PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

Rafael Henrique Pinotti

MODELO PARA PREDIÇÃO DE CORRENTES DE FUGA EM ISOLADORES DE DISTRIBUIÇÃO CLASSE 25 kV

Belo Horizonte
2020

Rafael Henrique Pinotti

PREDIÇÃO DE CORRENTES DE FUGA EM ISOLADORES DE DISTRIBUIÇÃO CLASSE 25 kV

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

Belo Horizonte

2020

LISTA DE FIGURAS

SUMÁRIO

1. Introdução

1.1. Contextualização

A qualidade com que a energia elétrica é fornecida às unidades consumidoras é uma preocupação constante por parte das concessionárias que fornecem este insumo à população.

Grande parte da qualidade sistema elétrico passa pelas linhas de transmissão e distribuição, que em sua maioria estão expostas às condições ambientais e aos contaminantes presentes na atmosfera.

Dentre os principais equipamentos utilizados no sistema estão os isolantes elétricos, que “têm [...] a função de limitar a passagem de corrente elétrica, entre um equipamento ou condutor energizado, e um potencial neutro [...], permitindo a operação de forma segura de dispositivos e sistemas elétricos” [4]. Alguns modelos estão ilustrados na figura 1.

Figura 1. Alguns modelos de isoladores. Fonte [4].

A classe de tensão de 25 kv (cujos isolantes são objetos deste estudo), são utilizadas em subestações e alimentadores aéreos de distribuição, com tensão alternada acima de 1000 v [1], sendo a classe utilizada nas redes de distribuição em Santa Catarina.

O presente artigo é baseado em [4], mas não pretende realizar comparações de resultados. O foco é utilizar algoritmos de aprendizado de máquina a fim de se criar um modelo com boa acurácia e que possa futuramente ser utilizado em outras soluções.

1.2. O problema proposto

As falhas em isoladores elétricos são responsáveis por muitos desligamentos (não programados) na rede elétrica.

O monitoramento da saúde dos isoladores é feito através da medição das correntes de fuga, o que permite uma manutenção preventiva destes equipamentos. Porém, alterações climáticas e ambientais podem prolongar ou encurtar o tempo de vida útil do isolamento, o que dificulta uma ação eficaz na prevenção de desligamentos não programados.

Os dados utilizados são provenientes de Dissertação de Mestrado [1] defendida na Universidade Regional de Blumenau (Furb).

Pretende-se com esta análise encontrar um modelo para predição da corrente de fuga em isoladores elétricos, baseado na coleta de dados atmosféricos, permitindo uma ação preventiva mais eficiente para este tipo de equipamento em sistemas de transmissão e distribuição de energia elétrica.

Os dados foram gerandos em uma Estação de Monitoramento de Sistemas Isolantes (EMSI) localizada na cidade de Itajaí em Santa Catarina e coletados diariamente durante o mês de Fevereiro de 2010.

Para o presente trabalho iremos focar em um único tipo de isolador (Pilar Polimérico - HTV) para criar o modelo de aprendizado de máquina. A figura 2 ilustra o modelo escolhido.

Figura 2. Isolador Polimérico. Fonte [3] .

2. Coleta de Dados

Os dados analisados foram coletados por sensores e equipamentos na EMSI instalada em subestação elétrica da Celesc (Centrais Elétricas de Santa Catarina) na Praia Brava em Itajaí/SC, gravando os resultados em *datasets* segmentados por isolador.

Os *datasets* disponibilizados são resultado da coleta de 40.320 dados (cada isolador), realizado durante 28 dias do mês de Fevereiro de 2010, com valores de corrente de fuga e dados meteorológicos.

A tabela abaixo exibe o dicionário de dados:

Coluna/campo	***Descrição***	***Tipo***
umidade	Percentual de umidade no ar	Inteiro
temperatura	Temperatura ambiente em ^{0}C	Decimal
pressao	Pressão atmosférica em mb	Inteiro
vento_ang	Direção do vento (coordenada em graus)	Inteiro
vent_veloc	Velocidade do vento em m/s	Decimal
chuva	Quantidade de chuva em mm/h	Decimal
corrente_fuga	Corrente de fuga em mA	Decimal

Tabela 1. Dicionário de Dados.

A massa de dados foi disponibilizada através de um arquivo do tipo CSV (*comma separated values*), e importado para um *dataframe* da biblioteca *Pandas* para que possa ser manipulado.

```
# Importar dados de arquivo csv, onde o separador de campos é '.' e o separador de casas decimais é ','
df = pd.read_csv('dados/dados_isolador40.csv', sep=';', decimal=',')
```

Figura 3. Importação dos dados.

3. Processamento/Tratamento de Dados

Ao inicar o tratamento dos dados, é possível verificar que os registros importados condizem com o dicionário fornecido.

```
df.head(5)
```

	umidade	temperatura	pressao	vento_ang	vent_veloc	chuva	corrente_fuga
0	70	29.6	1016	278	1.4	0	0.024992
1	69	29.7	1016	297	1.4	0	0.022767
2	70	29.8	1016	273	1.6	0	0.022607
3	68	29.8	1016	243	2.6	0	0.022426
4	68	29.7	1016	257	2.8	0	0.031797

Figura 4. Cinco primeiros registros no dataframe.

O comando *shape* traz o formato do *dataframe*, onde podemos verificar um total de 40.316 registros e 7 colunas.

```
# formato do dataframe (registros/colunas)
df.shape
```

```
(40316, 7)
```

Figura 5. Formato do dataframe.

Ao verificar tanto a quantidade de registros faltantes quanto de registros nulos, pudemos observar que o *dataset* não apresenta nenhum valor nulo ou *missing*.

```
# Valores missing
df.isnull().any()
```

```
umidade          False
temperatura      False
pressao          False
vento_ang        False
vent_veloc       False
chuva            False
corrente_fuga    False
dtype: bool
```

```
# Registros nulos
df.isnull().sum()
```

```
umidade          0
temperatura      0
pressao          0
vento_ang        0
vent_veloc       0
chuva            0
corrente_fuga    0
dtype: int64
```

Figura 6. Verificação de valores missing ou nulos.

Em seguida buscamos por registros com valor zero em algum atributo.

```
# Quantidade de registros com valor zero em cada atributo.
(df == 0).sum()
```

```
umidade              0
temperatura          0
pressao              0
vento_ang           92
vent_veloc       14511
chuva            38064
corrente_fuga        0
dtype: int64
```

Figura 7. Registros com valor zero em atributos.

Existem três atributos no *dataset* que possuem registros com valores zerados, ângulo do vento, velocidade do vendo e chuva (*vento_ang*, *vento_veloc* e *chuva* respectivamente).

Nenhum tratamento será realizado nestes dados visto que são condições atmosféricas, e a falta de chuva e vento é uma condição ambiental a ser considerada no modelo.

4. Análise e Exploração dos Dados

Inicialmente foi-se verificada a distribuição dos dados alvo do nosso modelo, para levantarmos a necessidade de tratamentos adicionais como normalização ou padronização das variáveis.

```
distplots_dataframe(df)
```

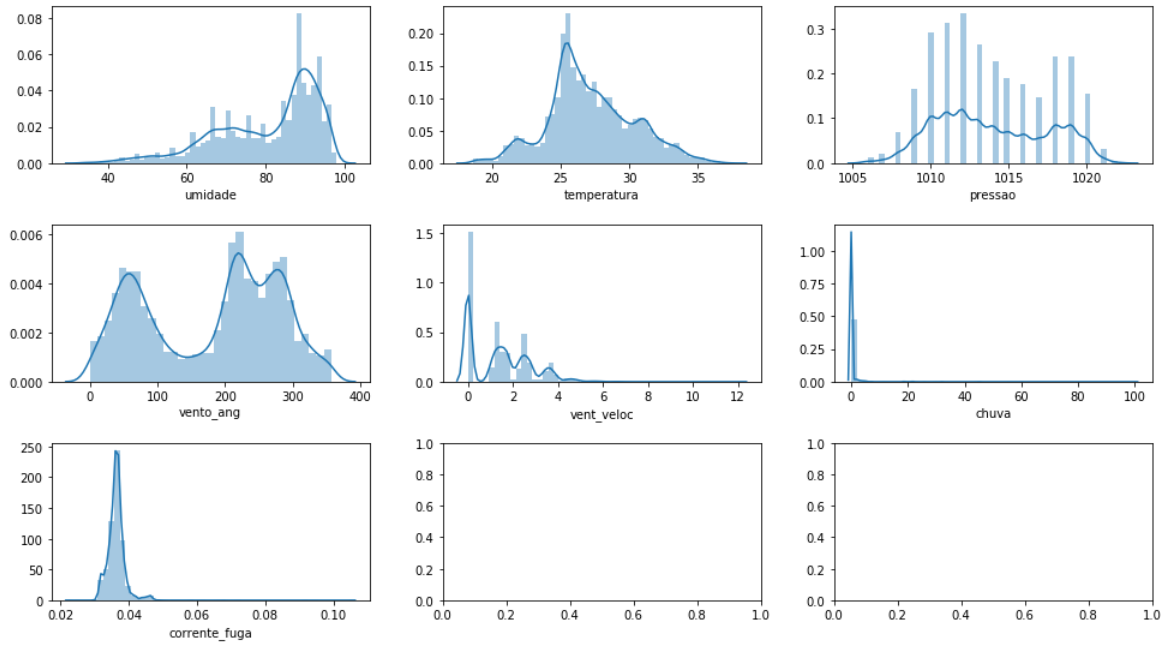

Figura 8. Distribuição dos dados coletados.

Os dados estão em escalas muito diversas, neste caso iremos normalizar para uma melhor eficiência do modelo.

```
min_scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = pd.DataFrame(min_scaler.fit_transform(df), columns=df.columns)
```

Figura 9. Padronização de escala das variáveis.

```
distplots_dataframe(scaled_data)
```

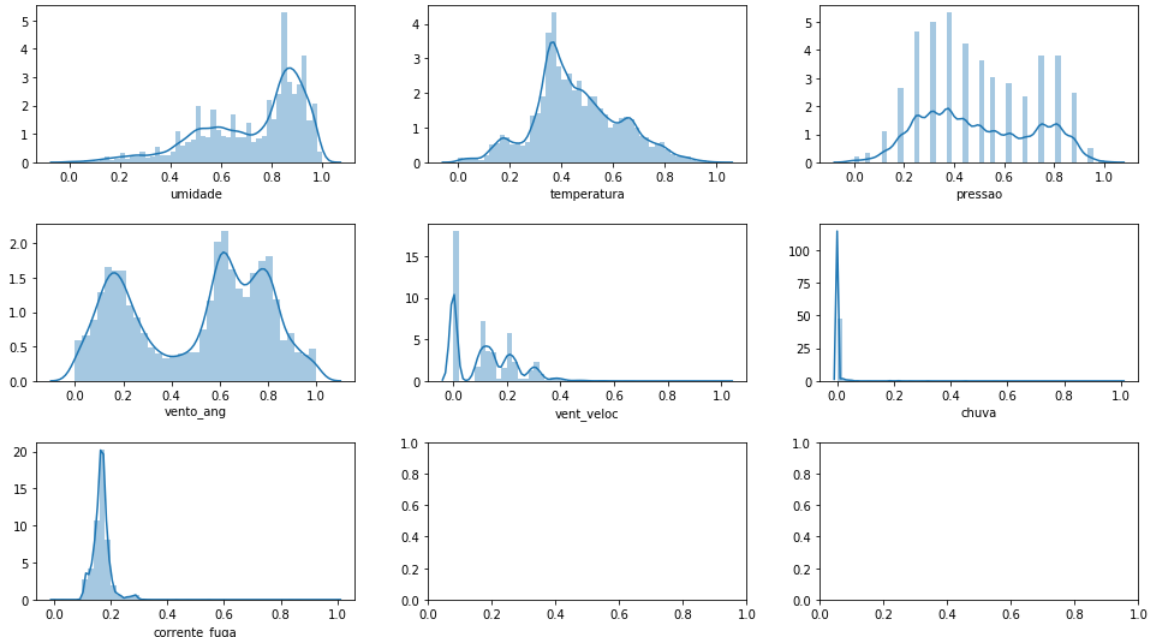

Figura 10. Dados padronizados para uma mesma escala.

Verificamos os dados estatísticos do *dataframe*, como média, desvio padrão e os valores dos quartis.

```
# Dados estatísticos do dataframe
df = scaled_data
df.describe()
```

	umidade	temperatura	pressao	vento_ang	vent_veloc	chuva	corrente_fuga
count	40316.000000	40316.000000	40316.000000	40316.000000	40316.000000	40316.000000	40316.000000
mean	0.726673	0.456358	0.495887	0.503842	0.120464	0.003483	0.167389
std	0.201368	0.166669	0.223528	0.278183	0.111802	0.027008	0.030226
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.578125	0.351064	0.312500	0.210084	0.000000	0.000000	0.152262
50%	0.796875	0.430851	0.437500	0.588235	0.117647	0.000000	0.167479
75%	0.890625	0.563830	0.687500	0.742297	0.201681	0.000000	0.179329
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Figura 11. Dados estatísticos do dataframe.

A seguir verificamos os dados em busca de *outliers* que possam invalidar o modelo podendo causar *overfitting* ou *underfitting*.

```
boxplots_dataframe(df)
```

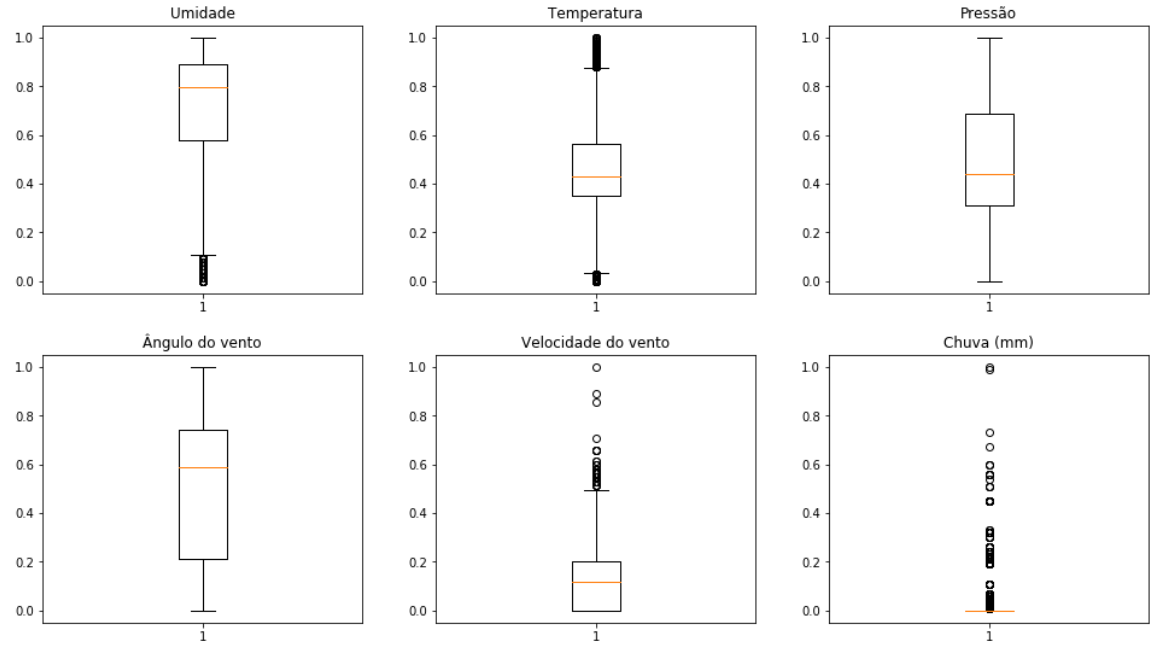

Figura 12. Boxplots das variáveis do dataframe.

As variáveis *umidade*, *temperatura*, *velocidade do vento* e *chuva*, apresentam uma grande quantidade de *outliers*, apesar disto elas serão mantidas no modelo por serem valores sujeitos a grande variação no período, por se tratarem de variáveis ambientais, e portanto, de alta volatilidade.

O próximo passo foi verificar de que forma as variavés se correlacionam.

```
# Plotando a correlação entre os atributos
df.corr().style.format("{:.5}").background_gradient(cmap=plt.get_cmap('coolwarm'), axis=1)
```

	umidade	temperatura	pressao	vento_ang	vent_veloc	chuva	corrente_fuga
umidade	1.0	-0.44157	-0.20368	0.32259	-0.50396	0.11214	0.69658
temperatura	-0.44157	1.0	-0.39874	-0.43925	0.45762	-0.11037	-0.34919
pressao	-0.20368	-0.39874	1.0	-0.011539	-0.045943	0.044076	-0.064254
vento_ang	0.32259	-0.43925	-0.011539	1.0	-0.34307	0.0424	0.22667
vent_veloc	-0.50396	0.45762	-0.045943	-0.34307	1.0	-0.0043842	-0.3078
chuva	0.11214	-0.11037	0.044076	0.0424	-0.0043842	1.0	0.29679
corrente_fuga	0.69658	-0.34919	-0.064254	0.22667	-0.3078	0.29679	1.0

Figura 13. Gráfico de correlação das variáveis.

Através do gráfico podemos notar uma correlação muito diversa entre as variáveis.

```
# Correlação da corrente de fuga com os demais atributos.
df.corr()['corrente_fuga']
```

```
umidade          0.696576
temperatura     -0.349192
pressao         -0.064254
vento_ang        0.226674
vent_veloc      -0.307804
chuva            0.296792
corrente_fuga    1.000000
Name: corrente_fuga, dtype: float64
```

Figura 14. Correlação entre as variáveis preditoras e a variável alvo.

A correlação é muito diferente entre as diversas variáveis preditoras e a variável alvo. Na próxima seção iremos testar de que forma estas correlações se comportam nos modelos a serem testados.

5. Criação de Modelos de Machine Learning

Por nossa variável *target* ser um valor numérico real, iremos trabalhar a classificação para realizar as previsões.

Nesta etapa optamos por trabalhar com três diferentes algoritmos de regressão, identificando qual melhor se adapata ao nosso problema.

Apesar dos baixos valores relacionados à correlação de algumas das variáveis preditoras com a variável alvo, iniciaremos o treinamento do modelo com todos os atributos.

5.1. Regressão Linear

O primeiro algoritmo a ser testado é o de *Regressão Linear*. Criamos uma função para treinar o modelo e calcular o *score* no final do processo.

```
def calcula_regressao(columns):
    X_train, X_test, y_train, y_test = train_test_split(df[columns], df['corrente_fuga'], test_size=0.20, random_state=42)
    lr = LinearRegression(normalize = False, fit_intercept=True)
    # Treinando o modelo
    model_lr = lr.fit(X_train, y_train)
    result_lr = model_lr.predict(X_test)
    # MSE: média do erro quadrado. Quanto menor o valor, mais acertivo é o modelo.
    mse_lr = mean_squared_error(y_test, result_lr)
    # RMSE: raiz quadrada do MSE (facilita a visualização do valor).
    rmse_lr = (np.sqrt(mse_lr))
    # A métrica score varia entre 0 e 1, e indica percentualmente o quanto o modelo consegue explicar os valores observados.
    # Quanto maior o valor mais explicativo é o modelo.
    score_lr = model_lr.score(X_test, y_test)
    print('Resultados do erro médio e score (acurácia do modelo) para o dataset de teste:\n  MSE: {}'.format(mse_lr)
          + '\n RMSE: {}'.format(rmse_lr)
          + '\nScore: {:2.4f}'.format(score_lr * 100) + '%')
    return score_lr;
```

Figura 15. Função para treinamento do modelo e cálculo do score.

Em nosso primeiro modelo, removemos apenas a coluna *corrente_fuga*, para verificar a acurácia com todos os atributos (exceto a variável *target*).

```
# Excluindo a variável target 'corrente_fuga'.
cols = df.columns.drop(['corrente_fuga'])
cols
```

```
Index(['umidade', 'temperatura', 'pressao', 'vento_ang', 'vent_veloc',
       'chuva'],
      dtype='object')
```

Figura 16. Remoção da variável target da lista de colunas do dataset.

Na sequência calculamos o *score* deste modelo com todas as variáveis preditoras presentes.

```
score_cols = calcula_regressao(cols)
```

```
Resultados do erro médio e score (acurácia do modelo) para o dataset de teste:
  MSE: 0.0003483160762296319
 RMSE: 0.01866322791560002
Score: 58.1681%
```

Figura 17. Cálculo do score com todas as variáveis preditoras presentes.

A seguir realizamos alguns testes removendo as colunas com baixa correlação, inicialmente as menores de 0,50.

```
cols_maior_05 = df.columns.drop(['temperatura','pressao','vento_ang','vent_veloc','chuva','corrente_fuga'])
cols_maior_05
```

```
Index(['umidade'], dtype='object')
```

```
score_cols_maior_05 = calcula_regressao(cols_maior_05)
```

```
Resultados do erro médio e score (acurácia do modelo) para o dataset de teste:
  MSE: 0.00038852908063290485
 RMSE: 0.019711141028182638
Score: 53.3386%
```

Figura 18. Cálculo do score ao remover colunas com correlação inferior a 0,50.

Em seguida as de correlação menor de 0,30.

```
cols_maior_03 = df.columns.drop(['pressao', 'vento_ang','chuva','corrente_fuga'])
cols_maior_03
```

```
Index(['umidade', 'temperatura', 'vent_veloc'], dtype='object')
```

```
score_cols_maior_03 = calcula_regressao(cols_maior_03)
```

```
Resultados do erro médio e score (acurácia do modelo) para o dataset de teste:
  MSE: 0.0003815019604008739
 RMSE: 0.01953207516883124
Score: 54.1825%
```

Figura 19. Cálculo do score ao remover colunas com correlação inferior a 0,30.

Depois, correlação menor que 0,10.

```
cols_maior_01 = df.columns.drop(['pressao','corrente_fuga'])
cols_maior_01
```

```
Index(['umidade', 'temperatura', 'vento_ang', 'vent_veloc', 'chuva'], dtype='object')
```

```
score_cols_maior_01 = calcula_regressao(cols_maior_01)
```

```
Resultados do erro médio e score (acurácia do modelo) para o dataset de teste:
  MSE: 0.00035207721393883953
 RMSE: 0.01876372068484392
Score: 57.7164%
```

Figura 20. Cálculo do score ao remover colunas com correlação inferior a 0,10.

Pudemos observar que a manutenção de todas as variáveis preditoras no *dataset* nos trouxe uma maior acurácia em nosso modelo. O gráfico abaixo ilustra esta comparação.

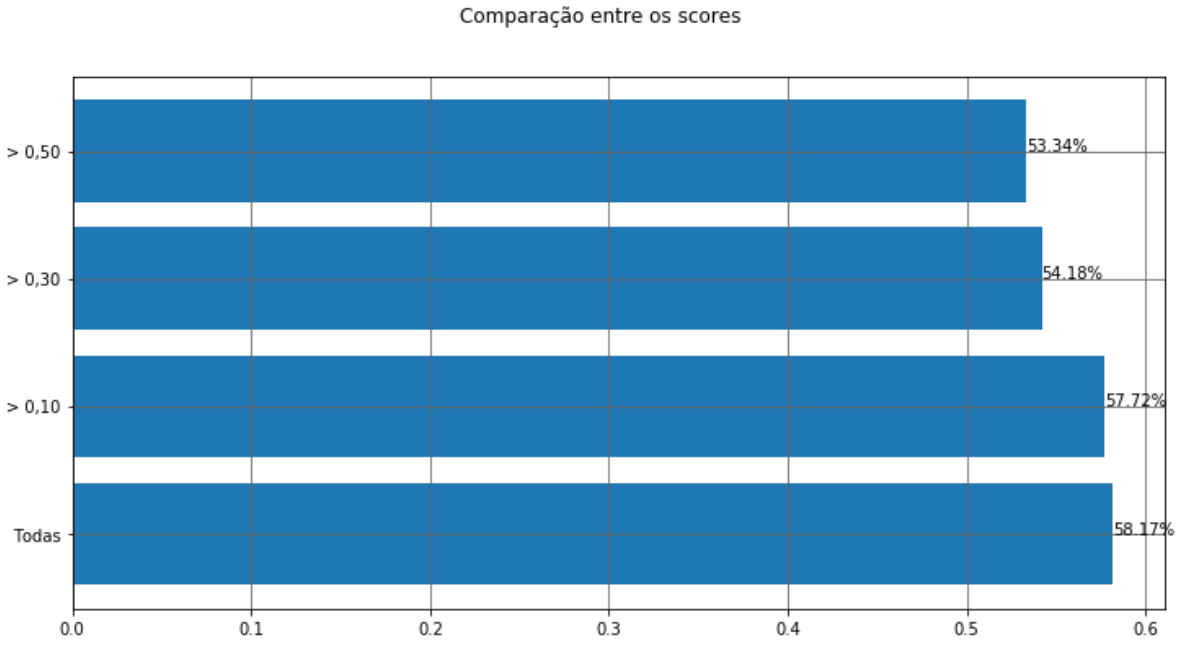

Figura 21. Comparação dos scores utilizando Regressão Linear.

Por termos um *score* maior utilizando todas as variáveis preditoras, optamos por mantê-las nos algoritmos a serem testados posteriormente.

5.2. Árvore de Decisão

O segundo algoritmo testado é o de *Árvore de Decisão*, que são métodos de aprendizado de máquinas supervisionados e não-paramétricos, utilizados em tarefas de classificação e regressão.

Separamos o *dataset* em treino e teste, onde deixamos 20% dos dados para testes no modelo.

```
X_train, X_test, y_train, y_test = train_test_split(df[colunas], df['corrente_fuga'], test_size=0.20, random_state=42)
```

Figura 22. Separação do dataset em treino e teste.

Criamos o objeto *tree*, ajustamos o modelo e realizamos as predições.

```
tree = DecisionTreeRegressor()
```

```
model_tree = tree.fit(X_train, y_train)
result_tree = tree.predict(X_test)
```

Figura 23. Ajustando o modelo e realizando as predições.

Ao calcular o *score* tivemos uma acurácia de 84,71% com este algoritmo.

```
mse_tree = mean_squared_error(y_test, result_tree)
rmse_tree = (np.sqrt(mse_tree))
score_tree = model_tree.score(X_test, y_test)
print('Resultados do erro médio e score (acurácia do modelo) para o dataset de treino:\n  MSE: {}'.format(mse_tree)
      + '\n RMSE: {}'.format(rmse_tree)
      + '\nScore: {:2.3f}'.format(score_tree * 100) + '%')
```

```
Resultados do erro médio e score (acurácia do modelo) para o dataset de treino:
  MSE: 0.00012725337306180857
 RMSE: 0.01128066368002382
Score: 84.717%
```

Figura 24. Cálculo do score para Árvore de Decisão.

5.3. Random Forest

Algoritmo de aprendizagem supervisionada, ele cria uma combinação (*ensemble*) de árvores de decisão, com o objetivo de reduzir a variância das previsões. Esta técnica é conhecida como *bagging*.

Com o algoritmo de *Random Forest* obtivemos uma acurácia ainda melhor, com 88,54% de assertividade.

```
X_train, X_test, y_train, y_test = train_test_split(df[colunas], df['corrente_fuga'], test_size=0.20, random_state=42)
```

```
tree_rf = RandomForestRegressor()
```

```
model_rf = tree_rf.fit(X_train, y_train)
result_rf = model_rf.predict(X_test)
```

```
mse_rf = mean_squared_error(y_test, result_rf)
rmse_rf = (np.sqrt(mse_rf))
score_rf = model_rf.score(X_test, y_test)
print('Resultados do erro médio e score (acurácia do modelo) para o dataset de treino:\n  MSE: {}'.format(mse_rf)
      + '\n RMSE: {}'.format(rmse_rf)
      + '\nScore: {:2.3f}'.format(score_rf * 100) + '%')
```

```
Resultados do erro médio e score (acurácia do modelo) para o dataset de treino:
  MSE: 9.534991560099171e-05
 RMSE: 0.009764728137587431
Score: 88.549%
```

Figura 25. Aplicação do algoritmo de Random Forest.

5.4. Random Forest com Ajustes nos Hiperparâmetros

Para uma acurária ainda maior, realizamos ajustes nos *hiperparâmetros* do algoritmo de *Random Forest*.

```
X_train, X_test, y_train, y_test = train_test_split(df[colunas], df['corrente_fuga'], test_size=0.20, random_state=42)
```

Figura 26. Separação dos dados de treino e teste.

Três parâmetros foram adicionados, e a cada parâmetro foi passado um conjunto de valores a serem testados. São eles:

- *min_samples_leaf*: número de amostras mínimas ao nível folha.
- *min_samples_split*: número de amostras mínimas para considerar um nó para divisão.
- *n_estimators*: número de árvores construídas pelo algoritmo antes de tomar uma votação ou fazer uma média de previsões.

```
param = {'min_samples_leaf':[1,3,5],'min_samples_split':[2,3,5],'n_estimators':[50,150,250]}
rf = RandomForestRegressor()
grid = GridSearchCV(rf, param, n_jobs=4)
```

Figura 27. Ajustes nos hiperparâmetros e criação dos objetos.

Os valores utilizados para os parâmetros foram escolhidos de forma a não tornar a árvore muito grande, desta maneira não temos um processamento muito pesado, podendo reproduzir os resultados em computadores pessoais.

Executamos o *grid* para treinar nosso modelo. Com os ajustes nos *hiperparâmetros* esta etapa levou um tempo consideravelmente maior com relação aos outros algoritmos testados.

```
# Executando o grid
grid.fit(X_train, y_train)
```

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
           oob_score=False, random_state=None, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=4,
       param_grid={'min_samples_leaf': [1, 3, 5], 'min_samples_split': [2, 3, 5], 'n_estimators': [50, 150, 250]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

Figura 28. Executando o grid.

O algoritmo também nos orienta na verificação da melhor combinação dentre os parâmetros testados.

```
# Verificando os melhores parâmetros
grid.best_params_
```

```
{'min_samples_leaf': 3, 'min_samples_split': 3, 'n_estimators': 250}
```

```
rf_best = grid.best_estimator_
print(rf_best)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=3, min_samples_split=3,
           min_weight_fraction_leaf=0.0, n_estimators=250, n_jobs=None,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

Figura 29. Verificando os melhores parâmetros.

Ao realizar as previsões e calcular o *score*, percebemos um aumento na acurácia do nosso modelo.

```
result_grid = rf_best.predict(X_test)
```

```
mse_grid = mean_squared_error(y_test, result_grid)
rmse_grid = (np.sqrt(mse_grid))
score_grid = rf_best.score(X_test, y_test)
print('Resultados do erro médio e score (acurácia do modelo) para o dataset de treino:\n  MSE: {}'.format(mse_grid)
      + '\n RMSE: {}'.format(rmse_grid)
      + '\nScore: {:2.3f}'.format(score_grid * 100) + '%')
```

```
Resultados do erro médio e score (acurácia do modelo) para o dataset de treino:
  MSE: 8.124078531074593e-05
 RMSE: 0.009013367035173145
Score: 90.243%
```

Figura 30. Realizando as previsões e calculando o score.

6. Apresentação dos Resultados

O canvas abaixo foi utilizado para orientar as etapas a serem seguidas para que fosse possível alcançar o objetivo do trabalho.

Title: Modelo de Machine Learning para Predição de Correntes de Fuga em Isoladores de Distribuição Classe 25kV

1 Problem Statement What problem are you trying to solve? What larger issues do the problem address? - Eliminar as quedas na distribuição de energia elétrica causadas por falhas nos isoladores elétricos. - Problemas nestes equipamentos causam rompimento na distribuição de energia elétrica.	**2 Outcomes/Predictions** What prediction(s) are you trying to make? Identify applicable predictor (X) and/or target (y) variables. - Antecipar o momento em que os isoladores podem falhar, através da previsão das correntes de fuga. - Analisando os fatores ambientais podemos predizer a corrente de fuga. - Variável alvo: corrente de fuga. - Variáveis preditoras: umidade, temperatura, pressão, ângulo do vento, velocidade do vento, quantidade de chuva.	**3 Data Acquisition** Where are you sourcing your data from? Is there enough data? Can you work with it? - Dados coletados em estação de monitoramento de sistemas isolantes (EMSI), na Praia Brava em Itajaí/SC. - Os dados disponibilizados são do período de 01 a 28/02 de 2010. - O dataset possui 7 atributos com 40.136 registros. - O dataset foi disponibilizado através de arquivo CSV.
4 Modeling What models are appropriate to use given your outcomes? - Os modelos de regressão são os ideais para este tipo de problema. - Três algoritmos serão testados: 1. Regressão Linear. 2. Árvore de Decisão. 3. Random Forest. - Na Regressão Linear serão criados diversos modelos de acordo com critérios de correlação. - Para o algoritmo de Random Forest serão aplicados ajustes nos hiperparâmetros.	**5 Model Evaluation** How can you evaluate your model's performance? - Para cada modelo é calculado um score, permitindo uma comparação através da acurácia dos modelos. - O score calculado é o coeficiente R^2 da predição.	**6 Data Preparation** What do you need to do to your data in order to run your model and achieve your outcomes? - A verificação e tratamento de campos nulos e vazios é necessária para que não hajam inconsistências. - O tratamento dos outliers pode ser necessário para garantir a qualidade do modelo.

Figura 31. Data Science Workflow Canvas [2].

O gráfico abaixo nos mostra uma comparação da acurácia dos algoritmos testados.

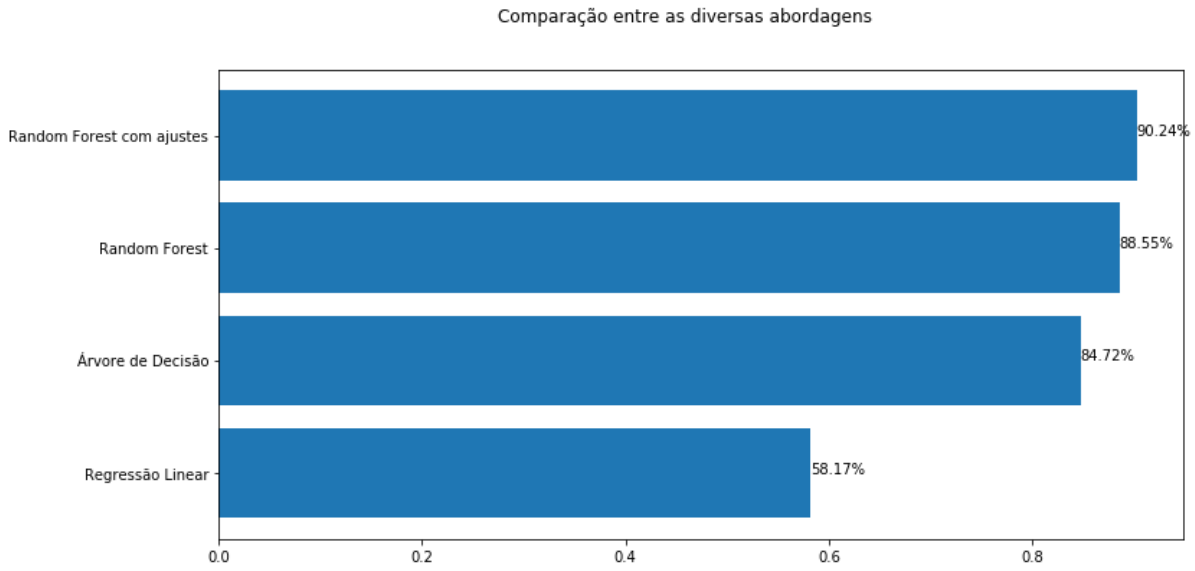

Figura 32. Comparação da acurácia das diversas abordagens.

Dentre os algoritmos de *machine learning* testados, o algoritmo de *Random Forest* com os parâmetros ajustados foi o que trouxe uma maior acurácia para os dados disponíveis.

A imagem abaixo mostra uma comparação entre o valor atual e o valor previsto pelo modelo dos dez primeiros registros.

	Atual	Previsto
4943	0.180253	0.175007
30083	0.201466	0.202482
25353	0.177424	0.177740
26959	0.183708	0.184010
18264	0.181854	0.179807
19244	0.164301	0.168664
30002	0.197662	0.199991
39360	0.160629	0.165408
2265	0.174859	0.175537
34139	0.129026	0.137633

Figura 33. Comparação dos dados atuais e previstos.

O gráfico abaixo traz uma comparação dos cem primeiros registros, onde podemos verificar a eficiência de nosso modelo.

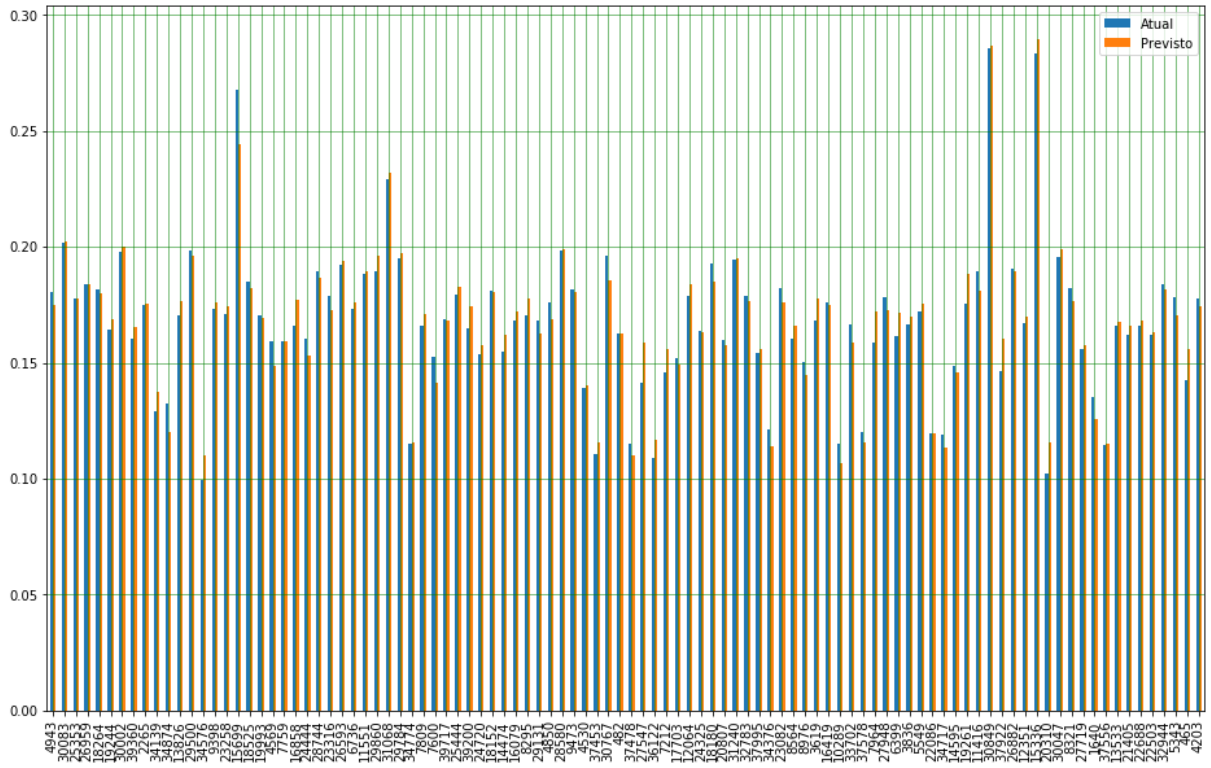

Figura 34. Comparação gráfica dos dados atuais e previstos.

Com uma taxa de acerto do 90,09%, o algoritmo de *Random Forest* com ajustes nos hiperparâmetros foi o que demonstrou os melhores resultados dentre os modelos criados

Este modelo poderá ser aplicado para outros tipos de isoladores, verificando se a acurácia é mantida, ou se novas análises e novos modelos serão necessários.

7. Links

- Código da solução: https://github.com/rhpinotti/pred_corr_fuga
- Dataset utilizado: https://github.com/rhpinotti/pred_corr_fuga/tree/master/dados
- Apresentação: https://www.youtube.com/watch?v=zsE7ZSn359Y&feature=youtu.be

8. Referências

[1] CELESC. **E-313.0031**. Especificação técnica. <https://www.celesc.com.br/arquivos/normas-tecnicas/especificacao-tecnica/e3130031.pdf>. Acesso em 15 abr 2020.

[2] VASANDANI, Jasmine. **A Data Science Workflow Canvas to Kickstart Your Projects**. <https://towardsdatascience.com/a-data-science-workflow-canvas-to-kickstart-your-projects-db62556be4d0>. Acesso em 10 mar 2020.

[3] Isolador Polimérico. <https://5.imimg.com/data5/FQ/HF/MY-10156667/polymer-pin-insulator-500x500.jpg>. Acesso em: 26 mar 2020.

[4] PINOTTI, Milton Augusto. **Modelo Matemático de Predição de Correntes de Fuga em Isoladores de Distribuição Classe 25 kV.** Dissertação de Mestrado em Engenharia Elétrica. Furb, 2015.