Relatório de Data Mining

Trabalho de Conclusão de Disciplina

Maio/2021

Professora Manoela Kohler

**Previsão de vendas para uma empresa varejista de calçados**

Participantes:

* Gustavo Dutra Martins
* Alex Marques Campos
* Fernanda Viviani de Paula

O arquivo jupyter utilizado em nosso projeto pode ser acessado no Github através do endereço:

<https://github.com/gdutramartins/bi-master/blob/main/Trabalhos-Curso/2021-05-DM/time-series.ipynb>

Anexaremos também o arquivo jupyter à entrega do trabalho.

# Introdução

Nosso grupo conseguiu uma parceria para obtenção das vendas de uma empresa varejista que vende calçados e acessórios.

Os dados fornecidos são de janeiro de 2019 até meados de abril de 2021, sendo um *dataset* analítico, ou seja, nenhum agrupamento foi realizado na geração do arquivo de exportação. O arquivo exportado tem aproximadamente 3,3 GB e está no formato *csv* e o separador utilizado foi “;”.

O objetivo do trabalho foi desenvolver modelos de *machine learning* que previssem as vendas de calçados femininos no Rio de Janeiro. O *dataset* inicialmente informado foi filtrado e utilizamos um subconjunto dos dados.

Os tratamentos realizados nos dos dados, bem como a análise dos hiperparametros nos modelos de séries temporais poderia ser utilizado em outro subconjunto dos dados, é muito simples realizar essa alteração em nosso arquivo *jupyter*, contudo nosso estudo se concentrou nas vendas de calçados femininos no Rio de Janeiro.

Os modelos de regressão utilizados em nosso projeto de estudo foram *Random Forest* e *SVM*.

# Tratamento dos Dados

O *dataset* com as vendas tem 3,3 GB e 19.559.118 registros. Os dados são analíticos, cada produto de todas as vendas realizadas desde janeiro de 2019.

Antes de mostramos as manipulações realizadas, vamos apresentar a estrutura do *dataset* de vendas. Embora outros dois *datasets* tenham sido fornecidos (produtos, lojas), somente a base de vendas foi utilizada para este trabalho.

## Leiaute do Conjunto de Dados (dataset) Original

A base de vendas continha os seguintes campos:

* NumeroBoleta – Boleta de Venda, poderia ser utilizado para agrupar itens de uma venda, mas não foi utilizado em nosso estudo.
* Loja – Loja que realizou a venda. Existe um *dataset* com o detalhamento de cada loja, mas não foi utilizado em nosso estudo.
* UF – Estado onde o produto foi vendido.
* CIDADE – Cidade onde o produto foi vendido.
* BAIRRO – Bairro onde o produto foi vendido.
* CriacaoReferencia – Data da venda.
* Produto\_Codigo – Código do produto vendido. Existe um *dataset* de produto, no entanto ele não foi utilizado em nosso estudo.
* SECAO – Primeiro nível de agrupamento dos produtos.
* GRUPO – Segundo nível de agrupamento dos produtos.
* CATEGORIA – Terceiro nível de agrupamento dos produtos.
* COR – Cor do produto vendido.
* Tamanho – Tamanho do calçado vendido.
* Quantidade – Quantidade vendida.
* PrecoVenda – Preço de venda do produto.
* PrecoTranscao – Preço aplicado a venda, quando são diferentes indica uma venda promocional (aniversariante do mês, promoção em loja, ...)

## Manipulação dos Dados para Modelos de Machine Learning

Não existiam *missing values* no *dataset* para os campos utilizados na análise da série temporal. Existiam valores não informados no *dataset*, mas eram detalhes de classificação.

O *dataset* foi filtrado da seguinte forma:

* Produtos que estejam na seção de Calçados;
* Somente produtos Femininos;
* Vendidos na cidade do Rio de Janeiro.

df\_venda\_preparado = df\_venda[(df\_venda.SECAO == 'CALCADOS') &

(df\_venda.GRUPO == 'FEMININO') &

(df\_venda.CIDADE == 'RIO DE JANEIRO')]

Analisamos inicialmente duas opções de período para agrupamento:

* Mensal – Vendas agrupadas por mês com forte sazonalidade no período de 11 meses (natal).

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

Gráfico

Descrição gerada automaticamente

* Semanal – Vendas agrupadas por semanas com correlações positivas fortes para 1,2, 5, 6 e 11 semanas anteriores.

Gráfico

Descrição gerada automaticamente

Gráfico

Descrição gerada automaticamente

Escolhemos a janela semanal que tinha correlações interessantes para o estudo, além de aumentar a quantidade de dados para treinamento do modelo. Se optássemos pelo período mensal teríamos poucos dados para treino e validação.

O projeto tem funções para montar diferentes tamanhos de janelas semanais (4, 12, 24, 36 e 52), dessa forma poderíamos comparar os resultados e encontrar a quantidade de semanas que gerasse a melhor previsão para nossa série.

def monta\_dataset\_por\_janela(dataset=None, window=12):

dataSize = len(dataset)

X = []

y = []

for i in range(window, dataSize):

X.append(dataset.iloc[i-window:i, 1])

y.append(dataset.iloc[i, 1])

X, y = np.array(X), np.array(y)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

return x\_train, x\_test, y\_train, y\_test

Com o objetivo de estudar uma abordagem diferente, montamos um vetor de entrada com a seguinte configuração:

* Penúltima e antepenúltima semanas anteriores;
* Mesma semana do ano anterior;
* Penúltima e antepenúltima semanas anteriores do ano anterior.

def monta\_dataset\_ano\_anterior(dataset):

df\_venda\_ml = dataset[dataset.AnoMesSemana >= '2020-01-02'].sort\_values(['AnoMesSemana'])

df\_venda\_agrupado\_ml = df\_venda\_ml.groupby('AnoMesSemana', as\_index=False).agg({

'Quantidade':'sum',

'AnoMesSemana1SemanaAnterior': 'max',

'AnoMesSemana2SemanaAnterior': 'max',

'AnoMesSemanaAnoAnterior': 'max',

'AnoMesSemanaAnoAnterior1Semana':'max',

'AnoMesSemanaAnoAnterior2Semana':'max'})

df\_venda\_ml\_final = pd.merge(df\_venda\_agrupado\_ml, df\_venda\_historico\_agrupado, left\_on="AnoMesSemanaAnoAnterior", right\_on="AnoMesSemana", suffixes=("","\_AnoAnterior"))

df\_venda\_ml\_final = pd.merge(df\_venda\_ml\_final, df\_venda\_historico\_agrupado, left\_on="AnoMesSemanaAnoAnterior1Semana", right\_on="AnoMesSemana", suffixes=("","\_AnoAnterior1Semana"))

df\_venda\_ml\_final = pd.merge(df\_venda\_ml\_final, df\_venda\_historico\_agrupado, left\_on="AnoMesSemanaAnoAnterior2Semana", right\_on="AnoMesSemana", suffixes=("","\_AnoAnterior2Semana"))

df\_venda\_ml\_final = pd.merge(df\_venda\_ml\_final, df\_venda\_historico\_agrupado, left\_on="AnoMesSemana1SemanaAnterior", right\_on="AnoMesSemana", suffixes=("","\_1SemanaAnterior"))

df\_venda\_ml\_final = pd.merge(df\_venda\_ml\_final, df\_venda\_historico\_agrupado, left\_on="AnoMesSemana2SemanaAnterior", right\_on="AnoMesSemana", suffixes=("","\_2SemanaAnterior"))

df\_venda\_ml\_final = df\_venda\_ml\_final[['AnoMesSemana','Quantidade\_1SemanaAnterior','Quantidade\_2SemanaAnterior','Quantidade\_AnoAnterior','Quantidade\_AnoAnterior1Semana','Quantidade\_AnoAnterior2Semana','Quantidade']]

np\_dataset = df\_venda\_ml\_final.to\_numpy()

X = np\_dataset[:,1:-1]

y = np\_dataset[:,-1]

np.random.seed(0)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

return x\_train, x\_test, y\_train, y\_test

Nossos treinamentos e validações foram realizados com as seguintes massas de dados:

* Últimas 4 semanas.
* Últimas 12 semanas.
* Últimas 24 semanas.
* Últimas 36 semanas.
* Últimas 52 semanas.
* Mesma semana do ano anterior e 2 últimas semanas desse ano e do ano anterior.

## Análise Estatística

Realizamos análises estatísticas para encontrarmos correlações entre os períodos de janelas e analisarmos os gráficos de previsão de alguns modelos estatísticos conhecidos, como média móvel, amortecimento exponencial e Arima.

No arquivo *jupyter* podem ser encontrados os estudos realizados para o período mensal, contudo no relatório colocaremos os gráficos semanais, já que foi o período escolhido pelo grupo.

### Média Móvel

Janelas de 2, 5 e 11 semanas

Gráfico, Histograma

Descrição gerada automaticamente

Interface gráfica do usuário, Gráfico, Histograma

Descrição gerada automaticamente

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

### Amortecimento Exponencial

Gráfico, Histograma

Descrição gerada automaticamente

### Arima

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

# Teste com Modelos de Machine Learning

Após a preparação dos dados, treinamos e validamos os resultados com *Random Forest e SVM*, buscando hiperparâmetros melhores com *GridSearch.*

**def** train\_predict\_with\_model(X\_train, X\_test, y\_train, y\_test, model):

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

calcula\_metricas\_erro(y\_pred=y\_pred, y\_test=y\_test, number\_features=X\_test.shape[1])

mostra\_grafico\_previsao(y\_pred=y\_pred, y\_test=y\_test)

*# imprimimos os parametros usados no modelo, caso tenha sido usado o GridSearchCV*

*# para determinar um modelo.*

**try**:

print(model.best\_params\_)

**except** **AttributeError**:

**pass**

## Random Forest

**def create\_tuned\_random\_forest\_regressor():**

# veja os parâmetros em:

# https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

tuned\_parameters = {

'n\_estimators': [1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110],

'criterion': ['mse', 'mae'],

'max\_depth': [None, 1, 2, 4, 6, 8],

'min\_samples\_leaf': [x for x in range(1,12,2)], # de 1 a 11 de 2 em 2

'random\_state': [0], # para facilitar a reprodução dos resultados observados

'n\_jobs': [-1] # usamos todos os processadores lógicos disponíveis no treino e na predição do modelo

}

# com o RandomForestRegressor, não podemos usar o scoring do GridSearchCV

# tanto o treinamento de cada árvore como a busca do melhor modelo são paralelizados

return GridSearchCV(RandomForestRegressor(), tuned\_parameters, verbose=1, n\_jobs=-1)

**def train\_predict\_random\_forest(X\_train, X\_test, y\_train, y\_test, best\_model=True):**

if best\_model:

regressor = create\_tuned\_random\_forest\_regressor()

else:

regressor = RandomForestRegressor(n\_estimators = 10, random\_state=0)

train\_predict\_with\_model(X\_train, X\_test, y\_train, y\_test, regressor)

## Support Vector Machine (SVM)

**def** create\_tuned\_svm():

*# veja os parâmetros em:*

*# https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html*

tuned\_parameters = {

'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],

'tol': [1e-3], *# default*

'C': list(np.arange(1.0, 18.0, 5.0)), *# C de 1 a 17 de 5 em 5*

'epsilon': [0.1] *# default*

}

*# a busca é paralelizada para todos os processadores disponíveis*

**return** GridSearchCV(svm.SVR(), tuned\_parameters, verbose=1, n\_jobs=-1)

**def** train\_predict\_svm(X\_train, X\_test, y\_train, y\_test, best\_model=**True**):

*# veja os parâmetros em:*

*# https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html*

**if** best\_model:

regressor = create\_tuned\_svm()

**else**:

regressor = svm.SVR(kernel='linear', C=15.0)

train\_predict\_with\_model(X\_train, X\_test, y\_train, y\_test, regressor)

Infelizmente, os hiper-parâmetros de busca utilizados para o SVM tiveram que ser bastante tolhidos, em virtude do longo tempo para treinar cada modelo durante a etapa de busca. Apesar dos grandes ganhos ao utilizar uma busca paralelizada, esse tempo era muito significativo – alguns modelos demoraram várias horas para serem treinados.

### Treinamento e Predição

O horizonte utilizado para a predição é de um valor apenas, mas o processo pode ser iterado em uma nova série que componha o valor previsto anteriormente, sendo o erro observado amplificado nessa etapa.

Para o estudo de caso, elaboramos o código abaixo, que gera os *datasets* de cada um dos cenários e faz o treinamento e predição com os algoritmos. Para cada cenário e cada algoritmo selecionado, são treinados dois modelos: um modelo com parâmetros “hard-coded” e um modelo que é selecionado a partir de um conjunto de restrições de hiper-parâmetros fornecidos ao GridSearchCV.

*# funções geradoras dos conjuntos de dados de cada cenário que iremos analisar*

f\_gerar\_ano\_anterior\_e\_corrente = **lambda**: monta\_dataset\_ano\_anterior(dataset=df\_venda\_preparado)

f\_gerar\_4\_semanas = **lambda**: monta\_dataset\_por\_janela(dataset=df\_venda\_historico\_agrupado, window=4)

f\_gerar\_12\_semanas = **lambda**: monta\_dataset\_por\_janela(dataset=df\_venda\_historico\_agrupado, window=12)

f\_gerar\_24\_semanas = **lambda**: monta\_dataset\_por\_janela(dataset=df\_venda\_historico\_agrupado, window=24)

f\_gerar\_36\_semanas = **lambda**: monta\_dataset\_por\_janela(dataset=df\_venda\_historico\_agrupado, window=36)

f\_gerar\_52\_semanas = **lambda**: monta\_dataset\_por\_janela(dataset=df\_venda\_historico\_agrupado, window=52)

*# um iterável ordenado de dicionários, contendo os nomes dos cenários e um lambda para gerar*

*# o conjunto de dados (evitamos incluir os dados no dicionário para não aumentar o footprint*

*# de memória)*

cenarios\_analise\_ml = (

{'nome':'ano anterior e corrente', 'f\_gerar\_dataset': f\_gerar\_ano\_anterior\_e\_corrente},

{'nome':'janela de 4 semanas', 'f\_gerar\_dataset': f\_gerar\_4\_semanas},

{'nome':'janela de 12 semanas', 'f\_gerar\_dataset': f\_gerar\_12\_semanas},

{'nome':'janela de 24 semanas', 'f\_gerar\_dataset': f\_gerar\_24\_semanas},

{'nome':'janela de 36 semanas', 'f\_gerar\_dataset': f\_gerar\_36\_semanas},

{'nome':'janela de 52 semanas', 'f\_gerar\_dataset': f\_gerar\_52\_semanas}

)

algoritmos\_ml = (

{'nome': 'Random Forest', 'f\_train\_predict': train\_predict\_random\_forest},

{'nome': 'SVM' , 'f\_train\_predict': train\_predict\_svm}

)

*# percorremos os cenários, treinando os algoritmos de IA e fazendo previsões nos dados*

*# cenários x algoritmos (com e sem otimização de hiper-parâmetros)*

**for** idx, cenario **in** enumerate(cenarios\_analise\_ml):

nome\_cenario = cenario['nome']

f\_gerar\_dataset = cenario['f\_gerar\_dataset']

X\_train, X\_test, y\_train, y\_test = f\_gerar\_dataset() *# aqui, chamamos a função lambda para obter os dados do cenário*

print(f'Cenário **{**idx + 1**}**: **{**nome\_cenario**}**' + os.linesep)

**for** alg **in** algoritmos\_ml:

nome\_alg = alg['nome']

f\_train\_predict = alg['f\_train\_predict']

**for** best\_model **in** (**False**, **True**):

s\_best\_model = 'melhores' **if** best\_model **else** 'padrão'

print(f'Algoritmo: **{**nome\_alg**}**')

print(f'Parâmetros do modelo: **{**s\_best\_model**}**.' + os.linesep)

%time f\_train\_predict(X\_train, X\_test, y\_train, y\_test, best\_model)

print()

print(('-' \* 50) + os.linesep)

Analisaremos, abaixo, a saída do código para os cenários com dados semanais (cenário 2 em diante).

### Cenário: janela de 4 semanas

Algoritmo: Random Forest

Parâmetros do modelo: padrão.

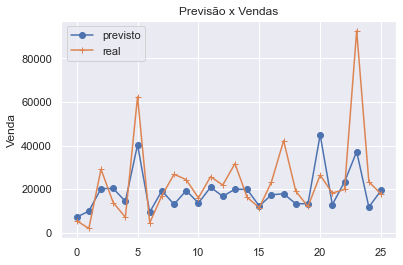
RMSE: 14461.610604379179

MSE: 209138181.27269232

MAPE: 50.85759481212 %

R2 Score: -1.5912754856166176

R2 Ajustado: -2.084851768591211



Wall time: 235 ms

Algoritmo: Random Forest

Parâmetros do modelo: melhores.

Fitting 5 folds for each of 864 candidates, totalling 4320 fits

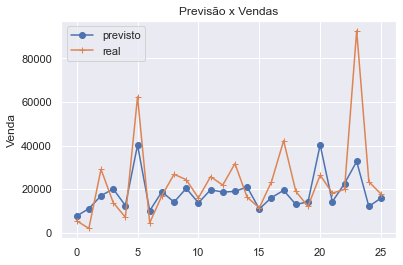
RMSE: 14941.055586510687

MSE: 223235142.0392022

MAPE: 52.50056069268344 %

R2 Score: -2.4416063454869903

R2 Ajustado: -3.0971504112940362



{'criterion': 'mse', 'max\_depth': 6, 'min\_samples\_leaf': 1, 'n\_estimators': 10, 'n\_jobs': -1, 'random\_state': 0}

Wall time: 4min 37s

Algoritmo: SVM

Parâmetros do modelo: padrão.

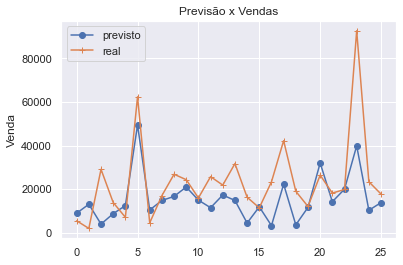
RMSE: 15012.720179550495

MSE: 225381767.18948263

MAPE: 63.35385143842193 %

R2 Score: -1.0389852552682015

R2 Ajustado: -1.4273633991288115



Wall time: 1min 48s

Algoritmo: SVM

Parâmetros do modelo: melhores.

Fitting 5 folds for each of 16 candidates, totalling 80 fits

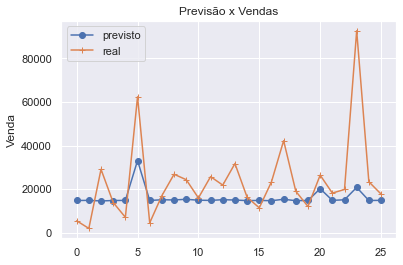
RMSE: 17889.03989027022

MSE: 320017748.1956792

MAPE: 71.75587952640521 %

R2 Score: -21.98891951613609

R2 Ajustado: -26.36776132873344



{'C': 6.0, 'epsilon': 0.1, 'kernel': 'poly', 'tol': 0.001}

Wall time: 17min 36s

### Cenário: janela de 12 semanas

Algoritmo: Random Forest

Parâmetros do modelo: padrão.

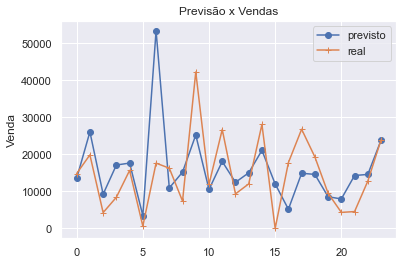
RMSE: 10272.769508186842

MSE: 105529793.36833334

MAPE: 10082.726909147641 %

R2 Score: -0.14219128048929042

R2 Ajustado: -1.388218131932153



Wall time: 246 ms

Algoritmo: Random Forest

Parâmetros do modelo: melhores.

Fitting 5 folds for each of 864 candidates, totalling 4320 fits

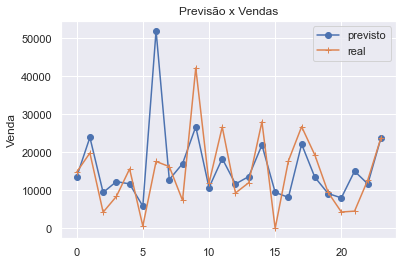
RMSE: 9380.44790687858

MSE: 87992802.93366273

MAPE: 8003.573658450593 %

R2 Score: -0.003480877958258688

R2 Ajustado: -1.0981872902763588



{'criterion': 'mse', 'max\_depth': 8, 'min\_samples\_leaf': 3, 'n\_estimators': 110, 'n\_jobs': -1, 'random\_state': 0}

Wall time: 5min 12s

Algoritmo: SVM

Parâmetros do modelo: padrão.

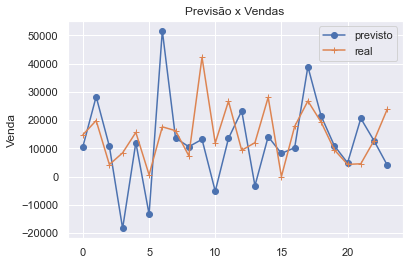
RMSE: 14566.898393059908

MSE: 212194528.7937313

MAPE: 7021.5222114811995 %

R2 Score: 0.0004402651522662504

R2 Ajustado: -1.0899885364998068



Wall time: 5min 12s

Algoritmo: SVM

Parâmetros do modelo: melhores.

Fitting 5 folds for each of 16 candidates, totalling 80 fits

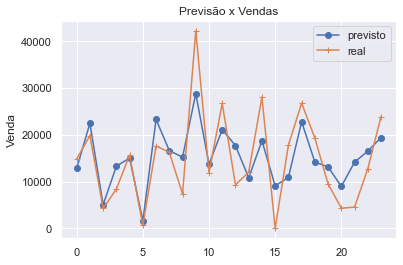
RMSE: 5908.1461446024

MSE: 34906190.86598021

MAPE: 7630.906575506338 %

R2 Score: 0.01997777015644131

R2 Ajustado: -1.0491373896728953



{'C': 1.0, 'epsilon': 0.1, 'kernel': 'linear', 'tol': 0.001}

Wall time: 1h 6min 22s

### Cenário: janela de 24 semanas

Algoritmo: Random Forest

Parâmetros do modelo: padrão.

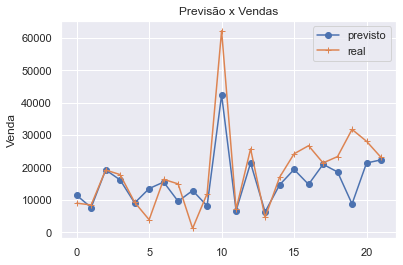
RMSE: 8171.667795749587

MSE: 66776154.564090915

MAPE: 76.04277639402994 %

R2 Score: -0.09604318401723155

R2 Ajustado: 8.672302288120623



Wall time: 241 ms

Algoritmo: Random Forest

Parâmetros do modelo: melhores.

Fitting 5 folds for each of 864 candidates, totalling 4320 fits

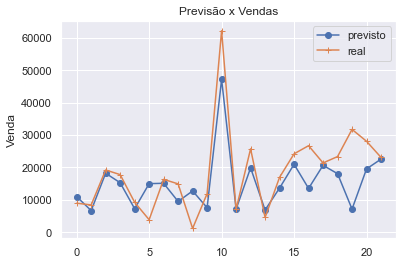
RMSE: 8243.90624286915

MSE: 67961990.14121695

MAPE: 80.38684687376285 %

R2 Score: 0.0969403086718178

R2 Ajustado: 7.3214178392972755



{'criterion': 'mse', 'max\_depth': 6, 'min\_samples\_leaf': 1, 'n\_estimators': 10, 'n\_jobs': -1, 'random\_state': 0}

Wall time: 4min 53s

Algoritmo: SVM

Parâmetros do modelo: padrão.

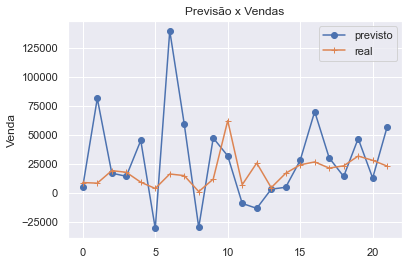
RMSE: 39146.01806121464

MSE: 1532410730.0489428

MAPE: 330.13225196406586 %

R2 Score: -0.04889082193484273

R2 Ajustado: 8.3422357535439



Wall time: 56min 34s

Algoritmo: SVM

Parâmetros do modelo: melhores.

Fitting 5 folds for each of 16 candidates, totalling 80 fits

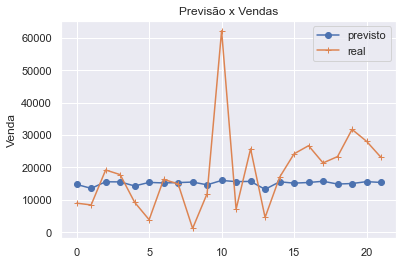
RMSE: 12881.25320153015

MSE: 165926684.04193074

MAPE: 112.89092237083729 %

R2 Score: -362.1199876673644

R2 Ajustado: 2542.8399136715507



{'C': 16.0, 'epsilon': 0.1, 'kernel': 'poly', 'tol': 0.001}

Wall time: 1h 51min 7s

### Cenário: janela de 36 semanas

Algoritmo: Random Forest

Parâmetros do modelo: padrão.

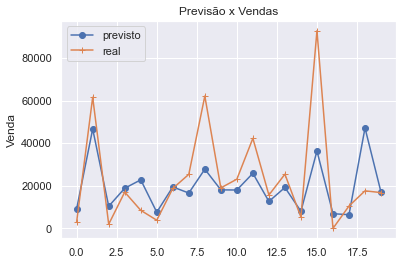
RMSE: 17704.300066706957

MSE: 313442240.852

MAPE: 17326.78812619131 %

R2 Score: -1.2735579587906196

R2 Ajustado: 3.541035365707163



Wall time: 294 ms

Algoritmo: Random Forest

Parâmetros do modelo: melhores.

Fitting 5 folds for each of 864 candidates, totalling 4320 fits

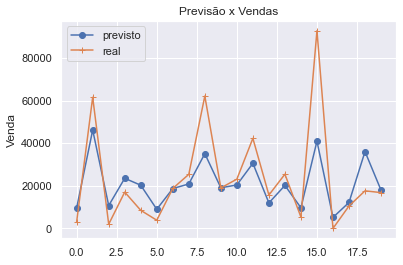
RMSE: 15113.466179371972

MSE: 228416859.95502043

MAPE: 12957.894514834075 %

R2 Score: -0.8348839994868873

R2 Ajustado: 3.050752705308874



{'criterion': 'mse', 'max\_depth': None, 'min\_samples\_leaf': 1, 'n\_estimators': 70, 'n\_jobs': -1, 'random\_state': 0}

Wall time: 5min 48s

Algoritmo: SVM

Parâmetros do modelo: padrão.

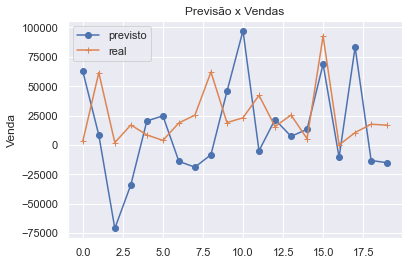
RMSE: 44540.980945571566

MSE: 1983898983.5937696

MAPE: 26982.848993099673 %

R2 Score: -0.2091253728305409

R2 Ajustado: 2.3513754166929575



Wall time: 25min

Algoritmo: SVM

Parâmetros do modelo: melhores.

Fitting 5 folds for each of 16 candidates, totalling 80 fits

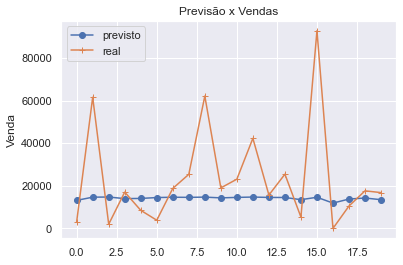
RMSE: 24808.690755232623

MSE: 615471136.9887645

MAPE: 29809.088191939696 %

R2 Score: -1281.5984223411028

R2 Ajustado: 1434.4923543812326



{'C': 16.0, 'epsilon': 0.1, 'kernel': 'poly', 'tol': 0.001}

Wall time: 5h 33min 41s

### Cenário: janela de 52 semanas

Algoritmo: Random Forest

Parâmetros do modelo: padrão.

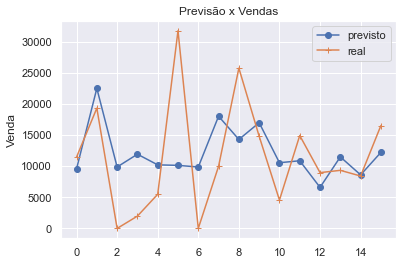
RMSE: 8209.14887861403

MSE: 67390125.31125

MAPE: 43212.80629533585 %

R2 Score: -3.4505137155356946

R2 Ajustado: 2.8042623171090657



Wall time: 269 ms

Algoritmo: Random Forest

Parâmetros do modelo: melhores.

Fitting 5 folds for each of 864 candidates, totalling 4320 fits

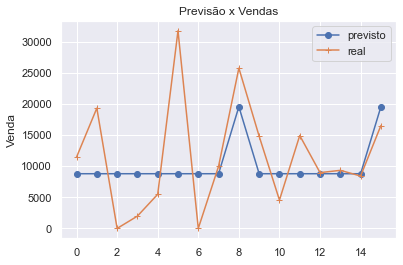
RMSE: 7896.978471445279

MSE: 62362268.97847022

MAPE: 38471.4680282622 %

R2 Score: -3.9272066669551418

R2 Ajustado: 2.9975162163331657



{'criterion': 'mse', 'max\_depth': 2, 'min\_samples\_leaf': 5, 'n\_estimators': 1, 'n\_jobs': -1, 'random\_state': 0}

Wall time: 5min 10s

Algoritmo: SVM

Parâmetros do modelo: padrão.

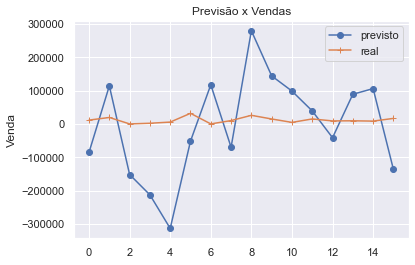
RMSE: 147549.69309644034

MSE: 21770911932.853737

MAPE: 622881.8929867689 %

R2 Score: 0.02455871053318015

R2 Ajustado: 1.3954491714054675



Wall time: 29min 47s

Algoritmo: SVM

Parâmetros do modelo: melhores.

Fitting 5 folds for each of 16 candidates, totalling 80 fits

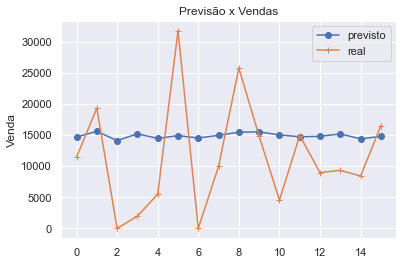
RMSE: 9081.967953085337

MSE: 82482141.90086909

MAPE: 62312.62685466279 %

R2 Score: -491.72101928153745

R2 Ajustado: 200.75176457359626



{'C': 16.0, 'epsilon': 0.1, 'kernel': 'poly', 'tol': 0.001}

# Conclusão

Para comparar o desempenho de cada algoritmo, escolhemos a métrica MAPE. A Tabela 1 - Comparativo de performance dos Algoritmos, abaixo, contém a métrica de cada par cenário x algoritmo.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algoritmo** | **MAPE (%)** | | | | |
| **4 semanas** | **12 semanas** | **24 semanas** | **36 semanas** | **52 semanas** |
| Random Forest (padrão) | 50,86 | 10082,73 | 76,04 | 17326,79 | 43212,81 |
| Random Forest (melhor) | 52,50 | 8003,57 | 80,39 | 12957,89 | 38471,47 |
| SVM (padrão) | 63,35 | 7021,52 | 330,13 | 26982,85 | 622881,89 |
| SVM (melhor) | 71,76 | 7630,91 | 112,89 | 29809,09 | 62312,63 |

Tabela 1 - Comparativo de performance dos Algoritmos

# Observamos, nos dados, que o Random Forest teve melhor desempenho do que o SVM com os parâmetros selecionados. Também observamos que o tamanho da janela utilizada influenciou muito no erro obtido e no tempo de processamento.

# Outra questão observada é que nem sempre o menor MAPE indicava um gráfico mais próximo da curva original para todos os pontos. Às vezes um *outlier* na predição, muito distante dos dados originais, acabava aumentando o MAPE de um algoritmo que havia produzido uma curva preditiva mais ajustada para a maioria dos outros pontos do conjunto de dados analisado.

A Tabela 2 - Comparação entre MAPE e fit visual, abaixo, contém, para cada um dos cenários, o algoritmo com menor MAPE e o algoritmo cuja curva preditiva avaliamos, visualmente, estar mais aderente aos dados originais. Também observamos nesses dados que o Random Forest apresentou melhores resultados do que o SVM.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Condição** | **4 semanas** | **12 semanas** | **24 semanas** | **36 semanas** | **52 semanas** |
| **Menor MAPE** | Random Forest (padrão) | SVM (padrão) | Random Forest (padrão) | Random Forest (melhor) | Random Forest (melhor) |
| **Gráfico mais ajustado aos dados** | Random Forest (melhor) | Random Forest (melhor) | Random Forest (padrão) | Random Forest (melhor) | Random Forest (padrão) |

Tabela 2 - Comparação entre MAPE e fit visual

Com base nos dados obtidos, avaliamos que a melhor opção para esse problema seria trabalhar com uma janela de 4 semanas para os dados, pois foi onde tivemos os melhores resultados (menores valores de MAPE) e os menores tempos de treinamento dos algoritmos. Também selecionamos o RandomForestRegressor como a melhor das opções analisadas no estudo.

# Considerações Finais

O trabalho foi muito interessante: aprendemos bastante sobre séries temporais, sem dúvida um tema desafiador. A utilização de uma base real de dados trouxe mais desafios ao trabalho - toda a análise que realizamos antes do início do projeto foi muito interessante. A leitura de artigos e a revisão do material ministrado no curso acrescentaram conhecimentos importantes a nossa formação.

Redes Recorrentes (RNN, LSTM, GRU) provavelmente teriam resultados melhores que os modelos de *machine learning* que utilizamos, mas o estudo de redes neurais estava fora do escopo do nosso trabalho.

Outras oportunidades de evolução, que não exploramos por conta do tempo, seriam a análise comparativa de outros algoritmos de *machine learning* que aprendemos no curso, como o KNN, e o refino continuado dos hiper-parâmetros utilizados para os algoritmos, dado que agora identificamos a janela de tempo mais adequada para utilizar nesse problema.