IA006 - Exercícios de Fixação de Conceitos

EFC 1 - 1s2019

Pedro Mariano Sousa Bezerra RA: 118383

Parte 1 - Atividades teóricas

Exercício 1

a)

$$P(A^C) = 1 - P(A) = 1 - \frac{1}{3} = \frac{2}{3}$$

b)

$$P(A^C \cup B) = P(A^C) + P(B) - P(A^C \cap B) = P(A^C) + P(B) - [P(B) - P(AB)]$$
$$P(A^C \cup B) = P(A^C) + P(AB) = \frac{2}{3} + \frac{1}{6} = \frac{5}{6}$$

c)

$$P(A \cup B^C) = P(A) + P(B^C) - P(A \cap B^C) = P(A) + 1 - P(B) - [P(A) - P(AB)]$$
$$P(A \cup B^C) = 1 - P(B) + P(AB) = 1 - \frac{1}{4} + \frac{1}{6} = \frac{11}{12}$$

d)

$$P(AB^C) = P(A) - P(AB) = \frac{1}{3} - \frac{1}{6} = \frac{1}{6}$$

e)

$$P(A^C \cup B^C) = P(A^C) + P(B^C) - P(A^C \cap B^C) = P(A^C) + P(B^C) - [P(B^C) - P(AB^C)]$$
$$P(A^C \cup B^C) = P(A^C) + P(AB^C) = \frac{2}{3} + \frac{1}{6} = \frac{5}{6}$$

Exercício 2

a) A função densidade de probabilidade $f_X(x)$ é dada por:

$$f_X(x) = \begin{cases} 1/2, & \text{se } 0 \leq x \leq 2 \\ 0, & \text{caso contrário} \end{cases}$$

A função cumulativa associada $F_X(x)$ é dada por:

$$F_X(x) = \int_{-\infty}^{x} f_X(\xi) d\xi = \begin{cases} 0, & \text{se } x < 0 \\ x/2, & \text{se } 0 \leq x \leq 2 \\ 1, & \text{se } x > 2 \end{cases}$$

b)

$$E(X) = \int_{-\infty}^{\infty} x f_X(x) dx = \int_0^2 \frac{1}{2} x dx = \left. \frac{x^2}{4} \right|_0^2 = 1$$

$$E(X^2) = \int_{-\infty}^{\infty} x^2 f_X(x) dx = \int_0^2 \frac{1}{2} x^2 dx = \left. \frac{x^3}{6} \right|_0^2 = \frac{4}{3}$$

$$E(X^3) = \int_{-\infty}^{\infty} x^3 f_X(x) dx = \int_0^2 \frac{1}{2} x^3 dx = \left. \frac{x^4}{8} \right|_0^2 = 2$$

Exercício 3

a) A variável X_2 possui maior entropia, pois como as probabilidades são iguais para todo o domínio de valores (eventos equiprováveis), a incerteza média associada à variável é maior.

b)

$$H(X) = -\sum_x p(x) \log_2[p(x)]$$

In [1]:
```
import numpy as np
p1 = np.array([0.1, 0.2, 0.3, 0.4])
H1 = -np.sum(p1*np.log(p1)/np.log(2))
print H1
```

```
1.84643934467
```

$$H(X_1) = -\{0{,}1 \cdot \log_2[0{,}1] + 0{,}2 \cdot \log_2[0{,}2] + 0{,}3 \cdot \log_2[0{,}3] + 0{,}4 \cdot \log_2[0{,}4]\} \cong 1{,}85$$

In [2]:
```
p2 = np.array([0.25]*4)
H2 = -np.sum(p2*np.log(p2)/np.log(2))
print H2
```

```
2.0
```

$$H(X_2) = -4 \cdot 0{,}25 \cdot \log_2[0{,}25] = 2$$

c)

$$D(P_1||P_2) = \sum_x P_1(x)\log_2\left[\frac{P_1(x)}{P_2(x)}\right] =$$
$$= 0{,}1 \cdot \log_2\left[\frac{0{,}1}{0{,}25}\right] + 0{,}2 \cdot \log_2\left[\frac{0{,}2}{0{,}25}\right] + 0{,}3 \cdot \log_2\left[\frac{0{,}3}{0{,}25}\right] + 0{,}4 \cdot \log_2\left[\frac{0{,}4}{0{,}25}\right] \cong 0{,}15$$

In [3]:
```
D1_2 = np.sum(p1 * np.log(p1/p2)/np.log(2))
print D1_2
```

```
0.153560655329
```

$$D(P_2||P_1) = \sum_x P_2(x)\log_2\left[\frac{P_2(x)}{P_1(x)}\right] =$$
$$= 0{,}25 \cdot \log_2\left[\frac{0{,}25}{0{,}1}\right] + 0{,}25 \cdot \log_2\left[\frac{0{,}25}{0{,}2}\right] + 0{,}25 \cdot \log_2\left[\frac{0{,}25}{0{,}3}\right] + 0{,}25 \cdot \log_2\left[\frac{0{,}25}{0{,}4}\right] \cong 0{,}18$$

In [4]:
```
D2_1 = np.sum(p2 * np.log(p2/p1)/np.log(2))
print D2_1
```

```
0.175687469707
```

Exercício 4

a) Para uma amostra, a verossimilhança é dada por:

$$p(x;\mu) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left[\frac{-(x-\mu)^2}{2\sigma^2}\right]$$

b) Para N amostras independentes, tem-se a seguinte expressão para a verossimilhança:

$$\mathcal{L}(\mu) = \prod_{k=1}^{N} p(x^{(k)};\mu) = \prod_{k=1}^{N}\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left[\frac{-(x^{(k)}-\mu)^2}{2\sigma^2}\right] = \frac{1}{(\sqrt{2\pi\sigma^2})^N}\exp\left[-\sum_{k=1}^{N}\frac{(x^{(k)}-\mu)^2}{2\sigma^2}\right]$$

c) O logaritmo da verossimilhança tem a seguinte expressão:

$$\log(\mathcal{L}(\mu)) = \log\left(\frac{1}{(\sqrt{2\pi\sigma^2})^N}\exp\left[\sum_{k=1}^{N}\frac{-(x^{(k)}-\mu)^2}{2\sigma^2}\right]\right) = -\frac{N}{2}\log(2\pi\sigma^2) - \sum_{k=1}^{N}\frac{(x^{(k)}-\mu)^2}{2\sigma^2}$$

Calculamos a derivada desta expressão:

$$\frac{d\log(\mathcal{L}(\mu))}{d\mu} = -\frac{d}{d\mu}\sum_{k=1}^{N}\frac{(x^{(k)}-\mu)^2}{2\sigma^2} = -\frac{1}{2\sigma^2}\sum_{k=1}^{N}\frac{d}{d\mu}(x^{(k)}-\mu)^2 =$$
$$= \frac{1}{\sigma^2}\sum_{k=1}^{N}(x^{(k)}-\mu) = \frac{1}{\sigma^2}\left(\sum_{k=1}^{N}x^{(k)} - N\mu\right)$$

O estimador de μ que maximiza o logaritmo da verossimilhança é dado pelo valor onde a derivada se anula:

$$\frac{d\log(\mathcal{L}(\hat{\mu}))}{d\hat{\mu}} = 0 \Leftrightarrow \sum_{k=1}^{N}x^{(k)} - N\hat{\mu} = 0$$

Assim, o estimador é dado por:

$$\hat{\mu} = \frac{1}{N}\sum_{k=1}^{N} x^{(k)}$$

Como esperado, o estimador de máxima verossimilhança da média da distribuição gaussiana corresponde à média das observações das N amostras.

Parte 2 - Atividade computacional

a) Leitura do arquivo de dados e pré-processamento:

In [5]:
```
sunspot = np.loadtxt('sunspot.txt',usecols=np.arange(12)+1).reshape(-1)

N = len(sunspot)
K = 20

X = np.ones((N-K,K+1))
Y = np.zeros(N-K)

for i in np.arange(N-K):
    X[i,1:] = sunspot[i:i+K]
    Y[i] = sunspot[i+K]

# Dados de treinamento

X_train = X[:-60,:]
Y_train = Y[:-60]

# Dados de teste

X_test = X[-60:,:]
Y_test = Y[-60:]
```

1. Aplicação do método dos mínimos quadrados irrestrito:

In [6]:
```
# Treinamento

A = np.transpose(X_train).dot(X_train)
B = np.transpose(X_train).dot(Y_train)
w = np.linalg.solve(A,B)
```

Avaliação da raiz quadrada do erro quadrático médio (RMSE):

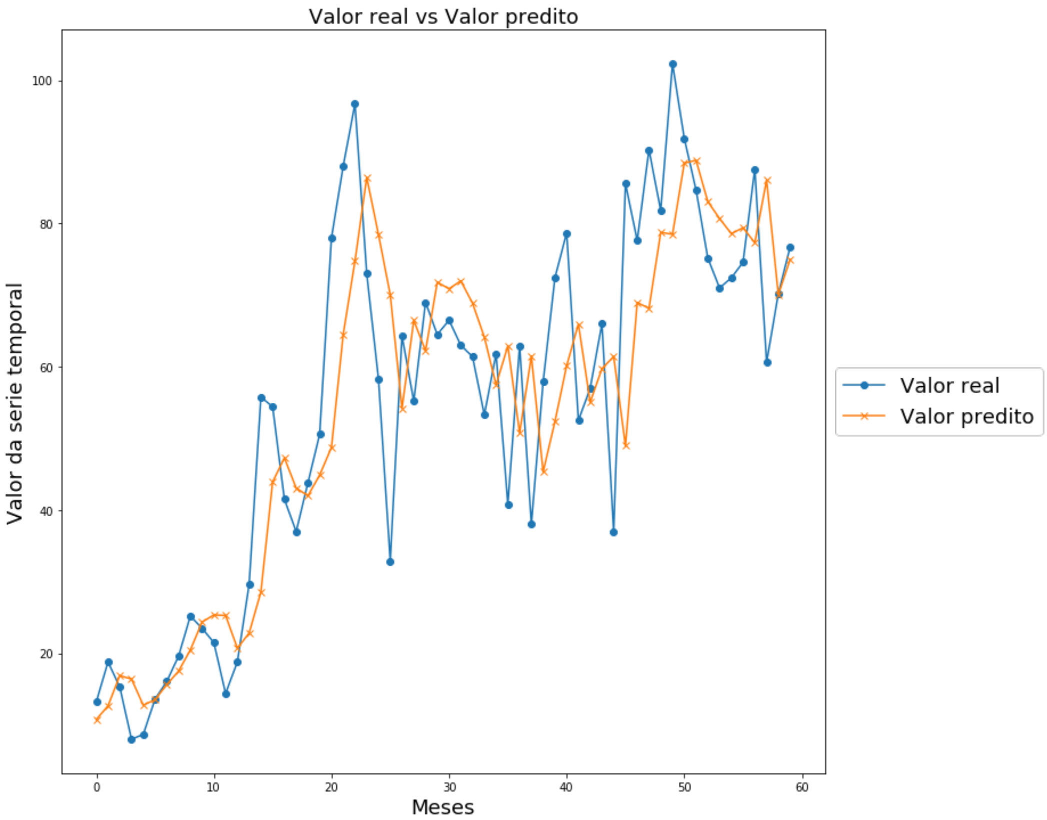
In [7]:
```
# Erro de treinamento

e_train = Y_train - X_train.dot(w)
RMSE_train = np.sqrt(np.mean(e_train**2))
print 'RMSE (treinamento) = ', RMSE_train

# Erro de teste

e_test = Y_test - X_test.dot(w)
RMSE_test = np.sqrt(np.mean(e_test**2))
print 'RMSE (teste) = ', RMSE_test
```

```
RMSE (treinamento) =  15.3702320885
RMSE (teste) =  14.2494511127
```

Observamos que o RMSE para o conjunto de dados de teste é ligeiramente inferior ao RMSE para o conjunto de dados de treinamento, o que significa que o preditor linear ótimo (no sentido de quadrados mínimos) teve um bom desempenho quando aplicado a dados novos.

2. Comparação das amostras de teste da série temporal com as respectivas estimativas geradas pelo preditor

In [8]:
```
import matplotlib.pyplot as plt
%matplotlib inline

Y_pred = X_test.dot(w) # estimativa gerada pelo preditor

plt.figure(figsize=(12, 12))
plt.plot(Y_test, label='Valor real', marker = 'o')
plt.plot(Y_pred, label='Valor predito', marker = 'x')
plt.xlabel('Meses', fontsize=18)
plt.ylabel('Valor da serie temporal', fontsize=18)
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), fontsize=18)
plt.title('Valor real vs Valor predito', fontsize=18)
plt.show()
```



O gráfico mostra que a série temporal obtida com as estimativas geradas pelo preditor linear se assemelha com a série temporal das amostras de teste, com valores relativamente próximos. No entanto, a série temporal estimada pelo preditor parece apresentar uma pequena defasagem com relação aos momentos de crescimento e decrescimento da série original.

b) O trecho de código a seguir executa a abordagem *backward elimination* utilizando validação cruzada do tipo *k-fold*.

In [9]:
```
## Parametros do k-fold

k = 5 # Numero de k-folds

np.random.seed(1)
ind = np.random.permutation(N-K) # Gera aleatoriamente uma permutacao do
s indices das linhas da matriz de entrada
ind_val = np.array_split(ind, k) # Contem os indices do conjunto de vali
dacao para cada fold
ind_train = [] # Contem os indices do conjunto de treinamento para cada
fold

X_train = []
Y_train = []

X_val = []
Y_val = []

# Construcao dos conjuntos de treinamento e validacao para cada fold

index = np.arange(k)
for fold in np.arange(k):
    # Concatena os pedacos de indval para compor os indices do conjunto
de treinamento
    ind_train.append(np.concatenate([ind_val[i] for i in np.delete(index
,fold)]))
    X_train.append(X[ind_train[fold],:])
    Y_train.append(Y[ind_train[fold]])

    X_val.append(X[ind_val[fold],:])
    Y_val.append(Y[ind_val[fold]])

# Conjunto completo de variaveis

RMSE_min = np.zeros(K) # RMSE minimo em funcao do numero de variaveis re
movidas

RMSE = np.zeros(k) # RMSE para cada fold

# k-fold
for fold in np.arange(k):

    # Treinamento

    A = np.transpose(X_train[fold]).dot(X_train[fold])
    B = np.transpose(X_train[fold]).dot(Y_train[fold])
    w = np.linalg.solve(A,B)

    # Erro de validacao

    e_val = Y_val[fold] - X_val[fold].dot(w)
    RMSE_val = np.sqrt(np.mean(e_val**2))
    #print 'RMSE (validacao) = ', RMSE_val
    RMSE[fold] = RMSE_val

RMSE_min[0] = np.mean(RMSE) # RMSE medio de validacao
#print RMSE_min[0]
```

In [17]:

```
## Backward elimination

remain_var = np.arange(1,K+1) # Lista de variaveis remanescentes
elim_var = [] # Lista de variaveis eliminadas

for nvar in np.arange(1,K): # Itera sobre todos os conjuntos possiveis d
e K - nvar variaveis

    RMSE_min_var = np.inf # RMSE minimo

    for var in remain_var:

        #print 'Var = ', var

        RMSE = np.zeros(k) # RMSE para cada fold
        ind_elim = elim_var + [var] # Lista de indices de colunas a sere
m removidas
        # print 'Colunas a serem removidas: ', ind_elim

        # k-fold
        for fold in np.arange(k):

            # Treinamento

            nX_train = np.delete(X_train[fold],ind_elim,1) # Remove as v
ariaveis da matriz de dados
            A = np.transpose(nX_train).dot(nX_train)
            B = np.transpose(nX_train).dot(Y_train[fold])
            w = np.linalg.solve(A,B)

            # Erro de validacao

            nX_val = np.delete(X_val[fold],ind_elim,1) # Remove as varia
veis da matriz de dados
            e_val = Y_val[fold] - nX_val.dot(w)
            RMSE_val = np.sqrt(np.mean(e_val**2))
            #print 'RMSE (validacao) = ', RMSE_val
            RMSE[fold] = RMSE_val

        nRMSE = np.mean(RMSE)
        # print 'RMSE medio = ', nRMSE
        if(nRMSE < RMSE_min_var):
            RMSE_min_var = nRMSE # Armazena o RMSE minimo
            best_var = var # Armazena a variavel eliminada que produziu
RMSE minimo

    #print 'melhor RMSE = ', RMSE_min_var
    #print 'melhor var = ', best_var
    RMSE_min[nvar] = RMSE_min_var

    remain_var = np.setdiff1d(remain_var,best_var) # Remove a variavel c
om melhor desempenho
    elim_var.append(best_var) # Atualiza a lista de variaveis removidas
    #print 'Variaveis restantes: ', remain_var
    print 'Variavel eliminada: %d - RMSE minimo = %f' % (best_var,RMSE_m
in_var)

#print RMSE_min
```

```
Variavel eliminada: 6 - RMSE minimo = 15.533124
Variavel eliminada: 11 - RMSE minimo = 15.517583
Variavel eliminada: 7 - RMSE minimo = 15.505234
Variavel eliminada: 9 - RMSE minimo = 15.494481
Variavel eliminada: 16 - RMSE minimo = 15.484665
Variavel eliminada: 2 - RMSE minimo = 15.482328
Variavel eliminada: 4 - RMSE minimo = 15.480163
Variavel eliminada: 13 - RMSE minimo = 15.479293
Variavel eliminada: 8 - RMSE minimo = 15.478668
Variavel eliminada: 14 - RMSE minimo = 15.478630
Variavel eliminada: 5 - RMSE minimo = 15.481437
Variavel eliminada: 10 - RMSE minimo = 15.488945
Variavel eliminada: 1 - RMSE minimo = 15.507312
Variavel eliminada: 18 - RMSE minimo = 15.560179
Variavel eliminada: 15 - RMSE minimo = 15.637978
Variavel eliminada: 19 - RMSE minimo = 15.814353
Variavel eliminada: 12 - RMSE minimo = 16.060805
Variavel eliminada: 3 - RMSE minimo = 16.131385
Variavel eliminada: 17 - RMSE minimo = 16.893431
RMSE minimo = 15.478630 para 10 variaveis removidas
Conjunto otimo de variaveis:  [ 1  3  5 10 12 15 17 18 19 20]
```

In [18]:

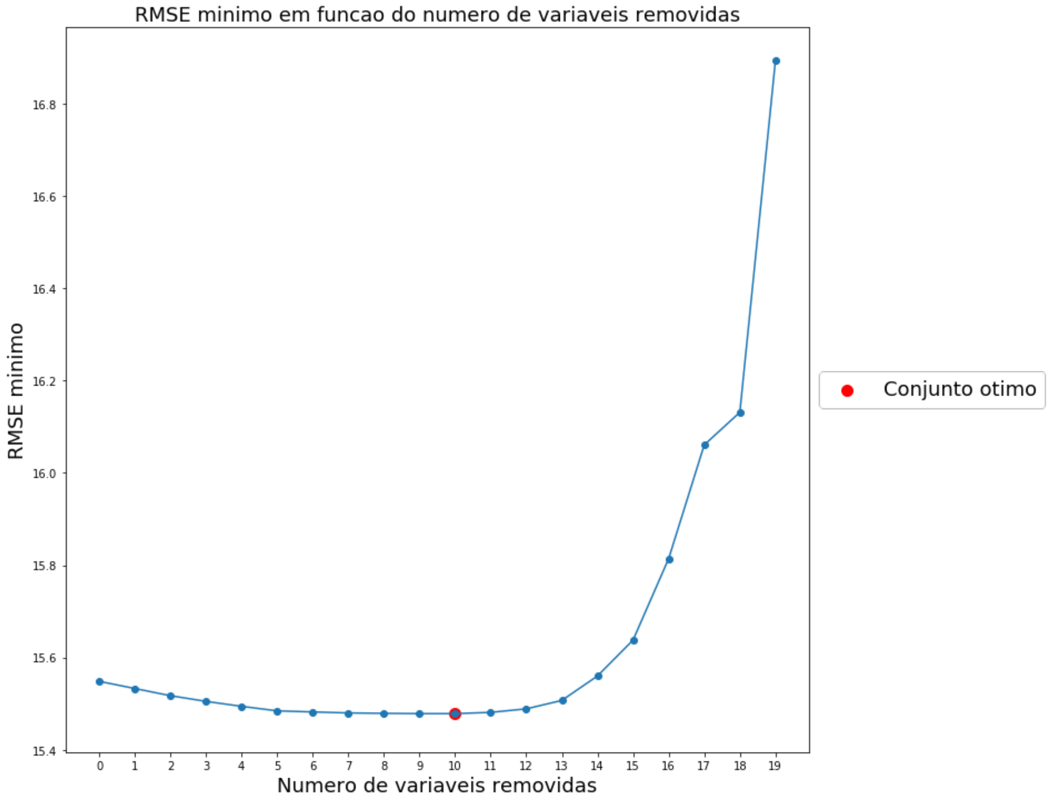
```
RMSE_opt = min(RMSE_min) # RMSE do conjunto otimo
nvar_opt = np.argmin(RMSE_min) # Numero de variaveis removidas no conjun
to otimo
print ('RMSE minimo = %f para %d variaveis removidas' % (RMSE_opt, nvar_
opt))

elim_var_opt = elim_var[:nvar_opt] # Variaveis eliminadas no conjunto ot
imo
var_opt = np.setdiff1d(np.arange(1,K+1),elim_var_opt) # Variaveis restan
tes no conjunto otimo
print 'Conjunto otimo de variaveis: ', var_opt
```

```
RMSE minimo = 15.478630 para 10 variaveis removidas
Conjunto otimo de variaveis:  [ 1  3  5 10 12 15 17 18 19 20]
```

O número de atrasos presentes no conjunto ótimo de atrasos identificado pelo *wrapper* é 10, metade do número de variáveis do conjunto completo. As variáveis presentes no conjunto ótimo são, a princípio, as mais relevantes para o preditor linear aqui utilizado - deve-se levar em conta, no entanto, que a abordagem *backward elimination* é gulosa e não leva em conta interações entre variáveis que já foram eliminadas e outras que ainda estão presentes no conjunto. Observa-se ainda que, neste caso, o RMSE mínimo (15,478630) foi ligeiramente maior do que o erro de treinamento (15,3702320885) e de teste (14,2494511127) obtidos no exercício anterior. No entanto, os resultados deste exercício apresentam maior confiabilidade, uma vez que trata-se de um desempenho médio de k execuções.

A seguir, mostramos a progressão do RMSE mínimo em função do número de variáveis removidas pelo *wrapper*.

In [11]:
```
plt.figure(figsize=(12, 12))
plt.plot(np.arange(K),RMSE_min, marker = 'o')
plt.xticks(np.arange(K))
plt.xlabel('Numero de variaveis removidas', fontsize=18)
plt.scatter(nvar_opt,RMSE_opt, marker ='o', color = 'r',s = 100, label='
Conjunto otimo')
plt.ylabel('RMSE minimo', fontsize=18)
plt.title('RMSE minimo em funcao do numero de variaveis removidas', font
size=18)
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), fontsize=18)
plt.show()
```



Como podemos observar, o RMSE mínimo em função do número de variáveis removidas inicialmente decresce de forma suave até atigingir o valor ótimo, e logo depois aumenta rapidamente. Acima de 14 variáveis removidas, o RMSE mínimo torna-se maior do que o RMSE do conjunto completo das variáveis - um comportamento esperado, visto que, quanto menos informação disponível, maior a dificuldade de se fazer uma boa predição.

Em seguida, executamos a etapa de regularização do preditor linear para cada conjunto de 1 a 20 variáveis encontrado pelo *wrapper* com RMSE mínimo. Definimos o parâmetro de regularizacação como $\lambda = 10^e$, variando e no intervalo $[-10, 5]$. Os parâmetros que produziram o menor RMSE para cada caso são apresentados na sequência.

In [12]:
```
# Parametro de regularizcao: lambda = 10^exp
exp = np.arange(-10,6, dtype=float) # Lista de expoentes para parametro
de regularizacao
param = np.zeros(K) # Lista de parametros para cada conjunto de variavei
s

for nvar in np.arange(K): # Itera sobre os conjuntos de K - nvar variave
is encontrados pelo wrapper

    ind_elim = elim_var[:nvar] # Lista de variaveis eliminadas na ordem
encontrada pelo wrapper

    RMSE_min_reg = np.inf

    for e in exp: # Itera sobre um intervalo de valores do parametro de
regularizacao

        # k-fold
        for fold in np.arange(k):

            # Treinamento

            nX_train = np.delete(X_train[fold],ind_elim,1) # Remove as v
ariaveis da matriz de dados
            A = np.transpose(nX_train).dot(nX_train) + 10**e*np.identity
(K-nvar+1)
            B = np.transpose(nX_train).dot(Y_train[fold])
            w = np.linalg.solve(A,B)

            # Erro de validacao

            nX_val = np.delete(X_val[fold],ind_elim,1) # Remove as varia
veis da matriz de dados
            e_val = Y_val[fold] - nX_val.dot(w)
            RMSE[fold] = np.sqrt(np.mean(e_val**2))
            #print 'RMSE (validacao) = ', RMSE_val

        RMSE_val = np.mean(RMSE) # RMSE medio de validacao

        # print 'RMSE = %f para expoente %d' % (RMSE_val,e)

        if(RMSE_val < RMSE_min_reg):
            RMSE_min_reg = RMSE_val
            best_reg = e

    param[nvar] = best_reg

    print ('%d variaveis: melhor parametro = %.1f, RMSE = %f' %(K-nvar,1
0**param[nvar],RMSE_min_reg))
```

```
20 variaveis: melhor parametro = 10.0, RMSE = 15.548537
19 variaveis: melhor parametro = 10.0, RMSE = 15.533013
18 variaveis: melhor parametro = 10.0, RMSE = 15.517500
17 variaveis: melhor parametro = 10.0, RMSE = 15.505140
16 variaveis: melhor parametro = 10.0, RMSE = 15.494378
15 variaveis: melhor parametro = 10.0, RMSE = 15.484589
14 variaveis: melhor parametro = 10.0, RMSE = 15.482254
13 variaveis: melhor parametro = 10.0, RMSE = 15.480091
12 variaveis: melhor parametro = 10.0, RMSE = 15.479221
11 variaveis: melhor parametro = 10.0, RMSE = 15.478596
10 variaveis: melhor parametro = 10.0, RMSE = 15.478558
9 variaveis: melhor parametro = 10.0, RMSE = 15.481362
8 variaveis: melhor parametro = 10.0, RMSE = 15.488867
7 variaveis: melhor parametro = 10.0, RMSE = 15.507236
6 variaveis: melhor parametro = 10.0, RMSE = 15.560100
5 variaveis: melhor parametro = 10.0, RMSE = 15.637898
4 variaveis: melhor parametro = 10.0, RMSE = 15.814295
3 variaveis: melhor parametro = 10.0, RMSE = 16.060755
2 variaveis: melhor parametro = 100.0, RMSE = 16.131299
1 variaveis: melhor parametro = 10.0, RMSE = 16.893386
```

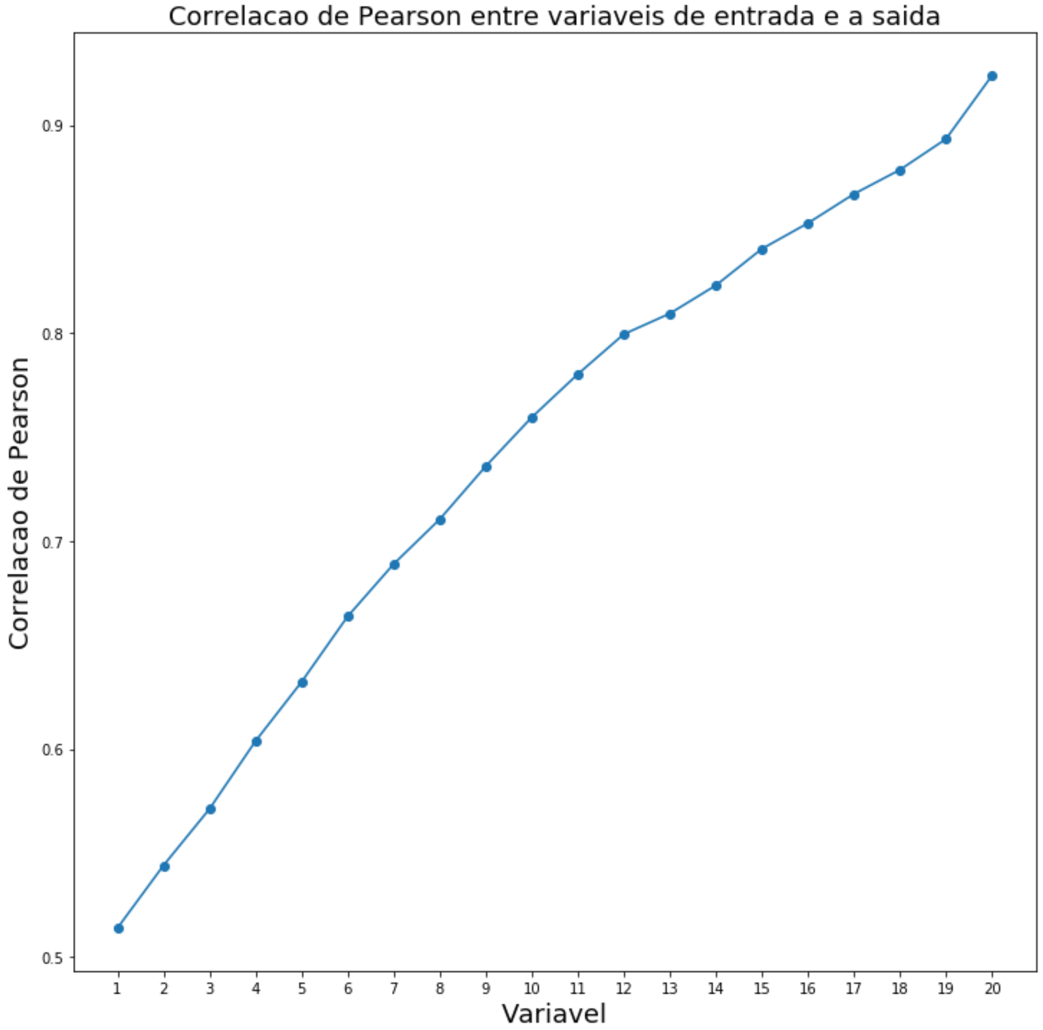
Como podemos ver, para a maioria dos conjuntos de variáveis encontradas pelo *wrapper*, o valor de λ que produziu o menor erro médio de validação foi 10^1. A exceção é para o caso do conjunto com 2 variáveis, onde $\lambda = 10^2$ é o valor ótimo encontrado. Como esperado, o erro obtido com regularização para o conjunto ótimo de 10 variáveis (15,478558) foi ligeiramente menor do que o erro obtido para o modelo não regularizado (15,478630).

c) Vamos utilizar a correlação de Pearson para a metodologia do tipo filtro. O trecho de código a seguir calcula o coeficiente de correlação de Pearson entre duas amostras de dados $\mathbf{x}$ e $\mathbf{y}$.

In [13]:
```
def pearson_corr(x,y):

    mx = np.mean(x)
    my = np.mean(y)

    return np.sum((x - mx)*(y-my))/np.sqrt(np.sum((x-mx)**2)*np.sum((y-m
y)**2))
```

A seguir, avaliamos a correlação de Pearson entre cada variável de entrada e a saída, e visualizamos os valores dos coeficientes em um gráfico.

In [14]:
```
corr = np.zeros(K)

for var in np.arange(1,K+1):

    corr[var-1] = pearson_corr(X[:,var], Y)

plt.figure(figsize=(12, 12))
plt.plot(np.arange(1,K+1),corr, marker = 'o')
plt.xticks(np.arange(1,K+1))
plt.xlabel('Variavel', fontsize=18)
plt.ylabel('Correlacao de Pearson', fontsize=18)
plt.title('Correlacao de Pearson entre variaveis de entrada e a saida',
fontsize=18)
plt.show()
```



A figura mostra que a correlação das variáveis de entrada com a saída aumenta conforme a proximidade temporal destes valores na série de dados. A fim de realizar uma seleção de variáveis, vamos considerar apenas aquelas que possuírem coeficiente de correlação acima de 0,7.

In [15]:
```
threshold = 0.7 # Limiar da correlacao de Pearson para selecao de variav
eis
sel_var = np.where(corr > threshold)[0]+1
print sel_var
```

```
[ 8  9 10 11 12 13 14 15 16 17 18 19 20]
```

As variáveis selecionadas pela abordagem filtro são as 13 últimas variáveis do conjunto. Diferentemente da abordagem anterior, as variáveis selecionadas aqui são consecutivas. Embora as últimas variáveis também tenham sido selecionadas pelo *wrapper*, este também apresenta variáveis que foram excluídas pelo filtro por apresentarem baixa correlação com a saída. Apesar disso, a interação dessas variáveis com outras pode levar o preditor a ter um melhor desempenho, o que levou o *wrapper* a retê-las no conjunto.

A seguir, avaliamos o desempenho médio do preditor linear com regularização para o conjunto de variáveis selecionadas. O expoente e do parâmetro de regularização $\lambda = 10^e$ é variado no intervalo $[-5, 6]$.

In [16]:
```
# Parametro de regularizcao: lambda = 10^exp
exp = np.arange(-5,7, dtype=float) # Lista de expoentes para parametro de regularizacao
param = np.zeros(K) # Lista de parametros para cada conjunto de variaveis

RMSE_min_reg = np.inf

ind_var = np.insert(sel_var,0,0) # Indices das colunas a serem selecionadas

for e in exp: # Itera sobre um intervalo de valores do parametro de regularizacao

    # k-fold
    for fold in np.arange(k):

        # Treinamento

        nX_train = X_train[fold][:,ind_var] # Seleciona as variaveis da matriz de dados
        A = np.transpose(nX_train).dot(nX_train) + 10**e*np.identity(len(ind_var))
        B = np.transpose(nX_train).dot(Y_train[fold])
        w = np.linalg.solve(A,B)

        # Erro de validacao

        nX_val = X_val[fold][:,ind_var] # Seleciona as variaveis da matriz de dados
        e_val = Y_val[fold] - nX_val.dot(w)
        RMSE[fold] = np.sqrt(np.mean(e_val**2))
        #print 'RMSE (validacao) = ', RMSE_val

    RMSE_val = np.mean(RMSE) # RMSE medio de validacao

    # print 'RMSE = %f para expoente %d' % (RMSE_val,e)

    if(RMSE_val < RMSE_min_reg):
        RMSE_min_reg = RMSE_val
        best_reg = e

param[nvar] = best_reg

print ('%d variaveis: melhor parametro = %.1f, RMSE = %f' %(len(sel_var),10**param[nvar],RMSE_min_reg))
```

```
13 variaveis: melhor parametro = 100.0, RMSE = 15.732782
```

O melhor resultado foi obtido para $\lambda = 10^2$, com RMSE = 15.732782. Observamos que a abordagem de seleção de variáveis do tipo filtro obteve o pior desempenho para o preditor entre todos os casos aqui analisados. Ainda assim, os valores de RMSE obtidos são relativamente próximos. Os resultados mostram um exemplo do compromisso existente entre a simplicidade computacional da abordagem de seleção de variáveis adotada e o desempenho do preditor obtido.