Trabalho Final Data Mining

Prof.: Manoela Kohler

Aluno: Otávio Ciribelli Borges

Email: otavio.ciribelli@gmail.com

Introdução

O trabalho final da disciplina de Data Mining propõe a implementação de um algoritmo para resolver um problema de classificação do tipo supervisionada em um conjunto de dados (*dataset*) sugerido. Um *dataset* base, disponível no arquivo horseTest.csv, poderá ser usado para entender situações acerca da sobrevivência de cavalos a partir de informações históricas vividas por cada animal. Porém, a proposição do trabalho deixa arbitrada aos alunos a escolha de diferentes bases para aplicar técnicas de exploração e tratamento de dados, seleção de atributos, transformações e classificações utilizando técnicas de aprendizagem de máquina.

Neste trabalho, o *dataset* adotado foi gerado pelo próprio autor a partir de condições climáticas registradas em dois ambientes domésticos. As medidas de temperatura e umidade foram tomadas em períodos compreendidos entre 2018 e 2021 e estão disponíveis em repositório do Github https://github.com/ciribelli/datamining

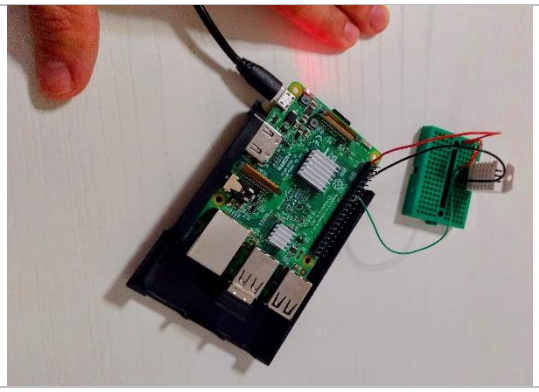
O objetivo de classificação deste trabalho consiste em prever, a partir dos registros base de temperatura e umidade, quando o aparelho de ar-condicionado dos relativos ambientes esteve ligado. Existem, portanto, duas classes de saída para o estado de ar-condicionado a serem previstas, sendo '1' para aparelho ligado e '0' quando o aparelho está desligado.

Em termos de propósito e abrangência mercadológica, a rotina de classificação implementada neste trabalho pode contribuir com a simplificação de aplicações do tipo IoT ou IIoT, podendo atender com a geração de informações adicionais àquelas fornecidas por sensores de variáveis primárias ou físicas.

Todo o material utilizado para a realização deste trabalho, bem como este relatório, está disponível no endereço https://github.com/ciribelli/datamining.

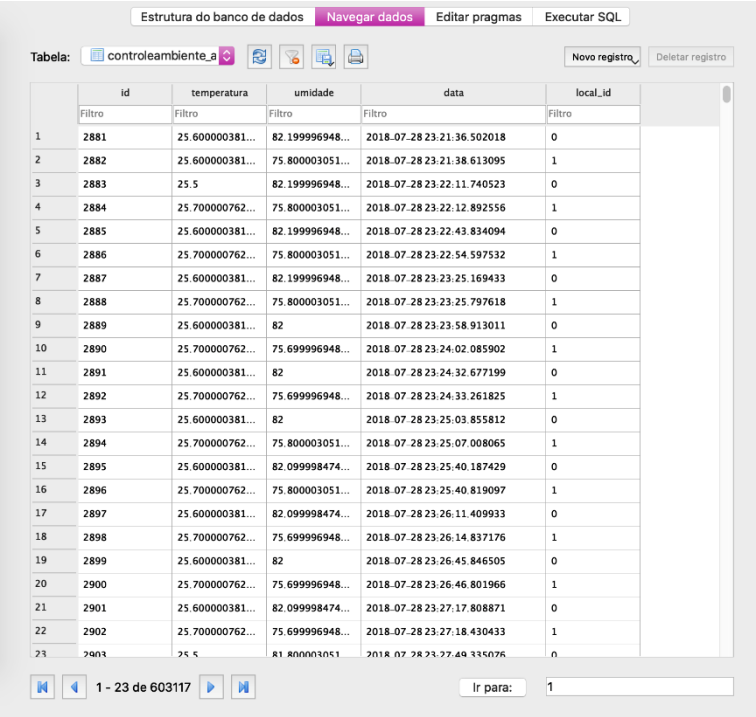
Medições e composição do *dataset*

As tomadas de medição de temperatura e umidade foram realizadas utilizando um computador do tipo Raspberry Pi 3 Model B que contou com a aplicação do sensor DHT22 (datasheet disponível em https://pdf1.alldatasheet.com/datasheet-pdf/view/1132459/ETC2/DHT22.html.

	
Raspberry Pi 3 Model B	*Raspberry Pi 3 integrado a um sensor do tipo DHT22*

Sobre o processo de captura:

- a rotina de captura está disponível no repositório https://github.com/ciribelli/autohome/blob/master/motorHome.py
- o período de amostragem dos dados é de sessenta (60) segundos, tanto para temperatura quanto para umidade
- as informações são registradas na base de dados sqlite3 no arquivo db.sqlite3 disponível em https://github.com/ciribelli/autohome/tree/master/home/db.sqlite3
- O arquivo db.sqlite3 tem um total de 603.117 registros
- a coluna 'local_id' indica a origem do local das medições (conforme mencionado anteriormente, trata-se de dois ambientes diferentes)
- a coluna data indica o *timestamp* do instante em que os registros são capturados do sensor
- temperaturas são registradas em graus Celsius
- umidades são registradas em valores percentuais



	id	temperatura	umidade	data	local_id
	Filtro	Filtro	Filtro	Filtro	Filtro
1	2881	25.600000381...	82.199996948...	2018-07-28 23:21:36.502018	0
2	2882	25.600000381...	75.800003051...	2018-07-28 23:21:38.613095	1
3	2883	25.5	82.199996948...	2018-07-28 23:22:11.740523	0
4	2884	25.700000762...	75.800003051...	2018-07-28 23:22:12.892556	1
5	2885	25.600000381...	82.199996948...	2018-07-28 23:22:43.834094	0
6	2886	25.700000762...	75.800003051...	2018-07-28 23:22:54.597532	1
7	2887	25.600000381...	82.199996948...	2018-07-28 23:23:25.169433	0
8	2888	25.700000762...	75.800003051...	2018-07-28 23:23:25.797618	1
9	2889	25.600000381...	82	2018-07-28 23:23:58.913011	0
10	2890	25.700000762...	75.699996948...	2018-07-28 23:24:02.085902	1
11	2891	25.600000381...	82	2018-07-28 23:24:32.677199	0
12	2892	25.700000762...	75.699996948...	2018-07-28 23:24:33.261825	1
13	2893	25.600000381...	82	2018-07-28 23:25:03.855812	0
14	2894	25.700000762...	75.800003051...	2018-07-28 23:25:07.008065	1
15	2895	25.600000381...	82.099998474...	2018-07-28 23:25:40.187429	0
16	2896	25.700000762...	75.800003051...	2018-07-28 23:25:40.819097	1
17	2897	25.600000381...	82.099998474...	2018-07-28 23:26:11.409933	0
18	2898	25.700000762...	75.699996948...	2018-07-28 23:26:14.837176	1
19	2899	25.600000381...	82	2018-07-28 23:26:45.846505	0
20	2900	25.700000762...	75.699996948...	2018-07-28 23:26:46.801966	1
21	2901	25.600000381...	82.099998474...	2018-07-28 23:27:17.808871	0
22	2902	25.700000762...	75.699996948...	2018-07-28 23:27:18.430433	1
23	2903	25.5	81.800003051...	2018-07-28 23:27:49.335076	0

Banco de dados utilizado para geração do dataset

Para a composição do *dataset*, foram geradas medidas indiretas a partir dos dados brutos disponíveis. Essas medidas consistem do resultado de operações de janelas deslizantes de diferentes tamanhos aplicadas sobre os dados disponíveis.

As operações foram realizadas utilizando a função *rolling* da biblioteca Pandas (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.rolling.html) com janelas que têm a seguinte composição:

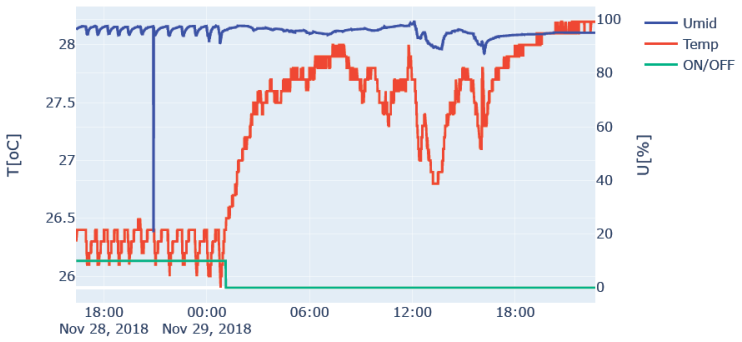
```
janela_0 = 15 # janela deslizante de 15 elementos
janela_1 = 30 # janela deslizante de 30 elementos
janela_2 = 60 # janela deslizante de 60 elementos
```

As variáveis ou atributos resultantes das operações são apresentados na Tabela 01.

TABELA 01 - VARIÁVEIS MEDIDAS E GERADAS COMO ATRIBUTOS DO CONJUNTO DE DADOS

	COLUNAS ORIGINAIS	COLUNAS COM ATRIBUTOS GERADOS
1	data	temp_amplitude_15
2	local_id	temp_amplitude_30
3	temperatura	temp_amplitude_60
4	umidade	temp_desvpad_15
5		temp_desvpad_30
6		temp_desvpad_60
7		temp_media_15
8		temp_media_30
9		temp_media_60
10		temp_variancia_15
11		temp_variancia_30
12		temp_variancia_60
13		umid_amplitude_15
14		umid_amplitude_30
15		umid_amplitude_60
16		umid_desvpad_15
17		umid_desvpad_30
18		umid_desvpad_60
19		umid_media_15
20		umid_media_30
21		umid_media_60
22		umid_variancia_15
23		umid_variancia_30
24		umid_variancia_60

Para a preparação do *dataset*, também foi necessário um trabalho de identificação dos períodos em que o ar-condicionado esteve ligado ou desligado para cada ambiente. Foi utilizada uma rotina de geração de gráficos das séries temporais históricas de temperatura e umidade e realizada a identificação visual de seis diferentes períodos de interesse. A classificação de ar-condicionado ligado ou desligado foi feita com base na experiência empírica característica revelada nas curvas das medições realizadas.



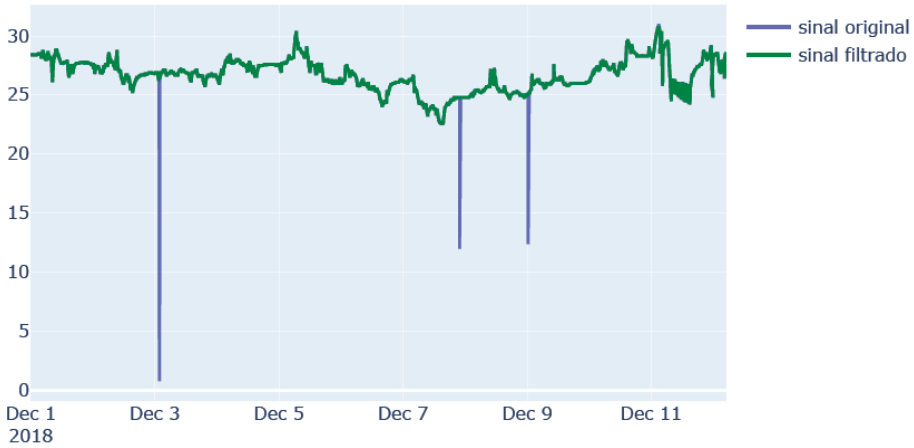
Os intervalos de interesse definidos são apresentados a seguir num formato de dicionário da linguagem Python:

```
intervalo_01 = {'t0': '2018-12-05 15:10', 'ti': '2018-12-05 16:28', 'tf': '2018-12-06 14:00', 'toff': '2018-12-06 01:36', 'local_id': 0}
intervalo_02 = {'t0': '2018-10-03 20:30', 'ti': '2018-10-03 21:43', 'tf': '2018-10-04 22:40', 'toff': '2018-10-04 02:43', 'local_id': 0}
intervalo_03 = {'t0': '2018-11-28 15:10', 'ti': '2018-11-28 16:23', 'tf': '2018-11-29 22:40', 'toff': '2018-11-29 01:07', 'local_id': 0}
intervalo_04 = {'t0': '2020-04-07 22:15', 'ti': '2020-04-07 23:19', 'tf': '2020-04-08 17:48', 'toff': '2020-04-08 10:30', 'local_id': 1}
intervalo_05 = {'t0': '2020-04-12 22:25', 'ti': '2020-04-12 23:30', 'tf': '2020-04-13 22:00', 'toff': '2020-04-13 10:46', 'local_id': 1}
intervalo_06 = {'t0': '2020-04-11 00:05', 'ti': '2020-04-11 01:12', 'tf': '2020-04-11 16:52', 'toff': '2020-04-11 11:32', 'local_id': 1}
```

Filtragem e segmentação

Para a conclusão do processo de criação do *dataset*, uma rotina implementada no arquivo sql_to_csv.py (https://github.com/ciribelli/datamining/) realiza as seguintes operações:

1. Carrega o arquivo do tipo SQL e constrói o *dataframe* com a biblioteca Pandas;
2. Aplica filtros para eliminação de dados espúrios, tanto para temperatura quanto para umidade
3. Adiciona uma coluna com a rotulagem de classificação utilizando ' 1' para ar-condicionado ligado e '0' para ar-condicionado desligado
4. Aplica operação de janela deslizante por meio da função *rolling* gerando colunas adicionais ao *dataframe*
5. Segmentação os dados resultantes eliminando bordas de dados não rotulados da série temporal
6. Concatena os intervalos de interesse e salva no arquivo datasetDM.csv



No trecho de código destacado abaixo, são apresentadas as operações de janela deslizante aplicadas aos dados:

```
def aplica_janela(daux):

    janela_0 = 15 # janela deslizante de 15 elementos
    janela_1 = 30 # janela deslizante de 30 elementos
    janela_2 = 60 # janela deslizante de 60 elementos

    l_janelas = [janela_0, janela_1, janela_2]

    for l_j in l_janelas:

        # para temperaturas
        s_arcon_t = daux['temperatura'].rolling(l_j).std()              # desvio padrao
        v_arcon_t = daux['temperatura'].rolling(l_j).var()              # variância
        m_arcon_t = daux['temperatura'].rolling(l_j).mean()             # média
        min_arcon_t = daux['temperatura'].rolling(l_j).min()            # calc. auxiliar de mínimo
        max_arcon_t = daux['temperatura'].rolling(l_j).max()            # calc. auxiliar de máximo
        a_arcon_t = max_arcon_t - min_arcon_t           # amplitude (max - min)

        # para umidade
        s_arcon_u = daux['umidade'].rolling(l_j).std()              # desvio padrao
        v_arcon_u = daux['umidade'].rolling(l_j).var()              # variância
        m_arcon_u = daux['umidade'].rolling(l_j).mean()             # média
        min_arcon_u = daux['umidade'].rolling(l_j).min()            # calc. auxiliar de mínimo
        max_arcon_u = daux['umidade'].rolling(l_j).max()            # calc. auxiliar de máximo
        a_arcon_u = max_arcon_u - min_arcon_u           # amplitude (max - min)

        daux['temp_desvpad_' + str(l_j)] = s_arcon_t
        daux['temp_variancia_' + str(l_j)] = v_arcon_t
        daux['temp_media_' + str(l_j)] = m_arcon_t
        daux['temp_amplitude_' + str(l_j)] = a_arcon_t
        daux['umid_desvpad_' + str(l_j)] = s_arcon_u
        daux['umid_variancia_' + str(l_j)] = v_arcon_u
        daux['umid_media_' + str(l_j)] = m_arcon_u
        daux['umid_amplitude_' + str(l_j)] = a_arcon_u
    return (daux)
```

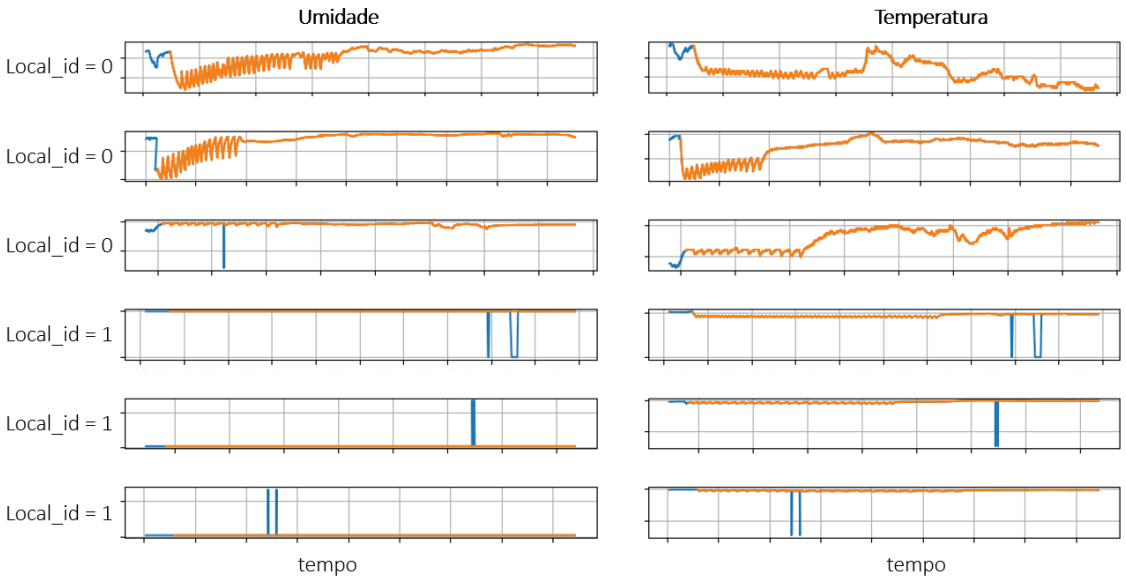
Os intervalos de segmentação foram definidos com base nos tempos anotados conforme:

- T0: tempo de início da janela (permite que janelas de até 60 elementos não retornem elementos nulos)
- Ti: início do período de ar-condicionado ligado
- Toff: fim do período de ar-condicionado ligado
- Tf: fim do período da janela

Para as operações de segmentação, foram aplicadas funções ‘loc’ utilizando os índices relativamente aos tempos de interesse de cada intervalo. Um exemplo é visto abaixo:

```
def segmentaInicio(u):
u = u.loc[i['ti']:i['tf']]
```

O resultado das operações de segmentação é apresentado abaixo na figura.



Algumas constatações podem ser registradas:

- pode-se avaliar nas séries temporais em azul diversas situações de ruído ou presença de *outliers* que foram eliminados com a operação de filtragem;
- as séries temporais com informações de umidade do local_id = 1 não podem ser utilizadas por falha no sensor
- as séries temporais com informações de temperatura do local_id = 1 são bem comportadas e parecem adequadas ao uso após filtragem
- a primeira série temporal de temperatura do local_id = 0 deve ser avaliada com maior cuidado
- séries temporais de umidade do local_id = 0 parecem adequadas ao uso após filtragem

As curvas em laranja que são as informações filtradas, foram então selecionadas para compor o *dataset* final que tem 13.847 elementos.

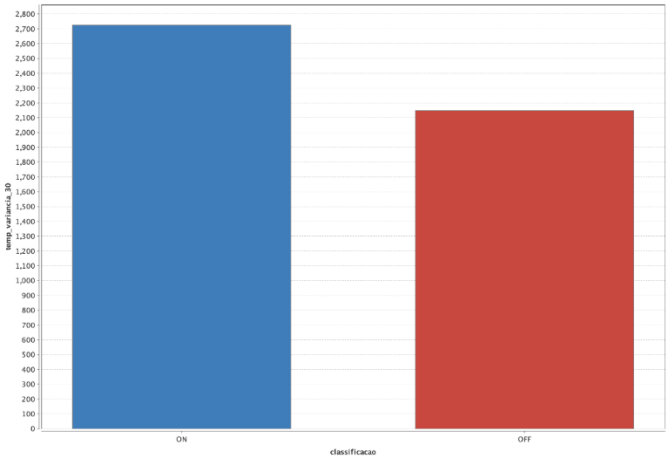
Uma vez consolidado o *dataset* após aplicação dos processos filtragem, classificação e segmentação, o arquivo com extensão do tipo csv com 13.847 linhas foi salvo para ser utilizado na realização da etapa de exploratória. Os dados foram avaliados em duas perspectivas, sendo a primeira por meio da biblioteca seaborn e a segunda utilizando o software RapidMiner.

A informação completa do arquivo sql_to_csv.py também está disponível no ANEXO I.

Análise de balanceamento e seleção de atributos

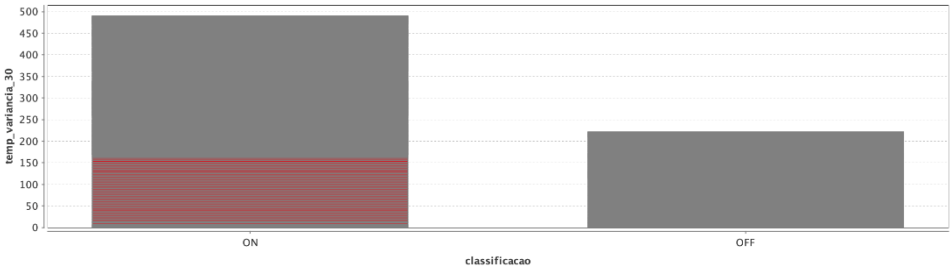
Com o suporte do RapidMiner, foi verificado um desbalanceamento das classes com favorecimento para amostras com ar-condicionado ligado. Por se tratar de um desbalanceamento pouco severo, e diante de uma boa oferta de amostras no banco de dados originário, foi decidido por não tomar providências relativamente à busca por maior equilíbrio

entre as bases e atuar reativamente caso os resultados de classificação apresentem desempenho ruim nas fases posteriores.

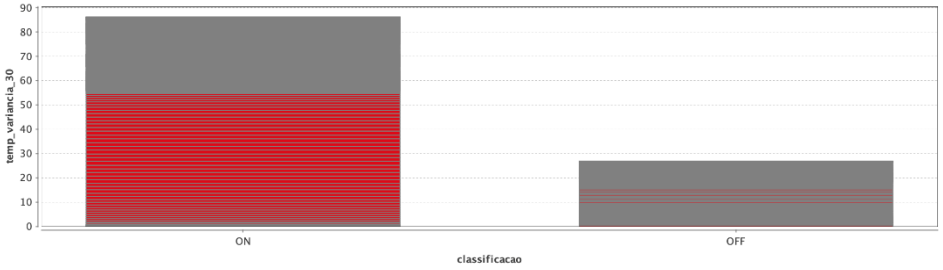


Análise de balanceamento das amostras (ON x OFF)

Ainda no contexto do RapidMiner, foram testadas diversas visualizações das quais destacam-se duas representadas abaixo onde são evidenciadas boas perspectivas de diferenciação das classes por meio da adoção de variáveis como desvio padrão, amplitude e variância dos dados. Percebe-se diferentes oportunidades de desempenho quando combinadas janelas de 30 e 60 amostras, principalmente.



Local_id == 1 :: Temp_desvpad30 vs temp_amplitude30

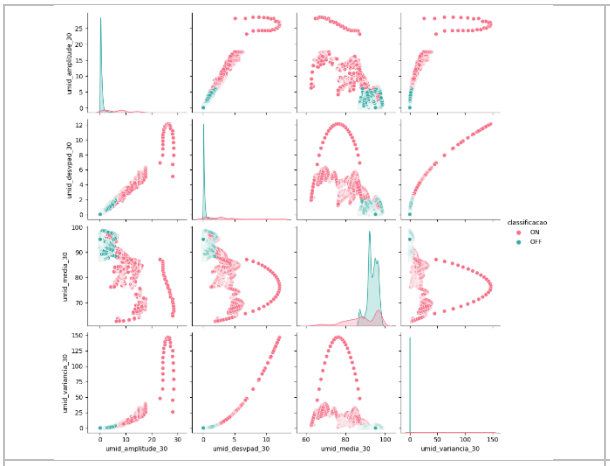


Local_id == 1 :: Temp_desvpad30 vs temp_variancia60

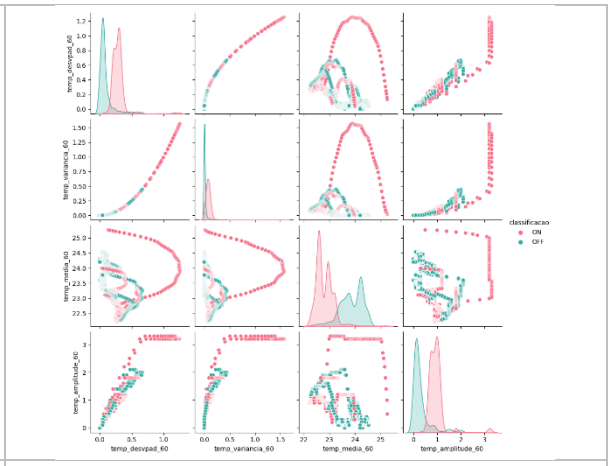
Para fins de seleção de *features*, a biblicoteca seaborn foi aplicada com a opção de visualização conhecida por *pairplot*. Nesta fase do processo, foram tomadas ações de cunho experimental testando diferentes combinações de conjuntos de variáveis, sejam elas de natureza de temperatura ou umidade. Também foram avaliadas diferentes situações em que os ambientes físicos 0 e 1 foram testados separadamente e em conjunto.

A análise visual dos gráficos do tipo *pairplot* resultou na seleção das *features* que são apresentadas na tabela abaixo. Os gráficos resultantes da aplicação do método *pairplot* para janelas de 15, 30 e 60 elementos estão apresentados na sequência a seguir. Deve ser registrado que a combinação de feartures validada para a próxima etapa do processo envolve a combinação de janelas de 15, 30 e 60 elementos. Importante também registrar que outras combinações foram avaliadas, mas todas com desempenho inferior às apresentadas na tabela.

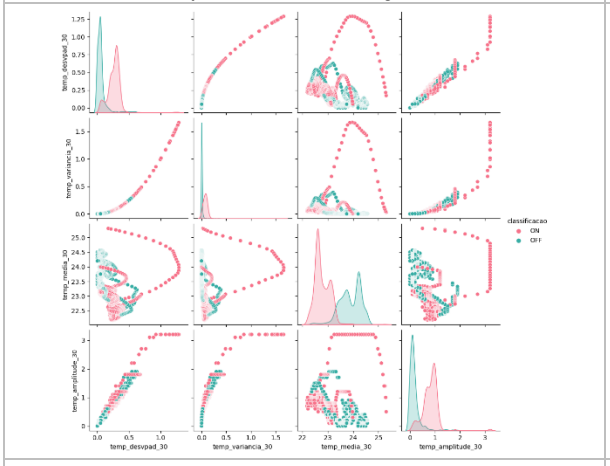
#	*Local físico*	*Combinação de features avaliadas*	*status*
1	local_id = 0	['classificacao', 'umid_amplitude_30','umid_desvpad_30', 'umid_media_30', 'umid_variancia_30']	✗
2	local_id = 1	['temp_desvpad_60','temp_variancia_60', 'temp_media_60', 'temp_amplitude_60', 'classificacao']	✗
3	local_id = 1	['temp_desvpad_30', 'temp_variancia_30', 'temp_media_30', 'temp_amplitude_30', 'classificacao']	✗
4	local_id = 1	['temp_media_15', 'temp_desvpad_60', 'temp_variancia_30', 'temp_media_30', 'temp_amplitude_60', 'classificacao']	✓



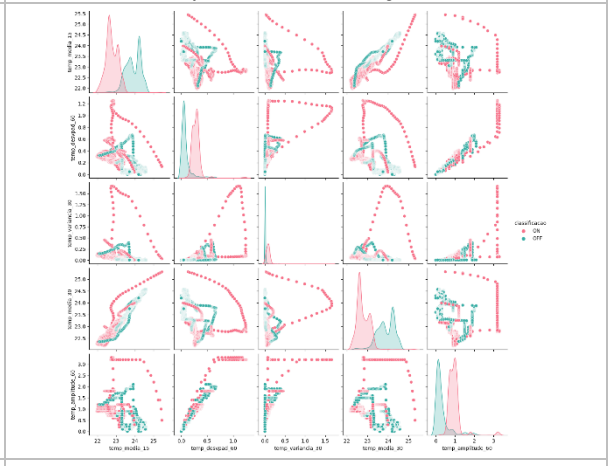
Pairplot da combinação 01



Pairplot da combinação 02



Pairplot da combinação 03



Pairplot da combinação 04 (opção aceita)

Os estudos de seleção de *features* foram registrados no arquivo feature_sel.py, disponível no ANEXO II.

Normalização

O estudo do impacto da normalização dos dados foi realizado de forma expedita e constado que, dentre os métodos de *machine learning* avaliados, apenas o método SVM se mostrou sensível à normalização. Vale destacar, no entanto, que, para um uso efetivo do método com abrangência para dados de diferentes localidades e estações do ano, seria prudente utilizar o artifício de normalização no processo de criação do algoritmo.

Abaixo, está apresentado o conjunto de instruções utilizados para o exercício de normalização:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Teste de Modelos de *Machine Learning*

Para cumprir com o objetivo do trabalho, foram selecionados quatro algoritmos de classificação supervisionada disponíveis na biblioteca *scikit-learn*. Eles são:

- *Random Forest*
- *Gradient Boosting*
- *Multi-layer Perceptron*
- *Support Vector Machine*

O arquivo machine_learning.py resume as operações que permitem concluir o objetivo de classificação supervisionada. De forma sucinta, estão contidas no arquivo operações de:

1. Carregamento dos dados do arquivo csv e criação do *dataset*
2. Transformação dos dados substituindo valores do tipo texto por inteiro na coluna de classificação
3. Separação dos dados de entrada e saída conforme *features* selecionadas
4. Separação dos dados para as operações de treino e teste
5. Treinamento do modelo para os 4 algoritmos selecionados neste trabalho
6. Predição das respostas para o conjunto de dados reservado para teste
7. Avalição do desempenho numérica e graficamente

O conjunto de instruções para esta etapa está apresentado no ANEXO III.

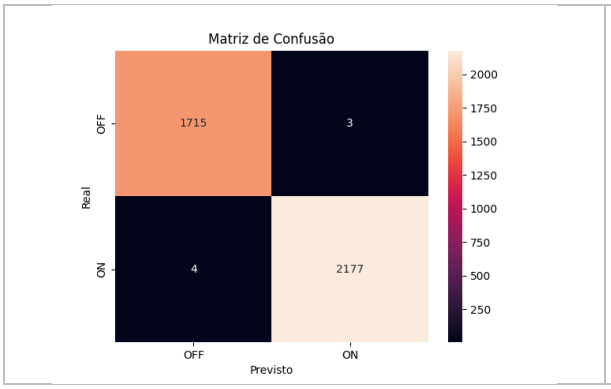
Validações e resultados

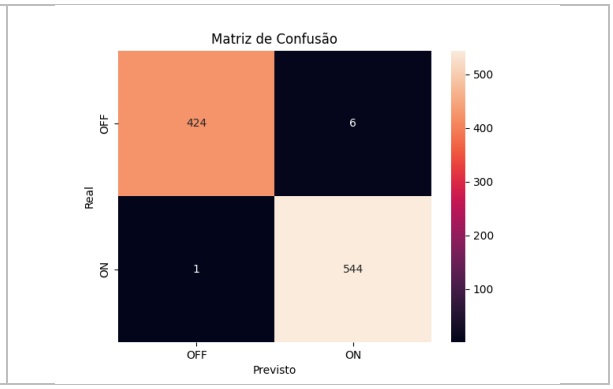
Na tabela abaixo, são apresentados os resultados de desempenho dos classificadores implementados. Foram relacionados na tabela três indicadores de desempenho largamente difundidos na área de inteligência artificial: precisão, revocação e f1-score.

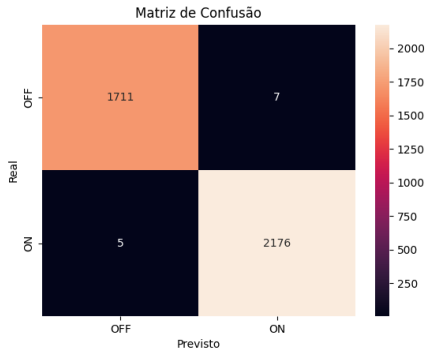
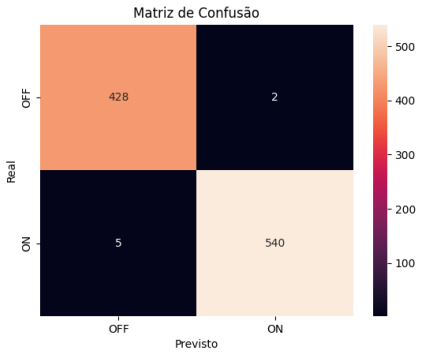
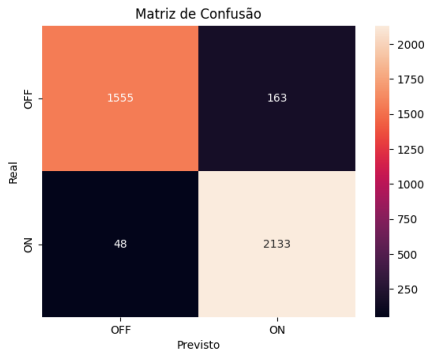
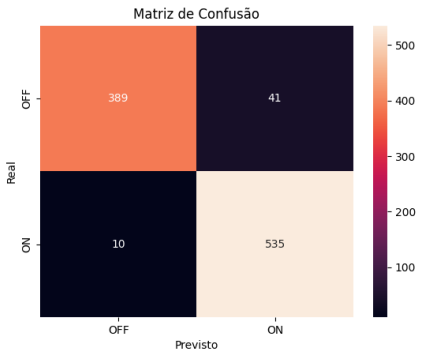
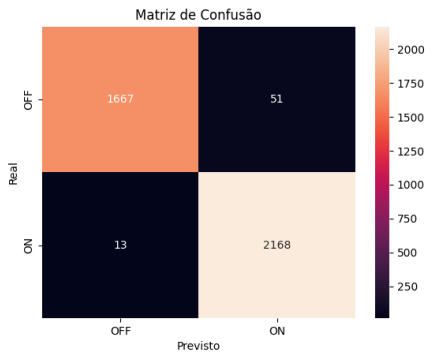
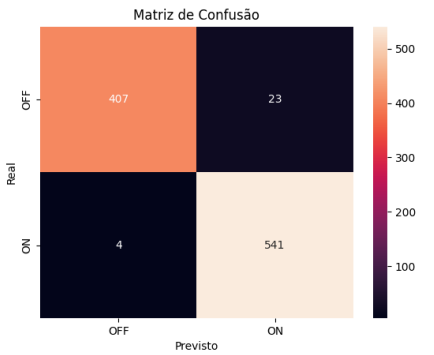
Apenas para o método de classificação SVM foram registrados resultados tanto para a versão sem normalização quanto para a versão com dados normalizados. Como mencionado anteriormente, para os demais classificadores os resultados não variaram significativamente.

		precision	recall	f1-score	support
RandomForestClassifier	0	100%	100%	100%	430
	1	100%	100%	100%	545
	accuracy			100%	975
	macro avg	100%	100%	100%	975
	weighted avg	100%	100%	100%	975
GradientBoostingClassifier	0	99%	99%	99%	430
	1	99%	99%	99%	545
	accuracy			99%	975
	macro avg	99%	99%	99%	975
	weighted avg	99%	99%	99%	975
MLPClassifier	0	96%	91%	94%	430
	1	93%	97%	95%	545
	accuracy			94%	975
	macro avg	95%	94%	94%	975
	weighted avg	95%	94%	94%	975
SVM	0	99%	92%	96%	430
	1	94%	100%	97%	545
	accuracy			96%	975
	macro avg	97%	96%	96%	975
	weighted avg	96%	96%	96%	975
SVM (normalizado)	0	99%	97%	98%	430
	1	97%	99%	98%	545
	accuracy			98%	975
	macro avg	98%	98%	98%	975
	weighted avg	98%	98%	98%	975

Na sequência de imagens reproduzidas abaixo, são apresentadas as matrizes de confusão enquanto visualização de desempenho para os classificadores definidos nesta fase do trabalho.

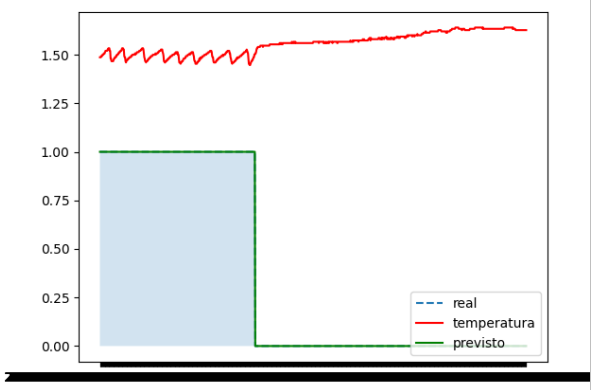




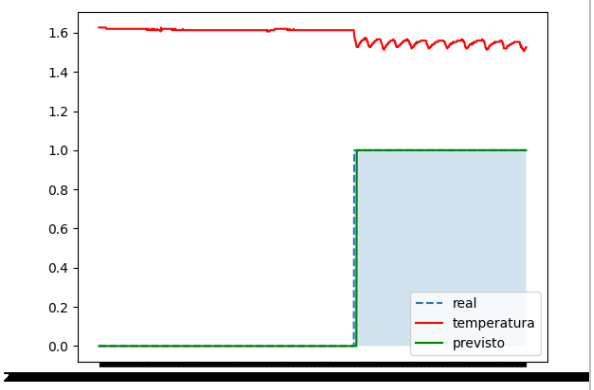
RF Treino	*RF Teste*
	
GB Treino	*GB Teste*
	
MLP Treino	*MLP Teste*
	
SVM Normalizado Treino	*SVM Normalizado Teste*

Complementarmente aos resultados de desempenho clássicos, foram geradas visualizações das séries temporais mediante à predição realizada pelos modelos definidos. O intuito dessa avaliação é transmitir um pouco da dinâmica física que o classificador teria em situais reais.

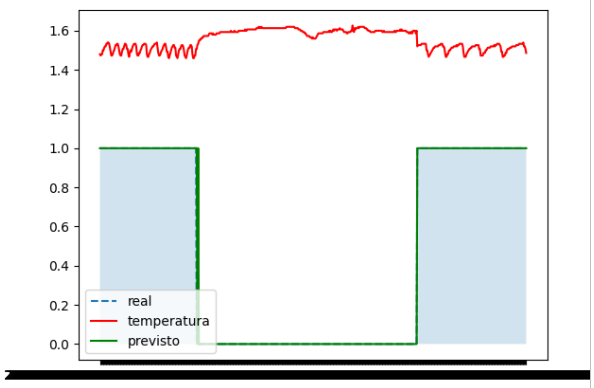
Na sequência de figuras a seguir, são demonstradas situações em que o classificador apresentou desempenho adequado quanto a falsos positivos, verdadeiros positivos, falsos negativos e verdadeiros negativos. Cabe ressaltar que as transições foram bem identificadas pelo classificador, com atrasos típicos da ordem de poucos minutos (geralmente menor que três minutos).



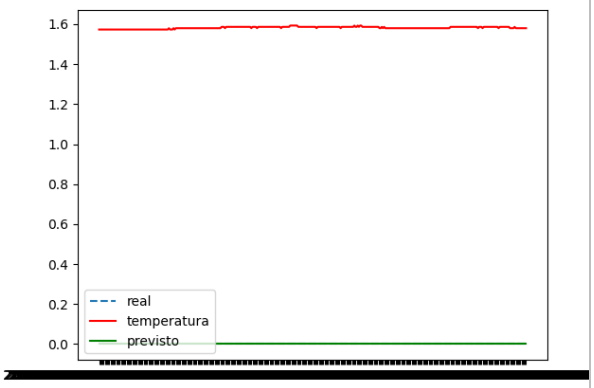
Identificação de transição ON para OFF



Identificação de transição OFF para ON



Pequeno falso positivo logo após transição



Sinal estável sem falsos positivos

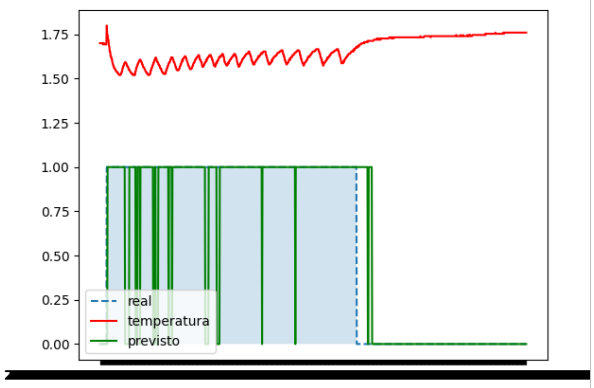
Validações adicionais

Validações adicionais foram realizadas por meio do confronto cruzado de um modelo construído com dados de um local físico avaliando dados de outro local. No contexto do trabalho, o modelo com melhor desempenho foi construído com os intervalos 4, 5 e 6 do local_id igual a '1' e com *features* de natureza térmica. Esse exercício adicional se propõe a avaliar o desempenho deste modelo estimando o comportamento do ar-condicionado de outro local físico, no caso, local_id igual a '0'.

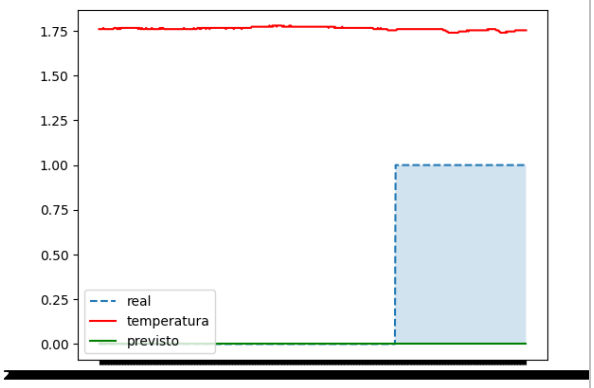
Para esta etapa, para efeito de simplificação dos resultados, foram registrados apenas informações de classificadores do tipo *Random Forest*. Os indicadores de desempenho quando avaliamos todos os intervalos (1, 2 e 3) do local_id = '0' são os seguintes:

		precision	recall	f1-score	support
RandomForestClassifier (bônus)	75%	75%	99%	85%	975
	88%	88%	22%	35%	2673
				76%	8973
	81%	81%	60%	60%	8973
	79%	79%	76%	70%	8973

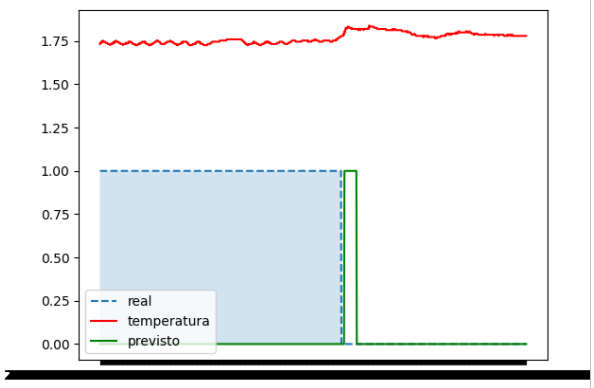
Na sequência de figuras a seguir, são demonstradas situações em que o classificador apresentou desempenho adequado quanto a falsos positivos, verdadeiros positivos, falsos negativos e verdadeiros negativos. Para o caso das validações cruzadas, e diferentemente do constatado na sessão anterior onde foi avaliada a validação convencional, as transições não foram bem identificadas pelo classificador. Atrasos típicos aconteceram com ordem de dezenas de minutos (geralmente menor que três minutos). Os resultados dessa validação adicional estão registradas no arquivo bonus.py reproduzido no ANEXO IV.



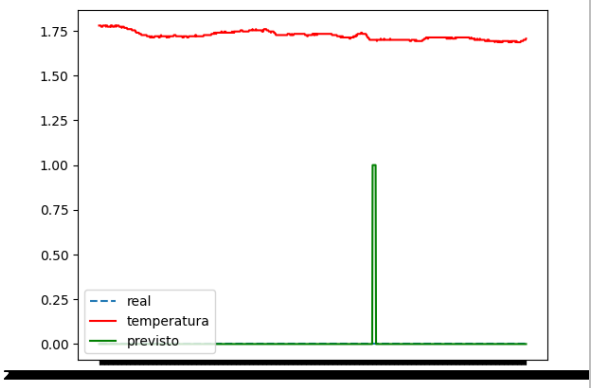
Sinal com falso negativo e atraso em transição



Transição não identificada



Intervalo 1 não identificado



Sinal com falsos positivos

Conclusão

A partir de um conjunto de dados de temperatura e umidade registrados em dois ambientes diferentes foi possível estimar por meio de algoritmos de *machine learning* os momentos em que o ar-condicionado se apresentou ligado ou desligado nesses amibentes. As estimativas apresentaram adequação estatística corroborando o processo de exploração e transformação dos dados, definição de *features* relevantes e desempenho avaliados por parâmetros consagrados na literatura.

Quanto ao desempenho das operações de classificação, pode-se concluir que o classificador apresentou desempenho adequado quando exposto aos dados intrínsecos ao ambiente em que foi construído. Quando um classificador foi exposto a dados de outro ambiente físico (validação cruzada), os resultados permitem concluir sobre um baixo desempenho de classificação. Para este último caso, demanda-se maiores investigações, seleções de novas

features ou mesmo a inclusão de variáveis exógenas, como por exemplo, a temperatura do ambiente externo.

ANEXO I

```
import pandas as pd
import sqlite3
import numpy as np
import matplotlib.pyplot as plt


def load_sql():
con = sqlite3.connect("/Users/ciribelli/Server/Data Mining/db.sqlite3")
saida = pd.read_sql_query("SELECT * from controleambiente_ambiente", con)
saida['data'] = pd.to_datetime(saida['data'], format='%Y-%m-%d %H:%M:%S')
con.close()
return saida


def filtra(s):
to_remove = s[s['temperatura'] < 15.0].index
s = s.drop(to_remove)
to_remove = s[s['umidade'] < 30.0].index
s = s.drop(to_remove)
return s


def plota(f, s, item, a):
axs[item, 0].plot(s.index, s['umidade'], f.index, f['umidade'])
axs[item, 1].plot(s.index, s['temperatura'], f.index, f['temperatura'])
#axs[item, 0].set(xlabel='tempo', ylabel='temp')
axs[item, 0].grid()
axs[item, 1].grid()
return(plt)


def classifica(df, i):
dfs = df.set_index('data')[i['t0']:i['tf']]
dfs = dfs.loc[dfs['local_id'] == i['local_id']]
dfs = dfs.drop(['id'], axis=1)
dfs['classificacao'] = 1
dfs['classificacao'] = np.where(dfs.index >= i['toff'], 0, dfs['classificacao'])
return dfs


def segmentaInicio(u):
u = u.loc[i['ti']:i['tf']]
return u


def aplica_janela(daux):


janela_0 = 15 # janela deslizante de 15 elementos
janela_1 = 30 # janela deslizante de 30 elementos
```

```
janela_2 = 60 # janela deslizante de 60 elementos

l_janelas = [janela_0, janela_1, janela_2]

for l_j in l_janelas:

# para temperaturas
s_arcon_t = daux['temperatura'].rolling(l_j).std() # desvio padrao
v_arcon_t = daux['temperatura'].rolling(l_j).var() # variância
m_arcon_t = daux['temperatura'].rolling(l_j).mean() # média
min_arcon_t = daux['temperatura'].rolling(l_j).min() # calc. auxiliar de mínimo
max_arcon_t = daux['temperatura'].rolling(l_j).max() # calc. auxiliar de máximo
a_arcon_t = max_arcon_t - min_arcon_t # amplitude (max - min)

# para umidade
s_arcon_u = daux['umidade'].rolling(l_j).std() # desvio padrao
v_arcon_u = daux['umidade'].rolling(l_j).var() # variância
m_arcon_u = daux['umidade'].rolling(l_j).mean() # média
min_arcon_u = daux['umidade'].rolling(l_j).min() # calc. auxiliar de mínimo
max_arcon_u = daux['umidade'].rolling(l_j).max() # calc. auxiliar de máximo
a_arcon_u = max_arcon_u - min_arcon_u # amplitude (max - min)

daux['temp_desvpad_' + str(l_j)] = s_arcon_t
daux['temp_variancia_' + str(l_j)] = v_arcon_t
daux['temp_media_' + str(l_j)] = m_arcon_t
daux['temp_amplitude_' + str(l_j)] = a_arcon_t
daux['umid_desvpad_' + str(l_j)] = s_arcon_u
daux['umid_variancia_' + str(l_j)] = v_arcon_u
daux['umid_media_' + str(l_j)] = m_arcon_u
daux['umid_amplitude_' + str(l_j)] = a_arcon_u
return (daux)

# Inicio
df = load_sql() # dataframe original
ds = filtra(df) # dataframe filtrado

intervalo_01 = {'t0': '2018-12-05 15:10', 'ti': '2018-12-05 16:28', 'tf': '2018-12-06 14:00', 'toff': '2018-12-06 01:36',
'local_id': 0}
intervalo_02 = {'t0': '2018-10-03 20:30', 'ti': '2018-10-03 21:43', 'tf': '2018-10-04 22:40', 'toff': '2018-10-04 02:43',
'local_id': 0}
intervalo_03 = {'t0': '2018-11-28 15:10', 'ti': '2018-11-28 16:23', 'tf': '2018-11-29 22:40', 'toff': '2018-11-29 01:07',
'local_id': 0}
```

```
intervalo_04 = {'t0': '2020-04-07 22:15', 'ti': '2020-04-07 23:19', 'tf': '2020-04-08 17:48', 'toff': '2020-04-08 10:30',
'local_id': 1}
intervalo_05 = {'t0': '2020-04-12 22:25', 'ti': '2020-04-12 23:30', 'tf': '2020-04-13 22:00', 'toff': '2020-04-13 10:46',
'local_id': 1}
intervalo_06 = {'t0': '2020-04-11 00:05', 'ti': '2020-04-11 01:12', 'tf': '2020-04-11 16:52', 'toff': '2020-04-11 11:32',
'local_id': 1}


l = [intervalo_01, intervalo_02, intervalo_03, intervalo_04, intervalo_05, intervalo_06]


fig, axs = plt.subplots(6, 2)
dF = pd.DataFrame()


# item eh o indice do for para geracao dos graficos // i eh o objeto intervalo
for item, i in enumerate(l):
# segmenta e adiciona classificacao
dff = classifica(df, i) # c/ o original
dfs = classifica(ds, i) # c/ o filtrado
# aplica janela
dfa = aplica_janela(dfs)
# ajusta inicio do segmento de t0 para ti
dfa = segmentaInicio(dfa)
# plota
plota(dfa, dff, item, axs)
# concatena
dF = pd.concat([dF, dfa])


plt.show()


# salva arquivo csv
dF.to_csv('/Users/ciribelli/Server/Data Mining/datasetDM.csv', index=True)
print('Arquivo salvo')
```

ANEXO II

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#df = pd.read_csv('/Users/ciribelli/Server/Data Mining/datasetDM.csv')
df = pd.read_csv('C://Users//Ciribelli//OneDrive//Documentos//BI
Master//DM//datamining//datasetDM.csv')

df = df.loc[df['local_id'] == 1]

# opcoes de janela para selecao de parametros
#df = df.loc[:,['classificacao',
'umid_amplitude_30',   'umid_desvpad_30',  'umid_media_30', 'umid_variancia_30']] #
local_id = 0
#df = df.loc[:,['temp_desvpad_60','temp_variancia_60', 'temp_media_60',
'temp_amplitude_60', 'classificacao']] # local_id = 1
#df = df.loc[:,['temp_desvpad_30', 'temp_variancia_30',  'temp_media_30',
'temp_amplitude_30', 'classificacao']] # local_id = 1

# opcao selecionada
df = df.loc[:,['temp_media_15', 'temp_desvpad_60', 'temp_variancia_30', 'temp_media_30',
'temp_amplitude_60', 'classificacao']] # local_id = 1

print(df.head())

sns.pairplot(df,diag_kind="kde",hue="classificacao",palette="husl")
plt.show()
```

ANEXO III

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
import seaborn as sns
import numpy as np

df = pd.read_csv('C://Users//Ciribelli//OneDrive//Documentos//BI
Master//DM//datamining//datasetDM.csv')
df=df.replace(to_replace="ON",value=1)
df=df.replace(to_replace="OFF",value=0)
df = df.loc[df['local_id'] == 1]
# features selecionadas
df = df.loc[:,['temp_media_15', 'temp_desvpad_60', 'temp_variancia_30', 'temp_media_30',
'temp_amplitude_60', 'classificacao']]
X = df.loc[:,df.columns != 'classificacao']  # Entrada
y = df.classificacao   # Saida
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)

############ NORMALIZACAO

# from sklearn.preprocessing import StandardScaler
# scaler = StandardScaler().fit(X_train)
# X_train = scaler.transform(X_train)
# X_test = scaler.transform(X_test)



# treinar modelo
from sklearn.ensemble import RandomForestClassifier
def train(X_train, y_train):
    model = RandomForestClassifier(min_samples_leaf=5, bootstrap=False) # tente mudar
parâmetro para evitar overfitting
    model.fit(X_train, y_train)
    return model

# treinar modelo
# from sklearn.ensemble import GradientBoostingClassifier
# def train(X_train, y_train):
#     model = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=3,
random_state=0)
#     model.fit(X_train, y_train)
#     return model

#treinar modelo
# from sklearn.neural_network import MLPClassifier
# def train(X_train, y_train):
#     model = MLPClassifier(hidden_layer_sizes=(100,100,100), max_iter=500, alpha=0.0001,
solver='adam', verbose=True, random_state=21 ,tol=0.000001)
#     model.fit(X_train, y_train)
#     return model

# from sklearn import svm
# def train(X_train, y_train):
#     model = svm.SVC()
#     model.fit(X_train, y_train)
#     return model


model = train(X_train, y_train)

def predict_and_evaluate(model, X_test, y_test):
```

```
    y_pred = model.predict(X_test) # inferência do teste

    # Acurácia
    from sklearn.metrics import accuracy_score
    accuracy = accuracy_score(y_test, y_pred)
    print('Acurácia: ', accuracy)

    # Kappa
    from sklearn.metrics import cohen_kappa_score
    kappa = cohen_kappa_score(y_test, y_pred)
    print('Kappa: ', kappa)

    # F1
    from sklearn.metrics import f1_score
    f1 = f1_score(y_test, y_pred)
    print('F1: ', f1)

    # Matriz de confusão
    # .: true negatives is C00, false negatives is C10, true positives C11 is  and false
positives is C01.
    from sklearn.metrics import confusion_matrix
    confMatrix = confusion_matrix(y_test, y_pred)

    ax = plt.subplot()
    sns.heatmap(confMatrix, annot=True, fmt=".0f")
    plt.xlabel('Previsto')
    plt.ylabel('Real')
    plt.title('Matriz de Confusão')

    # Colocar os nomes
    ax.xaxis.set_ticklabels(['OFF', 'ON'])
    ax.yaxis.set_ticklabels(['OFF', 'ON'])
    plt.show()

    from sklearn.metrics import classification_report
    print(classification_report(y_test, y_pred))

print('Resultados de Treino')
predict_and_evaluate(model, X_train, y_train)
print('Resultados de Teste')
predict_and_evaluate(model, X_test, y_test)
```

ANEXO IV

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pathlib
import joblib
from sklearn.metrics import classification_report



# Inicio

path = pathlib.Path(__file__).parent.resolve()
df = pd.read_csv(pathlib.PurePath(path, 'datasetDM.csv'))
df = df.loc[df['local_id'] == 0] # para fazer validacoes adicionais com a classe do
local_id = 0

model = joblib.load(pathlib.PurePath(path, 'ml.pkl'))

x = df.data
temperatura = df.temperatura
y = df.classificacao   # Saida
X = df.loc[:,['temp_media_15',  'temp_desvpad_60', 'temp_variancia_30', 'temp_media_30',
'temp_amplitude_60']]
y_pred = model.predict(X) # inferência do teste

print(classification_report(y, y_pred))

# ajuste da janela de plot para efeito de otimizacao do tempo
d = 1500
i = 0 + d
f = 1000 + d

fig, ax = plt.subplots()
ax.plot(x[i:f], y[i:f], '--', label = 'real')
ax.plot(x[i:f], temperatura[i:f]/15, '-', color = 'red', label = 'temperatura')
ax.fill_between(x[i:f], y[i:f], 0, alpha=0.2)
ax.plot(x[i:f], y_pred[i:f], '-', color='green', label = 'previsto')
plt.legend(loc="lower left")


plt.show()
```