or the preservation of favored races in the struggle for life**. Murray, London, UK. 1859.
- DCE – ESALQ - BASE DE DADOS DA ESTAÇÃO METEOROLÓGICA AUTOMATIZADA - ÁREA DE FÍSICA E METEOROLOGIA - DCE - ESALQ – USP. <http://www.esalq.usp.br/departamentos/lce/automatica/pagina4.html>. Acesso em 13 de fevereiro de 2006.
- DIANATI, M.; SONG, I.; TREIBER, M. An introduction to genetic algorithms and evolution strategies. 2002.
- DIETTERICH, T.G. ; MICHALSKI, R.S. Discovering patterns in sequences of events. **Artificial Intelligence**, v. 25, p. 187-232, 1985.
- DRUCKER, H. Improving Regressors using Boosting Techniques. In: FOURTEENTH INTERNATIONAL CONFERENCE ON MACHINE LEARNING, ed. Douglas H. Fisher, Jr. **Proceedings...** Morgan-Kaufmann, p. 107-115, 1997.

EFRON, B. Computer-intensive methods in statistical regression. **SIAM Review**, v. 30, n. 3, p. 421-449, 1988.

EO (EVOLVING OBJECTS) LIBRARY. <http://www.lri.fr/%7Emarc/EO/>. Acesso em 15 de março de 2005.

EVANS, R.M.; ALVIN, J.S. Relating numbers of processing elements in a sparse distributed memory model to learning rate and generalization. **ACM APL Quote Quad**, v. 21, n. 4, p. 166-173, 1991.

FAYYAD, U.; PIATETSKY-SHAPIRO, G.; SMYTH, P. The KDD process for extracting useful knowledge from volumes of data. **Communications of the ACM**, v.39 n.11, p.27-34, nov. 1996.

FOG, T.L. et al. Training and evaluation of neural networks for multi-variate time series processing. In: IEEE International Conference on Neural Networks. **Proceedings...** IEEE Press., 1995.

FOGEL, L.J.; Autonomous automata. **Industrial Research**, v. 4, p. 14-19, 1962.

FOGEL, L.J.; OWENS, A.J.; WALSH, M.J. **Artificial intelligence through simulates evolution**. New York: Wiley, 1966.

FRANK, E.; WITTEN, I.H. **Data mining: practical machine learning tools and techniques with java implementations**. San Mateo: Morgan Kaufmann. 1999.

FREUND, Y.; SCHAPIRE, R.E. **A decision-theoretic generalization of on-line learning and an application to boosting**. Murray Hill, NJ: AT&T Bell Laboratories, 1995. Relatório técnico.

FREUND, Y.; SCHAPIRE, R.E. A decision-theoretic generalization of on-line learning and an application to boosting. **Journal of Computer and System Sciences**, v. 55, n.1, p.119-139, ago. 1997.

FREUND, Y.; SCHAPIRE, R.E. A short introduction to boosting. **Journal of Japanese Society for Artificial Intelligence**, v. 14, n.5., p. 771-780, set. 1999. (em japonês, tradução de Naoki Abe).

GIROSI, F.; POGGIO, T. Networks and the best approximation property. **Biological Cybernetics**, v. 63, p. 169-176, 1990.

GOLDBERG, D.E. **Genetic algorithms in search, optimization and machine learning**. Addison-Wesley, Reading. 1989.

HAYKIN, S. **Neural networks: a comprehensive foundation**. New Jersey: Prentice-Hall, 1999.

HOLLAND, J.H. **Adaptation in natural and artificial systems**. 2. ed. MIT Press, 1992.

KADOUS, M.W. A general architecture for supervised classification of multivariate time series. 57p. 1998.

KEEL DATASET. Disponível em: <http://sci2s.ugr.es/keel-dataset/problemas/regresion/precio/datos/precio.dat>. Acesso em 20 jan. 2006.

KLEMA, J., KOUT, J., VEJMEĽKA, M. Predictive System for Multivariate Time Series. **Cybernetics and Systems 2004**. Vienna: Austrian Society for Cybernetics Studies, vol. 1,2, p. 723-728, 2004.

KOUT, J., VLCEK, T., KLEMA, J. Predictive System for Multivariate Time Series. **Automa International - Trade Journal for Industrial Automation**. n. 11, p. 6-8, 2005.

KOZA, J.R. **Genetic programming II: automatic discovery of reusable programs**. Cambridge, MA: MIT Press, 1994.

KOZA, J.R. **Genetic programming: on the programming of computers by means of natural selection**. Cambridge, MA: MIT Press, 1992.

KUSIAK, A. Evolutionary computation and data mining. In: SPIE CONFERENCE ON INTELLIGENT SYSTEM AND ADVANCED MANUFACTURING, SPIE, Vol.4192, 2000, Boston, MA. **Proceedings...** Boston, MA: B. Gopalakrishnan and Agunasekaran, nov. 2000. p.1-10.

LACHTERMACHER, G.; FULLER, J.D. 1995. Backpropagation in time series forecasting. **Journal of Forecasting**, v. 14, p. 381-393.

LIU, B.; MCKAY, B.; ABBASS, A. Improving genetic classifiers with a boosting algorithm. **CEC 2003**, 2003. p.2596-2602.

LUGER, G.F. **Inteligência artificial: estruturas e estratégias para a solução de problemas complexos**. George F. Luger; brad-Paulo Engel. 4. ed. Porto Alegre: Bookman, 2004.

MCCLELLAND, J.; RUMELHART, D. **Explorations in parallel distributed processing**. Cambridge, MA: MIT Press, 1988.

MINERVA, T.; PATERLINI, S. Evolutionary approaches for statistical modelling. In: CONGRESS ON EVOLUTIONARY COMPUTATION (CEC-2002), 4, 2002. **Proceedings...** 2002.

MORRETIN, P. A.; TOLOI, C. M. C. **Análise de Séries Temporais**. Ed. Edgard Blucher, 2004.

NIST/SEMATECH e-Handbook of Statistical Methods. Disponível em: <http://www.itl.nist.gov/div898/handbook/>. Acesso em 05 mar. 2006.

PARIS, G.; ROBILLIARD, D.; FONLUPT, C. Applying boosting techniques to genetic programming. In: INTELLIGENT ARTIFICIAL EVOLUTION, 6., 2001. **Proceedings...** 2001. p.315 – 326.

PEARLMUTTER, B. Gradient calculations for dynamic recurrent neural networks. **IEEE Transactions on Neural Networks**, v.6, n.5, p. 212–1228, 1995.

POVINELLI, R.J. Using genetic algorithms to find temporal patterns indicative of time series events, In: GECCO 2000 WORKSHOP: DATA MINING WITH EVOLUTIONARY ALGORITHMS, 2000. p. 80-84.

QUINLAN, J. Combining instance-based and model-based learning. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 1993. **Anais....** 1993. p. 236-243.

- RAMÍREZ, M.C.V.; VELHO, H.F.C.; FERREIRA, N.J. Artificial neural networks technique for rainfall forecasting applied to the São Paulo region. Elsevier. **Journal of Hydrology**. 2004.
- RECHENBERG, I. **Cybernetic solution path of an experimental problem**. Royal Aircraft Establishment, Library translation n°. 1122, Farnborough, Hants., UK, ago. 1965.
- RECHENBERG, I. Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen **Evolution**. **Frommann-Holzboog Verlag Stuttgart**, 1973.
- RIPLEY, B.D. Statistical aspects of neural networks. In: NETWORKS AND CHAOS – STATISTICAL AND PROBABILISTIC ASPECTS, 1993. eds O.E. Barndorff-Nielsen, J.L. Jensen and W.S. Kendall. Chapman and Hall, p. 40-123, 1993.
- RODDICK, J.F.; SPILIOPOULOU, M. A survey of temporal knowledge discovery paradigms and methods. **IEEE Transactions on Knowledge and Data Engineering**, v.14, n.4, p. 750-767, jul./ago. 2002.
- ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. **Psychological Review**, v. 65, p. 386-408, 1958.
- SARLE, W.S. Stopped training and other remedies for overfitting. In: SYMPOSIUM ON THE INTERFACE, 27, 1995. **Proceedings...** 1995.
- SCHAPIRE, R.E. The boosting approach to machine learning: an overview. In: MSRI Workshop on Nonlinear Estimation and Classification, 2002.
- SCHEFFE, H. The analysis of variance. Wiley, New York, 1959.
- SCHWEFEL, H.-P. **Evolutionsstrategie und numerische optimierung**. Dissertation, Technische Universität Berlin, maio 1975.
- SCHWEFEL, H.-P. **Numerical optimization of computer models**. Wiley, Chichester, 1981.
- SCHWEFEL, H.-P. Numerische optimierung von computer-modellen mittels der evolutionsstrategie. **Interdisciplinary Systems Research**, Birkhauser, Basel, v. 26, 1977.
- SIQUEIRA, T.G.; SOARES FILHO, S. Application of neural networks with radial basis activation function to the prediction of non-stationary time series, In: CONGRESSO BRASILEIRO DE AUTOMAÇÃO, 14. 2002. (em Português).
- SOLOMATINE, D. P.; SHRETHA, D. AdaBoost. RT: a Boosting Algorithm for Regression Problems. In: IEEE INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS, 2004, **Proceedings...** 2004, p.1163-1168.
- SOUZA, L. V. ; COSTA, E. ; POZO, A. T. R. . Análise da Capacidade da Programação Genética na Previsão de Séries Temporais. In: Congresso de Métodos Numéricos em Engenharia, 2005, Granada. Congresso de Métodos Numéricos em Engenharia, 2005.
- TORGO, L.; GAMA, J. Regression by classification. 3th In: BRAZILIAN SYMPOSIUM ON ARTIFICIAL INTELLIGENCE, 3., SBIA '96, Curitiba, Brazil, out. 1996. p. 51-60.
- WEISS, G. M. Timeweaver: a genetic algorithm for identifying predictive patterns in sequences of events. In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE, 1999, Orlando, Florida, edited by W. Banzhaf, J. Daida, A. Eiben, M.

Garzon, V. Honavar, and M. Jakiela, **Proceedings...** San Francisco, CA: Morgan Kaufmann, 1999. p. 719-725.

WEISS, G.M.; HIRSH, H. Learning to predict rare events in event sequences, In: INT'L CONF. KNOWLEDGE DISCOVERY AND DATA MINING (KDD '98), 4., R. Agrawal, P. Stolorz, and G. Piatetsky-Shapiro, eds., **Proceedings...** 1998. p. 359-363.

WEISS, S.; INDURKHYA, N. Rule-base regression. In: Australian Joint Conference on Artificial Intelligence, 5., 1993. **Proceedings...** 1993. p. 1072-1078.

WEISS, S.; INDURKHYA, N. Rule-based machine learning methods for functional prediction. **Journal of Artificial Intelligence Research (JAIR)**, v. 3, p.383-403, 1995.

WEKA SOFTWARE. <http://www.cs.waikato.ac.nz/ml/weka/>. Acesso em 15 de novembro de 2004.

WHIGHAM, P.A. Grammatically-based genetic programming. In: WORKSHOP ON GENETIC PROGRAMMING: FROM THEORY TO REAL-WORLD APPLICATIONS, 1995. p. 33-41.