d.s.c. UPE
Departamento de Sistemas e Computação

POLI
ESCOLA POLITÉCNICA DE PERNAMBUCO

Estimativa de Esforço em Projetos de Software com Extreme Learning Machines (ELM)

Trabalho de Conclusão de Curso

Engenharia da Computação

João Fausto Lorenzato de Oliveira

Orientador: Prof. Sérgio Castelo Branco Soares

UNIVERSIDADE DE PERNAMBUCO

ESCOLA
POLITÉCNICA DE
PERNAMBUCO

João Fausto Lorenzato de Oliveira

Estimativa de Esforço em Projetos de Software com Extreme Learning Machines (ELM)

Monografia apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Recife, novembro de 2009.

Dedico este trabalho aos meus pais, Sandra Rejane Barbosa Lorenzato e Severino Francisco de Oliveira Filho, e toda minha família.

Agradecimentos

Agradeço aos meus pais, Sandra Rejane Barbosa Lorenzato e Severino Francisco pela educação e apoio, que tornaram possível essa conquista.

À toda minha família, por sempre estar presente em todos os momentos da minha vida.

Aos meus amigos Arthur Fernandes, Caio Medeiros, Diego Siqueira, Gilliard Alan, Nathália Temudo, Rafael Galvão, pela amizade, apoio e conhecimentos construídos e compartilhados.

Aos meus orientadores Adriano Lorena e Sérgio Soares, pela paciência e orientações. Ao professor Mêuser Valença pelos valiosos conselhos, que proporcionaram melhores resultados para este trabalho.

Resumo

O software se tornou um insumo fundamental para o sucesso de várias empresas e de organizações maiores como governos. Com o aumento da demanda de software, as fábricas de software possuem a meta de desenvolvê-los o mais rápido possível e com o menor custo possível. Para tanto, o gerenciamento dos riscos e das incertezas durante o decorrer de um projeto tornou-se uma das prioridades dos gerentes de projetos de softwares.

Projetos de software podem ser afetados por vários eventos que podem mudar o seu andamento. Todos os fatores devem ser analisados pelo gerente de forma que seja possível realizar uma estimativa do esforço necessário para a conclusão do projeto dentro do prazo estabelecido.

Uma etapa fundamental para o processo de estimativa de esforço é o levantamento de requisitos, que representa a fase inicial do processo de desenvolvimento. Um bom entendimento dos requisitos reduz significativamente a incerteza sobre o projeto e aumenta a probabilidade de estimar com sucesso o esforço do projeto.

Neste trabalho, aplicaremos a técnica Extreme Learning Machine (ELM) ao problema da estimativa de esforço de software, e também serão realizadas comparações com outros trabalhos realizados.

Palavras-chave: Estimativa de Esforço de Software, Redes Neurais, Aprendizado de Máquina

Abstract

The software has become a key input for the success of several companies and larger organizations such as governments. With increasing demand for software, software factories have the goal of developing them as quickly as possible and with the lowest possible cost. Therefore, the management of risks and uncertainties during the course of a project became a priority for managers of software projects.

Software projects can be affected by various events that can change their progress. All factors must be analyzed by the manager, so it can be used to estimate the effort required to complete the project within the deadline.

A critical step in the process of estimating the effort is requirements elicitation, which represents the early stage of development. A good understanding of the requirements significantly reduces the uncertainty about the project and increases the probability of successfully estimating the project effort.

In this work, we will apply the technique Extreme Learning Machine (ELM) to the problem of estimating software effort, and will also make comparisons with other works.

Keywords: Software Effort Estimation, Neural Networks, Machine Learning

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

ELM – Extreme Learning Machine

NASA – National Aeronautics and Space Administration

COCOMO – Constructive Cost Model (Modelo de custo Construtivo)

LOC – Lines of Code (Linhas de código)

UML – Unified Modeling Language

RNA– Rede Neural Artificial

BP– Backpropagation

MLP – Multilayer Perceptron

SLFN – Single Layer Feedforward Networks

IPSONET – Improved Particle Swarm Optimization for Evolving Feedforward Artificial Neural Networks

PSO – Particle Swarm Optimization

MMRE – Mean Magnitude Relative Error

LOOCV – Leave-One-Out Cross Validation

Capítulo 1

1.1 Introdução

Um importante insumo no desenvolvimento de software é saber quanto ele vai custar. Os softwares tornam-se gradativamente mais complexos devido às demandas das empresas, portanto para viabilizar o processo de produção de software de qualidade dentro do prazo estabelecido é necessário haver o controle sobre sua produção. Para as áreas de controle e planejamento de desenvolvimento de software, onde os gerentes fazem cronogramas de projeto, a estimativa de esforço é um importante insumo [1].

Segundo [9], o esforço de software pode ser definido como a quantidade total de trabalho a ser executado em um projeto de software. Normalmente o esforço de software é medido em Homem-hora.

Durante a fase inicial do projeto de software, o levantamento de requisitos é crucial para o bom entendimento do projeto. Através deste entendimento é possível reduzir a quantidade de incerteza inicial do projeto, pois nessa fase será convencionado o escopo do projeto, e também é a fase mais passível à ocorrência de falhas. Devido às incertezas dos gerentes que estão estimando o esforço de um projeto de software em sua fase inicial, uma estimativa precisa é praticamente impossível [16]. Em [16] também é afirmado que, à medida que o projeto avança no tempo, a incerteza sobre ele diminui, possibilitando melhores estimativas.

O “Cone da Incerteza” é definido por [16] para ilustrar que à medida que o projeto de software avança, a incerteza sobre ele diminui, proporcionando melhores estimativas de esforço. O cone é mostrado na Figura 1.1, onde o grau de incerteza na fase inicial é de 4x e a precisão da estimativa é de 0,25x. Porém ao decorrer do projeto, essas medidas diminuem. Entretanto, para que haja a diminuição da incerteza durante o avanço do projeto, é necessário um bom monitoramento de projeto [16]. Seguindo o conceito do cone da incerteza, caso não seja feito o

gerenciamento de cada etapa do projeto, podem produzir nuvens de incerteza, como é ilustrado na Figura 2.

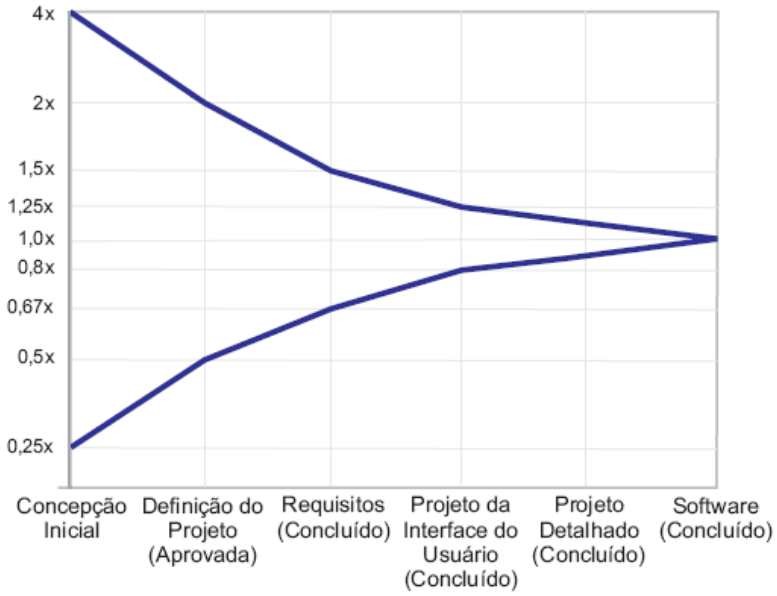

Figura 1. Incerteza X Etapas de desenvovimento do Software [16]

Através da Figura 2, podemos notar uma grande “mancha” no gráfico, que representa a falta de gerenciamento nas diversas etapas do projeto, ocasionando maior incerteza sobre o projeto. Como a incerteza do projeto ainda é elevada, isso reflete na precisão das estimativas, que tendem a ser menos precisas.

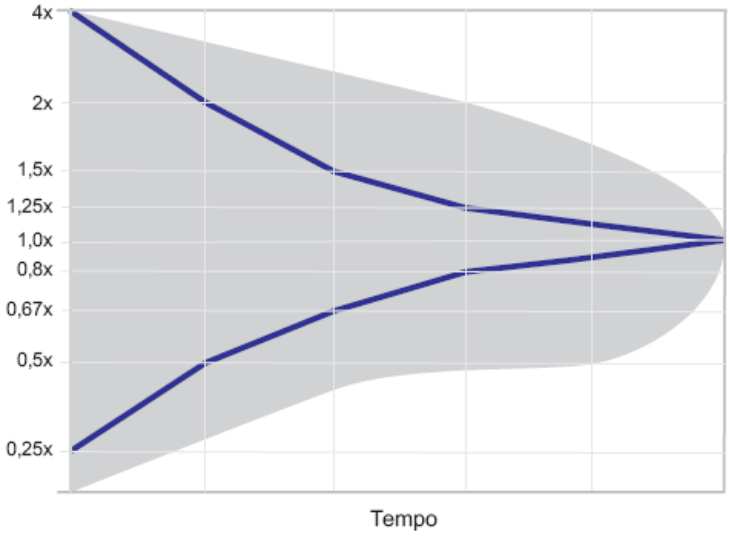

Figura 2. Nuvens de incerteza [16]

A estimativa de esforço é fundamental para o projeto de desenvolvimento de softwares nas empresas, pois viabiliza maior certeza do escopo do projeto em sua fase inicial, e conseqüentemente possibilita a entrega do produto com qualidade e dentro do prazo, garantindo assim a competitividade de mercado [25].

1.2 Motivação

Um dos motivos do sucesso das fábricas de software é o bom gerenciamento das suas atividades em todas as etapas. Para que a estimativa de esforço seja realizada, algumas métricas devem ser tomadas. Normalmente o tamanho do software [20] é utilizado como medida para a estimativa de esforço.

O tamanho do software não é a única métrica utilizada para se estimar esforço, também devem ser analisadas informações relevantes de projetos passados, para que seja possível estimar o custo de projetos futuros.

Dados da Associação Brasileira das Empresas de Software afirmam que o mercado brasileiro de software ocupa a 12ª posição no cenário mundial, tendo movimentado 5 bilhões de dólares em software em 2008, o que representou 1,68% do mercado mundial. E segundo o Ministério da Tecnologia em 2001 apenas 29% das empresas realizavam estimativas de tamanho e 45,7% realizavam estimativas de esforço de software [29].

Esses dados incentivam o desenvolvimento de novas técnicas que possibilitem melhores resultados, estimulando as empresas a utilizar a estimativa de esforço em projetos de software.

Técnicas de aprendizado de máquina têm sido muito utilizadas para o problema da estimativa de esforço [6,17]. Neste trabalho, propomos a aplicação da técnica Extreme Learning Machine para o treinamento de redes neurais. Entre as vantagens da sua utilização em relação a técnicas tradicionais, podemos citar a rapidez no treinamento, e boa capacidade de generalização.

1.3 Objetivos

Neste projeto aplicaremos o algoritmo Extreme Learning Machine[12] ao problema da estimativa de esforço em projetos de software. Na literatura, esta técnica ainda não foi utilizada no problema em questão.

Para o projeto estabelecemos os seguintes objetivos principais:

- Desenvolvimento do algoritmo ELM
- Aplicação do ELM ao problema de estimativa
- Execução de experimentos com as bases NASA, COCOMO e Desharnais.
- Análise e comparação dos resultados com outros trabalhos.

Através desse estudo, esperamos obter melhores estimativas de esforço em relação a projetos relacionados [6].

1.4 Estrutura da Monografia

Este trabalho está dividido em 5 capítulos. No primeiro capítulo foi iniciada a introdução ao tema da estimativa de esforço de software, assim como a motivação para pesquisas.

No capítulo 2, será dada ênfase ao problema da estimativa de esforço em engenharia de software. Abordaremos técnicas que possibilitam medir o tamanho de um software, visando estimar a quantidade de trabalho necessário para a conclusão do projeto. E também, serão apresentadas as bases de dados que serão utilizadas neste trabalho.

No capítulo 3, abordaremos o algoritmo de treinamento para redes neurais de uma única camada escondida, o Extreme Learning Machine. Primeiro explicaremos o algoritmo, em seguida mostramos um estudo comparativo realizado para comprovar a eficiência deste algoritmo em bases de dados conhecidas.

No capítulo 4, realizamos experimentos com o algoritmo explicado, utilizando as bases de estimativa de esforço, e também, realizaremos comparações com outros trabalhos realizados.

O capítulo 5 apresenta uma visão geral do que foi visto, e apresenta também propostas de trabalhos futuros.

Capítulo 2

Estimativa de Esforço de Software

Este capítulo abrange a importância da estimativa de esforço assim como as principais técnicas utilizadas pelas empresas para a medição de softwares.

2.1 Introdução

Em sistemas complexos, customizados, um erro grande de estimativa de software, pode fazer a diferença entre lucro e prejuízo [20].

A estimativa de esforço nunca será uma ciência exata [20], devido à imensa quantidade de variáveis que afetam o custo final do produto. De acordo com [4] 40% dos desenvolvedores de software continuam a fazer estimativas de esforço, e que o tamanho do software e o tempo de desenvolvimento são difíceis de estimar de forma precisa.

Pressman [20] relata que existem diversas formas de realizar estimativas mais confiáveis, uma delas é esperar até o final do projeto para fazer a estimativa. Nesse caso a espera é impraticável porque as estimativas são necessárias na fase inicial do projeto. Outra forma é se basear em estimativas de projetos antigos para estimar o projeto atual, que é a técnica que será utilizada neste trabalho. E também existe o uso de técnicas de decomposição para gerar estimativas de custo e esforço de projeto, que serão detalhadas na próxima seção.

A estimativa de esforço de software é um processo contínuo que pode ser aplicado em todas as etapas do ciclo de vida do projeto [1]. A estimativa de software engloba as seguintes etapas:

- Estimar Tamanho
- Estimar Custo e Esforço
- Estimar Cronograma
- Estimar Recursos Críticos Computacionais
- Avaliar Riscos

- Inspecionar/Aprovar Estimativas
- Reportar Resultados
- Medir e melhorar o processo

Para estimar o tamanho do software, primeiro deve-se executar a fase de análise de requisitos. Esta fase é crucial para o processo de estimativa, pois é onde o nível de incerteza sobre o projeto é mais elevado.

Antes que uma estimativa seja feita, o gerente deve entender bem o escopo do software a ser construído e gerar uma estimativa do seu tamanho.

2.2 Métricas de Tamanho de Software

O tamanho do software a ser desenvolvido é um importante insumo para a estimativa de esforço, pois a partir dele pode-se ter noção do trabalho necessário para a conclusão do projeto e pode ser medido utilizando as seguintes técnicas [20]:

- Métricas orientadas ao código fonte: diretamente relacionadas com o código fonte. Métrica por linhas de código (LOC) [20] é amplamente utilizada nas técnicas orientadas ao código fonte.
- Métricas Orientadas à Função: voltam-se para as funcionalidades que o software apresenta. O ponto de função [20] é a métrica mais utilizada entre as métricas orientadas à função.
- Métricas Orientadas a casos de uso: O UML [14] é amplamente utilizado para modelar os casos de uso. Para fazer a medição, a técnica de ponto por caso de uso, é necessário elaborar primeiramente um diagrama de casos de uso, para então realizar a medição.

2.2.1 Métrica por Linhas de Código

A métrica por linhas de código é bastante utilizada quando é necessário maior nível de detalhe sobre o tamanho do código [20], para isso é necessário realizar o delineamento do código a ser analisado. Quanto mais o software em questão for particionado, melhor e mais precisas serão as estimativas utilizando LOC.

No Entanto esta técnica depende da linguagem de programação a ser utilizada [20]. Como exemplo citamos o algoritmo de ordenação QuickSOrt, implementado na linguagem funcional Haskell [21], e na linguagem C [22] .

$$void\ QuickSort\ (char * item, int\ left, int\ right)$$

$$\{$$

$$register\ int\ i, j;$$

$$char\ x, y;$$

$$i = left; j = right;$$

$$x = item\left[\frac{left + right}{2}\right];$$

$$do\{$$

$$while\ (item[i] < x\ \&\&\ i < right)$$

$$i++;$$

$$while(x < item[j]\&\&\ j > left)$$

$$j++;$$

$$if(i \leq j)$$

$$\{$$

$$y = item[i];$$

$$item[j] = item[j];$$

$$item[j] = y;$$

$$i++;$$

$$j++;$$

$$\}$$

$$\}\ while(i \leq j);$$

$$if(left < j)$$

$$QuickSort(item, left, j);$$

$$if(i < right)$$

$$QuickSort(item, i, right);$$

```
}
```

Algoritmo QuickSort implementado em linguagem de programação C[22]

$$QuickSort[]=[]$$
$$QuickSort(x:xs) = QuickSort\ left\ ++[x]++QuickSort\ right$$
$$where$$
$$left\ = [y\ |y \leftarrow xs, y < x]$$
$$right = [y|y \leftarrow xs, y \geq x]$$

Algoritmo QuickSort Implementado em Haskell[21].

A partir dos algoritmos analisados, vemos que a mesma funcionalidade nas duas linguagens requer diferentes números de linhas de código, portanto em projetos onde há implementações utilizando linguagens de programação diferentes, poderá haver diferenças entre medições por linha de código.

2.2.2 Métrica de Pontos por Função

A métrica de pontos por função foi originalmente desenvolvida por Albrecht [2]. Utilizando dados históricos, esta técnica pode ser usada nas seguintes atividades [20]:

- Estimar o custo ou esforço necessário para projetar, codificar e testar o software.
- Prever o número de erros que vão ser encontrados durante o teste
- Prever o numero de componentes e ou numero de linhas de código projetadas.

Visto que a métrica por linhas de código é dependente da linguagem de programação, a abordagem de pontos por função visa a produtividade por funcionalidades apresentadas pelo software, tornando a métrica de pontos de função independente da linguagem de programação [30].

2.2.3 Métrica de Pontos por Caso de Uso

Os casos de uso descrevem funcionalidades e características que são requisitos básicos do sistema, sob o ponto de vista do usuário. Estas se assemelham com a métrica de pontos por função, tendo suas funcionalidades descritas pelo usuário. Porém, devido à alta abstração na descrição e pela falta de um padrão na elaboração de casos de uso o processo de desenvolvimento de estimativas utilizando casos de uso torna-se problemático [20].

A métrica por casos de uso também é independente da linguagem de programação utilizada

2.3 Bases de Dados

Nessa seção, iremos analisar as bases de dados que serão utilizadas. Selecionamos 3 bases de dados públicas de projetos de software [18], para realizarmos experimentos :

- COCOMO81
- Desharnais
- NASA

Essas bases são voltadas para o problema da estimativa de esforço, porém os atributos de cada base são diferentes.

2.3.1 COCOMO81

A base COCOMO81 contém informações sobre 63 projetos de software, e está disponível no repositório PROMISE [18]. Por padrão, os 63 projetos da base, são detalhados através de 19 atributos sendo **ADJKDSI** e **EFFORT** variáveis numéricas e os demais são definidos por categorias, em que cada um dos atributos possui um valor numérico.

Tabela 2.1. Variáveis da base de dados COCOMO81

Variáveis	Descrição
Rely	Confiabilidade do software
Data	Tamanho do banco de dados
Cplx	Complexidade do produto
Rvol	Volatilidade dos requisitos
Time	Restrição do tempo de execução
Stor	Restrição de armazenamento principal
Virt	volatilidade da máquina virtual
Turn	Tempo de execução da máquina
Acap	Capacidade do analista
Aexp	Experiência com aplicações
Pcap	Capacidade dos programadores
Vexp	Experiência com máquina virtual
Lexp	Experiência com linguagem de programação
Modp	Uso de práticas modernas de programação
Tool	Uso de ferramentas de software
Sched	Cronograma de desenvolvimento requerido
Mode	Modelo de desenvolvimento adotado
ADJKDSI	Número de linhas de código
Effort	Quantidade de esforço para o desenvolvimento do software

Entre os 19 atributos da base de dados COCOMO81, 16 deles são conhecidos como fatores de ajuste de esforço. Cada um desses fatores é classificado com valores numéricos que correspondem a uma escala: muito baixo, baixo, normal, alto, muito alto e extremamente alto.

Tabela 2.2. Classificação dos atributos da base COCOMO

Fatores de Ajuste de esforço	Classificação					
	Muito Baixo	Baixo	Médio	Alto	Muito Alto	Extremamente Alto
Rely	0.75	0.88	1.00	1.15	1.40	–
Data	–	0.94	1.00	1.08	1.16	–
Cplx	0.70	0.85	1.00	1.15	1.30	1.65
Rvol	–	0.91	1.00	1.19	1.38	1.62
Time	–	–	1.00	1.11	1.30	1.66
Stor	–	–	1.00	1.06	1.21	1.56
Virt	–	0.87	1.00	1.15	1.30	–
Turn	–	0.87	1.00	1.07	1.15	–
Acap	1.46	1.19	1.00	0.86	0.71	–
Aexp	1.29	1.13	1.00	0.91	0.82	–
Pcap	1.42	1.17	1.00	0.86	0.70	–
Vexp	1.21	1.10	1.00	0.90	–	–
Lexp	1.14	1.07	1.00	0.95	–	–
Modp	1.24	1.10	1.00	0.91	0.82	–
Tool	1.24	1.10	1.00	0.91	0.83	–
Sched	1.23	1.08	1.00	1.04	1.10	–

2.3.2 Desharnais

A base e dados Desharnais contém informações sobre 81 projetos de software desenvolvidos por empresas canadenses entre 1982 e 1988 [18]. É composta por 11 atributos, sendo que 9 deles são independentes, e 2 independentes *Effort* e *Length*. Os atributos são detalhados na tabela 2.5.

Tabela 2.3 Variáveis da base de dados Desharnais

Variáveis	Descrição
TeamExp	Experiência da equipe em anos
ManagerExp	Experiência do gerente do projeto em anos
YearEnd	Ano de conclusão do projeto
Transactions	Número de transações processadas
Entities	Número de entidades
PointsAdjust	Pontos de função ajustados
PointsNonAdjust	Pontos de função não ajustados
Envergure	Medida de complexidade derivada por fatores definidos pelo ambiente.
Language	Linguagem de programação adotada
Length	Duração do projeto em meses
Effort	Esforço medido em homens/hora

2.3.3 NASA

A base de dados é composta por 18 projetos oriundos da seção de desenvolvimento de sistemas da NASA (Goddard *Space Flight Center*). Cada projeto levou cerca de dois meses a dois anos para ser concluído, contando de 2 a 10 desenvolvedores para cada projeto.

A base possui dois atributos independentes (linhas desenvolvidas e metodologia), e uma variável dependente (esforço), mostradas na tabela 2.3.

Tabela 2.4. Variáveis da base de dados NASA

Variáveis	Descrição
Developed Lines	Total de linhas de código desenvolvidas
Methodology	Metodologia de desenvolvimento de software utilizada
Effort	Esforço medido em homem/mês

2.4 Comentários Finais

Neste capítulo, analisamos o problema da estimativa de esforço, assim como as etapas mais importantes para o processo de estimativa. Vimos as principais métricas utilizadas para medir o tamanho do software e também as bases de dados que serão utilizadas para simulações.

Capítulo 3

Extreme Learning Machine

3.1 Introdução

Muitos algoritmos de treinamento de RNAs são baseados em gradiente descendente. Nos últimos anos, esses algoritmos foram utilizados em diversas tarefas como otimizações e buscas. No entanto, os algoritmos baseados em gradiente descendente são geralmente lentos e facilmente convergem para mínimos locais. O treinamento é realizado iterativamente para conseguir uma melhor generalização, o que resulta em longos intervalos de tempo para treinar a rede e, além disso, na maioria dos algoritmos tradicionais de treinamento tais como *backpropagation* (BP), todos os parâmetros das redes, ou seja, todos os pesos e bias devem ser ajustados durante o treinamento.

No algoritmo *Extreme Learning Machine* (ELM) não é necessário ajustar os pesos de entrada e os bias da primeira camada escondida [12]. De fato, alguns experimentos [11] mostraram que, com as atribuições aleatórias destes parâmetros, o treinamento se torna mais rápido, e a rede produzida também atinge boa capacidade de generalização.

O método ELM proposto recentemente foi projetado para minimizar alguns problemas que ocorrem freqüentemente em algoritmos de aprendizagem baseados em gradiente descendente como o BP.

Na Aprendizagem do algoritmo *Extreme Learning Machine*, os pesos de entrada e os bias da camada escondida são atribuídos aleatoriamente, e os pesos de saída são determinados analiticamente. O algoritmo ELM só é aplicável em RNAs tais como MLPs, porém com uma única camada escondida [12]. Esta não é uma limitação importante na prática, pois em estudos realizados por [26] foi provado que redes neurais com apenas uma camada podem aproximar qualquer função contínua, e também podem ser aplicadas em muitas aplicações de classificação e regressão.

O ELM é um algoritmo de treinamento rápido que fornece baixos erros de treinamento, assim como boa capacidade de generalização. É baseado puramente em operações matriciais, e para seu pleno desenvolvimento utiliza a Matriz generalizada inversa de Moore-Penrose.

O ELM pode ser resumido em três etapas básicas: primeiro, geram-se aleatoriamente os pesos de entrada e bias dos neurônios escondidos. Em seguida, calcula-se a matriz de saída da camada escondida [23]. Finalmente, a matriz modificada é usada para obter a matriz de peso de saída, utilizando-se de Matrix generalizada inversa de Moore-Penrose.

3.2 Treinamento de Redes ELM

Dado N amostras de treinamento (Xi, Ti), onde $x_i = [x_{i1}, x_{i2}, \dots, x_{in}]^T \in R^n$ e $t_i = [t_{i1}, t_{i2}, \dots, t_{in}]^T \in R^n$, uma RNA com uma camada escondida e os N neurônios escondidos podem ser matematicamente representados como:

$$\sum_{i=1}^{\tilde{N}} \beta_i g_i\left(x_j\right) = \sum_{i=1}^{\tilde{N}} \beta_i g(w_i \cdot x_j + b_i) = o_j$$

Onde $j = 1, \dots, N$, $w_i = [w_{i1}, w_{i2}, \dots, w_{in}]^T$ é o vetor de peso que conecta o $i^{ésimo}$ neurônio escondido com os neurônios de entrada, $g(x)$ é a função sigmóide $g(x) = \frac{1}{1+e^{-x}}$, $\beta_i = [\beta_{i1}, \beta_{i2}, \dots, \beta_{im}]^T$ é o vetor de peso que conecta o $i^{ésimo}$ neurônio escondido para os neurônios de saída, b_i é o bias do $i^{ésimo}$ neurônio escondido e $w_i \cdot x_i$ é o produto interno entre w_i e x_i. Para tornar as saídas da rede em relação de igualdade com os resultados esperados, em outras palavras, para a realização de treinamento de erro igual a zero, deve haver $\beta_i, w_i \; and \; b_i$ de modo que a seguinte equação pode ser escrita como:

$$\sum_{i=1}^{\tilde{N}} \beta_i g_i\left(x_j\right) = \sum_{i=1}^{\tilde{N}} \beta_i g(w_i \cdot x_j + b_i) = t_j$$

Os algoritmos tradicionais de treinamento consistem na otimização dos pesos de entrada w_i, os pesos de saída β_i, e os bias b_i, de modo que a diferença entre a saída o_j obtidos e as t_j saídas esperadas também possam ser otimizadas. A grande diferença do ELM de algoritmos de treinamento tradicional é exatamente esta otimização.

A atribuição aleatória dos pesos w_i de entrada e os bias b_i, o algoritmo de treinamento ELM calcula a solução B do sistema linear $H\beta = T$, usando o método dos quadrados mínimos, onde

$$H = \begin{bmatrix} g(w_1 \cdot x_1 + b_1) & \cdots & g(w_{\tilde{N}} \cdot x_1 + b_{\tilde{N}}) \\ \vdots & \ddots & \vdots \\ g(w_1 \cdot x_N + b_1) & \cdots & g(w_{\tilde{N}} \cdot x_N + b_{\tilde{N}}) \end{bmatrix}_{NX\tilde{N}},$$

$$\beta = \begin{bmatrix} \beta_1^T \\ . \\ . \\ . \\ \beta_{\tilde{N}}^T \end{bmatrix}_{\tilde{N}Xm} \quad \text{e}$$

$$T = \begin{bmatrix} t_1^T \\ . \\ . \\ . \\ t_N^T \end{bmatrix}_{NXm}.$$

Ficou provado em [7] que a solução de norma mais baixa pelo método de quadrados mínimos para o sistema linear $H\beta = T$ é dada pela Eq. (6).

$$B = H^{\dagger}T \qquad (6)$$

Onde $H^{\dagger}$ é a matriz inversa generalizada de Moore-Penrose [19] da matriz H. Assim, o ELM lida analiticamente com o processo de treinamento como uma resolução de um sistema não-linear. Através da solução do sistema é possível determinaros pesos de saída através do cálculo da matriz inversa generalizada(Matriz Inversa de Moore Penrose) da matriz de saída da camada escondida.

Portanto, o algoritmo ELM pode ser resumido em três etapas principais (mostrado no seguinte algoritmo): (i) geração aleatória dos pesos de entrada e vieses da camada escondida (linhas 1 a 3), (ii) cálculo das saídas da camada escondida H (linha 5) e (iii) cálculo da *Moore-Penrose* da matriz inversa generalizada de H (linha 6).

$$Entrada: \ Conjunto\ de\ treinamento: \aleph = \{(x_i, t_i) | \ x_i \in R^n, t_i \in R^m, i = 1\ldots,N, função\ de\ ativação\ g x \tilde{N}\ neuronios\ escondidos$$

1. $for\ i \leftarrow 1\ to\ \tilde{N}\ do$
2. $w_i \leftarrow valor\ aleatório;$
3. $b_i \leftarrow valor\ aleatório;$
4. $end\ for$
5. $calcular\ a\ matriz\ H\ de\ saída\ da\ camada\ escondida;$
6. $calcular\ os\ pesos\ de\ saída\ \beta, onde$
   $\beta = H^{\dagger}T\ and\ T = [t_1 \ldots t_N]^T$

Neste trabalho foi utilizada a versão original do algoritmo ELM [12], porém existem muitas melhorias recentes, como as descritas em [15,13].

3.3 Técnicas para Avaliação de Desempenho

Logo após a conclusão do treinamento da rede neural, é bastante útil medir o desempenho obtido no conjunto de teste, pois, os exemplos deste conjunto não foram apresentados durante o treinamento, logo a avaliação será imparcial [27].

As métricas mais utilizadas são:

- *Holdout*
- Validação Cruzada

3.3.1 Holdout

Neste método, os dados que serão utilizados são divididos em dois conjuntos disjuntos (normalmente utiliza-se dois terços para o treinamento e um terço para o teste), chamados de conjunto de treinamento e conjunto de teste [27].

Durante o treinamento usa-se o conjunto de treinamento, para que a rede neural conclua o processo de aprendizado. Após o treinamento, o desempenho da rede é avaliado utilizando o conjunto de teste.

Esta técnica possui algumas desvantagens conhecidas. Quando os dados são particionados, nem todos os exemplos estão disponíveis para o treinamento, pois parte deles foi alocada para o conjunto de testes. Se o conjunto de dados for pequeno, então uma pequena porção deles estará disponível para realizar o treinamento, causando maior variância na rede [27]. E também o treinamento pode ter um desempenho menor do que seria obtido utilizando todos os dados disponíveis [27].

3.3.2 Validação Cruzada

O método de validação cruzada de K partições (k-*folds)* , divide o conjunto de dados original em K conjuntos de igual tamanho. Durante cada execução, um conjunto é escolhido para ser usado como conjunto de teste, enquanto que os demais são usados durante o treinamento.

Este processo é repetido K vezes, até que cada conjunto tenha sido utilizado como conjunto de testes, e o erro obtido é igual a média dos erros de todas as K execuções[27].

Um caso especial deste método é quando o conjunto de testes é constituído de apenas um exemplo. Esse caso chama-se de deixe-um-fora [27] (Leave-One-Out).

A vantagem deste método é que como os dados são divididos em k partições, há um aproveitamento de todos os dados durante o treinamento. Porém, como o procedimento é repetido K vezes, seu custo computacional é mais elevado.

3.4 Estudo Comparativo

Nesta seção iremos, discutir algumas simulações realizadas para medir o desempenho do algoritmo ELM em bases de dados conhecidas do UCI [3], compará-lo com outros algoritmos desenvolvidos e verificar a sua velocidade de treinamento.

Para as comparações, utilizamos os algoritmos IPSONET [32] e MLP-PSO [28]. Ambos os algoritmos, realizam a otimização da arquitetura da rede, então, como estamos utilizando a versão básica do ELM, executamos simulações com 8,13 e 20 neurônios nas bases apresentadas, e observamos o número de neurônios escondidos que produzisse a melhor taxa de acerto no conjunto de testes de cada base.

Tabela 3.1. Número de Neurônios utilizados em cada base.

Base de dados	Neurônios Escondidos
Card	20
Cancer	8
Diabetes	8
German	20
Heart	20
Pima	20
Iris	13

As bases utilizadas remetem problemas de classificação, porém nossa intenção nesta seção é avaliar o desempenho do ELM em termos da sua taxa de acerto e sua velocidade de treinamento.

As simulações para todos os conjuntos de dados foram realizadas com 5-*fold cross-validation* para fornecer uma comparação justa com os resultados do algortimo IPSONET, que usou essa configuração experimental. Os resultados de precisão de

classificação aqui relatados correspondem à média ao longo dos cinco folds (e o desvio-padrão). Em 5-*fold cross-validation*, o conjunto de dados é dividido em cinco subgrupos com o mesmo número de padrões. Em seguida, cinco simulações são executadas, em cada simulação um subconjunto diferente é usado como o conjunto de teste e os subconjuntos restantes são juntados para formar o conjunto de treinamento. A precisão do 5-*fold cross-validation* é a média das cinco precisões medidas em cada simulação (ou seja, em cada subgrupo).

O algoritmo ELM não precisa de um conjunto de validação [12]. Portanto, para cada *fold* (simulação) no 5-*fold cross-validation*, o conjunto de treinamento foi utilizado para ajustar os pesos de saída da rede e do conjunto de teste foi utilizado para medir a generalização da rede (ou seja, para calcular a precisão).

Tabela 3.2. Taxa de acerto e desvio padrão sobre conjunto de teste.

Data Set	**IPSONET**	**ELM**	**MLP-PSO**
Card	86.83±2.7	85.95±2.8	86.45±1.5
Cancer	97.07±0.5	96.70±1.9	98.68±0.6
Diabetes	77.63±0.7	77.48±2.1	76.22±2.2
German	75.52±3.6	71.60±1.3	70.80±2.3
Heart	82.62±5.2	81.49±1.6	82.54±2.5
Iris	96.00±3.5	98.00±1.6	83.34±15.2
Pima	76.68±2.0	76.46±3.8	73.97±2.7

Tabela 3.3. Tempo necessário para o treinamento

	Tempo de Treinamento	
Data Set	**ELM**	**MLP-PSO**
Card	0.9410	56.00
Cancer	0.3141	20.70
Diabetes	0.3723	24.43
German	2.0830	106.00
Heart	0.0577	10.76
Iris	0.0460	11.00
Pima	0.7860	59.00

Tabela 2 mostra a 5-*fold cross-validation* precisão para cada algoritmo e o conjunto de dados. Estes resultados mostram que os três algoritmos possuem precisão e desvio padrão muito semelhantes. Porém o ELM, não realizou nenhum tipo de otimização de parâmetros, ao contrário do IPSONET e o MLP-PSO. E também o ELM obteve tempos de treinamento mais baixos, portanto concluímos que o ELM mostrou-se superior aos dois algoritmos comparados, pois além de conseguir resultados similares, o treinamento é mais rápido que o MLP-PSO.

3.5 Regressão Com Extreme Learning Machine

Nesta seção iremos abordar conceitos básicos de regressão, e explicar como a regressão é feita no ELM.

A previsão é o principal propósito dos modelos de regressão. O objetivo dos modelos de regressão é construir uma função $f(x)$, a partir de um conjunto de variáveis independentes, para estimar o valor das variáveis dependentes [5]. Nosso objetivo é aplicar dados de projetos passados (variáveis independentes) de forma que seja possível realizar a estimativa de esforço de projetos.

O algoritmo ELM também pode ser aplicado em problemas de regressão. O treinamento da rede ocorre da mesma forma como foi mostrado. Segundo [12], as funções de ativação utilizadas nas camadas escondida e de saída são respectivamente, sigmóide e linear.

Um detalhe importante dessa abordagem é a necessidade de realizar a normalização das variáveis dependentes [12]. Para normalizar os dados de saída, utilizamos uma função de transformação linear:

$$f(x) = \frac{(b-a)(x_i - x_{min})}{(x_{max} - x_{min})} + a$$

onde: $f(x)$ é o valor normalizado x_i é o valor original, x_{min} é o valor mínimo da variável, x_{max} é o valor máximo da variável, a e b são os limites de normalização.

Para a visualização dos resultados obtidos, é feita a operação inversa através da função:

$$f(x) = \frac{(x_i - a)(x_{max} - x_{min})}{(b - a)} + x_{min}$$

A única diferença da função anterior, é que o x_i é o valor normalizado e $f(x)$ é o valor real.

Ressaltamos que para realizar a operação inversa, os limites devem ser os mesmos que foram estabelecidos durante a normalização.

3.6 Medidas de Desempenho

Para o problema da estimativa de esforço em projetos de software, utilizaremos as seguintes medidas de desempenho bastante utilizadas na literatura [17][24]:

- PRED
- MMRE

3.6.1 PRED

O PRED(x) é uma medida que analisa quantas estimativas se encontram dentro de um intervalo de erro aceitável x. Na literatura, é comum utilizar um valor de MMRE 0.25 como erro aceitável, neste trabalho utilizaremos o PRED(25).

E PRED é definido a seguir:

$$PRED(x) = \frac{1}{N} \sum_{i=1}^{N} \begin{cases} 1, se \ \frac{|Y_i^{estimado} - Y_i^{original}|}{Y_i^{original}} \leq \frac{x}{100} \\ 0, caso\ contrário \end{cases}$$

Onde: N é o valor total de exemplos no conjunto de testes, $Y_i^{estimado}$ é a estimativa de esforço estimada, $Y_i^{original}$ é a estimativa original e x é a medida de erro aceitável.

3.6.2 MMRE

Entre as duas medidas de desempenho utilizadas em estimativa de esforço de software, a mais usada é a MMRE (*Mean Magnitude Relative Error*). Conte et al. [8] considera MMRE<=0.25 um nível de precisão aceitável para modelos de estimativa de esforço.

O MMRE é definido a seguir:

$$MMRE = \frac{1}{N}\sum_{i=1}^{N}\frac{|Y_i^{estimado} - Y_i^{original}|}{Y_i^{original}}$$

Onde, os valores são idênticos aos definidos na equação do PRED.

Capítulo 4

Experimentos

4.1 Introdução

Neste capítulo descreveremos como os experimentos foram conduzidos nas bases de dados especificadas, juntamente com a análise dos resultados obtidos.

Durante a realização dos experimentos verificamos que diferentes normalizações das variáveis dependentes possibilitavam resultados diferenciados, então realizamos um estudo da influência da normalização nos resultados do ELM.

Em todas as simulações as entradas foram normalizadas no intervalo [0,1] e variamos a normalização das variáveis para possibilitar a nossa análise. As comparações são realizadas com diversas técnicas presentes em [6], cujos parâmetros são especificados na tabela abaixo.

Tabela 4.1. Parâmetros das técnicas presentes em [6] e do ELM

Técnicas	Parâmetros	Valores dos Parâmetros
M5P	R - (true) árvore de regressão ou (false) árvore de modelo	{true, false}
	I - Número mínimo de instâncias na folha	[4,15]
	P - Podar/Não podar a árvore	{true, false}
	S - Suavizar/Não suavizar a árvore	{true, false}
MLP	N - Número de neurônios na camada escondida	[5,15]
	E - Número de épocas do treinamento	{2000,2500}
	L - Taxa de aprendizado para o algoritmo backpropagation	$\{10^{-2}\}$
	M - Momentum para o algoritmo backpropagation	{0.2,.0.3,0.4,0.5,0.6}
RBF	N - Número de clusters para gerar o K-Means	[4,15]
	D - O mínimo desvio padrão para os clusters	$\{10^{-1}\}$
SVR RBF	C - parâmetro de complexidade	{1,10,100,1000}
	ε- limite que os desvios são tolerados	$\{10^{-3}, 10^{-4}, 10^{-5}\}$
	γ - parâmetro do kernel	$\{1, 10^{-1}, 10^{-2}\}$
SVR Linear	C - parâmetro de complexidade	{1,10,100,1000}
	ε - limite que os desvios são tolerados	$\{10^{-3}, 10^{-4}, 10^{-5}\}$
	S - tamanho de cada versão do conjunto de dados do	[70,100]

Bagging	treinamento I - número de iterações C - classificador ou modelo de regressão a ser usado	 [5,30] {LR, SVR, MLP, M5P, RBF}
AG com M5P	R - (true) árvore de regressão ou (false) árvore de modelo I - Número mínimo de instâncias na folha P - Podar/Não podar a árvore S - Suavizar/Não suavizar a árvore	{true, false} [4,15] {true, false} {true, false}
AG com MLP	N - Número de neurônios na camada escondida E - Número de épocas do treinamento L - Taxa de aprendizado para o algoritmo backpropagation M - Momentum para o algoritmo backpropagation	[5,20] [500,2500] $[10^{-4}, 10^{-1}]$ [0.1,0.8]
AG com SVR RBF	C - parâmetro de complexidade ε - limite que os desvios são tolerados γ - parâmetro do kernel	$[10^{-2}, 10^{3}]$ $[10^{-7}, 10^{-1}]$ $[10^{-5}, 0.5]$
AG com SVR Linear	 C - parâmetro de complexidade ε - limite que os desvios são tolerados	 $[10^{-2}, 10^{3}]$ $[10^{-7}, 10^{-1}]$
ELM	N- número de neurônios na camada escondida	[5,15]

4.2 Experimentos com a base Desharnais

A base de dados desharnais, como já mencionado anteriormente possui dados referentes a 81 projetos de software, contando com 10 variáveis, sendo 9 independentes e uma independente que se refere ao esforço envolvido no projeto. Em nossas simulações dividimos a base em dois conjuntos, um conjunto de treinamento e um conjunto de teste.

Para o conjunto de testes foram escolhidos aleatoriamente 18 projetos dos 81 pertencentes a base. Os projetos selecionados foram os seguintes: 2, 8, 12, 15, 19, 20, 22, 35, 38, 42, 46, 50, 56, 61, 63, 72, 76, 80, cuja numeração corresponde ao identificador do projeto na base de dados. Os 63 projetos restantes foram alocados para o conjunto de treinamento. Este mesmo procedimento foi utilizado por [6][7]. Esta base possui quatro projetos que possuem atributos de entrada ausentes, nós utilizamos um filtro o WEKA[31] para substituir os valores.

Para avaliar o desempenho do modelo utilizamos os seguintes medidores de desempenho: MMRE e PRED(25). Para selecionar o melhor modelo, executamos o algoritmo ELM 1000 vezes, e selecionamos o modelo que possuísse maior PRED(25) e menor MMRE.

Antes de selecionar o melhor modelo para esta base, primeiro realizamos simulações para analisar a influência da normalização das variáveis de saída nos resultados do ELM. Enfatizamos que a normalização das variáveis de saída é aplicada durante o treinamento da rede. A seleção do modelo é feita utilizando o conjunto de testes, com os dados reais (não normalizados).

Tabela 4.2. Resultados sem normalização das variáveis dependentes.

	5 Neurônios	10 Neurônios	15 Neurônios
PRED(25)	61,12	61,12	61,12
MMRE	0,4162	0,3831	0,2496

Tabela 4.3. Resultados com normalização no intervalo [0.3,0.7]

	5 Neurônios	10 Neurônios	15 Neurônios
PRED(25)	61,12	66,67	66,67
MMRE	0,4284	0,2108	0,281

Tabela 4.4. Resultados com normalização no intervalo [0.4,0.6]

	5 Neurônios	10 Neurônios	15 Neurônios
PRED(25)	61,12	72,22	72,22
MMRE	0,3533	0,2206	0,2645

Através das simulações realizadas, verificamos que a normalização das variáveis de saída afeta bastante os resultados obtidos. A simulação com 10 neurônios escondidos e normalização de saídas entre 0.4 e 0.6 produziu o melhor modelo de rede neural para o problema. E será o modelo utilizado para comparações com outros trabalhos.

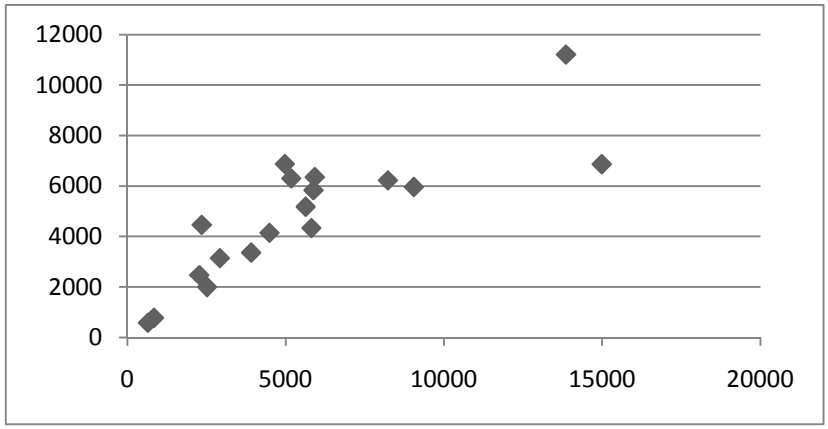

Figura 3. Distribuição de valores do melhor modelo gerado na base Desharnais.

O modelo de regressão gerado pode ser observado no gráfico de dispersão acima. Analisando o gráfico, observamos muitos pontos desalinhados, porém o MMRE obtido foi bastante satisfatório. Segundo [8], para valores de MMRE <=0.25, o modelo gerado é considerado acurado.

Tabela 4.5. Comparação do melhor resultado do ELM com os obtidos por [6].

Técnicas/Parâmetros	PRED(25)	MMRE
SVR Linear (C = 10^3, $\varepsilon = 10^{-3}$)	50.00	0.4898
SVR RBF (C =10, $\varepsilon = 10^{-3}$,$\gamma = 10^{-2}$)	55.56	0.4736
MLP (L = 10^{-2}, M = 0.3, N = 9, E = 2x10^3)	66.67	0.4069
M5P (R = true, P = true, S = false, N = 8) / RT	55.56	0.6135
M5P (R = false, P = true, S = false, N = 9) / MT	55.56	0.7545
RBF (N = 6, D = 0.1)	44.44	0.7550
Bagging (S = 88, C = MLP (L = 10^{-2}, M = 0.6, N = 15, E = 2x10^3), I = 10)	72.22	0.3778
Bagging (S = 81, C = M5P (R = true, P = true, S = true, N = 5), I = 9) / RT	72.22	0.4888
Bagging (S = 81, C = M5P (R = false, P = true, S = true, N = 5), I = 9) / MT	72.22	0.4888
Bagging (S = 83, C = SVR Linear (C = 10^3, $\varepsilon = 10^{-4}$), I = 5)	61.11	0.4226
Bagging (S = 85, C = SVR RBF (C = 10^2, ε = 10^{-3}, γ = 1), I = 5)	61.11	0.4153
Bagging (S = 91, C = RBF (N = 9, D = 0.1), I = 13)	66.67	0.8775
AG com SVR RBF	**76.11**	0.3571
AG com SVR linear	66.67	0.3583
AG com MLP	72.78	0.2967
AG com M5P	61.11	0.5945
ELM (N=10)	72.22	**0.2206**

Nossos resultados foram comparados com outros estudos realizados em estimativa de esforço de software. A tabela 4.4, mostra as comparações realizadas com o

trabalho desenvolvido por [6], onde foram empregadas várias técnicas para regressão.

A partir da tabela notamos que o ELM, obteve o menor MMRE, ou seja, a maioria das estimativas realizadas foram mais próximas aos valores reais.

4.3 Experimentos com a base NASA

A base de dados NASA (*National Aeronautics and Space Administration*), consiste de dois atributos, linhas desenvolvidas (*Developed Lines*) e metodologia (*Methodology*). Possui uma variável de saída: esforço (*Effort*).

Em nossas simulações utilizamos duas medidas de desempenho bastante utilizadas na literatura [17][24]. Para esta base a técnica utilizada foi a *leave-one-out* cross validation (LOOCV)[27][10], e o resultado foi dado pela média das execuções.

Tabela 4.6. Resultados sem normalização das variáveis dependentes.

	5 Neurônios	10 Neurônios	15 Neurônios
PRED(25)	94.44	94.44	72.22
MMRE	0.0312	0.0543	0.3233

Tabela 4.7. Resultados com normalização no intervalo [0.3,0.7].

	5 Neurônios	10 Neurônios	15 Neurônios
PRED(25)	94.44	94.44	77.77
MMRE	0.0299	0.0416	0.2944

Tabela 4.8. Resultados com normalização no intervalo [0.4,0.6].

	5 Neurônios	10 Neurônios	15 Neurônios
PRED(25)	94.44	94.44	77.77
MMRE	0.0139	0.0324	0.1120

Os resultados obtidos alcançaram baixos erros, sendo a simulação com normalização de saídas entre [0.4,0.6] com 5 neurônios na camada escondida, que obteve melhor desempenho, e será o modelo utilizado para comparações com outros trabalhos.

Na tabela 4.8 podemos observar as comparações realizadas com o trabalho desenvolvido por [6]. Observa-se que as taxas de PRED entre o ELM e o método proposto por [6] obtiveram valores iguais, porém o MMRE foi significativamente menor utilizando o ELM.

Tabela 4.9. Comparação do melhor resultado do ELM com os obtidos por [6]

Técnicas/Parâmetros	PRED(25)	MMRE
RBF-SG ($\delta = 0.1\%, \sigma = 0.85,\ m = 7$)	72.22	0.1907
RBF ($N = 18, D = 0.1$)	77.78	0.3945
Regressão Linear Múltipla	77.22	0.2330
SVR Linear ($C = 10, \varepsilon = 10^{-4}$)	88.89	0.1650
SVR RBF ($C = 10^2, \varepsilon = 10^{-3}, \gamma = 10^{-2}$)	83.33	0.1780
MLP ($L = 10^{-2},\ M = 0.2,\ N = 6, E = 2x10^3$)	**94.44**	0.2030
M5P ($R = true, P = true\ , S = true, N = 4)/RT$	66.67	0.2861
M5P ($R = true, P = true\ , S = false, N = 7)/MT$	83.33	0.1778
Bagging ($S = 77,\ C = Regressão\ Linear\ Múltipla, I = 9$)	88.89	0.1941
Bagging ($S = 89,\ C = MLP\ (L = 10^{-2},\ M = 0.3,\ N = 7, E = 2x10^3), I = 9$)	**94.44**	0.1735
Bagging ($S = 100,\ C = M5P\ (R = true, P = true\ ,\ S = true, N = 5), I = 17)/RT$)	77.78	0.2715
Bagging ($S = 83,\ C = M5P\ (R = false, P = false, S = true, N = 4), I = 13)/MT$)	88.89	0.1375
Bagging ($S = 70,\ C = SVR\ Linear(C = 10, \varepsilon = 10^{-3}), \mathrm{I} = 8$)	**94.44**	0.1590
Bagging ($S = 70, C = SVR\ RBF(C = 10, \varepsilon = 10^{-3}, \gamma = 1), \mathrm{I} = 8$)	**94.44**	0.1590
Bagging ($S = 77, C = RBF(N = 12, D = 0.1), \mathrm{I} = 26$)	83.33	0.3639
AG com SVR RBF	**94.44**	0.1624
AG com SVR linear	**94.44**	0.1636
AG com MLP	**94.44**	0.1873
AG com M5P	83.33	0.1778
ELM (N=5)	**94.44**	**0.0139**

4.4 Experimentos com a base Cocomo

A base Cocomo, é constituída de atributos que se classificam de acordo com o impacto que cada um gera no esforço necessário para projeto de software . O impacto é classificado em níveis, são eles: *Baixo, Muito Baixo, Médio, Alto, Muito Alto, Extremamente Alto*. Cada um possui um valor numérico associado que depende de cada atributo, como já foi visto na tabela 2.2.

Para nossos experimentos com a base de dados COCOMO, realizamos 6 simulações diferentes, dividindo a base de dados em 6 conjuntos de treinamento e

teste. Essa configuração de simulação foi selecionada visando realizar comparações justas com os trabalhos desenvolvidos por [6] e [29].

Na tabela a seguir mostramos a divisão dos conjuntos de treinamento e teste da base COCOMO.

Tabela 4.10. Divisão da base COCOMO em seis conjuntos diferentes.

Base de dados	Conjunto de Testes	Conjunto de Treinamento
1	1,7,13,19,25,31,37,43,49,55,61	Projetos Restantes
2	2,8,14,20,26,32,38,44,50,56,62	Projetos Restantes
3	3,9,15,21,27,33,39,45,51,57,63	Projetos Restantes
4	4,10,16,22,28,34,40,46 52,58	Projetos Restantes
5	5,11,17,23,29,35,41,47,53,59	Projetos Restantes
6	6,12,18,24,30,36,42,48,54,60	Projetos Restantes

Para esta base, realizamos simulações com normalizações das variáveis de saída entre [0.3,0.7], [0.4,0.6], e sem normalização alguma. Para cada simulação realizamos 1000 execuções do algoritmo ELM, e escolhemos o melhor modelo para as comparações.

Tabela 4.11. Resultados sem normalização das variáveis dependentes utilizando a base de dados COCOMO.

	5 Neurônios		10 Neurônios		15 Neurônios	
Conjunto	PRED(25)	MMRE	PRED(25)	MMRE	PRED(25)	MMRE
1	100.00	0.1161	100.00	0.1048	100.00	0.1093
2	100.00	0.1231	100.00	0.1083	100.00	0.0970
3	100.00	0.1407	100.00	0.0796	100.00	0.0984
4	100.00	0.1084	100.00	0.0989	100.00	0.0639
5	90.00	0.1782	90.00	0.1615	90.00	0.1460
6	90.00	0.1887	100.00	0.1512	100.00	0.1215

Tabela 4.12. Resultados com normalização entre o intervalo [0.3,0.7] das variáveis dependentes utilizando a base de dados COCOMO

	5 Neurônios		10 Neurônios		15 Neurônios	
Conjunto	PRED(25)	MMRE	PRED(25)	MMRE	PRED(25)	MMRE
1	100.00	0.1398	100.00	0.1196	100.00	0.0986
2	100.00	0.1579	100.00	0.0501	100.00	0.0758
3	100.00	0.1171	100.00	0.0883	100.00	0.0828
4	100.00	0.1015	100.00	0.0985	100.00	0.0940
5	90.00	0.1898	100.00	0.1354	100.00	0.0939
6	100.00	0.1337	100.00	0.0976	100.00	0.1602

Tabela 4.13. Resultados com normalização entre o intervalo [0.4,0.6] das variáveis dependentes utilizando a base de dados COCOMO.

	5 Neurônios		10 Neurônios		15 Neurônios	
Conjunto	PRED(25)	MMRE	PRED(25)	MMRE	PRED(25)	MMRE
1	100.00	0.1168	100.00	0.1014	100.00	0.0794
2	100.00	0.1105	100.00	0.0904	100.00	0.0791
3	100.00	0.1062	100.00	0.1004	100.00	0.0883
4	100.00	0.1077	100.00	0.1025	100.00	0.0964
5	80.00	0.2561	90.00	0.2134	90.00	0.1544
6	90.00	0.0844	100.00	0.1088	100.00	0.0720

A Tabela 4.13 compara os melhores resultados, em termos do MMRE, obtidos nas nossas simulações com os resultados obtidos por [6] e [29]. Esses resultados mostram que o ELM melhorou os resultados obtidos, em todos os conjuntos testados.

Tabela 4.14. Comparação dos resultados obtidos por [6] e [29] com os resultados obtidos pelo ELM em termos do MMRE.

Base de dados	Rede Neural Artificial [29]	AG com SVR Linear [6]	AG com MLP [6]	ELM
1	0.3647	0.1588	0.1684	0.0794
2	0.4652	0.2329	0.2231	0.0501
3	0.3795	0.1652	0.1355	0.0828
4	0.2519	0.1918	0.2136	0.0639
5	0.3987	0.1495	0.1546	0.0939
6	0.6317	0.1463	0.1578	0.0720

4.5 Comentários Finais

Nesta seção, realizamos experimentos utilizando o algoritmo ELM, sobre o problema de estimativa de esforço. Nas três bases utilizadas, o desempenho do ELM foi bastante satisfatório, produzindo bons valores de MMRE e PRED(25).

Segundo [1], os valores considerados precisos de uma estimativa de software têm MMRE≤0,25 e PRED(25)≥75%. É importante observar que o ELM obteve os menores valores de MMRE nas três bases e a única base cujo PRED(25) não foi maio que 75% foi a Desharnais.

Capítulo 5

Conclusão

O presente trabalho primeiramente apresentou uma visão geral sobre a estimativa de esforço de software, abordando aspectos como tamanho do software e a importância da fase de levantamento de requisitos, no processo de desenvolvimento do software. Observamos que a estimativa de esforço é bastante importante no desenvolvimento de um software. Com a estimativa é possível ter maior controle sobre o processo de desenvolvimento, elaborar cronogramas mais precisos sendo, portanto, um grande diferencial para o mercado de empresas de software.

No capítulo sobre estimativa de esforço de software, abordamos a importância das métricas de software mais utilizadas, e as principais diferenças entre elas. Abordamos as bases de estimativa de esforço que foram utilizadas neste projeto, NASA, COCOMO, Desharnais, e verificamos seus atributos.

Neste trabalho, utilizamos o algoritmo ELM aplicado ao problema da estimativa de esforço de software. Analisamos as etapas do algoritmo, realizamos um estudo comparativo com outras bases conhecidas para verificar o desempenho do ELM, e depois realizamos os experimentos nas bases de estimativa de esforço.

EM nossos experimentos, realizamos um estudo da influência da normalização das variáveis dependentes no desempenho alcançado pela rede neural em termos do MMRE e do PRED(25). Observamos que a normalização em muitos casos melhorou bastante a capacidade de generalização da rede em relação às execuções sem normalização, conseguindo obter bons resultados.

Portanto, concluímos que o algoritmo Extreme Learning Machine, a partir dos resultados obtidos nos experimentos, é bastante adequado ao problema da estimativa de esforço de software, e também, que a normalização das variáveis

independente, possibilita um aumento significativo de precisão dos resultados observados através do MMRE.

5.1 Trabalhos Futuros

Algumas possíveis extensões deste trabalho seriam:

- Aplicar outras técnicas de IA para encontrar a arquitetura ótima da rede, utilizando algoritmo ELM, e verificar seu desempenho em problemas de estimativa de esforço.
- Desenvolvimento de novas técnicas para regressão, aplicadas ao problema.
- Desenvolvimento de uma técnica para otimizar o intervalo de normalização de forma a maximizar os resultados.

Bibliografia

[1] Agarwal, R. ; Manish Kumar, Yogesh, S. Mallick, R. M. Bharadwaj, e D. Anantwar, **Estimating software projects**. SIGSOFT Software Engineering Notes. Vol. 26, nr. 4, p. 60-67, 2001.

[2] Albrecht, A., **Measuring application development productivity**. In Proc. Of the IBM Applications /development Symposium, pages 83-92, Outubro 1979.

[3] Asuncion, A.; Newman D. J., "**UCI Machine Learning Repository**", http://www.ics.uci.edu/~mlearn/MLRepository.html, 2007.

[4] Bennatan, E. M., **Software Project Management: A Practitioner's Approach**, McGraw-Hill, 1992.

[5] Braga, A. P.; Carvalho, A. P. L. F.; Ludermir, T. B.; **Redes Neurais Artificiais : Teoria e Aplicações**, 2 ed., LTC, 2007.

[6] Braga, P. L., **Ferramenta Para Estimativa de Esforço de Projetos de Software Baseada em Técnicas de Computação Inteligente**, Universidade de Pernambuco, 2008.

[7] Colin J. Burgess e Martin Lefley, **Can genetic programming improve software effort estimation? a comparative evaluation**. Information & Software Technology. Vol. 43, nr. 14, p. 863-873, 2001.

[8] Conte, S.D. ; Dunsmore, H.E.; Shen, V.Y., **Software Engineering Metrics and Models**. Menlo Park, Calif.: Benjamin/Cummings, 1986.

[9] Fenton, N.E. e Pfleeger, S.L., **Software metrics: A rigorous and practical approach**, 2 ed., PWS Publishing Co., Boston,MA, USA, Fevereiro 1998.

[10] Haykin, S., **Neural networks: A comprehensive introduction**, 2 ed.,Bookman, 2008.

[11] Huang, G. B.; Zhu, Q. Y. ; Siew C. K.. **Real-time learning capability of neural networks**. IEEE Transactions on Neural Networks, volume 17, p. 863–878, 2006.

[12] Huang, G.; Zhu, Q.; Siew, C.,"**Extreme Learning Machine: Theory and applications**", Neurocomputing 70, 2006, pp 489–501.

[13] Huang, G.B.; Chen, L.; "**Convex incremental extreme learning machine**", Neurocomputing 70 (2007) , pp,3056–3062.

[14] Larman, C. , **Utilizando UML e Padrões**, 3 ed., Bookman, 2007.

[15] Li,M.B. et al.,"**Fully Complex Extreme Learning Machine**", Neurocomputing 68 (2005),pp 306–314

[16] McConnell, S., **Software estimation: Demystifying the black art**, Microsoft Press, 2006.

[17] Oliveira , A. L. I., **Estimation of software project effort with support vector regression**. Neurocomputing. Vol. 69, nr. 13-15, p. 1749-1753, 2006.

[18] PROMISE Data sets, Techinical report, **PROMISE Data**, Software disponível em: http://promisedata.org/, Acesso em 10 de outubro de 2009.

[19] Rao C.R., Mitra S.K., **Generalized Inverse of Matrices and its Applications**, Wiley, New York, 1971.

[20] Roger S. Pressman, **Engenharia de software**, 6 ed., McGraw-Hill, 2006.

[21] Sá, C. C., Silva M. F., **Haskell : Uma Abordagem Prática**, Novatec, 2006.

[22] Schildt, H. , **C Completo e total**, 3 ed., Makron Books,2005.

[23] Serre, D., **Matrices: Theory and Applications**, Springer, New York,2002.

[24] Shin, M. e Goel, A.L., **Empirical data modeling in software engineering using radical basis functions**. IEEE Transactions on Software Engineering. Vol. 26, nr. 6, p. 567-576, 2000.

[25] Simões, C. A., **Sistemática de métricas, qualidade e produtividade**, Technical report, BFPUG, http://www.bfpug.com.br/Artigos/sistematica_metricas_simoes.htm,1999.

[26] Tamura S.; Tateishi M., **Capabilities of a four-layered feedforward neural network: four layers versus three**, IEEE Trans. Neural Networks 8 (2) (1997) 251–255.

[27] Tan , P. ; Steinbach, M. ; Kumar, V. , **Introdução ao Data Mining**, Editora Ciência Moderna, 2009.

[28] Teixeira, L.A.; Oliveira, F.T.G.; Oliveira, A.L.I. ; Bastos, C.J.A., "**Adjusting Weights and Architecture of Neural Networks Through PSO with Time-Varying Parameters and Early Stopping**", 10th Brazilian Symposium on Neural Networks (SBRN), pp.33-38, 2008, IEEE Computer Society Press.

[29] Tronto, I. F. B.; Silva, J.D.S.; Sant'Anna,N., **Uma investigação de modelos de estimativas de esforço em gerenciamento de projeto de software**. Simpósio Brasileiro de Engenharia de Software, p. 224-240, 2006.

[30] Vazquez, C. E.; Simões,G. S. ; Albert,R. M. **Análise de pontos de função**, 5 ed., Érica, 2006.

[31] WEKA, Techinical report, Universidade de Waikato, Software disponível em: http://www.cs.waikato.ac.nz/ ml/weka/, Acesso em 20 de outubro de 2009.

[32] Yu, J.; Xi, L.; Wang, S., "**An Improved Particle Swarm Optimization for Evolving Feedforward Artificial Neural Networks**", Neural Processing Letters, 2007, pp. 217-231.