
# Exercício de Fixação de Conceitos 1

## EA072 - Inteligência Artificial em Aplicações Industriais

Gustavo **CIOTTO PINTON** - **RA 117136**
Campinas, 1 de outubro de 2015

# 2 Seleção de variáveis empregando filtros e *wrappers*

1. A quantidade de dados e de variáveis de entrada podem representar verdadeiros obstáculos no treinamento e validação dos pesos sinápticos de redes neurais, principalmente ao que se refere aos recursos disponíveis de processamento. Sendo assim, algoritmos e técnicas que possam ser capazes de determinar o grau de importância das variáveis em relação à saída e distinguir os dados mais pertinentes tornam-se indispensáveis a essas operações. Destacam-se duas técnicas: a primeira, a chamada técnica ***filter***, busca a classificar as variáveis de acordo com algum critério, seja ele a *correlação* ou a *informação mútua* entre as variáveis de entrada $\boldsymbol{x_j}$ e as saídas $\boldsymbol{y_i}$. Tais técnicas independem do modelo de predição e são aplicadas durante a fase de pré-processamento. A segunda, chamada ***wrapper***, utiliza uma máquina de aprendizado qualquer como uma caixa preta e avalia os subconjuntos de variáveis de acordo com suas respectivas qualidades de predição. Tais características podem impor algumas dificuldades a essa últmia técnica, à medida que a avaliação dos resultados de predição pode não ser tão trivial e o método de construção dos subconjuntos pode apresentar uma complexidade elevada. Destacam-se, portanto, os métodos de *forward selection* e *backward elimination*.

2. Seja $\mathbb{S}$ o conjunto de variáveis de entrada cujo efeito na saída do modelo $\hat{\boldsymbol{y}}_{\boldsymbol{k}}$ seja pertinente em relação à saída esperada $\boldsymbol{y_k}$. A abordagem de *forward selection* consiste a aumentar progressivamente $\mathbb{S}$, à medida que uma variável se mostre importante ao modelo. A importância de uma variável pode ser determinada através de alguns critérios, como por exemplo o cálculo de $J(\mathbb{S} \cup \{x_i\}) = \sum_{k=1}^{m} (\hat{\boldsymbol{y}}_{\boldsymbol{k}} - \boldsymbol{y_k})^2$, sendo $x_i$ um variável canditata à inserção. Neste caso, compara-se $J(\mathbb{S} \cup \{x_i\})$ e $J(\mathbb{S})$ e, caso o efeito dessa variável seja positivo, isto é, $J(\mathbb{S} \cup \{x_i\})$ menor, a acrescentamos em $\mathbb{S}$. Para *forward selection*, $\mathbb{S}$ começa vazio.

   A abordagem de *backward elimination*, por sua vez, elimina de $\mathbb{S}$ gradativamente as variáveis menos pertinentes ao modelo. $\mathbb{S}$ é, portanto, inicializado com todas as variáveis. Analogamente ao caso anterior, pode-se calcular $J(\mathbb{S} \setminus \{x_i\})$ e compará-lo com $J(\mathbb{S})$. Caso $J(\mathbb{S} \setminus \{x_i\})$ seja inferior, elimina-se de $\mathbb{S}$ a variável $\{x_i\}$.

   As duas abordagens acima não garantem a melhor combinação de entradas pelo fato da possível existência de *mínimos locais* da função $J(\mathbb{T})$, a função que associa o erro com as variáveis presentes no conjunto $\mathbb{T}$. Dependendo das condições e ordem de verificação das variáveis $x_i$, o método pode tender a diferentes mínimos, que podem ser eventualmente os melhores ou não.

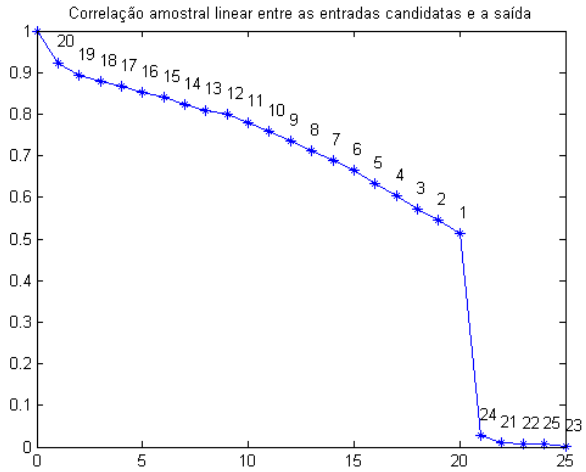
3. (a) A tabela 1 a seguir descreve algumas características estatísticas da série temporal. Todos os valores foram calculador através do MATLAB. Além delas, a série é formada por 3180 entradas, sendo compostas pelos dados de todos os meses de 1749 até 2013.

Tabela 1: Características da série temporal

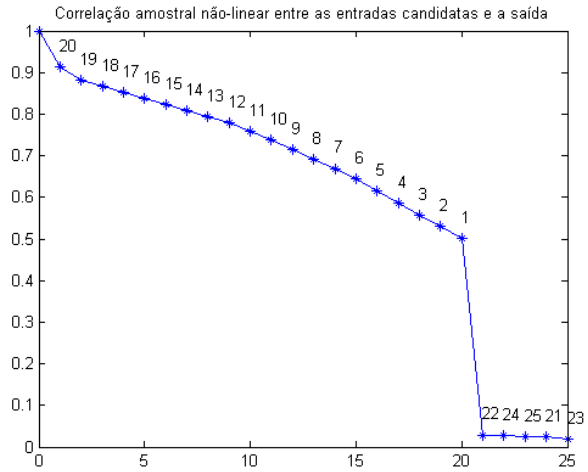
| Propriedades | Valor |
|---|---|
| Média | 51.9949 |
| Valor máximo | 253.8 |
| Valor mínimo | 0 |
| Desvio padrão | 41.116 |

   Em relação ao evento físico, conhece-se que a atividade das *sunspots* possui um ciclo de aproximadade 11 anos. O ponto de maior atividade durante o ciclo é chamado de *solar maximum* e o ponto de menor atividade, *solar minimum*. Este período de 11 anos também é observado para outros fenomênos solares e é ligado à variação no campo magnético que altera a polaridade durante ente período.

   (b) Para o *filtro linear*, as execuções de *filtro_lin(dados1.mat)* e *filtro_nlin(dados1.mat)* resultam nas figuras 1a e 1b, respectivamente.




(a) Resultado do *filtro linear*.




(b) Resultado do *filtro não linear*.

Figura 1: Resultados das execuções das funções de filtro.

Conclui-se portanto que as variáveis de 1 a 20 apresentam as maiores correlações, tanto a linear $R_{linear}$ quanto a não linear $R_{nlinear}$, em relação à saída, sendo a $20^a$ a mais correlata. A ordem de relevância em que elas aparecem também é a mesma, isto é, a sêquencia decrescente de 20 até 1 é observada em ambos os casos, e a maior diferença percentual entre $R_{linear}$ e $R_{nlinear}$ é de 3.12%, correspondente à $13^a$ variável.

Observa-se ainda que as variáveis 21 até 25 apresentam $R_{linear}$ e $R_{nlinear}$ muito inferiores em relação aos valores das outras varíaveis e que a sua ordem de relevância é diferente: para o filtro *linear* temos a sequência 24, 21, 22, 25 e 23, enquanto que para o *não linear* obtemos 22, 24, 25, 21 e 23.

(c) Para o método ***forward selection*** [1], obtem-se para as 5 execuções os seguintes dados:

Tabela 2: Resultados da execução ***1***.

| Entradas | 20 | 18 | 17 | 3 | 12 | 19 | 15 | 1 | 16 | 11 | 23 | 5 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Erro mínimo | 1.1765 | | | | | | | | | | | | |
| No. de Variáveis | 13 | | | | | | | | | | | | |

Tabela 3: Resultados da execução ***2***.

| Entradas | 20 | 18 | 17 | 3 | 12 | 19 | 15 | 1 | 10 | 5 | 22 | 7 | 21 | 25 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Erro mínimo | 1.1745 | | | | | | | | | | | | | | |
| No. de Variáveis | 15 | | | | | | | | | | | | | | |

Tabela 4: Resultados da execução ***3***.

| Entradas | 20 | 18 | 17 | 3 | 12 | 15 | 19 | 1 | 16 | 23 | 5 | 10 | 13 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Erro mínimo | 1.1727 | | | | | | | | | | | | | |
| No. de Variáveis | 14 | | | | | | | | | | | | | |

Tabela 5: Resultados da execução ***4***.

| Entradas | 20 | 18 | 17 | 3 | 12 | 15 | 19 | 1 | 16 | 10 | 5 | 23 | 24 | 13 | 21 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Erro mínimo | 1.1707 | | | | | | | | | | | | | | |
| No. de Variáveis | 15 | | | | | | | | | | | | | | |

[1] As imagens referentes a estas execuções encontram-se na figura 9 na seção **Anexos** no fim do documento

Tabela 6: Resultados da execução ***5***.

| Entradas | 20 | 18 | 17 | 3 | 12 | 19 | 15 | 1 | 10 | 11 | 5 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Erro mínimo | 1.1768 | | | | | | | | | | | |
| No. de Variáveis | 12 | | | | | | | | | | | |

Considerando somente as primeiras cinco variáveis selecionadas em cada execução, percebe-se que, em realidade, elas são todas iguais: 20, 18, 17, 3 e 12, nessa sequência. Na sexta posição, encontra-se a variável 19 (3 ocasiões) ou a 15 (2 ocasiões) e na sétima, a variável 1. À partir dessa posição, as entradas selecionadas variam a cada iteração.

Para o método ***backward elimination***[2], obtem-se:

Tabela 7: Resultados da execução ***1***.

| Entradas | 23 | 25 | 24 | 5 | 22 | 16 | 21 | 1 | 18 | 15 | 19 | 12 | 3 | 17 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Erro mínimo | 1.1711 | | | | | | | | | | | | | | |
| No. variáveis restantes | 15 | | | | | | | | | | | | | | |

Tabela 8: Resultados da execução ***2***.

| Entradas | 8 | 25 | 11 | 7 | 5 | 16 | 1 | 15 | 18 | 19 | 12 | 3 | 17 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Erro mínimo | 1.1717 | | | | | | | | | | | | | |
| No. variáveis restantes | 14 | | | | | | | | | | | | | |

Tabela 9: Resultados da execução ***3***.

| Entradas | 8 | 7 | 23 | 21 | 5 | 16 | 11 | 1 | 18 | 15 | 19 | 12 | 3 | 17 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Erro mínimo | 1.1744 | | | | | | | | | | | | | | |
| No. variáveis restantes | 15 | | | | | | | | | | | | | | |

Tabela 10: Resultados da execução ***4***.

| Entradas | 6 | 21 | 7 | 8 | 10 | 24 | 16 | 25 | 1 | 18 | 15 | 19 | 12 | 3 | 17 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Erro mínimo | 1.1733 | | | | | | | | | | | | | | | |
| No. variáveis restantes | 16 | | | | | | | | | | | | | | | |

Tabela 11: Resultados da execução ***5***.

| Entradas | 7 | 21 | 13 | 16 | 5 | 10 | 1 | 18 | 15 | 19 | 3 | 12 | 17 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Erro mínimo | 1.1748 | | | | | | | | | | | | | |
| No. variáveis restantes | 14 | | | | | | | | | | | | | |

Observa-se que o número de variáveis restantes para as execuções do *backward elimination* é ligeiramente superior em relação ao método anterior, apresentando uma média de 15 variáveis escolhidas contra aproximadamente 14 do *forward selection*. A média do erro quadrado médio na construção do modelo é inferior para *backward elimination*: 1.1731 contra 1.1742. Nota-se ainda que algumas variáveis foram escolhidas em todas as execuções de ambos, sendo elas, por exemplo, a 20, 17, 3, 12 e a 15.

(d) Entradas que possuem as maiores correlações não são as primeiras a serem adicionadas ao modelo pela presença de *redundância* entre elas. Se uma variável é altamente correlata com uma outra, eu não precisaria, em príncipio, conhecer as duas, já que a partir de um única, eu sou capaz de determinar a outra. Em outras palavras, variáveis *redundantes* adicionam pouca informação ao sistema. Por exemplo, a correlação entre as variáveis 20 e 19 do arquivo `dados1.mat` vale 0.9239 (valor obtido pelo comando `corr(X(:,20), X(:,19))` do MATLAB). Isso signfica que se $X_{20}$ é alto, então $X_{19}$ também o é e, portanto, nenhuma outra informação é adicionada ao modelo. É por essa razão que nas execuções acima, $X_{20}$ é escolhida inicialmente e $X_{19}$, só depois de algumas iterações.

[2]As imagens referentes a estas execuções encontram-se na figura 10 na seção **Anexos** no fim do documento

(e) Variáveis de baixa correlação com a saída podem proporcionar um grande poder de separação, se consideradas juntamente com outras[3]. Desta maneira, as variáveis de menor correlação são separadas mais facilmente em relação àquelas de maior correlação, permitindo assim a detecção de classes com maior precisão. Em outras palavras, entradas mais próximas das variáveis de baixa correlação podem ser classificadas mais facilmente.

(f) As variáveis geradas aleatoriamente ($X_{21} \cdots X_{25}$) apresentam correlações em relação à saída praticamente nulas (os resultados de `corr (X(:,21), S)` ... `corr (X(:,25), S)` estão mostrados na tabela 12). Dessa maneira, elas são escolhidas em ambos os modelos pelos mesmos motivos daqueles discutidos nos dois itens anteriores (*redundância* e *poder de separação*).

Tabela 12: Correlação das variáveis aleatórias execução 1 do *forward selection*.

| Variável $X_i$ | `corr (X(:,i), S)` |
|---|---|
| $X_{21}$ | 0.001347309212350 |
| $X_{22}$ | 0.011839959350690 |
| $X_{23}$ | -0.037287334573179 |
| $X_{24}$ | -0.007005116183944 |
| $X_{25}$ | 0.001963170472808 |

4. (a) O arquivo `wineq.mat` é composto por duas estruturas de dados. A primeira, matriz $X_{1593\times11}$, possui 11 colunas, correpondentes a cada uma das variáveis de entrada, e 1593 linhas, simbolizando 1593 dados disponíveis. De acordo com o *website* de onde tais dados foram retirados, essas variáveis correspondem a diversas propriedades que podem ser extraídas de um vinho, sendo elas *fixed acidity*, *volatile acidity*, *citric acid*, *residual sugar*, *chlorides*, *free sulfur dioxide*, *total sulfur dioxide*, *density*, *pH*, *sulphates* e *alcohol*. Foram considerados variações de vinhos tintos e brancos do vinho português *Vinho Verde*. A tabela abaixo mostra algumas características dessas propriedades.

Tabela 13: Informações correpondentes às variáveis de `wineq.mat`.

| Variável | Média | Max. | Min. | *Standard Deviation* |
|---|---|---|---|---|
| *fixed acidity* | 0.52324 | 1 | 0.28931 | 0.10932 |
| *volatile acidity* | 0.33385 | 1 | 0.075949 | 0.11333 |
| *citric acid* | 0.27116 | 1 | 0 | 0.19495 |
| *residual sugar* | 0.16374 | 1 | 0.058065 | 0.090957 |
| *chlorides* | 0.14321 | 1 | 0.01964 | 0.077153 |
| *free sulfur dioxide* | 0.2205 | 1 | 0.013889 | 0.14537 |
| *total sulfur dioxide* | 0.16052 | 1 | 0.020761 | 0.11379 |
| *density* | 0.022051 | 1 | 0.0098643 | 0.096464 |
| *pH* | 0.82574 | 1 | 0.68329 | 0.038451 |
| *sulphates* | 0.32903 | 1 | 0.165 | 0.084846 |
| *alcohol* | 0.69949 | 1 | 0.56376 | 0.071404 |

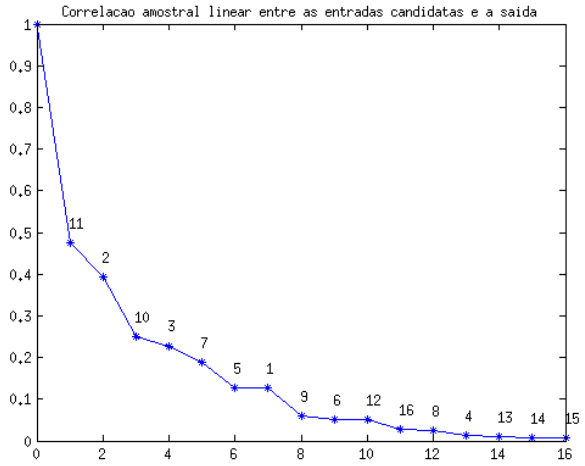
Para a saída, que mede o nível de qualidade de cada vinho, baseado em um teste sensitivo, encontra-se:

Tabela 14: Informações correpondentes à saída de `wineq.mat`.

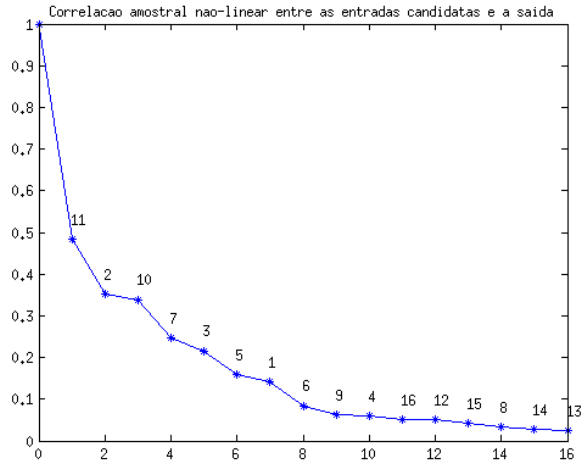
| Variável | Média | Max. | Min. | *Standard Deviation* |
|---|---|---|---|---|
| Saída | 0.70425 | 1 | 0.375 | 0.10101 |

[3]No artigo *[ Guyon, I.; Elisseeff, A. "An introduction to variable and feature selection", Journal of Machine Learning Resear ch, vol. 3, pp. 1157 - 1182 2003 ]*, o autor dá um exemplo dessa afirmação na seção 3.3.

(b) As execuções de `filtro_lin('dados2.mat')` e `filtro_nlin('dados2.mat')` são mostradas na figuras 2a e 2b a seguir.




(a) Resultado do *filtro linear*.




(b) Resultado do *filtro não linear*.

Figura 2: Resultados das execuções das funções de filtro.

Neste caso, observa-se uma maior diferença entre as filtragens linear e não linear em relação ao estudo de caso anterior. Percebe-se inicialmente que a correlação da variável mais correlata ($< 0.5$) aqui é muito inferior àquela mais correlata ($> 0.9$) para os dados referentes aos *Sunspots*.

A ordem decrescente de correlação em que as variáveis aparecem também muda do caso linear para o não linear. A tabela seguinte evidencia esta afirmação. $\Delta = \frac{(R_{nlin} - R_{lin})}{R_{nlin}}$, onde $R$ é o valor da correlação linear ou não linear, representa a variação percentual entre as respectivas correlações.

Tabela 15: Ordens de aparição das variáveis.

| | Ordem | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Linear | 11 | 2 | 10 | 3 | 7 | 5 | 1 | 9 | 6 | 12 | 16 | 8 | 4 | 13 | 14 | 15 |
| *Não* linear | 11 | 2 | 10 | 7 | 3 | 5 | 1 | 6 | 9 | 4 | 16 | 12 | 15 | 8 | 14 | 13 |
| $\Delta$ | 0.01 | -0.12 | 0.26 | 0.07 | 0.13 | 0.2 | 0.11 | 0.31 | 0.18 | 0.16 | 0.46 | 0.5 | 0.68 | 0.73 | 0.75 | 0.72 |

Observa-se que a ordem muda sobretudo no fim da sequência, onde as variações percentuais são superiores.

(c) Para o método ***forward selection***[4], obtem-se para as 5 execuções os seguintes dados:

Tabela 16: Resultados da execução ***1***.

| Entradas | 11 | 2 | 10 | 7 | 5 | 9 | 6 | 16 |
|---|---|---|---|---|---|---|---|---|
| Erro mínimo | 1.0484 | | | | | | | |
| No. de Variáveis | 8 | | | | | | | |

Tabela 17: Resultados da execução ***2***.

| Entradas | 11 | 2 | 10 | 7 | 5 | 9 | 6 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|
| Erro mínimo | 1.0491 | | | | | | | | |
| No. de Variáveis | 9 | | | | | | | | |

Tabela 18: Resultados da execução ***3***.

| Entradas | 11 | 2 | 10 | 7 | 5 | 9 | 6 | 13 |
|---|---|---|---|---|---|---|---|---|
| Erro mínimo | 1.0495 | | | | | | | |
| No. de Variáveis | 8 | | | | | | | |

Tabela 19: Resultados da execução ***4***.

| Entradas | 11 | 2 | 10 | 7 | 5 | 9 | 6 | 14 |
|---|---|---|---|---|---|---|---|---|
| Erro mínimo | 1.0509 | | | | | | | |
| No. de Variáveis | 8 | | | | | | | |

[4]As imagens referentes a estas execuções encontram-se na figura 11 na seção **Anexos** no fim do documento

Tabela 20: Resultados da execução ***5***.

| Entradas | 11 | 2 | 10 | 7 | 5 | 9 | 6 | 13 |
|---|---|---|---|---|---|---|---|---|
| Erro mínimo | 1.0509 | | | | | | | |
| No. de Variáveis | 8 | | | | | | | |

Para o método ***backward elimination***[5], obtem-se:

Tabela 21: Resultados da execução ***1***.

| Entradas | 13 | 6 | 9 | 5 | 7 | 10 | 2 | 11 |
|---|---|---|---|---|---|---|---|---|
| Erro mínimo | 1.0496 | | | | | | | |
| No. de Variáveis | 8 | | | | | | | |

Tabela 22: Resultados da execução ***2***.

| Entradas | 15 | 3 | 14 | 9 | 5 | 7 | 10 | 2 | 11 |
|---|---|---|---|---|---|---|---|---|---|
| Erro mínimo | 1.0558 | | | | | | | | |
| No. de Variáveis | 9 | | | | | | | | |

Tabela 23: Resultados da execução ***3***.

| Entradas | 15 | 12 | 3 | 6 | 9 | 5 | 7 | 10 | 2 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|
| Erro mínimo | 1.0532 | | | | | | | | | |
| No. de Variáveis | 10 | | | | | | | | | |

Tabela 24: Resultados da execução ***4***.

| Entradas | 3 | 1 | 14 | 6 | 9 | 5 | 7 | 10 | 2 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|
| Erro mínimo | 1.0548 | | | | | | | | | |
| No. de Variáveis | 10 | | | | | | | | | |

Tabela 25: Resultados da execução ***5***.

| Entradas | 12 | 6 | 14 | 9 | 5 | 7 | 10 | 2 | 11 |
|---|---|---|---|---|---|---|---|---|---|
| Erro mínimo | 1.0497 | | | | | | | | |
| No. de Variáveis | 9 | | | | | | | | |

Observa-se que para a técnica de *forward selection* as primeiras sete entradas selecionadas foram as mesmas para as 5 execuções, sendo elas a 11, 2, 10, 7, 5, 9 e 6. As restantes escolhidas fazem parte das entradas aleatórias, calculadas para cada execução. Em média, foram escolhidas 8 variáveis e produziu-se um erro de 1.04976.

Em relação à técnica de *backward elimination*, as mesmas variáveis citadas no parágrafo anterior estiveram presentes, com exceção da 6, que esteve ausente somente na execução 2. Foram selecionadas, em média, um número maior de variáveis, 9, e um erro superior, 1.05262. As variáveis restantes foram escolhidas dentre as geradas aleatoriamente.

Finalmente, as váriaveis que foram escolhidas nos primeiros lugares na *forward selection* seriam eliminadas por último na *backward elimination*, confirmando assim um certo nível de semelhância nos resultados das duas técnicas.

# 3 Séries temporais e a tarefa de predição

## 3.0 Normalização dos dados

De acordo com enunciado disponibilizado, os dados devem ser normalizados de maneira a obter média nula e desvio padrão unitário. Entradas que excursionam em intervalos muito extensos prejudicam o desempenho das redes neurais MLP.

Sendo assim, faremos uso da seguinte equação para atingir as características necessárias:

$$X_i = \frac{V_i - \hat{\mu}}{\hat{\sigma}} \tag{1}$$

em que $\hat{\mu}$ é a média de todas as entradas (calculada através de `mean(dengue_SP)`) e $\hat{\sigma}$ é o estimador do desvio padrão (calculado por `std(dengue_SP)`). Obtem-se, assim, um novo vetor

[5]As imagens referentes a estas execuções encontram-se na figura 12 na seção **Anexos** no fim do documento

cuja média e variância valem, respectivamente, $1.8288 \times 10^{-16}$ e 1. A figura à seguir mostra um histograma dos dados normalizados. Observa-se que a maioria dos dados encontram-se na vizinhança de 0, mas há uma quantidade razoável deles superior à unidade, correspondentes aos períodos de chuva, onde há, naturalmente, mais casos da doença.

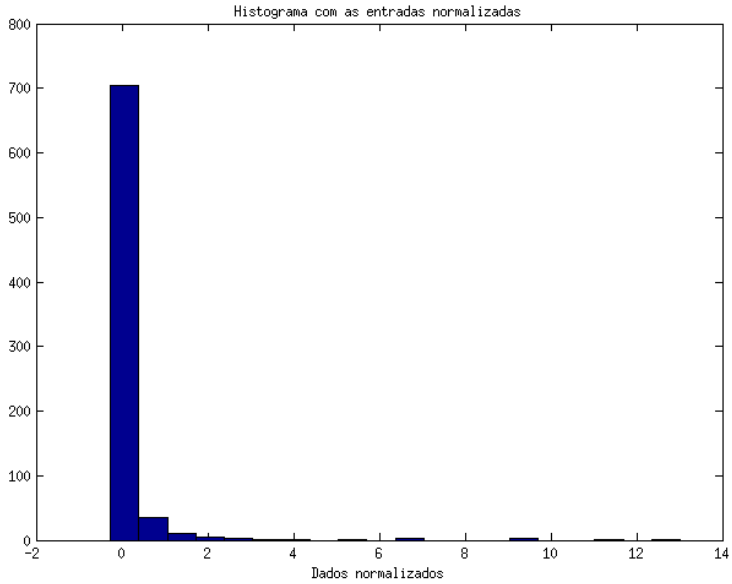



Figura 3: Dados normalizados.

## 3.1 Variáveis participantes do modelo - Filtro de correlação

A fim de obter um resultado mais representativo nos preditores a serem realizados, aplica-se um filtro de correlações nas 20 variáveis que inicialmente foram propostas para determinar a previsão do número de casos de dengue para a próxima semana. Para isso, usamos o programa `calc_corr2.m`, disponibilizado pelo professor. Obtem-se o gráfico mostrado na figura 4 a seguir:

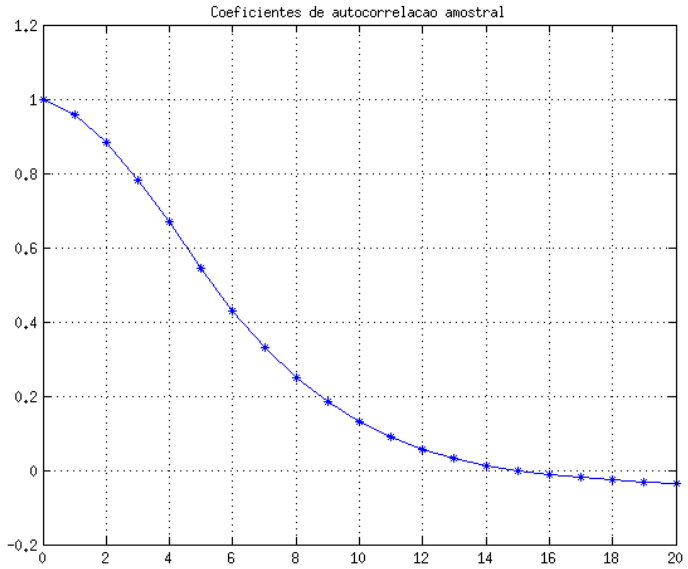



Figura 4: Correlação das 20 variavés escolhidas como entrada.

Observa-se neste gráfico que as cinco primeiras variáveis utilizadas no modelo são as mais correlatas à saída. Em termos numéricos, as variáveis a partir da sexta apresentam correlações inferiores

a 0.5. Conclui-se, portanto, que em termos de pertinência, as cinco variáveis iniciais são as mais importantes para compor o valor a ser predito.

## 3.2 Síntese de um preditor linear

Para esta etapa, define-se $M = 5$ a dimensão do espaço de entrada (consultar seção 3.1) e $R = 1$, a de saída. Em outras palavras, utilizaremos dados de 5 semanas anteriores para determinar o resultado da 'semana seguinte'. As funções utilizadas para a construção da série e nos cálculos dos preditores encontram-se, respectivamente, nos programas 1 e 2 na seção **Anexos** no fim deste documento.

Utilizaremos a estratégia de *k-folds cross-validation* para estimar o melhor valor do parâmetro $c$. Neste caso, adota-se $k = 10$.

### 3.2.1 Caso não regularizado

A resolução de $\overrightarrow{b} = \left(A^T A\right) A^T Y$ utilizando apenas o conjunto de treinamento produz o vetor, cujos coeficientes encontram-se na tabela abaixo, e um erro quadrático médio de **0.2500**.

$$\overrightarrow{b}^T_{nreg} = \begin{bmatrix} -0.1714 & 0.2349 & -0.3289 & -0.0528 & 1.2298 & -0.0000 \end{bmatrix}$$

### 3.2.2 Caso regularizado

A utilização de um parâmetro $c$ adicional e da estratégia *k-fold cross-validation* permite a adequação mais correta dos dados de entrada aos de saída, conforme observado em sala de aula. A execução do programa `resolve_sistema_k_folds.m` da seção **Anexos** gera o gráfico, em escala semi logarítmica, contido na figura 5, que relaciona $c$ com a média do erro quadrático médio junto ao conjunto de validação em cada uma das 10 execuções do programa (uma execução para cada uma das pastas de validação).

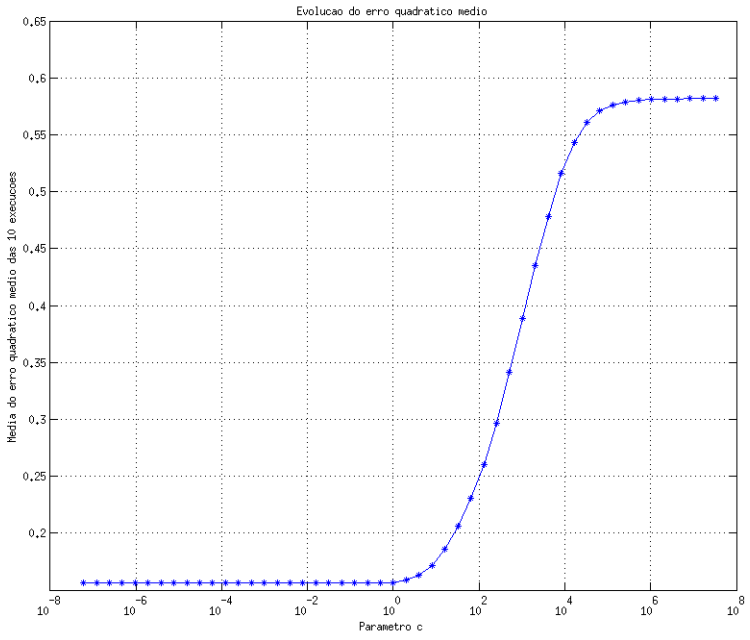



Figura 5: Erro quadrático médio em função do parâmetro c.

Ressalta-se a presença de um mínimo local, para $c_1 = 2^{-1} = 0.5$, valendo **0.15602**. Esse erro é inferior a aquele obtido ao caso não regularizado, já que o preditor obtido neste caso adapta-se melhor a todo conjunto dos dados e não somente a aqueles que só foram usados no treinamento.

Enfim, explicitamos os vetores $\overrightarrow{b}$ para $c = 2^{-1}$:

| | | | | | | |
|---|---|---|---|---|---|---|
| $\overrightarrow{b}_1$ | -0.1673 | 0.2193 | -0.3182 | -0.0397 | 1.2160 | 0.0023 |
| $\overrightarrow{b}_2$ | -0.1685 | 0.2244 | -0.3230 | -0.0418 | 1.2195 | 0.0014 |
| $\overrightarrow{b}_3$ | -0.1669 | 0.2195 | -0.3199 | -0.0379 | 1.2158 | 0.0015 |
| $\overrightarrow{b}_4$ | -0.1674 | 0.2194 | -0.3185 | -0.0390 | 1.2155 | 0.0025 |
| $\overrightarrow{b}_5$ | -0.1680 | 0.2217 | -0.3191 | -0.0400 | 1.2164 | 0.0008 |
| $\overrightarrow{b}_6$ | -0.1675 | 0.2199 | -0.3191 | -0.0390 | 1.2159 | 0.0018 |
| $\overrightarrow{b}_7$ | -0.1662 | 0.2329 | -0.3552 | -0.0222 | 1.2199 | -0.0014 |
| $\overrightarrow{b}_8$ | -0.1676 | 0.2352 | -0.3412 | -0.0347 | 1.2221 | 0.0008 |
| $\overrightarrow{b}_9$ | -0.1664 | 0.2202 | -0.3225 | -0.0365 | 1.2160 | 0.0009 |
| $\overrightarrow{b}_{10}$ | -0.0832 | -0.0739 | 0.1328 | -0.1357 | 1.0721 | -0.0105 |

Destaca-se que os valores para as componentes de uma mesma coluna não apresentam uma grande variação entre si, isto é, os coeficientes do modelo autoregressivo que produzem o menor erro médio possuem um comportamento bem definido.

## 3.3 Síntese de uma rede neural MLP

Os resultados contidos nas seções 3.3.1 e 3.3.2 foram obtidos do programa `numberNeuronsMLP.m`, contido no trecho de código 3 na seção **Anexos** no fim deste documento. Em poucas palavras, este programa automatiza o processo de treinamento de análise dos resultados de uma rede MLP para diferentes quantidades de neurônios e iterações. Ele utiliza as funções disponibilizadas pelo professor, que foram adaptadas somente para receber parâmetros de entrada no lugar de requirir os dados ao usuário. Neste programa, há dois laços: o mais externo é reponsável por modificar o número de iterações do algoritmo otimizador, escolhendo valores no conjunto $i \in \{50, 100, 150, 200, 300, 400 \dots 1000\}$ e o mais interno, o número de neurônios na camada intermediária no conjunto $n \in \{5, 6, 7 \dots 20\}$. Para cada valor de $i$ e de $n$ treina-se 10 MLPs (uma para cada configuração das *folds*) e calcula-se as médias dos seus desempenhos. Após obtermos as MLPs para todos os valores de $n$, construimos um gráfico com as médias dos erros de validação e de teste e, após todos os valores de $i$, desenhamos um último gráfico, com os desempenhos médios para cada valor de $i$. As próximas seções serão dedicadas às devidas explicações sobre os resultados.

### 3.3.1 Determinação do número de iterações

A figura 6 mostra um gráfico que relaciona o desempenho médio das MLPs (treinadas para cada valor de $n$) com o número de iterações. Observa-se que os comportamentos dos erros de teste e de validação são similiares, isto é, quando uma apresenta um valor elevado, a outra também apresentará. Esta afirmação explica-se pela capacidade de generalização das MLPs: se uma rede é treinada de forma que o conjunto de validação seja utilizado, é possível atingir um maior grau de flexibilidade a todos os dados e, assim, o erro para dados novos não levados em consideração (como aqueles do conjunto de teste) não se distanciará fortemente do erro de validação. Sendo assim, utilizaremos o valor que possui o maior custo benefício entre os dois erros, isto é, $i = 1000$, que produz um erro médio de validação igual a **0.1588** e um erro médio de teste valendo **0.1813**.

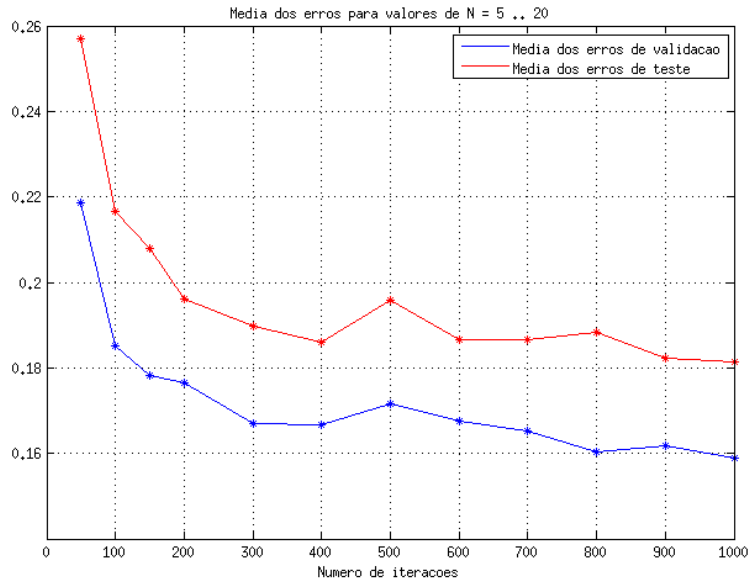



Figura 6: Média de todas as MLPs, calculadas para todo valor de $n$, para cada $i$.

### 3.3.2 Determinação do número de neurônios na camada intermediária

Uma vez determinado o melhor número de iterações, é possível estabelecer o número de neurônios que melhor se adequa à aplicação. Para isso, utilizamos o gráfico da figura 7.

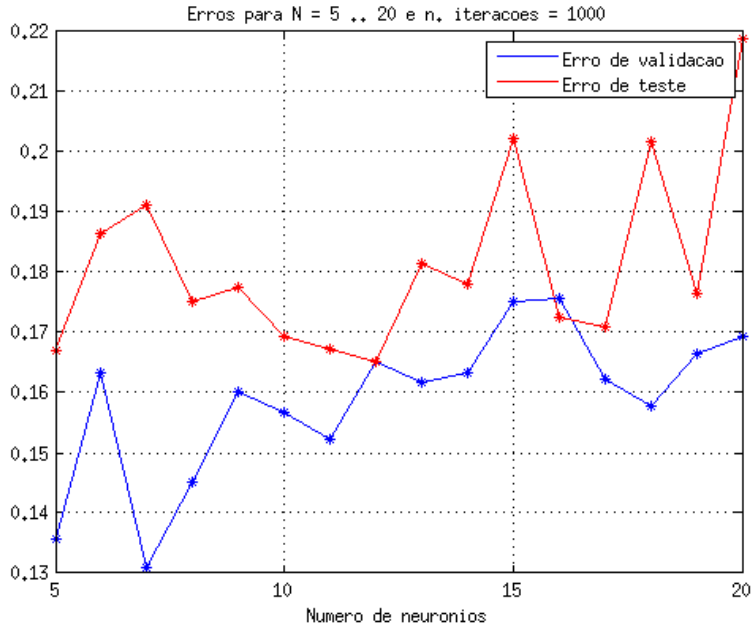



Figura 7: Média de todas as MLPs, calculadas para cada valor de $n$, para $i = 1000$.

O melhor custo benefício entre os dois erros é, neste caso $n = 5$, apresentando erro médio de validação de **0.1356** e erro médio de teste de **0.1671**. Destaca-se que, para $n = 7$, temos o valor mínimo do erro de validação, mas, ao mesmo tempo, um erro relacionado ao teste muito elevado. Por esta rezão, este número de neurônios não foi escolhido. Observa-se também que as duas curvas não apresentam comportamentos definidos, isto é, os erros para diferentes valores de $n$ são muito distintos entre si.

Os demais resultados para outros valores de $i$ e $n$ podem ser observados na figura 13 na seção **Anexos**.

## 3.4 Síntese da Máquina de Aprendizado Máximo - *ELM*

Neste exercício, adotaremos uma *ELM* com apenas uma camada intermediária com um número de neurônios que determinaremos a seguir. Para tal, utilizamos o programa 4, presente no fim deste documento na seção **Anexos**. Neste trecho de código, iteramos o número de neurônios no conjunto $\{100, 150, 250, 400, 500, 1000\}$ e o parâmetro regularizador $c$ e, obtemos um gráfico da média do erro quadrático médio de validação das 10 execuções (uma para cada *fold*) para valor do número de neurônios utilizado.

Após a execução no ambiente MATLAB, encontrou-se que o menor erro quadrático médio junto ao conjunto de validação foi **0.3036**, referente a $N = 400$ neurônios e $c = 2^{-6} = 0.015625$. A figura a seguir mostra o resultado da execução nestas configurações:

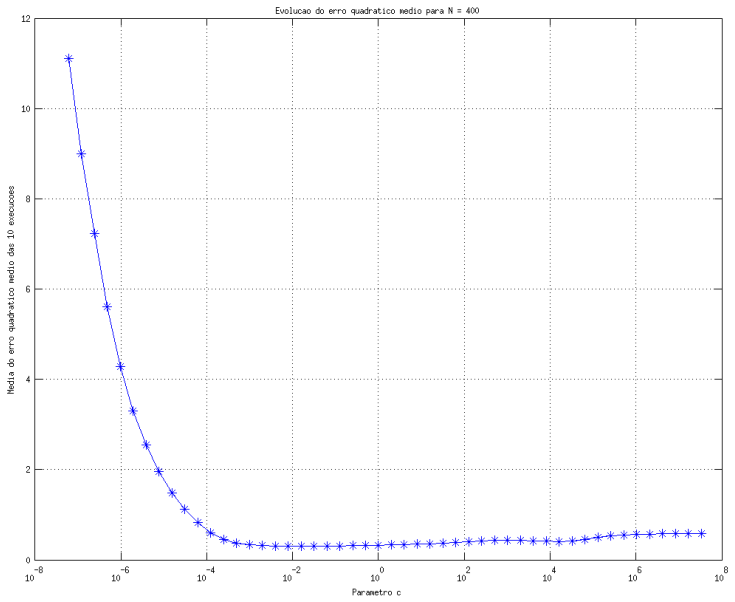



Figura 8: Erro quadrático médio de validação em função de $c$, para $N = 400$.

Percebe-se que o erro cresce à medida que $c$ se aproxima de 0 e depois se estabiliza para $c$ tendendo a infinito. Os resultados das demais execuções encontram-se na figura 14 na seção **Anexos**.

## 3.5 Análise dos resultados dos preditores lineares, MLPs e ELMs

A pior performance entre os quatro preditores implementados nesta lista foi a do *ELM*, que obteve um erro quadrático médio de **0.3036**. Tal resultado é um tanto quanto surpreendente, já que esse desempenho é inferior a aquele dos preditores lineares, fato este que não era esperado. Possíveis explicações para este fato são a quantidade insuficiente de camadas utilizadas, a inicialização dos pesos sinápticos da camada intermediária, que, foi neste caso, determinada aleatoriamente segundo uma lei normal de média 0 e variância 1, ou algum erro eventual no programa 4.

O terceiro lugar, *preditor linear não regularizado*, para qual só foi utilizado o conjunto de treinamento para o cálculo do erro quadrático médio, obteve um erro de **0.2500**. Uma vez o conjunto de validação não foi utilizado, a capacidade de generalização do preditor é comprometido e, assim, o preditor apresenta um erro relativamente alto.

Em segundo lugar, destaca-se *preditor linear regularizado*, cujo erro quadrático médio vale **0.15602**. A introdução de um parâmetro $c$ adicional e a sua determinação ótima juntamente ao conjunto de validação provaram que o desempenho obtido pode ser melhorado de maneira significativa, visto que

o modelo linear mantem-se o mesmo. Isso significa que nesta oportunidade, observa-se uma maior flexibilidade do preditor a todos os dados.

Enfim, para o caso das MLPs, obtem-se um erro médio quadrático de validação de **0.1356**, para $i = 1000$ e $n = 5$. É necessário dizer que este resultado foi conseguido com base em uma quantidade de processamento muito superior aos dois casos precedentes, uma vez que, no total, foram treinadas $10 * \mathbf{card}(n) * \mathbf{card}(i) = 10 * 15 * 12 = 1800$ MLPs (uma para cada configuração das *folds*, cada valor de $n$ e $i$). Foram utilizados conjuntos de validação e teste para otimizar ainda mais a adequação da rede aos dados. A rede neural é, portanto, a melhor opção para predizer a série temporal da dengue em São Paulo.

# 4 Conclusões Finais

A partir dos exemplos práticos desenvolvidos nessa lista, foi possível explorar as principais vantagens, desvantagens e recursos de alguns conceitos abordados durante a aula. Primeiramente, para a seleção de variáveis e filtros constatou-se que, ao contrário destes últimos, nem sempre as variáveis mais correlatas com a saída são escolhidas para participar do modelo. Adicionalmente, apesar dos métodos *wrappers* apresentarem uma complexidade computacional maior que os filtros, eles selecionam as variáveis de modo que o máximo de informação fique contido no modelo, isto é, o erro calculado junto ao modelo linear seja mínimo. Observou-se que os resultados para o *backward elimination* e *forward selection* foram bastante similares, diferenciando-se somente em um número bem pequeno de variáveis que foram selecionadas em um método, mas não no outro. Essa afirmação reforça a ideia que ambos os modelos selecionam as melhores variáveis para compor o modelo.

Em relação aos preditores implementados, nos deparamos com o compromisso "recursos computacionais $\times$ resultados", isto é, à medida que adicionamos a um determinado método algum "grau de liberdade"(como número de neurônios ou de iterações, por exemplo), tendemos a encontrar uma solução apresentando um erro menor. Isso foi observado principalmente no duelo entre os preditores lineares e as redes neurais MLP. Para estas últimas, realizou-se vários treinamentos distintos (da ordem de 1800 treinamentos - rever seção 3.3), mas, ao fim, encontra-se um resultado melhor em relação a ambos os preditores lineares (regularizado e não regularizado). Um resultado que foi constatato e que não era esperado é o fato de que as ELMs treinadas na seção 3.4 apresentarem performance inferior até mesmo em relação aos preditores lineares. Neste caso, foram treinadas várias ELMs com grandes variações no número de neurônios nas suas camadas intermediárias e no parâmetro regularizador $c$. Mesmo com essa quantidade de processamento, os resultados encontrados foram insuficientes, se comparados às MLPs e preditores lineares.

Enfim, de maneira geral, este exercício foi bastante abrangente, permitindo aos alunos desenvolver suas habilidades ligadas à programação de redes neurais em MATLAB.

# Referências bibliográficas


- `https://en.wikipedia.org/wiki/Sunspot`. Acessado às 19:22 29/09/2015.
- Guyon, I.; Elisseeff, A. "An introduction to variable and feature selection", Journal of Machine Learning Resear ch, vol. 3, pp. 1157 - 1182 2003.
- `http://archive.ics.uci.edu/ml/datasets/Wine+Quality`. Acessado às 21:27 30/09/2015.

# 5 Anexos

## 5.1 Programas

Programa 1: `gera_dados.m` - cria a sequência temporal.

```
%% Funcao gera_dados
%  Esta funcao recebe uma serie temporal contida no arquivo 'dados'
%  e retorna uma matriz X com m colunas e uma matriz Y com r colunas. Ambas
%  as matrizes sao salvas no arquivo 'saida' igualmente.
%  Autor: Gustavo CIOTTO PINTON
function [X, Y] = gera_dados(dados, saida, m, r)

    load(dados);

    data = dengue_SP;

    % Transforma os dados de maneira que sua media seja 0 e desvio padrao
    % 1.
    data = (data - mean(data))/std(data);
    n_data = length (data);

    X = [];
    Y = [];
    Z = [];

    % Insere-se sequencialmente os dados no vetor auxiliar Z, conforme
    % descrito no enunciado.
    % Linha 1: x1 x2 ... xm   xm+1 ... xm+r
    % Linha 2: x2 x3 ... xm+1 xm+2 ... xm + 2 + r
    % Linha n: ...
    for j=(m+r):n_data,
        Z = [Z; data( (j-m-(r-1)):j, 1 )'];
    end

    % Seleciona X e Y de acordo com parametros
    X = Z(:, 1:m);
    Y = Z(:, (m+1):(m+r));

    save(saida, 'X', 'Y');

end
```

Programa 2: `resolve_sistema_k_folds.m` - calcula preditores lineares.

```
%% Funcao resolve_k_folds
%  Esta funcao calcula os preditores lineares regularizados e nao
%  regularizado com auxilio do metodo crossfold validation para o caso
%  regularizado. Ela recebe como parametro a quantidade k de pastas a serem
%  utilizadas e o arquivo 'dados' onde os dados estao contidos. Retorna:
%  b_nreg :     o vetor b para o caso nao regularizado
%  b_k_folds:   uma matriz tridimensional contendo todos os vetores b
%               regularizados para cada fold e cada c.
%  erro_nreg:   o erro quadratico medio para o caso nao regularizado.
%  erro_reg_k_folds: os erros quadraticos medios de validacao para todas as
%                    execucoes (para cada configuracao das folds e parametro c)
% erro_reg_tr_k_folds : os erros quadraticos medios de treinamento para todas as
%                    execucoes (para cada configuracao das folds e
%                    parametro c)
% AUTOR: Gustavo CIOTTO PINTON

function [b_nreg, erro_nreg, b_k_folds, erro_reg_k_folds, erro_reg_tr_k_folds] =...
                        resolve_sistema_k_folds(dados,k)
```

```
19
20 close all;
21
22 k = 10;
23 M = 5; % Numero de entradas do modelo
24 R = 1;
25
26 % Gera matriz de entrada X e de saida Y
27 [X, Y] = gera_dados('dengue_SP.mat', 'resultado.mat', M, R);
28
29 % Calcula tamanhos dos conjunto de TReinamento, Validacao e TEste.
30 Tr = length(X(:,1));
31 V = round(Tr/k); % O conjunto de validacao correponde a 1/k do conj. de treinamento.
32 Te = 0;
33
34 % Controi-se matriz para o caso nao-regularizado. Utiliza-se neste caso
35 % todo o conjunto de dados.
36 A_nreg = [ X ones(Tr ,1) ];
37 ATA = (A_nreg'*A_nreg)\eye(M + 1);
38
39 % Resolve-se metodo dos minimos quadrados.
40 b_nreg = ATA*A_nreg'*Y;
41 % Calcula-se erro quadratico medio.
42 erro_nreg = sqrt((norm(A_nreg*b_nreg - Y))^2/Tr);
43
44 erro_reg_k_folds = [];
45 erro_reg_tr_k_folds = [];
46 b_k_folds = [];
47
48 for i = 1:k
49
50     % gera a ko. pasta para ser usada como validacao
51     k_fold_set = (i - 1)*V + 1 : i*V;
52
53     % pega o restante para treinamento. Setdiff realiza a diferenca de
54     % conjuntos entre { 1 2 .... Tr } e os indices selecionados para a
55     % validacao
56     conjunto_treinamento = setdiff(1:Tr, k_fold_set);
57
58     % Controi as matrizes A e Y somente com dados de treinamento
59     Atr = [ X(conjunto_treinamento, :) ones(Tr - V - Te ,1) ];
60     ATA = (Atr'*Atr)\eye(M + 1); % Calcula inversa
61     Ytr = Y (conjunto_treinamento,:);
62
63     b_nreg = ATA*Atr'*Ytr;
64     erro_nreg = sqrt((norm(Atr*b_nreg - Ytr))^2/(Tr - V - Te));
65
66     Aval = [ X(k_fold_set, :) ...
67         ones(V,1) ];
68     Yval = Y(k_fold_set, :);
69
70     % Inicializa variaveis que serao utilizadas para armazenar erros.
71     erro_reg = [];
72     erro_reg_tr = [];
73     b_reg = [];
74     n_b_reg = []; % Vetor que armazena as normas dos vetores b a serem calculados.
75
76     for c_teste = -24:25
77
78         % Calcula caso regularizado segundo enunciado (pag. 6)
79         ATA_ = (Atr'*Atr + (2^c_teste)*eye(M + 1)) \ eye(M + 1);
80
81         b = ATA_*Atr'*Ytr; % Vetor coluna com M + 1 linhas
```

```
82
83          % Armazena vetor b que acabou de ser calculado e sua norma. b_reg
84          % eh uma matriz M+1 x 50 portanto em que cada coluna eh uma vetor b
85          % calculado para um valor de c
86          b_reg = [b_reg b];
87          n_b_reg = [n_b_reg norm(b)];
88
89          % Calcula e armazena erros quadraticos medios de validacao e
90          % treinamento. erro_reg e erro_reg_tr sao vetores linhas (1x50), em cada
91          % coluna eh o erro associado ao parametro c.
92          erro_reg = [erro_reg  sqrt((norm(Aval*b - Yval)^2)/V)];
93          erro_reg_tr = [erro_reg_tr sqrt((norm(Atr*b - Ytr)^2)/(Tr-V))];
94      end
95
96      % Guarda a matriz b_reg calculada nas iteracoes passadas numa matriz
97      % tridimensiona, que contera 10 matrizes b_reg, calculadas para cada
98      % uma das configuracoes das folds.
99      b_k_folds = cat(3, b_k_folds, b_reg);
100
101     % Guarda os vetores erro_reg e erro_reg_tr nas matrizes
102     % erro_reg_k_folds e erro_reg_tr_k_folds, respectivamente, de forma que
103     % cada linha dessas matrizes corresponda a uma das configuracoes da
104     % folds. Cada coluna corresponde ao erro calculado para um c. Exemplo:
105     % a linha 1 e coluna 4 dessas matrizes contem os erros correspondentes
106     % a situcao em que a primeira 1/k parte dos dados foi usada para validacao
107     % e c eh igual a % 2^-20.
108     erro_reg_k_folds = [erro_reg_k_folds; erro_reg];
109     erro_reg_tr_k_folds = [erro_reg_tr_k_folds; erro_reg_tr];
110 end
111
112
113 % Faz o grafico das perfomances medias dos erros em funcao do parametro c.
114 % mean (matriz) retorna um vetor linha com as medias de cada coluna da
115 % matriz. Neste caso, obteremos a media dos erros para todos os c.
116 figure
117 semilogx( 2.^(-24:25), mean(erro_reg_k_folds));
118 hold on;
119 semilogx( 2.^(-24:25), mean(erro_reg_k_folds),'*');
120 hold off;
121 title('Evolucao do erro quadratico medio');
122 ylabel('Media do erro quadratico medio das 10 execucoes');
123 xlabel('Parametro c');
124 grid on
125
126 % Seleciona o menor erro dentre as medias dos erros quadraticos medios e o
127 % respectivo index (que representa c)
128 [menor_erro, menor_index] = sort(mean(erro_reg_k_folds), 'ascend');
129
130 % Imprime os vetores e valores relacionados aos minimos
131 disp(['Vetores b para c = ' int2str(menor_index(1) - 25) ' e erro medio de ' num2str(
        menor_erro(1))]);
132 b_k_folds(:,menor_index(1),:)
133
134 end
```

Programa 3: `numberNeuronsMLP.m` - Automatiza treinamento e análise de redes MLP.

```
1 %% Funcao numberNeuronsMLP
2 %  Esta funcao utiliza os programas utilizados pelo professor
3 %  (nn1h_k_folds.m e analysis.m) para automatizar o processo de teste para
4 %  diversos valores de neuronios na camada intermediaria e numeros de
5 %  iteracoes a serem utilizadas pelo algoritmo otimizador. Recebe como
6 %  parametro o valor maximo do numero de neuronios.
```

```
%  AUTOR: Gustavo CIOTTO PINTON
function numberNeuronsMLP (N)

close all;

erro_tot_avg_iter = [];
mean_eqmv_min_avg_itr = [];

% Conjuntos a serem testados para o numero de neuronios na camada
% intermediaria e iteracoes do algoritmo otimizador
neurons_interval = 5:N;
iter_interval = [50:50:200  300:100:1000];

% Para cada nukmero de iteracoes
for n = iter_interval

    % Vetor linha que contera todas as medias dos erros quadraticos medios
    % de validacao das MLPs treinadas.
    mean_eqmv_min_avg_array = [];

    % Vetor linha que contera todas as medias dos erros quadraticos medios
    % de teste das MLPs treinadas.
    error_tot_avg_array = [];

    for i=neurons_interval

        disp(['==== ' num2str(n)  ' ITERACOES ==== ' num2str(i) ' NEURONS =====']);

        % Treimanento da rede MLP utilizando o algoritmo fornecido pelo
        % professor. As unicas alteracoes realizadas foram que, ao inves de
        % pedirmos ao usuario os dados, a funcao os recebe como parametro.
        % 'matrizes' eh o arquivo contendo as matrizes X e S. 1 eh a
        % resposta para o metodo 'Start the training from a random initial
        % condition' e 10, o numero de folds
        nn1h_k_folds('matrizes', 10, i, 1, n);

        % Da mesma forma, analysis.m foi alterada para retornar como
        % parametro os erros que foram calculados durante sua execucao.
        % Parametros tambem substituem pedidos de entrada pelo teclado
        [error_tot_avg mean_eqmv_min_avg eqm_ens] = analysis('matrizes', 1, 10);

        % Armazena erros no respectivos vetores
        mean_eqmv_min_avg_array = [mean_eqmv_min_avg_array mean_eqmv_min_avg];
        error_tot_avg_array = [error_tot_avg_array error_tot_avg];

    end

    % erro_tot_avg_iter eh a matriz que armazena todos o erros gerados para
    % cada valor de iteracao. Cada coluna dessa matriz representa os erros
    % de teste gerados para um valor n (iteracao) para todos o valores de neuronios
    % (i) na camada intermediaria. mean_eqmv_min_avg_itr representa a mesma
    % coisa, mas para os erros de validacao
    erro_tot_avg_iter = [erro_tot_avg_iter error_tot_avg_array'];
    mean_eqmv_min_avg_itr = [mean_eqmv_min_avg_itr mean_eqmv_min_avg_array'];

    % Determinacao da menor media dos erros quadraticos medios de validacao
    % para este valor de n. Eh possivel determinar portanto o valor de i
    % correspondente a essa media.
    [A B] = sort(mean_eqmv_min_avg_array, 'ascend');
    disp(sprintf('Numero de neuronios para minimizar erro de validacao: %d', B(1) +
        min (neurons_interval) - 1));

    % Constroi-se o grafico para cada valor das iteracoes n das medias dos
```

```
    % erros quadraticos medios de validacao e teste.
    figure
    plot(neurons_interval, mean_eqmv_min_avg_array);
    xlabel('Numero de neuronios');
    hold on;
    plot(neurons_interval, error_tot_avg_array, 'r');
    legend('Erro de validacao', 'Erro de teste');
    plot(neurons_interval, mean_eqmv_min_avg_array, '*');
    plot(neurons_interval, error_tot_avg_array, 'r*');
    title(sprintf('Erros para N = 5 .. %d e n. iteracoes = %d', i, n));

    grid on;
    hold off;
end

% Enfim, controi-se o ultimo grafico contendo as medias das performances
% calculadas anteriormente em funcao do numero de iteracoes.
figure
plot (iter_interval, mean(mean_eqmv_min_avg_itr));
hold on;
plot (iter_interval, mean(erro_tot_avg_iter), 'r');
legend('Media dos erros de validacao', 'Media dos erros de teste');
xlabel('Numero de iteracoes');
plot (iter_interval, mean(mean_eqmv_min_avg_itr),'*');
plot (iter_interval, mean(erro_tot_avg_iter), '*r');
title('Media dos erros para valores de N = 5 .. 20');
hold off;
grid on;

end
```

Programa 4: `elm.m` - Automatiza treinamento e análise de redes *ELMs*.

```
%% Funcao elm
%  Esta funcao treina maquinas de aprendizado extremo para diversos valores
%  do numero de neuronios na camada intermediaria. Utilizamos aqui o metodo
%  de crossfold validation e a otimizacao dos pesos sinapticos da ultima
%  camada eh realizada atraves do metodo de minimos quadrados regularizado.
%  AUTOR: Gustavo CIOTTO PINTON
function elm

close all;

E = 5 + 1; % Numero de entradas em cada neuronio (5 entradas e mais 1 constante).

% Numero de folds a ser utilizado
folds = 10;

% Obtem os dados a partir de denge_SP.mat
M = 5;
R = 1;
[X, Y] = gera_dados('dengue_SP.mat', 'resultado.mat', M, R);

% C eh o tamanho do conjunto e V, do conjunto de validacao
C = length (X(:,1));
V = round(C/folds);

% Variaveis que armazenarao o menor erro encontrado e os respectivos
% numeros de neuronios e parametro c.
m_erro = inf;
n_menor = 0;
c_menor = 0;

```

```
for n=[100 150 250 400 500 1000]

    % Inicia aleatoriamente os pesos da camada intermediaria. w_int eh uma
    % matriz em que a n-esima coluna representa os pesos sinapticos do
    % n-esimo neuronio.
    w_int = randn(E, n);

    erro_reg_k_folds = [];
    erro_reg_tr_k_folds = [];
    w_k_folds = [];

    for k = 1:folds

        % gera a ko. pasta para ser usada como validacao
        k_fold_set = (k - 1)*V + 1 : k*V;

        % pega o restante para treinamento. Setdiff realiza a diferenca de
        % conjuntos entre { 1 2 .... C } e os indices selecionados para a
        % validacao
        conjunto_treinamento = setdiff(1:C, k_fold_set);

        % Inicia matrizes H
        Htr = zeros (C - V, n );
        Hval = zeros (V, n );


        % Inicializa matriz H usada para o treinamento. Cada elemento ixk desta
        % matriz eh da forma f( wk * xi ) , em * eh o produto interno dos
        % pesos sinapticos do ko. neuronio com i-esima linha da entrada
        aux = 1;
        for i = conjunto_treinamento

            for k = 1:n

                Htr(aux, k) = tanh([ X(i,:) 1] * w_int(:, k));

            end

            aux = aux + 1;
        end

        % Adiciona um coluna de 1s referente a entrada constante do
        % neuronios da ultima camada
        Htr = [Htr ones(C-V, 1)];


        % Inicializa matriz H usada para o validacao. Mesmo raciocinio
        % utilizado no caso precedente
        aux = 1;
        for i = k_fold_set

            for k = 1:n

                Hval(aux, k) = tanh([ X(i,:) 1] * w_int(:, k));

            end
            aux = aux + 1;
        end

        % Adiciona um coluna de 1s referente a entrada constante do
        % neuronios da ultima camada
        Hval = [Hval ones(V, 1)];

```

```
94          Ytr = Y (conjunto_treinamento,:);
95          Yval = Y (k_fold_set,:);
96
97          erro_reg = [];
98          erro_reg_tr = [];
99          w_reg = [];
100
101         for c = -24:1:25
102
103             % Calcula caso regularizado segundo enunciado (pag. 6)
104             HHt_tr_inv = (Htr'*Htr + (2^c)*eye(n+1)) \ eye (n+1);
105             w = HHt_tr_inv * Htr'*Ytr;
106
107             % Armazena vetor w que acabou de ser calculado. w_reg
108             % eh uma matriz n x 50 portanto em que cada coluna eh uma vetor b
109             % calculado para um valor de c
110             w_reg = [w_reg w];
111
112             % Calcula e armazena erros quadraticos medios de validacao e
113             % treinamento. erro_reg e erro_reg_tr sao vetores linhas (1x50), em cada
114             % coluna eh o erro associado ao parametro c.
115             erro_reg = [erro_reg  sqrt((norm(Hval*w - Yval)^2)/V)];
116             erro_reg_tr = [erro_reg_tr sqrt((norm(Htr*w - Ytr)^2)/C)];
117
118         end
119
120
121         % Guarda a matriz w_reg calculada nas iteracoes passadas numa matriz
122         % tridimensional, que contera 10 matrizes w_reg, calculadas para cada
123         % uma das configuracoes das folds.
124         w_k_folds = cat(3, w_k_folds, w_reg);
125
126
127         % Guarda os vetores erro_reg e erro_reg_tr nas matrizes
128         % erro_reg_k_folds e erro_reg_tr_k_folds, respectivamente, de forma que
129         % cada linha dessas matrizes corresponda a uma das configuracoes da
130         % folds. Cada coluna corresponde ao erro calculado para um c. Exemplo:
131         % a linha 1 e coluna 4 dessas matrizes contem os erros correspondentes
132         % a situcao em que a primeira 1/k parte dos dados foi usada para validacao
133         % e c eh igual a % 2^-20.
134         erro_reg_k_folds = [erro_reg_k_folds; erro_reg];
135         erro_reg_tr_k_folds = [erro_reg_tr_k_folds; erro_reg_tr];
136     end
137
138     % Faz o grafico das perfomances medias dos erros em funcao do parametro c.
139     % mean (matriz) retorna um vetor linha com as medias de cada coluna da
140     % matriz. Neste caso, obteremos a media dos erros para todos os c.
141     fig = figure
142     semilogx( 2.^(-24:25), mean(erro_reg_k_folds));
143     hold on;
144     semilogx( 2.^(-24:25), mean(erro_reg_k_folds),'*');
145     hold off;
146     title(['Evolucao do erro quadratico medio para N = ' num2str(n)]);
147     ylabel('Media do erro quadratico medio das 10 execucoes');
148     xlabel('Parametro c');
149     grid on
150
151     % Salva figura diretamente em arquivo
152     print(fig,sprintf('elm_%d_neurons', n),'-dpng');
153
154     [menor_erro, menor_index] = sort(mean(erro_reg_k_folds), 'ascend');
155
156     w_k_folds(:,menor_index(1),:)
```

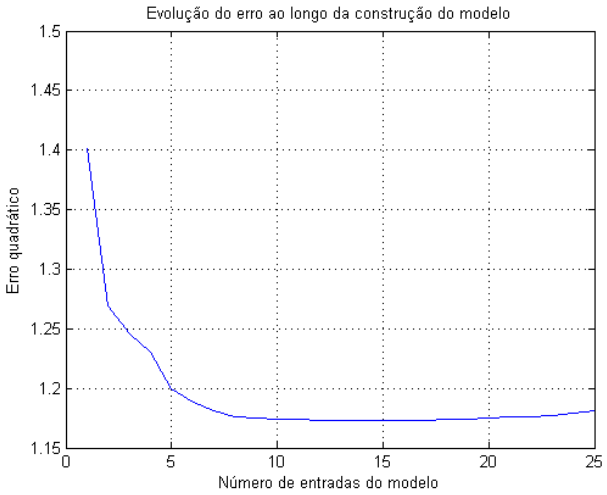
```
157     disp(['Vetores b para c = ' int2str(menor_index(1) - 25) ' e erro medio de '...
158                                                 num2str(menor_erro(1))]);
159 
160     % Guarda menor erro encontrada ate aqui, juntamente com as informacoes
161     % referentes a ele.
162     if m_erro > menor_erro(1)
163 
164         m_erro = menor_erro(1);
165         n_menor = n;
166         c_menor = menor_index(1) - 25;
167 
168     end
169 end
170 
171 % Imprime menor erro encontrado e informacoes referentes a ele.
172 disp(sprintf('Menor erro quadratico medio obtido %d para N = %d e c = 2^%d = %d', ...
173      m_erro, n_menor, c_menor, 2^c_menor));
174 end
```

## 5.2 Figuras

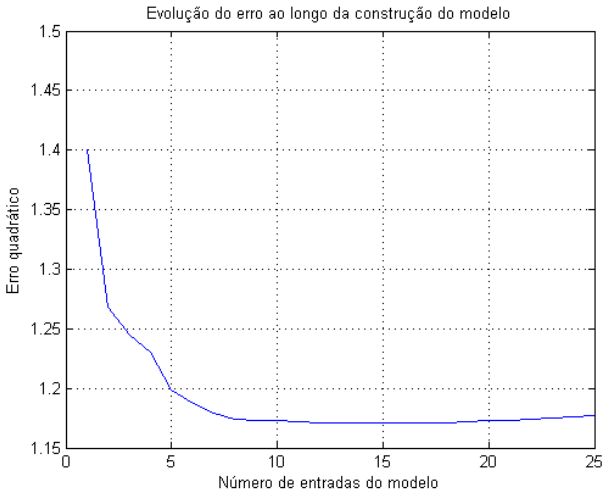
A figura 9 possui todos os resultados das execuções do *forward selection* para a série dos *sunspots*. Observa-se que o comportamento das curvas em todas elas é muito parecido, apresentando um mínimo para um número de variáveis próximo de 15. *(Rever análise completa na seção 2 item 3c).*


(a) Resultado da execução 1.


(b) Resultado da execução 2.




(c) Resultado da execução 3.

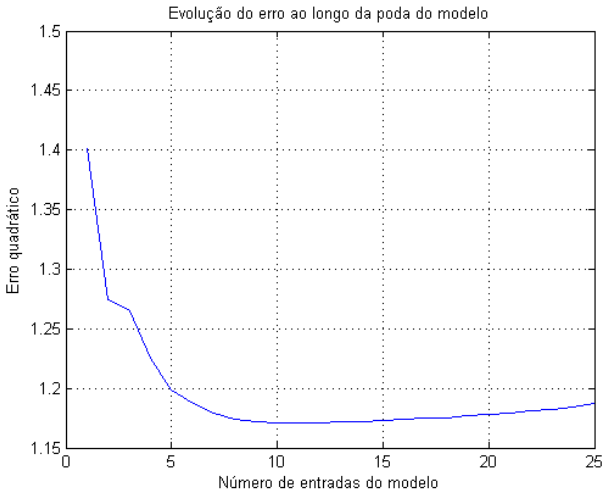



(d) Resultado da execução 4.

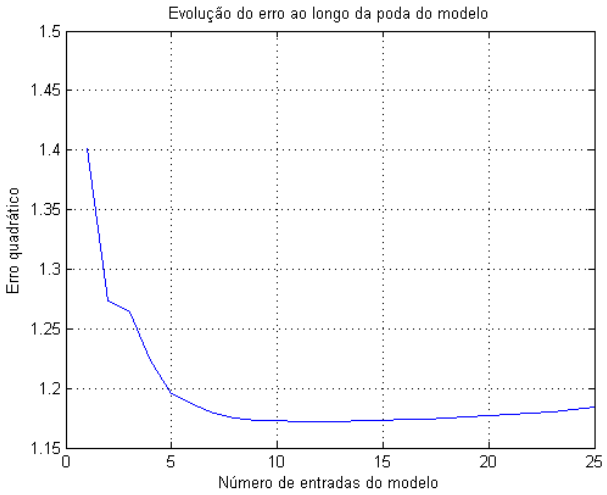

(e) Resultado da execução 5.

Figura 9: Resultados das execuções da *forward selection* para os dados de `sunspot.mat`.

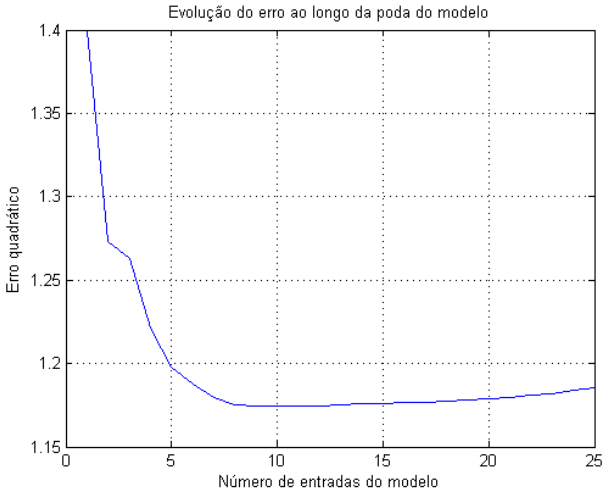
A figura 10 possui todos os resultados das execuções do *backward elimination* para a série dos *sunspots*. Assim como o caso anterior, observa-se que o comportamento das curvas em todas elas é muito parecido, apresentando um mínimo para um número de variáveis próximo de 15. *(Rever análise completa na seção 2 item 3c).*



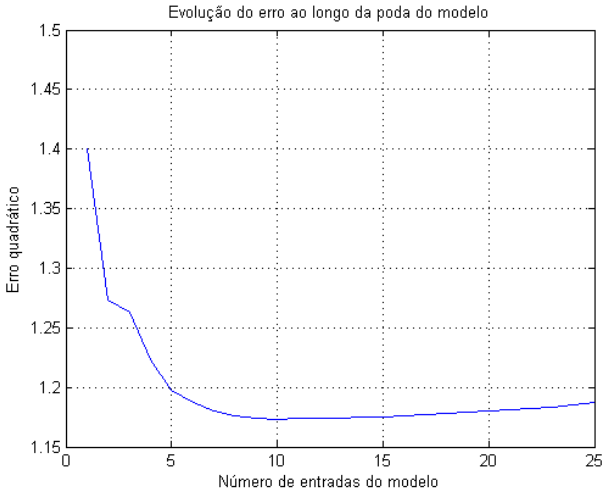

(a) Resultado da execução 1.




(b) Resultado da execução 2.




(c) Resultado da execução 3.

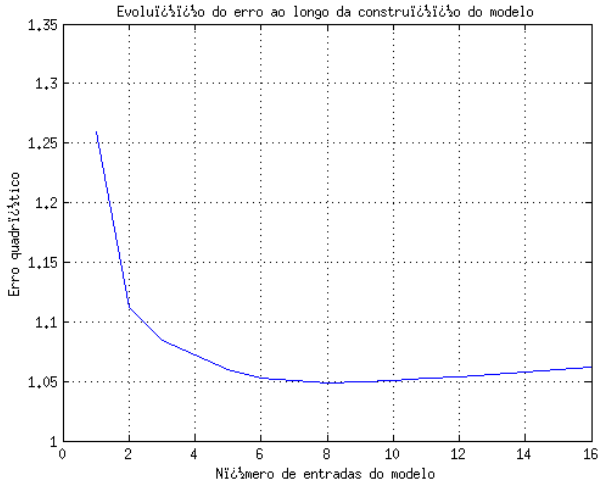



(d) Resultado da execução 4.

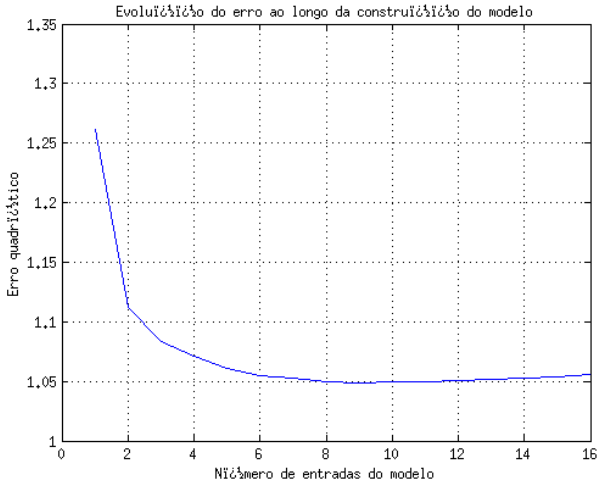

(e) Resultado da execução 5.

Figura 10: Resultados das execuções da *backward elimination* para os dados de `sunspot.mat`.

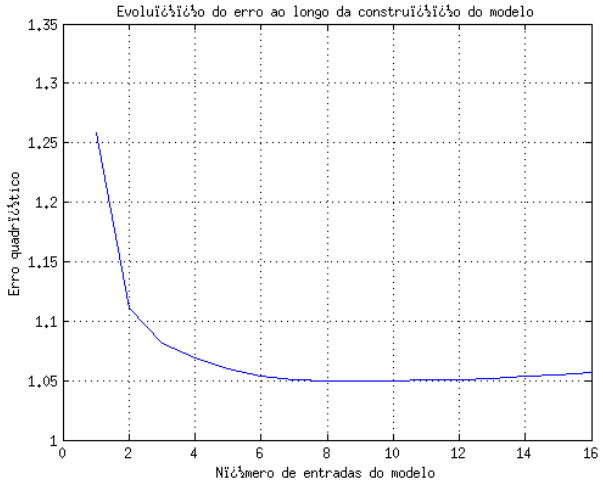
A figura 11 possui todos os resultados das execuções do *forward selection* para os dados contidos em `wineq.mat`. Observa-se que o comportamento das curvas em todas elas é muito parecido, apresentando um mínimo para um número de variáveis próximo de 8. *(Rever análise completa na seção 2 item 4c).*



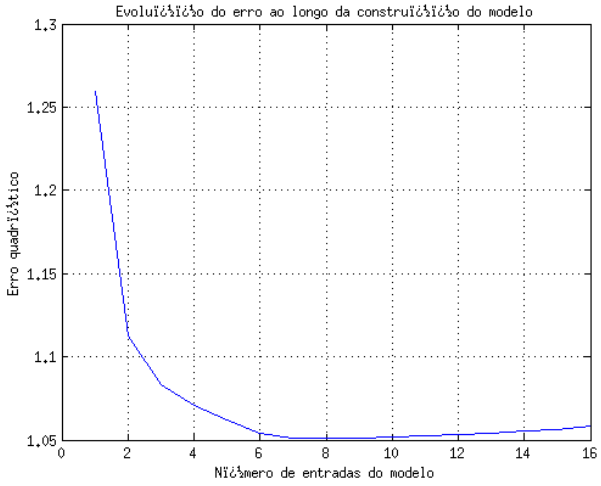

(a) Resultado da execução 1.



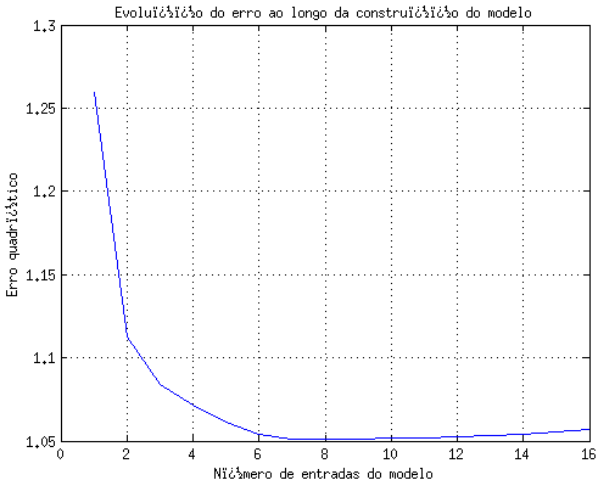

(b) Resultado da execução 2.




(c) Resultado da execução 3.



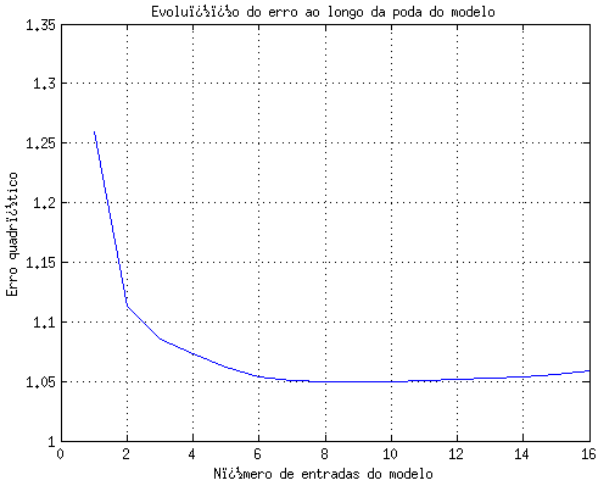

(d) Resultado da execução 4.



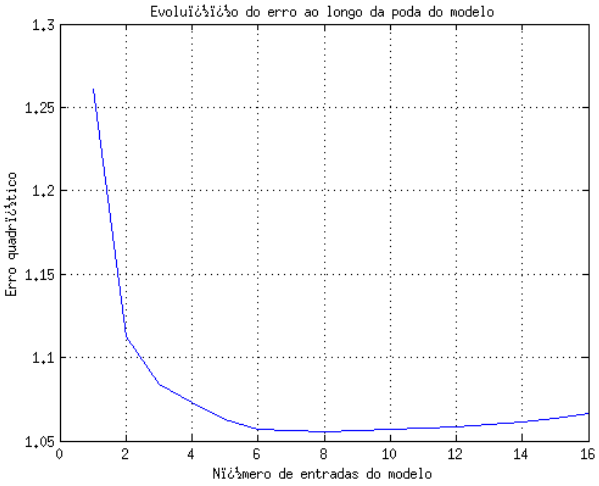

(e) Resultado da execução 5.

Figura 11: Resultados das execuções da *forward selection* para os dados de `wineq.mat`.

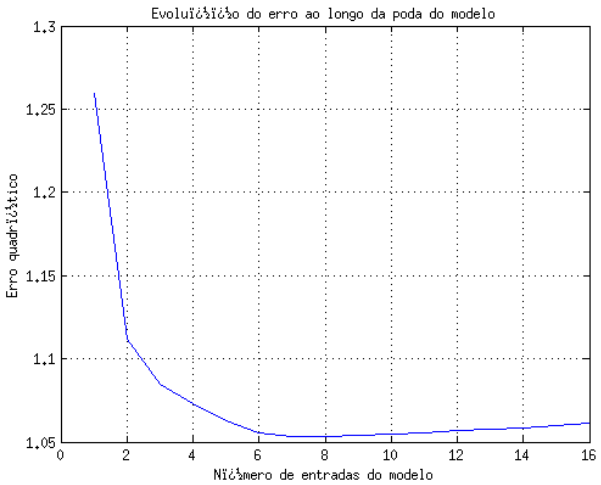
A figura 12 possui todos os resultados das execuções do *backward elimination* para os dados contidos em `wineq.mat`. Observa-se que o comportamento das curvas em todas elas é muito parecido, apresentando um mínimo para um número de variáveis próximo de 10. *(Rever análise completa na seção 2 item 4c).*



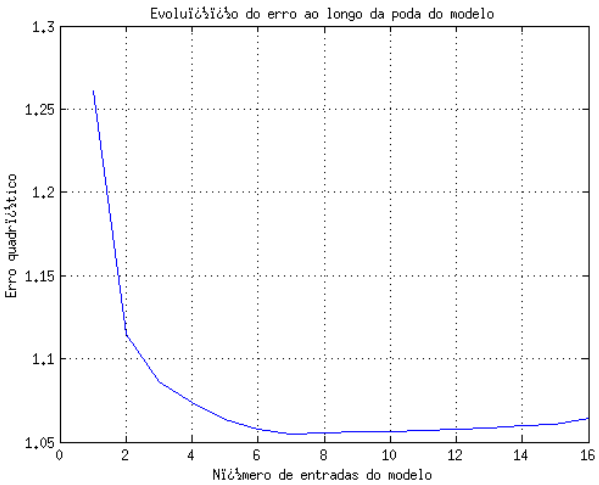

(a) Resultado da execução 1.



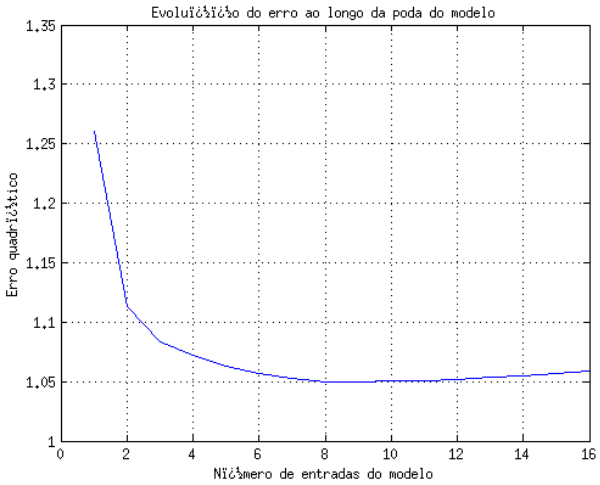

(b) Resultado da execução 2.




(c) Resultado da execução 3.



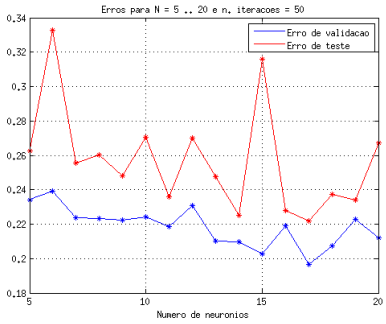

(d) Resultado da execução 4.



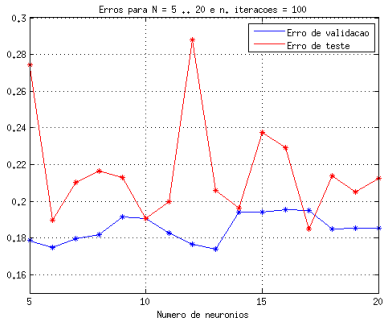

(e) Resultado da execução 5.

Figura 12: Resultados das execuções da *backward elimination* para os dados de `wineq.mat`.

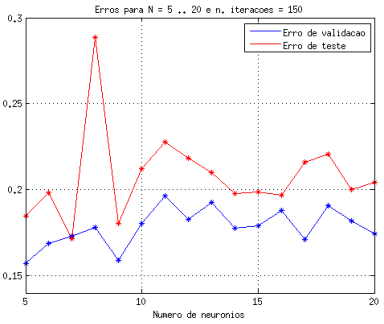
Resultados obtidos para diversos valores do número de iterações do algoritmo otimizador. Destaca-se que os resultados não apresentam nenhum comportamento bem definido, mas tendem a diminuir com o aumento de $i$. *(Rever análise completa na seção 3.3.1.)*



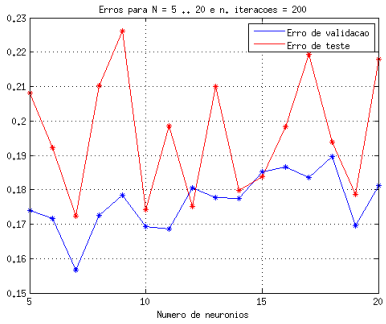

(a) $i = 50$.



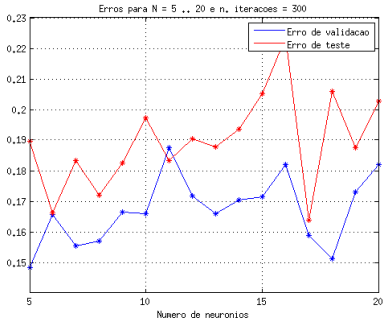

(b) $i = 100$.



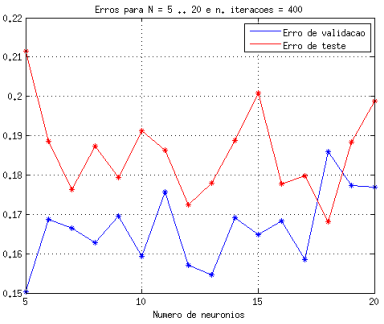

(c) $i = 150$.



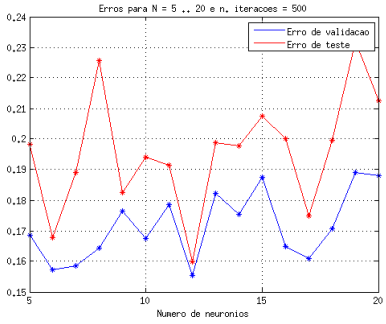

(d) $i = 200$.



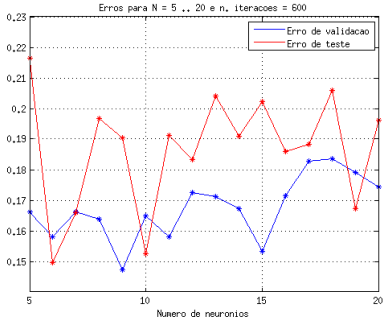

(e) $i = 300$.



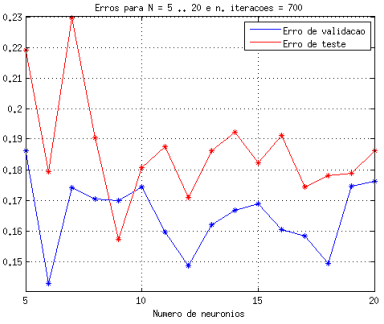

(f) $i = 400$.



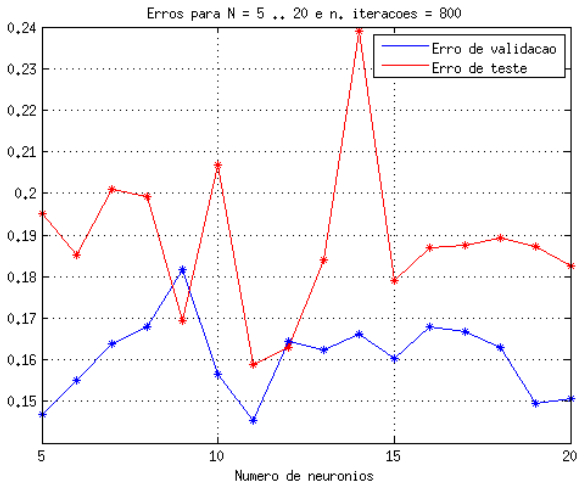

(g) $i = 500$.




(h) $i = 600$.




(i) $i = 700$.

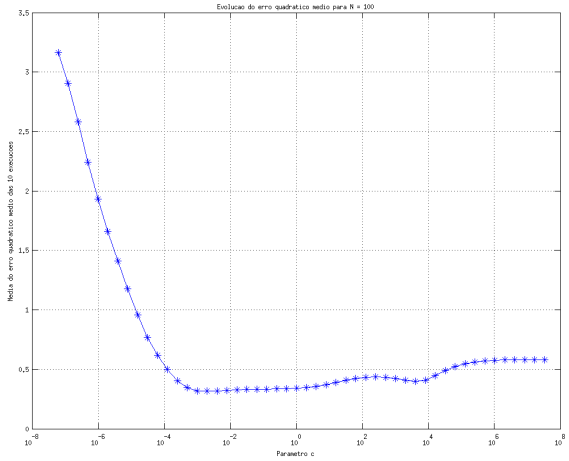



(j) $i = 800$.

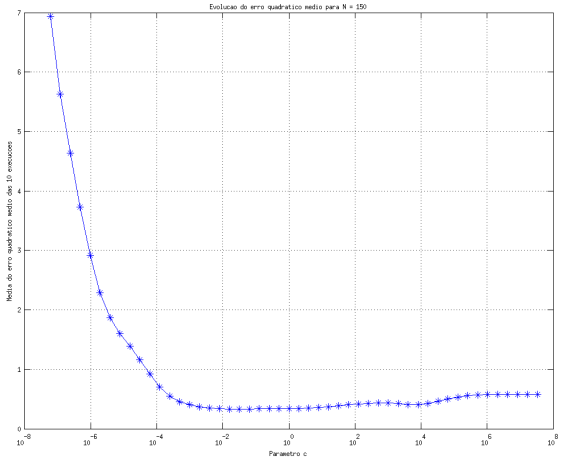

(k) $i = 900$.

Figura 13: Médias dos erros de validação e de teste para vários valores possíveis $i$ de iteração.

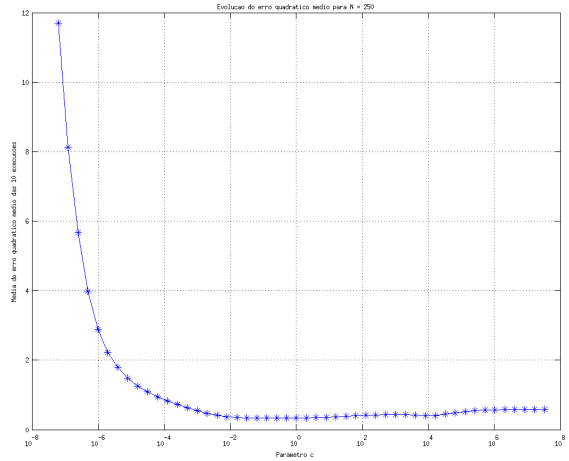
Observa-se que, para valores de $c$ que tendem a 0, as ELMs produzem um grande erro quadrático médio junto aos dados de validação. Nota-se também que o erro tende a se estabilizar para $c$ muito grande.



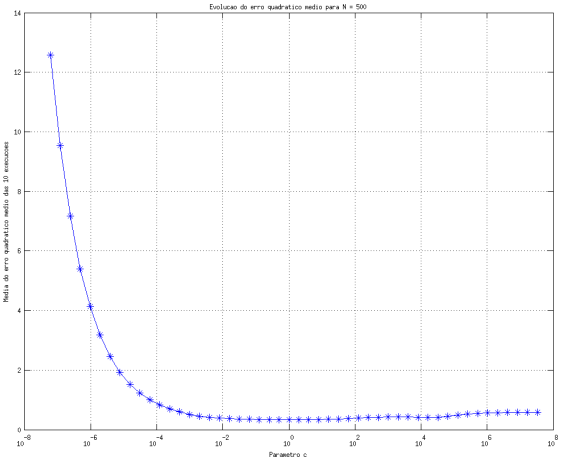

(a) $N = 100$



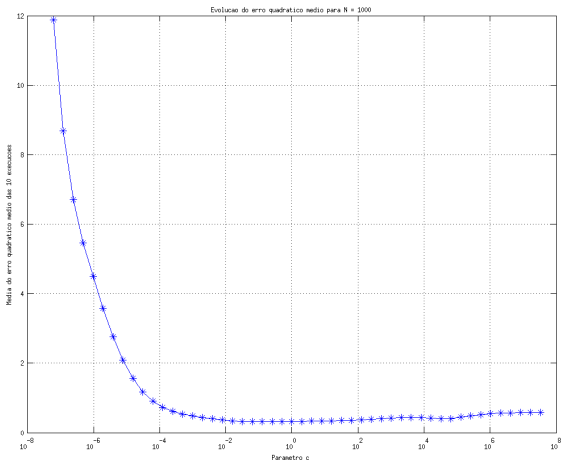

(b) $N = 150$




(c) $N = 250$




(d) $N = 500$




(e) $N = 1000$

Figura 14: Resultados das execuções de treinamento e validação de ELMs para diversos números de neurônios.