



**CENTRO UNIVERSITÁRIO CHRISTUS
PÓS-GRADUAÇÃO (*LATO SENSU*) UNICHRISTUS**

**Francisco Renato Gomes - POS19100137
Diego Herbester Sancho de Oliveira - POS19100119
Roberto Denny M. Nogueira - POS19100120**

**Projeto de Machine Learning: Modelo de árvore de
decisão para análise histórica de dados na criação de
uma ferramenta preditiva e estratégica.**

PROFESSOR: FELIPE VIANA

**FORTALEZA
2020**

Relatório do projeto de ciência de dados com Machine Learning, utilizando o modelo de árvore de decisão em um conjunto de dados histórico como ferramenta preditiva e estratégica de renovação de estoque.

Este arquivo contém informações sobre o projeto que está sendo executado, nesse caso, auxiliar na tomada de decisão utilizando o modelo de árvore de decisão e um conjunto de dados histórico como ferramenta preditiva e estratégica de renovação de estoque.

1. Entendimento comercial

- NOTA: Este é um exemplo de tutorial, portanto, o escopo, o plano etc. não correspondem necessariamente a um projeto real de ciência de dados que aborda uma questão comercial específica. Em um projeto real, é provável que a seção de definição do problema, escopo, plano e pessoal seja muito mais detalhada.

Definição de problema

A manutenção do estoque de uma empresa é um processo de suma importância tendo em vista que nele consta o capital investido, e sua má administração pode causar sérios problemas na saúde financeira do negócio. Neste cenário, o analista de compra precisa analisar várias informações de forma manual para decidir quando e quais produtos comprar, e isto lhe demanda muito tempo.

A finalidade deste projeto é auxiliar na decisão de compra ou não de um determinado produto pra renovação de estoque. Para isso, a empresa dispõe de uma base histórica de mais de 10 anos com os dados de compra e venda por produto. Nesta base, features como quantidade vendida, valor unitário, custo unitário, estoque e margem de contribuição, podem auxiliar na previsão de venda e lucro do produto.

Com esse objetivo em mente, utilizaremos algoritmos de machine learning, data analysis e estatística que estão armazenados nas diversas bibliotecas do Python. Escolhemos a Scikit Learn, da qual importaremos o estimador DecisionTreeClassifier, que é um modelo de árvore de decisão. Ele é um tipo de algoritmo de aprendizagem supervisionada, muito utilizada em problemas de classificação, e será muito útil para analisar o histórico de compra e vendas dos produtos no treinamento do modelo.

Escopo

- A Idealização do projeto se deu através da análise das necessidades estratégicas da empresa, utilizando a modelagem de painéis como Business Model Canvas;
- O escopo deste exemplo é criar um modelo de aprendizado de máquina de classificação binária que resolva o problema descrito a cima;
- Utilizaremos algoritmos de machine learning, data analysis e estatística que estão armazenados nas diversas bibliotecas do Python. Escolhemos a Scikit Learn, da qual importaremos o estimador DecisionTreeClassifier, que é um modelo de árvore de decisão;
- Operacionalizamos a solução para uso do analista de compras, para que o mesmo tenha controle eficiente do estoque e possa tomar melhor decisão na hora de efetuar a compra de materiais.

Plano

Seguimos o modelo de árvore de decisão e organizamos a documentação, o código e a base de dados de forma alinhada com a estratégia da empresa. A documentação sobre o trabalho e as descobertas em cada um dos estágios do ciclo de vida está incluída abaixo. O código foi escrito em python e disponibilizamos para análise a base de dados histórica com 12 meses.

Equipe Pessoal

O projeto é executado por três cientistas de dados, cada um possui um papel fundamental para a realização do projeto, um fica responsável pela documentação, outro pela implementação e desenvolvimento do projeto e outro para testar e corrigir as falhas identificadas.

- NOTA: Em um projeto do cliente, pessoal adicional, tanto de uma equipe de ciência de dados quanto da organização do cliente, pode estar envolvido.

2. Aquisição e entendimento de dados

Dados não tratados

O conjunto de dados utilizados no projeto tem origem do sistema de gestão da empresa, contendo um histórico de mais de 10 anos. Para o desenvolvimento e testes, utilizamos uma base de dados reduzida ao período de um ano.

Os dados são extraídos diretamente do SGBD Relacional Microsoft SQL Server, através de um script e salvos em uma instância online do MySQL Server, tornando possível efetuar a conexão com os mesmos de forma remota.

O script de exportação foi elaborado para enviar apenas as informações elegidas como influenciadoras no processo decisório e estarão disponíveis por tempo indeterminado na instância do MySQL. Disponibilizamos também em formato CSV, em um volume menor para permitir testes e execução do código Python desenvolvido.

Esses dados podem ser encontrado em: <https://github.com/fcorenato/mlpreditivo>

Há um total de 16.458 instâncias (antes de qualquer filtragem), contendo dados de natureza contínua e discreta.

CARACTERÍSTICAS: código, número_do_pedido, data_emissão, mês, ano, nome_do_cliente, produto, quantidade_vendida, preço_unitário, total, custo_unitário, margem, estoque, compra.

Exploração de dados

A exploração dos dados é realizada usando as seguintes bibliotecas do Python:

- NumPy que suporta arrays e matrizes multidimensionais, possuindo uma larga coleção de funções matemáticas para trabalhar com estas estruturas.
- Pandas que é uma biblioteca de software criada para a linguagem Python para manipulação e análise de dados. Em particular, oferece estruturas e operações para manipular tabelas numéricas e séries temporais.
- Matplotlib é uma biblioteca de softwares para criação de gráficos e visualizações de dados em geral.

- Seaborn que atua em cima do matplotlib e ajuda a melhorar o visual dos gráficos, dando uma aparência mais bem acabada.

Abaixo algumas análises efetuadas:

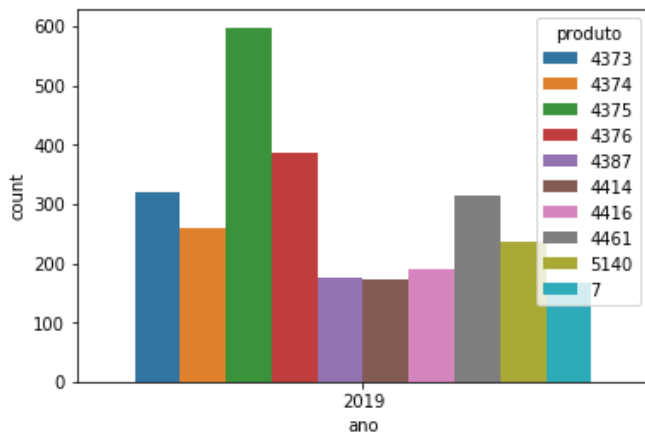


Figura 1 : Vendas por produto

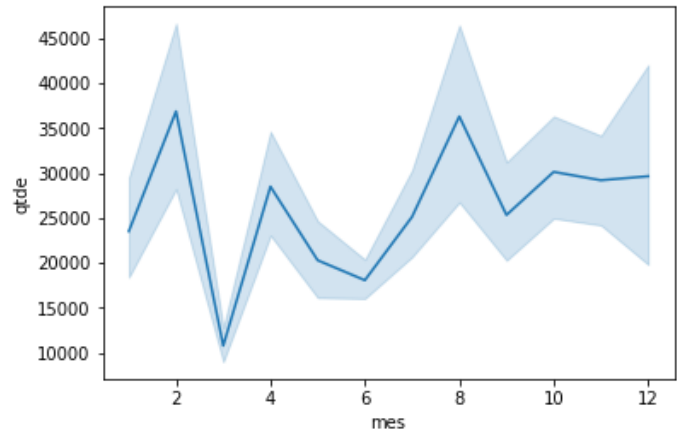


Figura 2: Vendas por mês

3. Modelagem

Engenharia de Recursos

Limpeza de dados: removendo colunas e linhas

Antes da engenharia de recursos, analisamos os dados afim de identificar colunas com valores nulos ou inválidos. Utilizando o Seaborn, criamos um gráfico de temperatura que possibilitou identificar a ausência de dados na coluna "número_do_pedido". Removemos esta coluna, já que a mesma não é relevante.

Em análises posteriores também decidimos remover as seguintes colunas: "cód.", "data_emissão", "nome_cliente".

Recursos categóricos de codificação one-hot-encoding

A feature "compra" foi codificados utilizando a técnica one-hot-encoding, que significa transformá-la em variável(coluna) e binária com o objetivo de melhorar a performance do modelo.

Salvando conjuntos de dados processados para entrada de modelagem

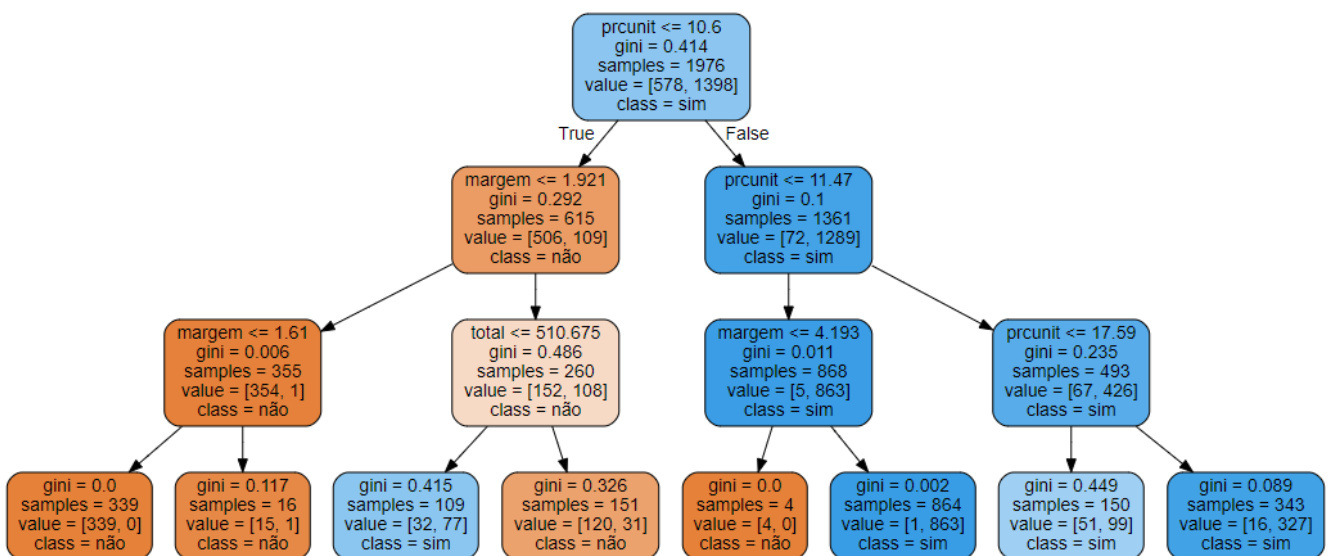
Os conjuntos de dados de treinamento e teste, foram selecionados e salvos em sub-datasets para entrada na modelagem (dados de treinamento) e avaliação ou implantação do modelo (dados de teste).

Treinamento do modelo

Criamos o modelo de teste e através da função FIT, utilizando 70% para treinamento e 30% para teste.

Cada linha de dados, representa um evento de venda, onde no final foi indicado a compra ou não do produto para renovar estoque. Então treinamos o modelo para que ele possa em eventos futuro indicar a compra ou não de um item.

Ao implementar o DecisionTreeClassifier, utilizamos o parâmetro max_depth para definir a profundidade máxima da árvore de decisão. Após alguns testes definimos max_depth = 3, que se mostrou com melhor resultado.



Abaixo segue árvore de decisão impressa:

Figura 3: Árvore de Decisão

Avaliação do modelo

A precisão do modelo com o conjunto de dados de teste se mostrou bastante eficiente, conforme os resultados abaixo, porém isto pode indicar um overfit do modelo, onde ele se mostra extremamente eficiente no treino , para

um determinado conjunto de dados, mas não tão indicado para a aplicação prática em ambiente de produção.

```
: print(classification_report(y_test,predictions))
```

	precision	recall	f1-score	support
0	0.93	0.81	0.87	248
1	0.92	0.98	0.95	599
micro avg	0.93	0.93	0.93	847
macro avg	0.93	0.89	0.91	847
weighted avg	0.93	0.93	0.93	847

Figura 4: Precisão do modelo

Plotamos a matriz de confusão, para termos uma visão gráfica, constatando o resultado obtido pelo modelo:

```
plot_confusion_matrix(cnf_matrix, classes=['Compra','NaoCompra'],
                      title='Confusion matrix, without normalization')
```

Confusion matrix, without normalization
 [[200 48]
 [14 585]]

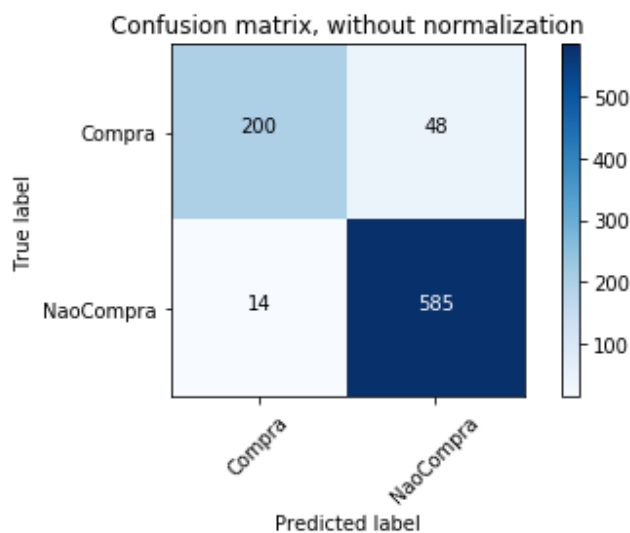


Figura 5: Matriz de Confusão

Validação do modelo

Para validar a capacidade de generalização do modelo, ou seja, avaliar o quão preciso é o modelo na prática e confirmar o seu desempenho para um novo

conjunto de dados, utilizamos a técnica de Validação Cruzada, que é amplamente empregada em problemas onde o objetivo da modelagem é a predição.

Este método consiste basicamente em dividir o conjunto total de dados em k subconjuntos mutuamente exclusivos do mesmo tamanho e, a partir daí, um subconjunto é utilizado para teste e os k-1 restantes são utilizados para estimação dos parâmetros, fazendo-se o cálculo da acurácia do modelo. Este processo é realizado k vezes alternando de forma circular o subconjunto de teste.

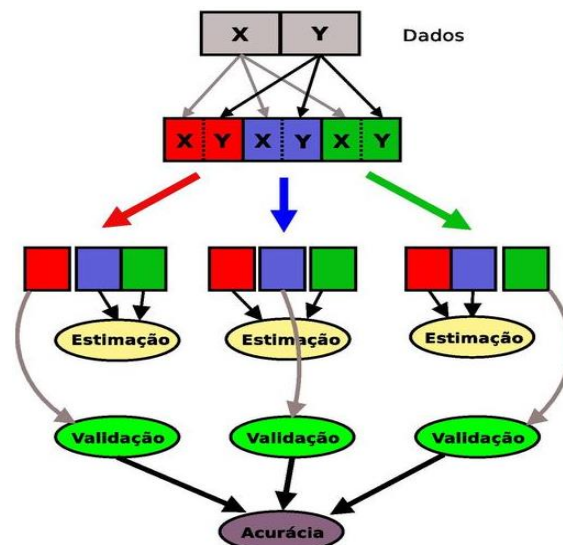


Figura 6: Validação Cruzada com método K-fold.

Antes de submeter os dados para validação, ordenamos o conjunto pela feature "compra" de forma proposital para complexar ainda mais a análise por parte do modelo.

Executando o a validação cruzada:

```
from sklearn.model_selection import cross_validate
modelo = DecisionTreeClassifier(max_depth=3)
results = cross_validate(modelo, X_train, y_train, cv = 10, return_train_score=False)
media = results['test_score'].mean()
desvio_padrao = results['test_score'].std()
print("Accuracy com cross validation, 10 = [%.2f, %.2f]" % ((media - 2 * desvio_padrao)*100,
                                                         |(media + 2 * desvio_padrao) * 100))
```

Accuracy com cross validation, 10 = [89.08, 96.95]

Figura 7: Validação Cruzada com método K-fold.

O Modelo apresentou uma precisão satisfatória variando entre 89% e 96%, mostrando que mesmo para um novo conjunto de dados ele permaneceu

preciso, a aplicação prática do modelo é viável de modo que podemos confiar nas previsões estatísticas realizadas pelo mesmo.

Ainda foi efetuado um teste supervisionado para validar a precisão do modelo através de um novo dataset real, porém controlado com o resultado previamente definido. O resultado mostrou está condizente com os obtidos nos treinos e testes anteriores.

```
#carregando dados para teste:
dt_teste = pd.read_csv('Teste-01.csv')
print("Qtde Registros: ",len(dt_teste))
```

Qtde Registros: 135

```
dt_teste.head(3)
```

```
#tratando a base de teste:
#convertendo tipos
dt_teste['mes'] = dt_teste['mes'].astype(int)
dt_teste['ano'] = dt_teste['ano'].astype(int)
dt_teste['produto'] = dt_teste['produto'].astype(int)

#excluindo colunas
dt_teste.drop(['numpedido','cod','dataemissao','nomecliente'],axis=1, inplace=True)

#one hot encoding
dt_teste["compra"] = dt_teste["compra"].replace("COMPRA", "1")
dt_teste["compra"] = dt_teste["compra"].replace("NAOCOMPRA", "0")

dt_teste.head(3)
```

```
#array contendo o resultado já conhecido do dataset de teste
testes_resultados = dt_teste['compra'].values
testes_resultados
```

```
#dropando a coluna COMPRA para enviar o dt_teste_pred para o modelo
dt_teste_pred = dt_teste
dt_teste_pred.drop(['compra'],axis=1, inplace=True)
dt_teste_pred.head(3)
```

```
# efetuando previsões
predicoes = dtc.predict(dt_teste_pred)
predicoes
```

```
# Calculando taxa de acertos
acertos = (predicoes == testes_resultados).sum()
total_registros = len(testes_resultados)
taxa_acertos = acertos/total_registros
print("taxa de acerto = ",taxa_acertos *100, "%")
```

taxa de acerto = 91.85185185185185 %

Figura 8: Teste com dados reais

Ainda foi possível identificar as correlações mais fortes entre as features:

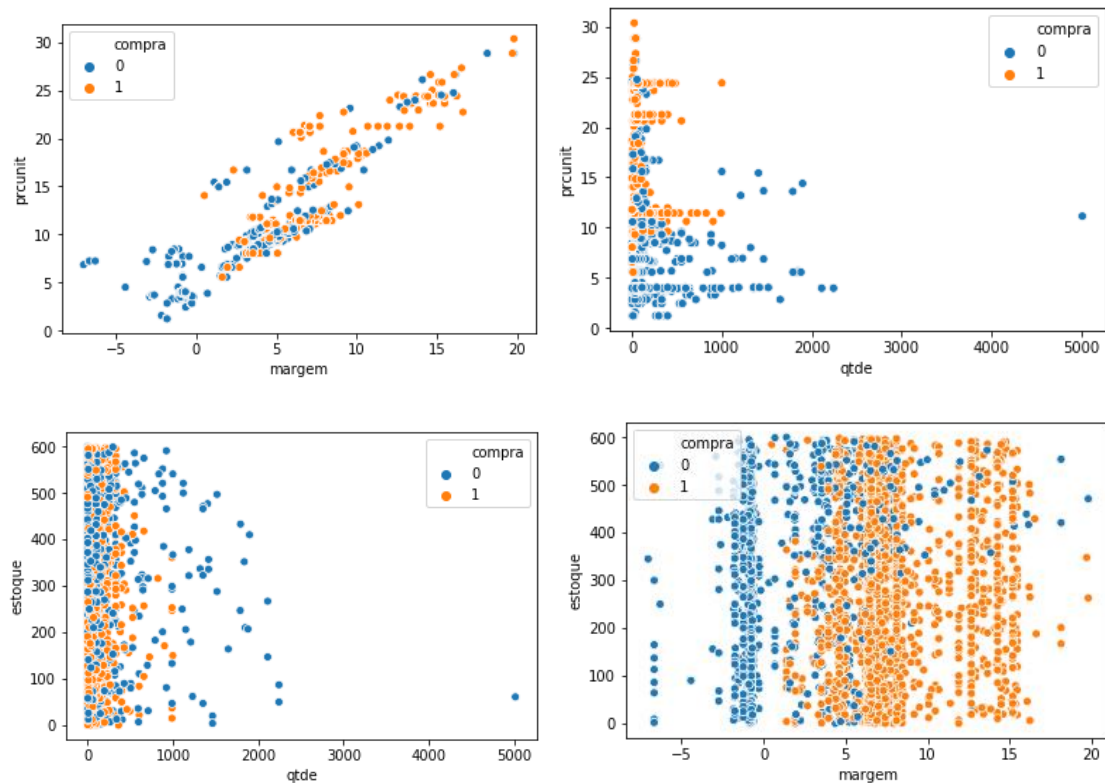


Figura 9: Correlação Features

4. Arquitetura, Ambientes e Execução de Código

Neste exemplo, executamos código apenas no ambiente de computação local utilizando o Jupyter notebook.

A utilização do modelo em ambiente de produção ocorrerá de forma automática, onde a aplicação utilizada pelo usuário submete os dados para o modelo durante a digitação de um pedido. Os dados são enviados através de uma API, e recebe como resposta do modelo um valor binário, sendo 0 = não compra e 1 = compra.

A implantação pode ser simulada através de um teste unitário que consiste na de entrada de dados conforme trecho de código abaixo:

```
#Teste unitário do modelo.

train.head(0)

mes ano produto qtde prcunit total custounit margem estoque compra
-----
# Criar um produto1 contendo informações do produto a ser analisado pelo modelo.
produto1 = [1,2019,4391,11,4.11,45.21,6.4655,-2.3555,260]
produto2 = [8,2019,7,11,4.11,45.21,8.55,4.52,160,]

#Submetendo o Produto1 para o modelo analisar se comprar ou não. [0] = Nao compra [1] = Compra
dtc.predict([produto1])

array(['0'], dtype=object)

#Submetendo o Produto2 para o modelo analisar se comprar ou não. [0] = Nao compra [1] = Compra
dtc.predict([produto2])

array(['1'], dtype=object)
```

Figura 10: Correlação Features

Para a utilização do modelo preditivo, foi criado uma API REST que receberá os parâmetros através de chamada GET, permitindo o envio dos dados e tendo como retorno a predição no formato JSON. Este formado permitirá que a aplicação que consumiu o serviço, possa receber e tratar a resposta de forma a atender sua demanda.

```
#Exportando o modelo e colocando em produção
from sklearn.externals import joblib
joblib.dump(dtc, 'decision_tree.pkl')

#Criando API para prover o serviço de classificação
from flask import Flask, jsonify, request

# [1] importo o deserializador
from sklearn.externals import joblib

# [2] Carrego a classe de predição do diretório local
dtc = joblib.load('decision_tree.pkl')

app = Flask(__name__)

@app.route('/compra_predictor')
def compra_predictor():

    # [3] Recupero as informações de uma Flor
    mes = int(request.args.get('mes'))
    ano = int(request.args.get('ano'))
    produto = int(request.args.get('produto'))
    qtde = int(request.args.get('qtde'))
    prcunit = float(request.args.get('prcunit'))
    total = float(request.args.get('total'))
    custounit = float(request.args.get('custounit'))
    margem = float(request.args.get('margem'))
    estoque = int(request.args.get('estoque'))

    event = [mes, ano, produto, qtde, prcunit, total, custounit, margem, estoque]
    target_names = ['NaoCompra', 'Compra']

    # [4] Realiza predição com base no evento
    prediction = dtc.predict([event])[0]

    res = int(prediction[0])
    result = target_names[res]

    return jsonify(result), 200

app.run()
```

Figura 11: API

A chamada deverá ser feita no seguinte molde:

https://servidor_da_aplicacao.com/compra_predictor?mes=1&ano=2019&produto=4391&qtde=11&prcunit=4.11&total=45.21&custounit=6.4655&margem=-2.3555&estoque=260

Repositório de Controle de Versão

Foi criado um repositório no GitHub para controlar o conteúdo e versionamento deste projeto.

Execução de código

Referências

- [Repositório do Projeto no GitHub](#)