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

Dataset utilizado contem os dados de vendas dos produtos da empresa X de 2016 a 2019. O produto A, B e C são importados e levam 3 meses para serem entregues pelo fornecedor na china.

Objetivo

1 - Verificar o hisorico de vendas e estoque disponivel de um produto e analisar se deve ser descontinuado ou adiquirido, tendo em vista as necesidades e estrategias comerciais da empresa.

O Dicionário de Dados

cod: Código sequencial de inserção no banco de dados

numpedido: Numero do documento de vendas

dataemissao:Data em que a venda foi efetuada

mes: Mês da venda

ano: Ano da venda

nomecliente: Nome generico do cliente

produto: Codigo do Produto

qtde: Quantidade vendida do produto

prcunit: Preço unitário do produto

total: Valor total (qtde * prcunit)

custounit: Preço unitário do produto

margem: Custo unitário do produto no dia da venda levando em consideração cambio de dolar, frete, estoque, despesas operacionais e impostos.

estoque: Estoque atual do produto no ato da venda

compra: Indica se deve ser emitido ordem de compra do produto.

Importando as bibliotecas

In [ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

Acessando os dados

In [ ]:
```
# Acessando banco de dados Mysql
# Obs: Antes será necessário instalar o mysql.connector via commando no cmd (pip install mysql-connec
```

In [ ]:
```
from datetime import date
import mysql.connector

db_connection = mysql.connector.connect(host="108.167.132.74", user="vetro057_dcroot", passwd="@dc202

data = pd.read_sql('SELECT * FROM fat19', con=db_connection)

db_connection.commit()
db_connection.close()
```

In [ ]:
```
#Exibindo os primeios 5 registros do dataset
data.head(5)
```

In [ ]:
```
data.corr()
```

In [ ]:
```
#convertendo alguns tipos de dados:

data['mes'] = data['mes'].astype(int)
data['ano'] = data['ano'].astype(int)
data['produto'] = data['produto'].astype(int)
```

Exploração dos dados

In [ ]:
```
# Informaçõe gerais.
data.info()
```

In [ ]:
```
# Analisando se existe alguma coluna com dados NULL utilizando grafico de temperatura.
```

In [ ]:
```
sns.heatmap(data.isnull(), yticklabels=False,cbar=False,cmap='viridis')
```

In [ ]:
```
#Removendo a coluna numpedido por conter alguns valores nulos e não ser um feature importante para a
data.drop(['numpedido'],axis=1, inplace=True)
data.head(5)
```

In [ ]:
```
# Vendas por Produto
```

In [ ]:
```
sns.countplot(x='ano',hue='produto',data=data)
```

In [ ]:
```
# Clintes que compraram no perído ordenado por quantidade comprada, idenficando os 10 princiais.
```

In [ ]:
```
by_cliente_produto = data[['nomecliente','produto','qtde']].groupby(["nomecliente"]).sum()
topcliente = by_cliente_produto.sort_values(by=["qtde"],ascending=False)
topcliente.head(10)
```

In [ ]:
```
# Calculando a media de venda por mes dos produtos
```

In [ ]:
```
by_produto_ano = data[['produto','mes','qtde']].groupby(["produto","mes"]).mean()
by_produto_ano.sort_values(by='produto',ascending=True)
```

In [ ]:
```
# Grafico de vendas por mes(todos os anos)
```

In [ ]:
```
sns.lineplot(x="mes", y="qtde", data=data, estimator=np.sum)
```

In [ ]:
```
# Vendas mensal(todos os anos) e Produto
```

In [ ]:
```
sns.countplot(x='mes',hue='produto',data=data)
```

Tratando o dataset para treino

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt # biblioteca de visualização utilizada pelo pandas e pelo seaborn
import seaborn as sns # biblioteca de visualização com mais opções de gráficos
 #comando necessário para que as imagens sejam exibidas aqui mesmo no notebook
%matplotlib inline
```

In [ ]:
```
df = data
df.head(5)
```

In [ ]:
```
# Eliminar os atributos que nao influencia na analise
```

In [ ]:
```
df.drop(['cod','dataemissao','nomecliente'],axis=1, inplace=True)
df.head(5)
```

In [ ]:
```
# One Hot Encode -  Subtistuir valor da coluna Compra( Compra = 1 e NaoCompra = 0)
```

In [ ]:
```
df["compra"] = df["compra"].replace("COMPRA", "1")
df["compra"] = df["compra"].replace("NAOCOMPRA", "0")
```

In [ ]:
```
#Verificando a coluna compra que antes estava nominal (COMPRA E NÃO) e agora está binária (0 e 1)
df.head(5)
```

In [ ]:
```
# Grafico mostrando a relação de decisões de compra ou não no dataset analisado.
sns.countplot(x='ano',hue='compra',data=df)
```

2. Treinar o Classificador

In [ ]:
```
import itertools
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
```

In [ ]:
```
train = df # Transferindo o dataset DF para train, para efetuar os treinos
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(train.drop(['compra'],axis=1),
                                                    train['compra'], test_size=0.30,
                                                    random_state=101)
```

In [ ]:
```
# Criar um objeto do classificador DecisionTreeClassifier()
dtc = DecisionTreeClassifier(max_depth=3)
```

In [ ]:
```
# Treinar o modelo dtc chamando a função fit
dtc.fit(X_train, y_train)
```

In [ ]:
```
# Fazer as predições passanto o X_TEST
predictions = dtc.predict(X_test)
```

In [ ]:
```
# Matrix de confusão
cnf_matrix = confusion_matrix(y_test, predictions)
cnf_matrix
```

In [ ]:
```
#Plotar matriz de confusão
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

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

In [ ]:
```
print(classification_report(y_test,predictions))
```

Reavaliando o Modelo com validação cruzada

In [ ]:
```
# validação cruzada para verificar se o modelo está em overfit
#ordenando os dados de proposito para dificultar o trabalho do modelo
```

In [ ]:
```
train2 = df.sort_values("compra", ascending=True)  # Transferindo o dataset DF para train, para efetu
train2.head(5)
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(train2.drop(['compra'],axis=1),
                                                    train2['compra'], test_size=0.30,
                                                    random_state=101)
```

In [ ]:
```
from sklearn.model_selection import cross_validate

results = cross_validate(dtc, 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))
```

In [ ]:
```
#Variando de 89% a 96%, o cross validation confirmou a eficiência do modelo.

#imprimir arvore de decisao
```

Treino final e arvovre de de decisão

In [ ]:
```
from sklearn.tree import export_graphviz
import graphviz

#treina o modelo final
dtc.fit(X_train, y_train)
features = X_train.columns
dot_data = export_graphviz(dtc, out_file=None, filled=True, rounded=True,
                class_names=["não", "sim"],
                feature_names =  features)

graph = graphviz.Source(dot_data)
```

In [ ]:
```
graph
```

In [ ]:
```
# Rank dos tributos mais relevantes
```

In [ ]:
```
df_import_features = dict(zip(train.drop(['compra'],axis=1), dtc.feature_importances_))
df_import_features = pd.DataFrame.from_dict(df_import_features, orient='index', columns = ['value'])
df_import_features = df_import_features.sort_values(['value'], ascending=False)
df_import_features
```

In [ ]:
```
# Plotar atributos mais relevantes
```

In [ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
ax = sns.scatterplot(x=train["margem"], y=train["prcunit"], hue="compra",data=train)
```

In [ ]:
```
ax = sns.scatterplot(x=train["qtde"], y=train["prcunit"], hue="compra",data=train)
```

In [ ]:
```
ax = sns.scatterplot(x=train["qtde"], y=train["estoque"], hue="compra",data=train)
```

In [ ]:
```
ax = sns.scatterplot(x=train["margem"], y=train["estoque"], hue="compra",data=train)
```

Teste unitário do modelo.

In [ ]:
```
train.head(0)
```

In [ ]:
```
# Criando produtos ficticios contendo informações dos produtos 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,]
```

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

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

Teste com dados reais

In [129]:
```
#carregando dados para teste:
dt_teste = pd.read_csv('Teste-01.csv')
print("Qtde Registros: ",len(dt_teste))
```

```
Qtde Registros:  135
```

In [ ]:
```
dt_teste.head(3)
```

In [ ]:
```
#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 enconding
dt_teste["compra"] = dt_teste["compra"].replace("COMPRA", "1")
dt_teste["compra"] = dt_teste["compra"].replace("NAOCOMPRA", "0")

dt_teste.head(3)
```

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

In [ ]:
```
#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)
```

In [ ]:
```
# efetuando predições
predicoes = dtc.predict(dt_teste_pred)
predicoes
```

In [135]:
```
# 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 %
```

Produto Final

In [ ]:
```
#Exportando o modelo e colocando em produção
from sklearn.externals import joblib
joblib.dump(dtc, 'decision_tree.pk1')
```

In [ ]:
```
#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.pk1')

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()
```

In [ ]:
```
#Testando a chamada a API
```

In [ ]:
```
http://127.0.0.1:5000/compra_predictor?mes=8&ano=2019&produto=7&qtde=11&prcunit=4.11&total=45.21&cust
```

In [ ]:
```
http://127.0.0.1:5000/compra_predictor?mes=1&ano=2019&produto=4391&qtde=11&prcunit=4.11&total=45.21&c
```