Método de regressão utilizando o algoritmo de *Random Forest* para previsão de temperaturas com base em dados históricos – 2021/1*

*Nota: Projeto da discplina Aprendizado de Máquina do 1o. Semestre de 2021

Aluno: Carlos Fonseca
Programa de Pós Graduação em Eng. Elétrica
PPGEE/UFAM
Manaus/AM, Brasil
henrique_fonseca@hotmail.com

Aluno: Willian Guerreiro
Programa de Pós Graduação em Eng. Elétrica
PPGEE/UFAM
Manaus/AM, Brasil
wguerreiro31@gmail.com

***Resumo*—As variações de temperatura tem efeitos adversos nas principais capitais do mundo, principalmente, nas área de saúde e agricultura. Para que empresas e governos tomem medidas de precaução que atenuem esses impactos, os Institutos de Meteorologia estão, continuamente, aperfeiçoando seus métodos de previsão climática buscando alcançar a maior precisão possível. Neste trabalho, foi implementado o algoritmo *Random Forest*, ou floresta aleatória, para fazer a predição da temperatura máxima na cidade de Seattle com base em dados históricos de temperatura, velocidade do vento, precipitação pluviométrica, entre outros, disponibilizados no portal NOAA (*National Oceanic and Atmospheric Administration*). O algoritmo foi implementado em Python e, visando otimizar os hiperparâmetros da floresta, foram utilizadas bibliotecas do Scikit Learn para realizar a procura em grade (*grid search*) e, como resultados, obteve-se uma precisão de 93,8%, erro absoluto médio de 3,66ºF e erro médio quadrático de 21,70.**

***Index Terms*—Predição de temperatura, Random Forest, hiperparâmetros, procura em grade, algoritmos de regressão.**

I. Introdução

A temperatura máxima prevista para os próximos dias, em uma determinada localidade, é um dos principais fatores que integram os estudos climáticos realizados, diariamente, por Institutos de Meteorologia, haja vista seus impactos à saúde, consumo de energia elétrica, cultivo de espécies na agricultura, queimadas de áreas de preservação florestal e outros setores da economia. Esses estudos diários tornaram-se essenciais pois permitem que governos e empresas privadas planejem-se e apliquem medidas contingenciais quando necessário. Neste trabalho, escolheu-se a cidade de Seattle, nos Estados Unidos, devido às mudanças significativas de temperatura ao longo dos doze meses do ano, conforme pode ser observado na Figura 1. Os dados fornecidos para o treinamento do modelo correspondem a todos os dias do período compreendido entre 2011 a 2016. Dados mais recentes podem, também, ser obtidos diretamente do portal NOAA (*National Oceanic and Atmospheric Administration*), na guia *Climate Data Online* [4], selecionando-se a localidade de interesse.

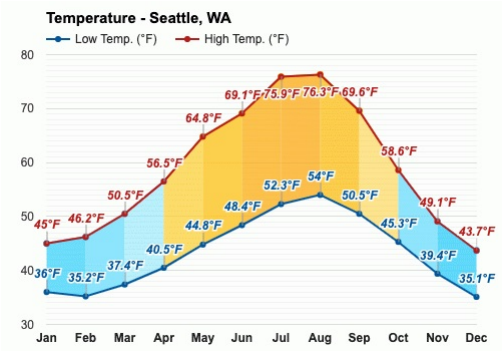

Figura 1. Temperaturas Máximas e Mínimas em Seattle

A partir desse conjunto de dados, pretende-se implementar um modelo de regressão utilizando o algoritmo de aprendizado de máquina supervisionado, conhecido como Floresta Aleatória ou, do inglês, o *Random Forest*. Tais algoritmos tem sido amplamente utilizados em problemas de classificação e regressão.

Feng e Wang[1], por exemplo, em seu trabalho, compararam os resultados dessa técnica com o desempenho de um modelo convencional de regressão linear múltipla aplicado ao problema de previsão de demanda de aluguel de bicicletas na China. Para isso, utilizaram dados históricos de temperatura, umidade relativa do ar, velocidade do vento, estação do ano, contagem de aluguéis, etc. A partir da análise dos dados, concluíram que o modelo de florestas aleatórias alcançou uma acurácia de 82%, bem acima dos resultados obtidos com a técnica de regressão múltipla.

De forma análoga, neste trabalho, pretende-se avaliar a precisão de um modelo da predição da temperatura máxima em Seattle, utilizando-se a técnica *Random Forest*, a partir dos dados históricos de diversos parâmetros, tais como a velocidade do vento, ano do registro e temperaturas máximas alcançadas até o dia anterior.

Na seção 2 será apresentada uma breve fundamentação teórica sobre o algoritmo de florestas aleatórias com procura em grade para obtenção dos melhores hiper-parâmetros do modelo. Na seção 3, a metodologia adotada para a implementação em Python com uso da biblioteca *Scikit Learn*, na seção 4 a comparação dos resultados encontrados e, por fim, na seção 5 as conclusões sobre a aplicação do método no problema proposto.

II. Fundamentação Teórica

As árvores de decisão são algoritmos de aprendizado de máquina supervisionado que podem ser utilizados em problemas de regressão e classificação. Neste trabalho, nosso foco é a implementação de uma modelo de regressão, ou seja, quando o resultado previsto pode ser considerado um número real, a exemplo da temperatura. As árvores de decisão são os blocos de construção do modelo de floresta aleatória.

A. Árvores de Regressão

Em geral, uma árvore é um conjunto de nós e arestas organizados de acordo com uma hierarquia sem loops, em que cada nó dividido armazena uma função de teste a ser aplicada aos dados de entrada. Os nós finais são chamados de folhas da árvore. Cada folha armazena o resultado final do teste, ou resposta conforme ilustrado na figura 2.

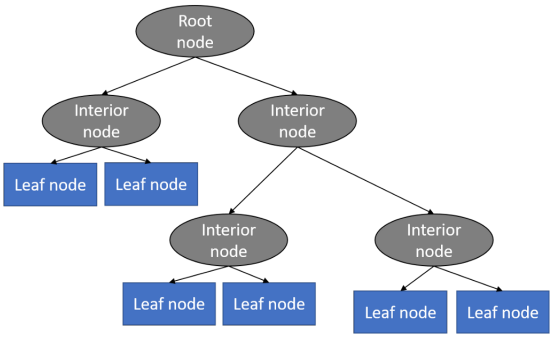

Figura 2. Estrutura de uma Árvore de Decisão

Seja X o vetor de entrada contendo p características e Y valores escalares das classes de saída. S_n é o conjunto de treinamento com n observações (X_i, Y_i), dado por:

$$S_n = \{(X_1, Y_1), ..., (X_n, Y_n)\}, X \in \mathbf{R}^p, Y \in \mathbf{R}. \quad (1)$$

O objetivo ao escolher uma variável X é minimizar a variância $Var(Y)$ entre os nós filhos, ou, equivalentemente, maximizar as médias ponderadas nos nós filhos.

$$Var(Y) = \frac{1}{|Y|} \sum_{y=Y} (y - \bar{y})^2, \quad (2)$$

em que $|Y|$ corresponde ao número de elementos de Y.

B. Floresta Aleatória (Random Forest)

A floresta aleatória (RF) é um método de montagem (*ensemble*) que combina a previsão de muitas árvores de decisão [3]. Baseia-se no princípio de *bagging*, em que uma amostra de tamanho n do conjunto de treinamento S_n é selecionada, aleatoriamente, e ajustada a uma árvore de regressão. Esta amostra é chamada de *bootstrap* e é escolhida com substituição, ou seja, a mesma observação (X_i, Y_i) pode aparecer mais de uma vez.

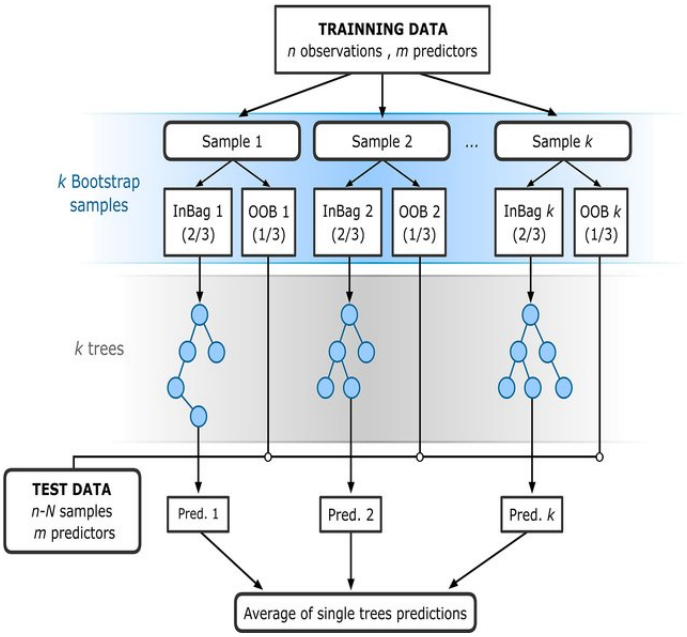

Figura 3. Fluxo em uma *Random Forest*

Na figura 3, destacamos uma das características principais que caracterizam a floresta aleatória: o erro *out-of-bag* ('OOB'), também chamado de erro de generalização, funciona como um tipo de validação cruzada interna, em que as amostras classificadas como tal farão parte do conjunto de testes, ou seja, uma estimativa ou predição de uma amostra (X_i, Y_i) será feita agregando apenas as árvores construídas sobre amostras de bootstrap que não contêm (X_i, Y_i). O OOB é muito útil estimar a capacidade de generalização do modelo construído.

C. Otimização de Hiperparâmetros com Procura em Grade

Uma boa prática para aumentar o resultado de qualquer algoritmo de classificação ou regressão é ajustar e otimizar os seus hiperparâmetros. Em uma floresta aleatória, exemplos de hiperparâmetro são: o número de árvores a serem inclusas na floresta, a quantidade nós cada árvore pode ter, a profundidade ou quantidade de níveis das árvores, dentre outros. São obtidos por tentativa e erro. Todas as combinações de valores de hiperparâmetros são testadas e, em seguida, os resultados são comparados para encontrar a melhor configuração. O método de procura em grade possui alto custo computacional. Uma alternativa para otimização desse processo, com menor custo, é a busca randômica de hiperparâmetros ou *Random Search*.

III. METODOLOGIA

Nesta seção serão descritos os procedimentos metodológicos necessários à solução do problema proposto. Serão apresentadas as ferramentas de software, o método utilizado para estimação dos melhores hiperparâmetros tendo em vista a regressão com *random forest*, a topologia de árvore adotada, bem como as métricas de avaliação utilizadas.

A. Ferramentas computacionais

A principal ferramenta utilizada neste trabalho foi o módulo *scikit-learn* da linguagem Python, capaz de prover recursos simples e eficientes para análise de dados.

1) Biblioteca NumPy: A biblioteca NumPy é fundamental para a computação científica em Python. Através desta biblioteca é possível manipular objetos do tipo array, com múltiplas dimensões, incluindo suas variações como é o caso das matrizes. Além disso, são fornecidas operações de álgebra linear, estatística e muitas outras.

2) Biblioteca Pandas: Pandas é uma biblioteca de código livre que fornece métodos práticos de se operar com estruturas de dados. Para o estudo de ciência de dados, é especialmente útil na manipulação de *datasets*. Dentre os recursos oferecidos estão: redimensionamento, agrupamento, substituições, entre outros. As extensões de arquivos permitidas incluem os formatos ".xlsx", ".xls", ".csv', por exemplo.

3) RandomForestRegressor: Esta é uma classe do módulo *scikit-learn*. Esta classe proporciona o uso de árvores aleatórias na tarefa de regressão baseando-se na média para mensurar a qualidade de cada divisão entre nós da árvore.

4) RandomizedSearchCV: Esta é uma classe do módulo *scikit-learn*. Esta classe proporciona a procura randômica dos hiperparâmetros do modelo selecionado, para isto implementa métodos de validação cruzada entre parâmetros. Em contraste com *GridSearchCV*, nem todos os parâmetros são combinados, sendo limitado a um número fixo de iterações previamente estabelecido.

5) GridSearchCV: Esta é uma classe do módulo *scikit-learn*. Esta classe proporciona uma busca extensa sobre todos os parâmetros estabelecidos para um estimador. Os hiperparâmetros são otimizados através de validação cruzada na matriz de parâmetros.

B. Descrição do problema proposto

A base de dados denominada "temps_extended.xlsx" é composta de 2191 observações, acumulando registros de 6 anos de dados climáticos da cidade de Seatle. Dentre os atributos de cada observação, serão analisados o ano em que o registro ocorreu, a velocidade média do vento no dia anterior, a temperatura máxima 1 dia antes, a temperatura máxima 2 dias antes e a média histórica de temperaturas máximas.

Com estas características, almeja-se modelar um algoritmo de *random_forest* capaz de estimar com o menor erro possível a temperatura máxima para o dia seguinte, estando o modelo munido de dados passados. Este modelo deverá ter seus hiperparâmetros obtidos através do método de procura em grid, estando 75% dos dados reservados para treinamento e 25% para teste. Como métrica de avaliação, serão avaliados o erro absoluto médio e o erro quadrático médio no conjunto de teste, objetivando-se obter maior precisão.

C. Implementação do modelo

A implementação deste trabalho foi realizada através do desenvolvimento de um script na linguagem Python, tendo o suporte dos pacotes Numpy, Pandas e *scikit-learn*. Para a edição e execução do código foi utilizado o ambiente de desenvolvimento Visual Studio Code.

Inicialmente foi realizada a leitura do dataset através do médodo *read_excel* do pacote Pandas, seguida da remoção de colunas não utilizadas nesta implementação, a exemplo da coluca "actual", que para este conjunto de dados representa a temperatura máxima a ser predita para o dia seguinte. Dividiu-se o conjunto de dados de forma que 75% das informações destinaram-se ao treinamento e ajuste de hiperparâmetros e 25% para teste.

Como ponto de partida do processo de busca de hiperparâmetros, foi utilizada a tabela 1 como referência e inicialização do algoritmo.

Tabela I
PRIMEIRA DEFINIÇÃO DE PARÂMETROS

Hiperparâmetros	**Possíveis valores**
Bootstrap	True, False
Max_Depth	*[10, 100] ou 'None'
Max_features	'auto', 'sqrt'
Min_samples_leaf	1, 2, 4
Min_samples_split	2, 5, 10
N_estimators	**[200, 2000]

*Incrementos de 10 **Incrementos de 200

Os parâmetros apresentados na tabela I possuem relação direta com a topologia a ser definida para a árvore aleatória. Seus respectivos significados são tais como se segue:

- *Bootstrap*: Este parâmetro indica o uso de uma técnica de reamostragem estatística que envolve amostragem aleatória de um conjunto de dados com substituição. Caso este parâmetro assuma o estado falso, todo o dataset será utilizado para a construção de cada árvore.
- *Max_Depth*: Este parâmetro configura a profundidade máxima de uma árvore, caso assuma o valor *None*, os nós são expandidos até que se atinja o número mínimo de amostras requeridas para a divisão.
- *Max_features*: O número de características a serem consideradas na busca pela melhor divisão.
- *Min_samples_split*: Especifica o número mínimo de amostras que um nó precisa ter para ser dividido.
- *Min_samples_leaf*: Um ponto de divisão em qualquer nó só será considerado se uma quantidade mínima de amostras estiver presente nos ramos direito e esquerdo da divisão.
- *N_estimators*: Quantidade de árvores(estimadores).

A procura em grade tende a ser caracterizada por uma busca extensa, o que demanda maior tempo de processamento. Visando obter maior celeridade na obtenção dos hiperparâmetros, adotou-se uma busca randômica limitada a 100 iterações, de forma que os melhores hiperparâmetros obtidos na procura randômica foram imediatamente inseridos na procura em grade. A redução do tempo de busca da procura em grade dá-se pela utilização de um conjunto reduzido de parâmetros, localizados em uma vizinhança dos melhores parâmetros obtidos no *Random Search*. Ao fim de cada iteração, o desempenho do modelo mediante cada conjunto de hiperparâmetros foi mensurado em termos da precisão e do erro quadrático médio. O diagrama em blocos que descreve esse sistema pode ser visualizado na figura 4.

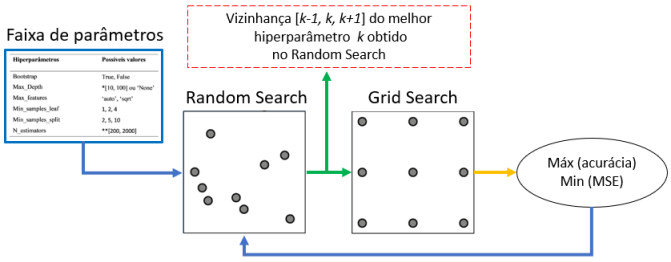

Figura 4. Metodologia de busca de hiperparâmetros

IV. Resultados

Como passo inicial do processo de obtenção dos hiperparâmetros ótimos, foi realizada a visualização da distribuição dos dados do conjunto de treinamento do algoritmo de árvores aleatórias. Para fins de visualização, foi fixada a variável alvo denominada *actual* no eixo y, representando a temperatura máxima, e foram variadas as demais características preditoras no eixo x, a citar: temperatura máxima no dia anterior (*temp_1*), temperatura máxima dois dias no passado (*temp_2*), velocidade do vento (*ws_1*) e temperatura média(*average*). Os gráficos da figura 5 descrevem a maneira como estas variáveis se relacionam.

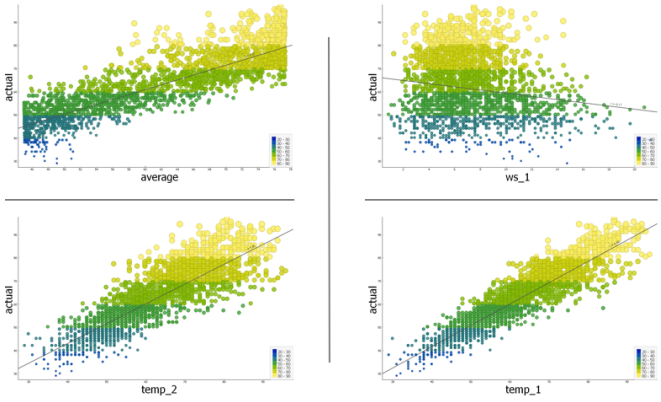

Figura 5. Gráficos de correlação entre as variáveis

Tabela II
Análise de correlação entre variáveis

Variáveis relacionadas	**r**
(*temp_1*, *actual*)	0.92
(*average*, *actual*)	0.87
(*temp_2*, *actual*)	0.85
(*ws_1*, *actual*)	-0.17

Como indicado pela tabela II, há correlação muito forte entre as variáveis *temp_1* e *actual*, correlação moderada a forte para as variáveis *temp_2* e *average*. Para a característica *ws_1* notou-se uma correlação ou influência praticamente desprezível sob a temperatura máxima indicada pela variável *actual*.

A tabela III indica os resultados obtidos em cada experimento de busca de hiperparâmetros. O procedimento de procura em grade realizado sobre todos os parâmetros fornecidos teve como característica um tempo excessivo de processamento. Ao final de 5 horas, alcançou-se uma precisão de 93,78%. Como método de otimização da procura em grade, foram realizadas buscas randômicas sobre o conjunto de parâmetros, sendo cada busca limitada a 100 iterações. O critério de parada assumido foi tal que uma precisão de pelo ao menos 93,78% fosse alcançada. Finalizada a busca randômica, foi iniciada uma busca em grade nas redondezas dos hiperparâmetros obtidos no passo anterior, fato este que contribuiu para um acréscimo na precisão, estabilizando-se em 93,80%.

Tabela III
Hiperparâmetros obtidos em cada experimento

Hiperparâmetros	**Grid_Search** (Total)	**Random_Search** (passo 1)	**Grid_Search** (passo 2)
Bootstrap	True	True	True
Max_depth	90	80	100
Max_features	sqrt	sqrt	sqrt
Min_samples_leaf	4	4	4
Min_samples_split	10	10	12
N_estimators	200	200	300
Precisão	93,78%	93,78%	93,80%
Erro quadrático médio	21,92	21,81	21,70
Erro absoluto médio	6,22ºF	6.22ºF	3,66ºF
Tempo de processamento	5h	131s	135s

Estando a topologia da árvore definida, foi possível realizar inferências sobre o conjunto de testes. No gráfico da figura 6 os pontos em azul representam os dados reais, enquanto que os pontos vermelhos indicam o resultado do modelo de árvore de regressão.

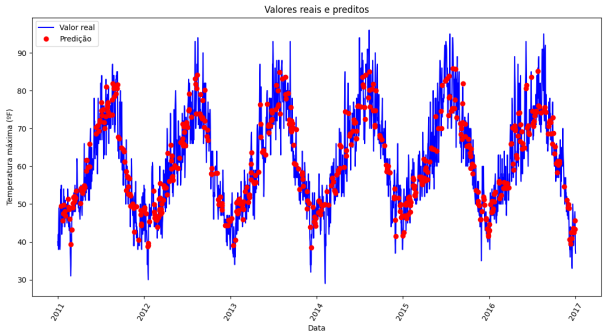

Figura 6. Previsão de temperatura máxima de acordo com o modelo

V. Conclusões

A implementação do algoritmo de árvores aleatórias mostrou-se eficaz na resolução do problema de predição de temperatura máxima. A diversidade de hiperparâmetros configuráveis proporcionou um entendimento sobre a importância de métodos de busca eficientes e que fornecessem combinações ótimas. Estas combinações de parâmetros puderam ter seus desempenhos mensurados através da avaliação da precisão e erro absoluto médio do modelo gerado. Observou-se que o método de procura em grade tende a exigir maior tempo de processamento à medida que o conjunto de hiperparâmetros aumenta, uma vez que são realizadas combinações randômicas destes. Como maneira de produzir resultados a curto prazo, adotou-se a estratégia de busca randômica com um número de iterações limitadas, em seguida, realizou-se procura em grade nas vizinhanças dos melhores hiperparâmetros obtidos no passo anterior. Este procedimento resultou em uma busca mais assertiva, com precisão alcançada de 93,80%. A análise dos dados também se mostrou uma etapa fundamental, uma vez que características que pouco influem nas decisões da árvore, podem eventualmente não serem levadas em consideração em possíveis otimizações do modelo. A figura 5 e a tabela II ilustram a correlação entre as características, indicando forte correlação entre a temperatura máxima de um dia com respeito ao dia anterior, um fato coerente em termos práticos. Por fim, através da figura 6 é possível observar que as inferências do modelo, apontadas pelos pontos em vermelho, seguem a tendência história de temperatura ao longo dos anos, o que sinaliza conformidade e coerência com os dados reais. Com uma precisão de 93,80% e erro absoluto médio de 3,66ºF, vê-se necessidade de inserção de novas características à base de dados caso haja necessidade de aumento de precisão, visto que o modelo atual acumula dados de 6 anos de temperatura.

Referências

[1] FENG, Y., WANG, S. ”A Forcast for Bicycle Rental Demand Based on Random Forests and Multiple Linear Regression”, International Conference on Information Systems (ICIS), May 24-26, 2017, Wuhan, China.

[2] XIE, Y., Li, X., NGAI, E., YING, W. ”Customer churn prediction using improved balanced random forests”, Expert Systems with Applications, Elsevier, 2009.

[3] LOUPPE, G. ”Understanding Random Forests - From Theory to Practice”, PhD Thesis, University of Liège, Faculty of Applied Sciences, Department of Electrical Engineering Computer Science, 2014.

[4] ”NOAA - National Oceanic and Atmospheric Administration”, Climate Data Online https://www.ncdc.noaa.gov/cdo-web/

```
################### PPGEE 2021/01 ####################
############## Aprendizado de m quina ################
# Trabalho 5: Regress o com Random Forest
# Alunos: Carlos Henrique
#         Willian Guerreiro Colares

# Import de bibliotecas para manipula  o e visualiza  o de dados
import time
import math
import pandas as pd
import numpy as np
import matplotlib as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.model_selection import RandomizedSearchCV

#Fun  o de avalia  o do modelo j  treinado
def evaluate(model, test_features, test_labels):
    #Erro absoluto m dio
    predictions = model.predict(test_features)
    errors = abs(predictions - test_labels)
    mape = 100 * np.mean(errors / test_labels)
    #Acur cia
    accuracy = 100 - mape
    #Erro m dio quadr tico
    N = len(predictions)
    Sum = 0
    for i,element in enumerate(predictions):
        Sum = Sum + math.pow((element - test_labels[i]),2)
    mse = Sum / N

    print('Model Performance')
    print('Average Error: {:0.4f} degrees'.format(np.mean(errors)))
    print('MAPE: {:0.2f}%'.format(mape))
    print('Accuracy = {:0.2f}%'.format(accuracy))
    print('MSE :{:0.2f}%'.format(mse))

# Leitura dos dados
features = pd.read_excel('temps_extended.xlsx')

# Exibi  o das 5 primeiras observa  es
print(features.head(5))

# Vari veis a serem utilizadas:  year, ws_1, temp2,temp1,average

# Labels s o os valores que ser o preditos (sa da)
labels = np.array(features['actual'])

# Labels retirados de features, bem como os dados que n o ser o utilizados
features = features.drop('actual', axis=1)
features = features.drop('month', axis=1)
features = features.drop('day', axis=1)
features = features.drop('weekday', axis=1)
features = features.drop('friend', axis=1)
features = features.drop('prcp_1', axis=1)
features = features.drop('snwd_1', axis=1)

# Armazenamento dos nomes das caracter sticas para uso futuro
feature_list = list(features.columns)

features = np.array(features)

# Cria  o do conjunto de treino e teste
```

```
train_features, test_features, train_labels, test_labels =
train_test_split(features, labels, test_size=0.25,   random_state=42)


################## TREINAMENTO ##############################

# Number of trees in random forest
n_estimators = np.arange(200,2200,200)  #start = range de 200 a 2200(aberto), com saltos de 200
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = list(np.arange(10,110,10))
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 12]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}


# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 50 different combinations, and use all available cores

rf_random = GridSearchCV(estimator = rf, param_grid = random_grid,
                            scoring = 'neg_mean_absolute_error', cv = None,
                            n_jobs = -1, verbose = 2)
'''
rf_random = RandomizedSearchCV(estimator=rf, param_distributions=random_grid,
                               n_iter = 50, scoring='neg_mean_absolute_error',
                                 cv = None, verbose=2, random_state=42, n_jobs=-1)
'''
# Fit the random search model
t_1 = time.time()
rf_random.fit(train_features, train_labels)
t_2 = time.time()

best_random = rf_random.best_estimator_
print('Best parameters Ramdom search: ',best_random.get_params())
print('Tunning time: ',t_2 - t_1,' seconds')
evaluate(best_random, test_features, test_labels)
print("\n")
parameters = best_random.get_params()
opt_max_depth = parameters['max_depth']
opt_min_samples_leaf = parameters['min_samples_leaf']
opt_min_samples_split = parameters['min_samples_split']
opt_n_estimator = parameters['n_estimators']

#Grid search 1
while True:
    Input = input("Enter para o grid 1")
    if opt_max_depth == 10:
        opt_max_depth = 20
    elif opt_max_depth == 100:
        opt_max_depth = 90
```

```
if opt_n_estimator == 200:
    opt_n_estimator = 400
elif opt_n_estimator == 2000:
    opt_n_estimator = 1800

if opt_min_samples_leaf == 1 or opt_min_samples_leaf == 4:
    opt_min_samples_leaf = 2

if opt_min_samples_split == 2 or opt_min_samples_split == 10:
    opt_min_samples_split = 5

param_grid = {
    'bootstrap': [True],
    'max_depth': [opt_max_depth-10, opt_max_depth, opt_max_depth+10],
    'max_features': ['sqrt'],
    'min_samples_leaf': [opt_min_samples_leaf-1,opt_min_samples_leaf,opt_min_samples_leaf+2],
    'min_samples_split': [opt_min_samples_split-3,opt_min_samples_split,opt_min_samples_split+5],
    'n_estimators': [opt_n_estimator-200, opt_n_estimator, opt_n_estimator+200]
}


# Create a based model
#rf = RandomForestRegressor()

# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                           scoring = 'neg_mean_absolute_error', cv = None,
                           n_jobs = -1, verbose = 2)

# Fit the grid search to the data
t_1 = time.time()
grid_search.fit(train_features, train_labels)
t_2 = time.time()
best_grid = grid_search.best_estimator_

print('Best parameters Grid search 1: ')
print(best_grid.get_params())
print('Tunning time: ',t_2 - t_1,' seconds')
#evaluate(best_grid, test_features, test_labels)
evaluate(best_grid, test_features, test_labels)

parameters = best_grid.get_params()
opt_max_depth = parameters['max_depth']
opt_min_samples_leaf = parameters['min_samples_leaf']
opt_min_samples_split = parameters['min_samples_split']
opt_n_estimator = parameters['n_estimators']

print("\n")
```