PONTIFÍCIA UNIVERSIDADE CATÓLICA – PUC-SP
FACULDADE DE ESTUDOS INTERDISCIPLINARES – FACEI
CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL

FABIO GUSTAVO GOMES VAZ
FELIPE FERNANDES ROJAS

RELATÓRIO DE PROJETO ACADÊMICO
“ANÁLISE DE PERSONALIDADE DE CONSUMO”

São Paulo
2021

FABIO GUSTAVO GOMES VAZ | RA00282997
FELIPE FERNANDES ROJAS | RA00282999

RELATÓRIO DE PROJETO ACADÊMICO
“ANÁLISE DE PERSONALIDADE DE CONSUMO”

Projeto apresentado como requisito parcial para avaliação na disciplina de Projeto Integrado: Redes Sociais e Marketing, no curso de Ciência de Dados e Inteligência Artificial – PUC-SP, sob a orientação do Profº Jefferson de Oliveira Silva.

São Paulo

2021

Sumário

1 Descrição do projeto

Na disciplina “Projeto Integrado: Redes Sociais e Marketing” requisitou-se que cada grupo escolhesse um contexto e especificasse um objetivo de negócio para uma organização. Em seguida, que obtivesse dados para o treinamento dos modelos de aprendizagem de máquina. Os dados não necessitavam ser reais, mas deveriam ser trabalhados para ser o mais realista possível. De posse dos dados, deveriam-se treinar classificadores e/ou preditores, cuidando para que o uso do modelo ajude a atingir o objetivo de negócio proposto, com a análise das métricas do modelo.

1.1 Contexto de negócio

Sabe-se que, hoje em dia, as organizações que dependem de clientes para sustentar suas atividades estão se deparando com enormes bases de dados de indivíduos que são (ou podem vir a ser) consumidores dos serviços que a empresa oferece e dezenas, centenas ou milhares de características que indicam se as pessoas são propensas a comprar um produto, a adquirir um serviço, a assinar um plano, a se fidelizar a uma atividade fornecida ou outras inúmeras formas de consumo, a depender do objetivo de negócio da empresa. Com o advento da ciência de dados, da inteligência artificial e suas vertentes, É de senso comum que um extenso volume de dados sobre âmbitos que interessam a uma corporação pode levar a análises e predições extraordinárias, capazes de conduzir a estratégias de negócios gradativamente mais inteligentes, devido ao poder de aprendizado de máquina, apta a realizar predições com maior precisão que a análise humana. Em 2013, 98% das informações armazenadas em todo o mundo já estavam sob o formato digital[1], provando a imprescindibilidade de se adequar às ações modernas baseadas em dados.

[1] MAYER-SCKOENBERGER; CUKIER, 2013

No caso deste projeto, o contexto de negócio que objetivamos atingir é o de marketing para incremento em vendas de produtos. Nota-se que, com uma grande quantidade de instâncias de indivíduos, é inviável realizar estratégias de negócio personalizadas para se adequar a cada cliente. A solução mais acessível para esse problema é a divisão desses clientes em grupos, onde os participantes seriam os mais similares possíveis entre si e a quantidade de táticas para retenção de consumidores seria reduzida e ainda eficaz.

1.2 Objetivo de negócio

Conforme a conjuntura identificada e a solução de um contratempo proposta na seção “Contexto de negócio”, a pretensão é identificar comunidades de potenciais clientes para traçar estratégias personalizadas para cada público. Ao utilizar o produto criado neste projeto, um exemplo de estratégia de negócio seria: em vez de gastar dinheiro para comercializar um novo produto para cada cliente no banco de dados da empresa, pode-se analisar qual segmento de clientes tem maior probabilidade de comprá-lo e, em seguida, comercializá-lo apenas naquela ramificação específica.

1.3 Metodologia

A base da pesquisa foi um conjunto de dados de 2240 clientes de uma empresa, ao qual foram aplicados métodos estatísticos através da linguagem de programação *Python* para analisar a personalidade consumidora do cliente, modelar predições de comportamento perante a organização e agrupar em comunidades para aprimorar as estratégias de marketing a cada público específico. Esses públicos serão definidos por meio de um algoritmo de aprendizagem de máquina não supervisionada, denominado *k-means*, e posteriormente serão treinados modelos de aprendizagem de máquina supervisionada: árvore de decisão (também conhecido como *Decision Tree*) e *SVM (Support Vector Machine)*, capazes de prever o grupo de novos indivíduos. O projeto foi implementado em um *Jupyter Notebook*.

1.3.1 Definição do *k-means*

"O algoritmo *k-means* agrupa os dados tentando separar amostras em *n* grupos de variância igual, minimizando um critério conhecido como *inércia* ou soma dos quadrados dentro do grupo. Este algoritmo requer que o número de grupos seja especificado. Ele se adapta bem a um grande número de amostras e tem sido usado em uma grande variedade de áreas de aplicação em muitos campos diferentes. O algoritmo *k-means* divide um conjunto de *n* amostras *x* em *k* aglomerados disjuntos *c*, cada um descrito pela média *μ* das amostras no grupo. Os meios são comumente chamados de "centróides" do grupo; note que eles não são, em geral, pontos de *x*, embora vivam no mesmo espaço."[2]

1.3.2 Definição da Árvore de decisão

"Uma árvore de decisão é um mapa dos possíveis resultados de uma série de escolhas relacionadas. Permite que um indivíduo ou organização compare possíveis ações com base em seus custos, probabilidades e benefícios. Podem ser usadas tanto para conduzir diálogos informais quanto para mapear um algoritmo que prevê a melhor escolha, matematicamente. Uma árvore de decisão geralmente começa com um único nó, que se divide em possíveis resultados. Cada um desses resultados leva a nós adicionais, que se ramificam em outras possibilidades. Assim, cria-se uma forma de árvore."[3]

1.3.3 Definição do *Support Vector Machine*

"Support Vector Machine (que pode ser traduzido para o português-brasileiro como Máquina de Vetores de Suporte) é um conceito para um conjunto de métodos de aprendizado de máquina supervisionado que analisam os dados e reconhecem padrões, usado para classificação e análise de regressão. O SVM padrão toma como entrada um conjunto de dados e prediz, para cada entrada dada, qual de duas possíveis

---

[2] *Sci-kit learn 1.0.1 User Guide – 2.3.2 K-means*. https://scikit-learn.org/stable/modules/clustering.html#k-means. Acesso em 3 de dezembro de 2021, às 15h.

[3] https://www.lucidchart.com/pages/pt/o-que-e-arvore-de-decisao

classes a entrada faz parte, o que faz do SVM um classificador linear binário não probabilístico. Dados um conjunto de exemplos de treinamento, cada um marcado como pertencente a uma de duas categorias, um algoritmo de treinamento do SVM constrói um modelo que atribui novos exemplos a uma categoria ou outra. Um modelo SVM é uma representação de exemplos como pontos no espaço, mapeados de maneira que os exemplos de cada categoria sejam divididos por um espaço claro que seja tão amplo quanto possível. Os novos exemplos são então mapeados no mesmo espaço e preditos como pertencentes a uma categoria baseados em qual o lado do espaço eles são colocados."[4]

1.4 Atributos dos clientes

Pessoais

ID: identificador único do cliente;
idade: idade do cliente em 2015;
educação: nível de educação do cliente;
estado civil: estado civil do cliente;
renda: renda familiar anual do cliente;
crianças em casa: número de crianças na casa do cliente;
adolescentes em casa: número de adolescentes na casa do cliente;
crianças e adolescentes: soma de crianças e adolescentes na casa do cliente;
inscrição: data de inscrição do cliente na empresa;
última compra: número de dias desde a última compra do cliente; e
reclamação: 1 se o cliente reclamou nos últimos dois anos, 0 caso contrário.

Produtos

gasto em vinhos: quantia gasta em vinhos nos últimos dois anos
gasto em frutas: quantia gasta com frutas nos últimos dois anos;
gasto em carnes: quantia gasta com carnes nos últimos dois anos;
gasto em frutas: quantia gasta em peixes nos últimos dois anos;
gasto em doces: quantia gasta em doces nos últimos dois anos;

[4] https://pt.wikipedia.org/wiki/M%C3%A1quina_de_vetores_de_suporte

gasto em ouro: quantia gasta em ouro nos últimos dois anos; e
gasto total: quantia total gasta nos produtos acima nos últimos dois anos.

Promoção

compras c/ desconto: número de compras feitas com desconto;
aceitou - 1: 1 se o cliente aceitou a oferta na 1ª campanha, 0 caso contrário;
aceitou - 2: 1 se o cliente aceitou a oferta na 2ª campanha, 0 caso contrário;
aceitou - 3: 1 se o cliente aceitou a oferta na 3ª campanha, 0 caso contrário;
aceitou - 4: 1 se o cliente aceitou a oferta na 4ª campanha, 0 caso contrário;
aceitou - 5: 1 se o cliente aceitou a oferta na 5ª campanha, 0 caso contrário; e
resposta final: 1 se o cliente aceitou a oferta na última campanha, 0 caso contrário.

Lugar

compras pela web: número de compras feitas através do site da empresa;
compras c/ catálogo: número de compras feitas usando um catálogo;
compras na loja física: número de compras feitas diretamente nas lojas;
total de compras: número total de compras feitas pelo cliente; e
visitas no site: Número de visitas ao site da empresa no último mês.

Classe predita

grupo: identificação do grupo ao qual o cliente pertence, de acordo com o modelo classificador treinado neste projeto.

2 Implementação do método

2.1 Funções de automatização

Com o intuito de tornar o código do projeto o mais automatizado possível e, consequentemente, mais legível e intuitivo para que outros cientistas de dados, programadores, professores, especialistas, entusiastas, estudantes ou qualquer outro público possa ter acesso e entender a implementação, foram criadas funções

para simplificar o uso dos métodos disponibilizados pelas bibliotecas disponíveis em *Python*. Abaixo, observa-se os nomes e as sinopses de cada uma delas:

```
def criar_dataframe(endereco, delimitador):
  """ Função que lê arquivo em .csv com a biblioteca pandas e o armazena na variável desejada.

      endereço: inserir string do local em que o arquivo está armazenado;
      delimitador: inserir string do delimitador deste arquivo .csv;
```

Figura 1: função “criar_dataframe”.

```
def traducao_atributos(dataframe):
  """ Função para traduzir os nomes dos atributos do Dataframe para o Português-brasileiro.
  Basta aplicá-la no dataset deste projeto.

      dataframe: inserir variável contendo o nome do DataFrame.
  """
```

Figura 2: função “traducao_atributos”.

```
def traducao_dados(dataframe):
  """ Função para traduzir todos os dados do Dataframe para o Português-brasileiro.
  Basta aplicá-la no dataset deste projeto.

      dataframe: inserir variável contendo o nome do DataFrame.
  """
```

Figura 3: função “traducao_dados”.

```
def traduzir(dataframe):
  """ Função que traduz o DataFrame através da composição das funções de tradução dos
  atributos e dos dados. Basta aplicá-la no dataset deste projeto.

      dataframe: inserir variável contendo o nome do DataFrame.
  """
```

Figura 4: função “traduzir”.

```
def preencher_nulos(dataframe, atributo, metodo):
  """ Função que preenche todos os valores nulos de uma coluna por algum método estatístico
  aplicado aos valores existentes naquela coluna.

      dataframe: inserir variável contendo o nome do DataFrame;
      atributo: inserir string contendo o nome da coluna do DataFrame a ter seus valores
  nulos preenchidos;
      metodo ('Média', 'Moda', 'Máximo', 'Mínimo', 'Mediana'): inserir string contendo o
  método estatístico a ser aplicado aos valores existentes para preenchimento dos valores nulos,
  conforme opções indicadas entre parênteses.
  """
```

Figura 5: função “preencher_nulos”.

```
def filtrar(dataframe, atributo, classificacao):
    """ Função para filtrar o DataFrame para mostrar apenas as instâncias com uma classificação
    única desejada em um determinado atributo.

        dataframe: inserir variável contendo o nome do DataFrame;
        atributo: inserir string contendo o nome da coluna do DataFrame a ser filtrada;
        classificacao: inserir valor único desejado dentro das classificações daquele atributo a
    ser filtrado.
    """
```

Figura 6: função “filtrar”.

```
def contar_valores(dataframe, atributo, classificacao=None):
    """ Função para contar valores de cada classificação em um determinado atributo. É possível
    contar os valores de apenas uma determinada classificação. É usado o método 'value_counts()'
    da biblioteca pandas.

        dataframe: inserir variável contendo o nome do DataFrame;
        atributo: inserir string contendo o nome da coluna do DataFrame a ser contada;
        classificacao (opcional): inserir valor único desejado dentro das classificações daquele
    atributo.
    """
```

Figura 7: função “contar_valores”.

```
def histogramar_qualitativo(dataframe, atributo, tamanho, titulo=None, cor=None, nome_do_arquivo=None):
    """ Função que realiza a plotagem de um gráfico de barras como um histograma, com base na contagem
    de ocorrências de um atributo. Recomenda-se usá-lo em atributos com valores qualitativos.

        dataframe: inserir variável contendo o nome do DataFrame;
        atributo: inserir string com o nome da coluna do DataFrame a ser plotada;
        tamanho: inserir tupla referente ao tamanho desejado do gráfico;
        título (opcional): inserir string com o nome do título desejado do gráfico;
        cor (opcional): inserir string com o nome da cor desejada no gráfico;
        nome_do_arquivo (opcional): inserir string com o nome desejado para armazenar a imagem do
    gráfico neste diretório e com o formato escolhido. Exemplo: "arquivo.png".
    """
```

Figura 8: função “histogramar_qualitativo”.

```
def histogramar_quantitativo(dataframe, atributo, tamanho, eixo=None, titulo=None, cor=None,
                             logaritmo=False, nome_do_arquivo=None):
    """ Função que desenvolve um gráfico de histograma com base na frequência de valores de um atributo
    específico em um dataframe. Recomenda-se usá-lo em atributos com valores quantitativos.

        dataframe: inserir variável contendo o nome do DataFrame;
        atributo: inserir string com o nome da coluna do DataFrame a ter a frequência de valores contada;
        tamanho: inserir tupla referente ao tamanho desejado do gráfico;
        eixo (opcional): inserir string com o nome do eixo desejado no gráfico;
        título (opcional): inserir string com o nome do título desejado do gráfico;
        cor (opcional): inserir string com o nome da cor desejada no gráfico;
        logaritmo (opcional): inserir 'True' caso deseje uma escala logarítmica. Por padrão, 'False'
    representa escala não-logarítmica;
        nome_do_arquivo (opcional): inserir string com o nome desejado para armazenar a imagem do gráfico
    neste diretório e com o formato escolhido. Exemplo: "arquivo.png".
    """
```

Figura 9: função “histogramar_quantitativo”.

```
def distribuir(dataframe, atributo, titulo=None, cor=None, nome_do_arquivo=None):
    """ Função que desenvolve um gráfico de distribuição de valores. Usa-se as bibliotecas seaborn e matplolib.

        dataframe: inserir variável contendo o nome do DataFrame;
        atributo: inserir string com o nome da coluna do DataFrame a ter os valores distribuídos;
        título (opcional): inserir string com o nome do título desejado do gráfico;
        cor (opcional): inserir string com o nome da cor desejada no gráfico;
        nome_do_arquivo (opcional): inserir string com o nome desejado para armazenar a imagem do gráfico neste
    diretório e com o formato escolhido. Exemplo: "arquivo.png".
    """
```

Figura 10: função “distribuir”.

```
def comparar_atributos(dataframe, atributo_1, atributo_2, cor=None, eixo_x=None, eixo_y=None, pontos_x=None,
                       pontos_y=None, titulo=None, nome_do_arquivo=None):
    """ Função que desenvolve um gráfico do agrupamento de dois atributos desejados em função da média destes.

        dataframe: inserir variável contendo o nome do DataFrame onde há os atributos desejados;
        atributo_1: inserir string com o nome da coluna do DataFrame a ser usada como base do agrupamento;
        atributo_2: inserir string com o nome da coluna do DataFrame a ser o parâmetro para medição do primeiro
    atributo;
        cor (opcional): inserir string com o nome da cor desejada no gráfico;
        eixo_x (opcional): inserir string com o nome do eixo das abscissas desejado no gráfico;
        eixo_y (opcional): inserir string com o nome do eixo das ordenadas desejado no gráfico;
        pontos_x (opcional): inserir lista com os pontos que deseja demonstrar no eixo das abscissas.
        pontos_y (opcional): inserir lista com os pontos que deseja demonstrar no eixo das ordenadas. Deve ser
    divisível pela quantidade de barras;
        titulo (opcional): inserir string com o nome do título desejado do gráfico;
        nome_do_arquivo (opcional): inserir string com o nome desejado para armazenar a imagem do gráfico neste
    diretório e com o formato escolhido. Exemplo: "arquivo.png".
    """
```

Figura 11: função “comparar_atributos”.

```
def soma_quadratica_erro(dataframe, grupos, conjunto_padronizado, tamanho, estilo='ggplot',
                         inicio_busca=1, fim_busca=10, nome_do_arquivo=None):
    """ Função que gera um gráfico da soma quadrática do erro, ou seja, a perda de similaridade entre
    os indivíduos conforme a quantidade de grupos de clusterização escolhida para o conjunto de dados.

        dataframe: inserir variável contendo o nome do dataframe;
        grupos: inserir string contendo o nome da coluna a qual pertence o grupo das instâncias;
        conjunto_padronizado: inserir variável contendo o conjunto de dados padronizados;
        tamanho: inserir tupla com o tamanho desejado para o gráfico;
        estilo (opcional): inserir string contendo o estilo desejado para o gráfico. Por padrão, 'ggplot';
        inicio_busca (opcional): inserir número inteiro com o início da busca do algoritmo. Por padrão, 1;
        fim_busca (opcional): inserir número inteiro com o fim da busca do algoritmo. Por padrão, 10;
        nome_do_arquivo (opcional): inserir string com o nome desejado para armazenar a imagem do gráfico
    neste diretório e com o formato escolhido. Exemplo: "arquivo.png".
    """
```

Figura 12: função “soma_quadratica_erro”.

```
def arredondar(lista, casas):
    """ Função para arredondar elementos de uma lista, conforme o número de casas decimais desejados.
        lista: inserir uma lista unidimensional apenas com floats (números não inteiros);
        casas: inserir o número de casas decimais desejadas, apenas números naturais.
    """
```

Figura 13: função “arredondar”.

```
def imprimir_ordenado(dicionario, decrescente=True):
    """ Função para imprimir um dicionário em ordem crescente ou descrescente.
        dicionario: inserir uma variável contendo um dicionário;
        descrescente: 'True' por padrão. Inserir 'False' caso queira a ordem crescente.
    """
```

Figura 14: função “imprimir_ordenado”.

```
def arvore_de_decisao(criterio, no_raiz, profundidade, min_amostras_no, min_amostras_folha=0.2):
    """ Função que cria uma árvore de decisão com parâmetros editáveis.

        instancias: inserir array contendo as instancias dos dados (sem a classe) a serem analisados;
        classes: inserir array contendo as classificações das instâncias passadas anteriormente;
        criterio: inserir string contendo o critério a ser utilizado ('gini' ou 'entropy');
        no_raiz: inserir string contendo o critério para escolha do nó raiz, entre o de melhor resultado
    ou aleatório ('best' ou 'random');
        profundidade: inserir
        min_amostras_no: inserir o número de amostras mínimas desejado para considerar um nó para divisão;
        min_amostras_folha (opcional): inserir o número de amostras mínimas no nível folha. Por padrão, 0.2.
    """
```

Figura 15: função “criar_dataframe”.

```
def matriz_de_confusao(teste, predicao):
    """ Função para criar um array da matriz de confusão com base no conjunto de teste e no conjunto
    predito. É usado método 'confusion_matrix' da biblioteca 'scikt-learn'.

        teste: inserir a lista contendo o conjunto de teste predefinido;
        predicao: inserir a lista contendo o conjunto predito pelo modelo.
    """
```

Figura 16: função “matriz_de_confusao”.

```
def dataframe_matriz(teste, predicao, legendas=None):
    """ Função que cria um DataFrame com base na matriz de confusão. Composta com a função
    'matriz_de_confusao()'.

        teste: inserir a lista contendo o conjunto de teste predefinido;
        predicao: inserir a lista contendo o conjunto predito pelo modelo;
        legendas (opcional): inserir a lista de strings contendo o nome das classificações
    (deve ter o mesmo comprimento da dimensão de classificações).
    """
```

Figura 17: função “dataframe_matriz”.

```
def mapa_de_calor(teste, predicao, legendas=None, titulo=None):
    """ Função que produz um mapa de calor da matriz de confusão com a biblioteca 'seaborn'.
    Composta com as funções 'matriz_de_confusão()' e 'dataframe_matriz()'

        teste: inserir a lista contendo o conjunto de teste predefinido;
        predicao: inserir a lista contendo o conjunto predito pelo modelo;
        legendas (opcional): inserir a lista de strings contendo o nome das classificações.
        titulo (opcional): inserir string com o nome do título desejado do gráfico.
    """
```

Figura 18: função “mapa_de_calor”.

```
def avaliacao(teste, predicao):
    """ Função que avalia o modelo preditivo com quatro métricas: f1_score, precision,
    recall e accuracy. Imprime as quatro pontuações.

        teste: inserir a lista contendo o conjunto de teste predefinido;
        predicao: inserir a lista contendo o conjunto predito pelo modelo.
    """
```

Figura 19: função “avaliacao”.

```
def validacao_cruzada(modelo, x_treino, y_treino, variacoes=20):
    """ Função que realiza a validação cruzada do modelo desejado. Retorna todos os valores obtidos,
    com destaque para a maior acurácia, a média e o desvio delas.

        modelo: inserir variável contendo o nome do modelo a ser validado;
        x_treino: inserir array contendo o conjunto de dados definidos como x_treino;
        y_treino: inserir array contendo o conjunto de dados definidos como y_treino;
        variacoes (opcional): inserir número inteiro de variações de divisão desejado. Por padrão, 20.
    """
```

Figura 20: função “validacao_cruzada”.

2.2 Tratamento e manipulação dos dados

Para que o conjunto de dados possa estar em ótimas condições de análise e modelagem, é necessário passar por um processo chamado de “tratamento de dados”, isto é, a remoção ou substituição de dados faltantes, incoerentes, redundantes ou em tipos diferentes do que realmente são, para que essas inconsistências não caracterizem um ruído dentro do projeto e a modelagem possa ser desenvolvida com a melhor performance possível.

No presente conjunto, foram identificados alguns ruídos como: idioma inglês (é um ruído caso os desenvolvedores e/ou o público-alvo da análise não possua fluência no idioma), estados civis inexistentes de acordo com a Constituição Federal Brasileira (apesar de os dados serem de indivíduos residentes nos Estados Unidos, optou-se por considerar a lei brasileira, apesar de os estados civis considerados como ruídos não estarem presentes em nenhuma das constituições supracitadas: “*YOLO*”, “*Alone*” e “*Absurd*”), cerca de 24 valores de rendas ausentes e tipo da coluna “inscrição” como *string* (texto).

Para a resolução das inconsistências, realizou-se a tradução do conjunto de dados do inglês para o português-brasileiro, a substituição dos estados civis inexistente por “solteiro(a)”, a incrementação da média da renda dos clientes onde esses valores eram ausentes e a conversão da coluna “inscrição” para o tipo *datetime*.

Ademais, manipularam-se os dados para a criação de novas colunas:

- “idade”: ano em que o conjunto de dados foi coletado subtraído pelo ano de nascimento;

- “gasto total”: soma de todas as colunas que correspondem aos gastos dos clientes;
- “total de compras”: soma de todos as colunas que correspondem aos tipos de compras realizadas;
- “crianças e adolescentes”: soma das colunas “crianças em casa” e “adolescentes em casa”; e
- “nível de renda”: divisão em quartis das rendas dos indivíduos;

2.3 Análise exploratória dos dados

Subcapítulo destinado a demonstração de gráficos que pretendem ilustrar um panorama das características dos clientes, conforme os atributos a que se tem acesso.

2.3.1 Crianças em casa

O gráfico abaixo demonstra que o fato de não ter crianças em casa corrobora para que o cliente gaste mais que o dobro dos clientes que cuidam de jovens. Fundamental ressaltar que estes valores correspondem às compras de carnes, frutas, vinhos, doces e ouro.

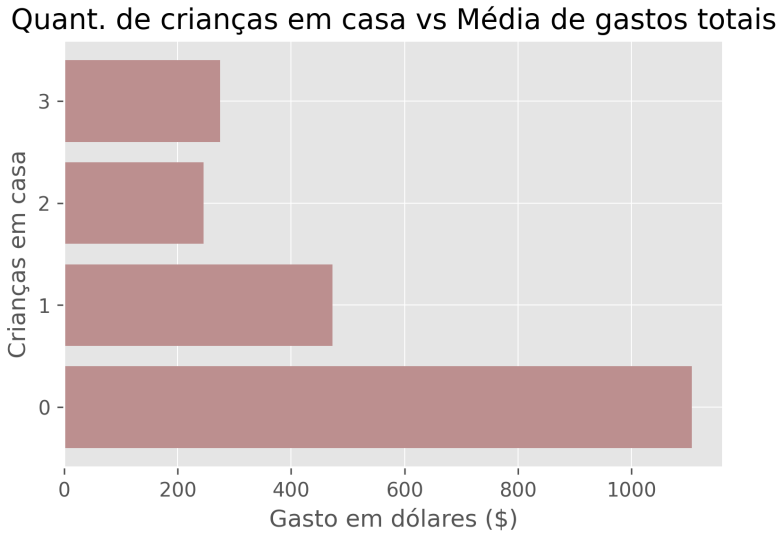

Figura 21: gráfico que mostra a média de gastos totais de acordo com a quantidade de crianças em casa.

2.3.2 Idade

O gráfico abaixo permite visualizar que as idades dos clientes variam majoritariamente entre 20 e 80 anos, apesar de haver *outliers* (dados incomuns) com mais de 100 anos. A maior parte dos indivíduos tem entre 35 e 50 anos.

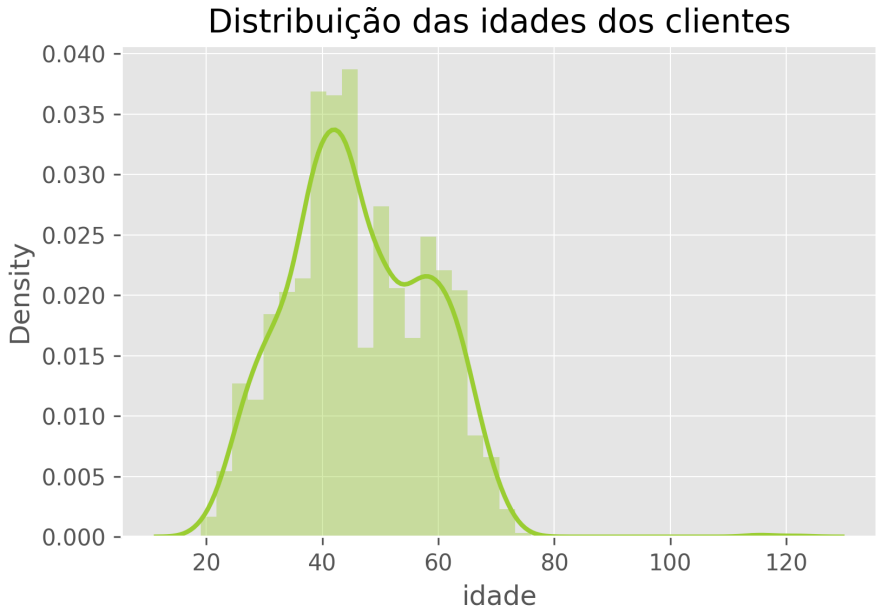

Figura 22: gráfico com a distribuição das idades dos clientes.

2.3.3 Total de compras

O gráfico abaixo ilustra que a maioria dos clientes tem certa fidelidade com a organização. A maioria fez mais de quatro compras, mas poucos fizeram mais de 35 compras.

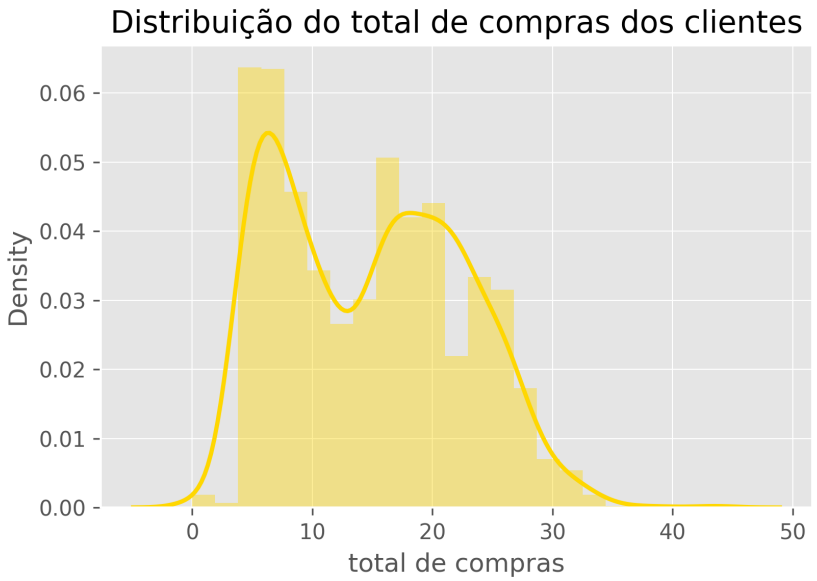

Figura 23: gráfico que ilustra a distribuição do total de compras dos clientes.

2.3.4 Estado civil

O gráfico abaixo demonstra que o estado civil tem certa influência na média de gastos totais, onde os viúvos gastaram cerca de $120 a mais do que os indivíduos em outros tipos de relacionamento.

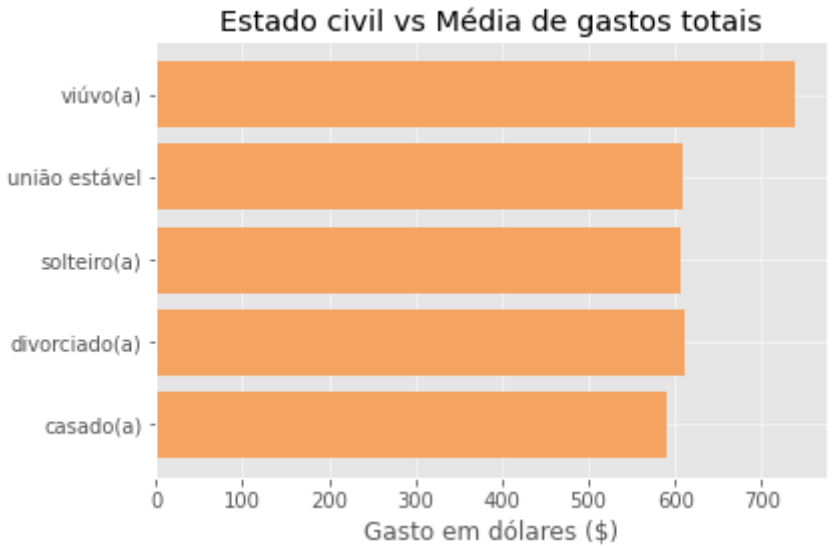

Figura 24: gráfico que mostra a média de gastos totais de acordo com o estado civil.

2.3.5 Grau de educação

O primeiro gráfico abaixo permite visualizar a quantidade de clientes em cada grau de educação dentro da base de dados, importante para analisar o viés existente no segundo gráfico, que demonstra a intervenção que o grau de educação do cliente tem em relação ao total de compras concretizadas na empresa. Nota-se que quanto mais avançado o grau, maior é o número de aquisições.

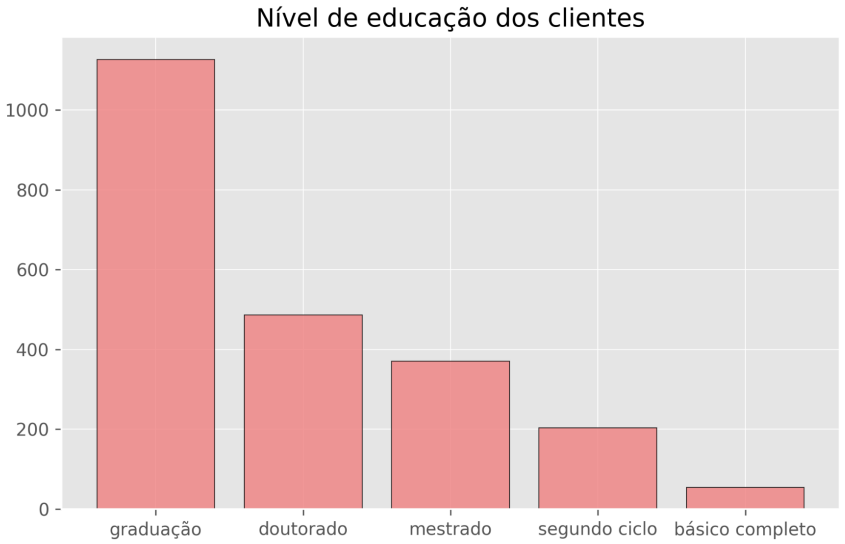

Figura 25: quantidade de clientes em cada grau de educação

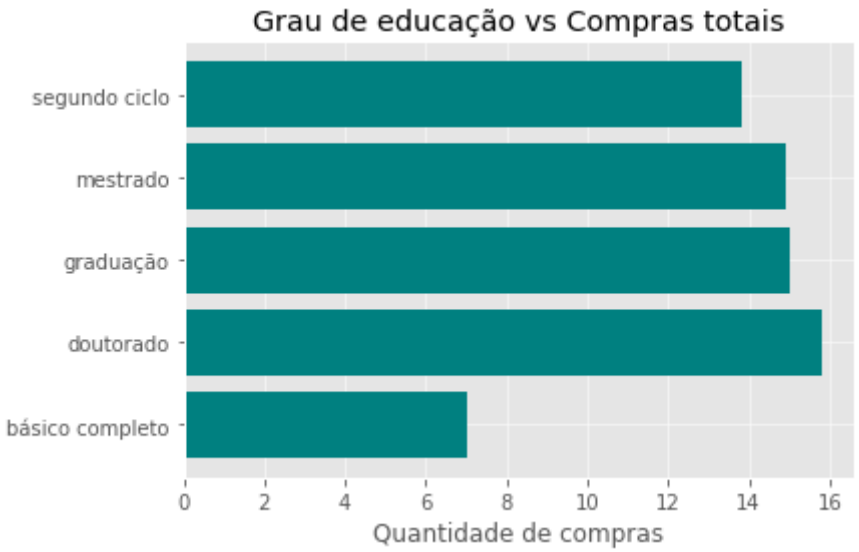

Figura 26: gráfico que mostra o total de compras de acordo com o grau de educação.

2.3.6 Renda

O gráfico abaixo mostra a frequência de cada renda anual dos clientes. Percebe-se que a maioria dos clientes recebe menos do que $70.000 anuais, mas há uma quantidade significativa de indivíduos de classe ainda mais alta, recebendo até $130.000 anuais. Há *outliers* com renda de cerca de $200.000 anuais e um com renda de $600.000 por ano.

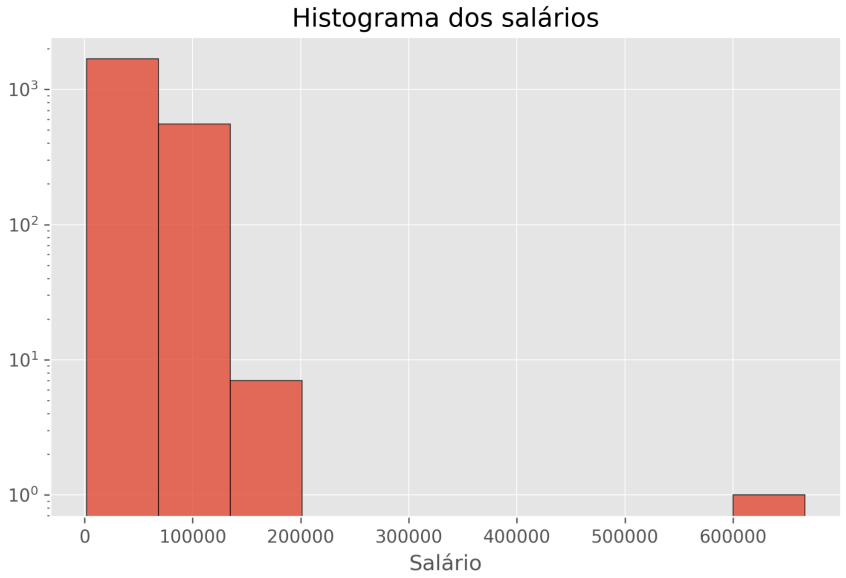

Figura 27: histograma das rendas dos clientes.

2.3.7 Média de compras pelo nível de renda

No gráfico abaixo, dividiu-se os clientes em quartis de níveis de renda, em que o primeiro nível é a menor classe financeira, enquanto o quarto nível é a maior classe. Nota-se que a média de compras é proporcional à renda do consumidor.

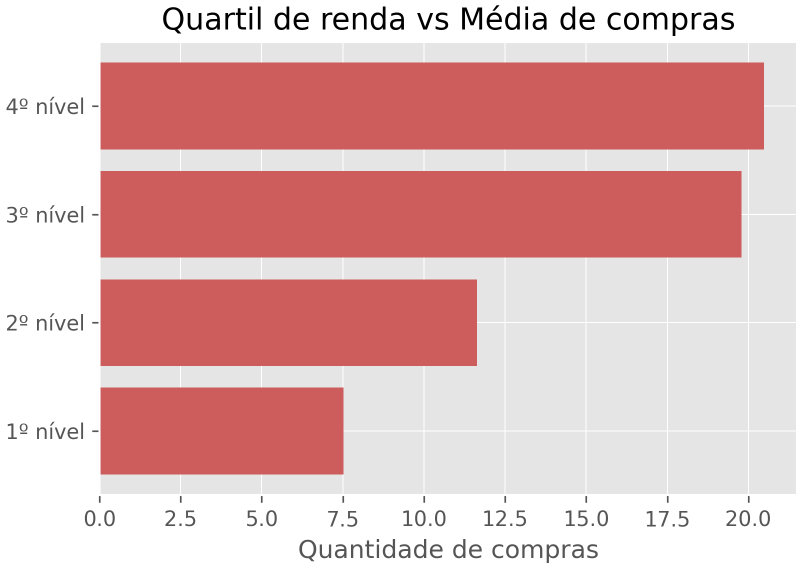

Figura 28: média de compras de cada nível de renda dos consumidores.

2.3.8 Média de gastos pelo nível de renda

Aproveitando-se dos níveis de renda definidos no subcapítulo supracitado, o gráfico abaixo demonstra que a média de gastos também é proporcional à renda do consumidor.

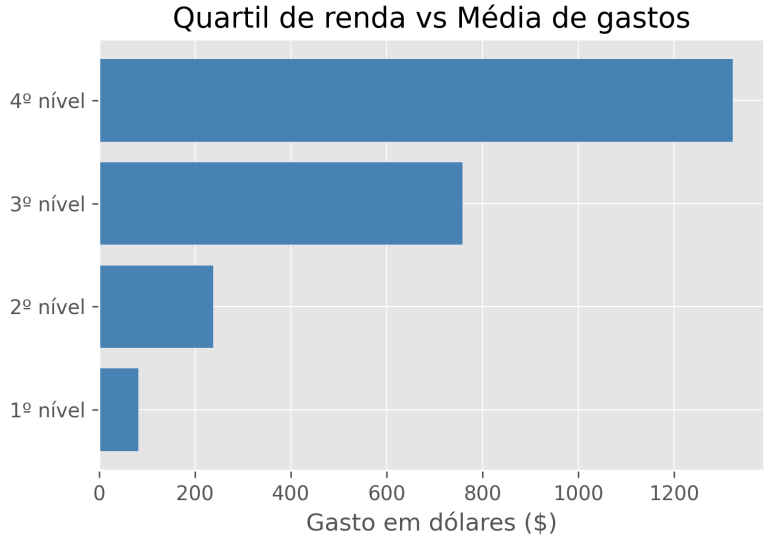

Figura 29: média de gastos de cada nível de renda dos consumidores.

2.4 Modelagem de dados

2.4.1 Avaliação do algoritmo *k-means*

Através do método *elbow*, aplicável após desenvolvimento do gráfico abaixo, definimos que o ganho de similaridade já não era tão alto de três grupos em diante, motivo pelo qual optou-se por determinar dois segmentos de clientes.

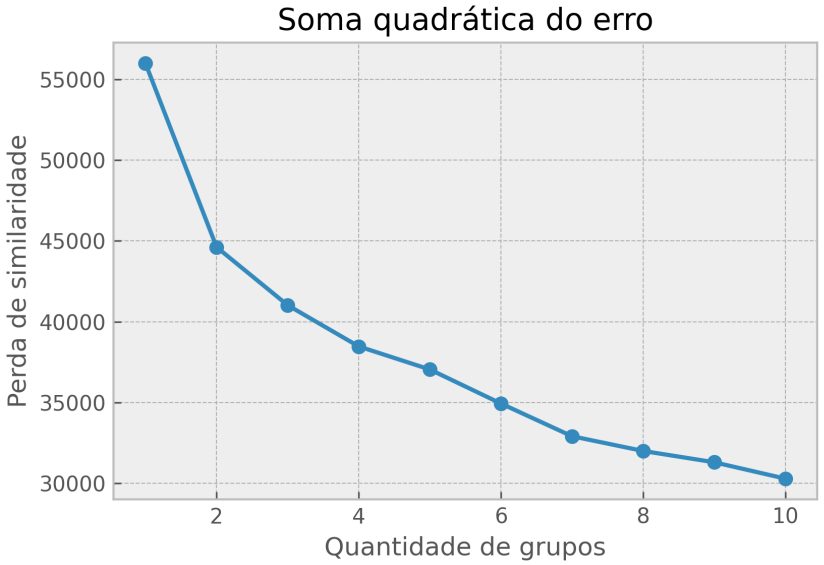

Figura 30: soma quadrática da perda de similaridade entre os indivíduos conforme a quantidade de grupos aos quais eles foram segmentados.

Outrossim, os gráficos abaixo cruzam os atributos em dispersão para entender a importância de cada um na escolha dos grupos:

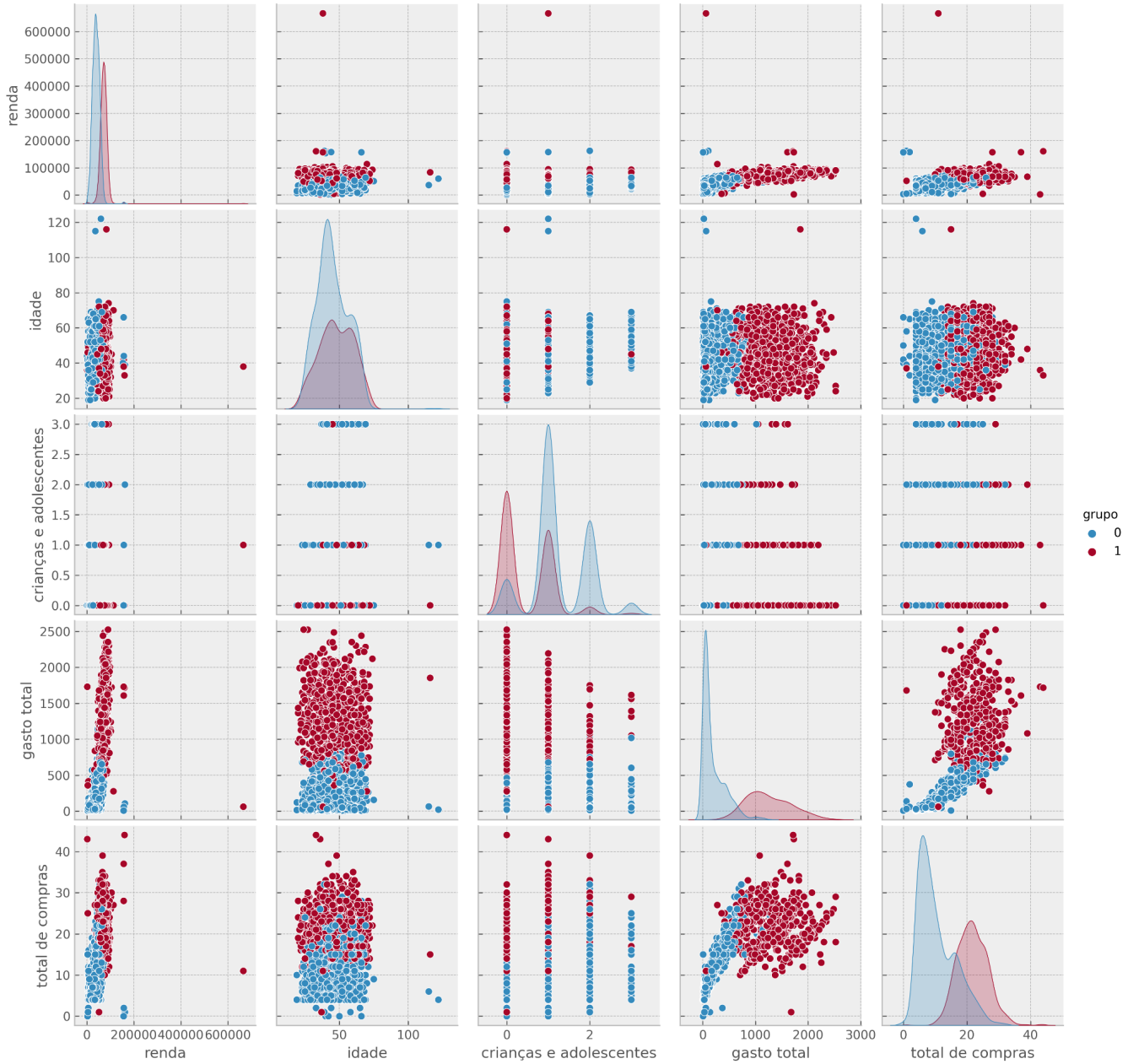

Figura 31: gráficos de dispersão que comparam cada dupla de atributos de maneira cruzada.

2.4.2 Avaliação do algoritmo Árvore de decisão

Validação cruzada

Maior: 0.96 | Média: 0.94 | Desvio: 0.022

Importância de cada atributo para a modelagem preditiva

compras c/ catálogo: 0.81

compras na loja física: 0.08

gasto em carnes: 0.07

gasto em doces: 0.03

renda: 0.01

** Todos os outros atributos demonstraram importâncias menores do que 0.01.*

Modelos com melhores desempenhos

Critério: entropia

```
f1: 96.25%
precisão: 96.48%
revocação: 96.03%
acurácia: 95.24%
```

Figura 32: avaliação por métricas da árvore de decisão com o critério de entropia.

Critério: gini

```
f1: 96.38%
precisão: 96.27%
revocação: 96.5%
acurácia: 95.39%
```

Figura 33: avaliação por métricas da árvore de decisão com o critério de gini.

Modelo com maior visibilidade e mais intuitivo

Critério: gini

Profundidade: 3

Mínimo de amostras nó: 86

Mínimo de amostras folha: 33

```
f1: 93.82%
precisão: 96.37%
revocação: 91.4%
acurácia: 92.71%
```

Figura 34: avaliação por métricas da árvore de decisão com o critério de gini, profundidade de 3, mínimo de amostras para o nó em 86 e mínimo de amostras para a folha de 33.

2.4.3 Avaliação do algoritmo *Support Vector Machine*

Validação cruzada

Maior: 1.0 | Média: 0.99 | Desvio: 0.0048

Métricas de avaliação

```
f1: 99.63%
precisão: 99.5%
revocação: 99.75%
acurácia: 99.55%
```

Figura 35: avaliação por métricas do modelo SVM.

Matriz de confusão

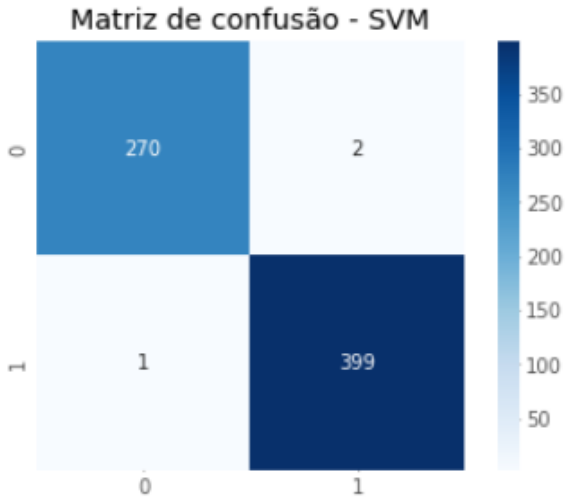

Figura 36: Matriz de confusão do modelo SVM.

3 Como a implementação atinge o objetivo

Tendo em vista que, para os mais de dois mil clientes de uma empresa, foi possível realizar a divisão deles em dois grupos com a maior similaridade possível, onde foi comprovado que, caso seja do desejo de um especialista, é viável aumentar a quantidade de grupos para adquirir maior similaridade entre os componentes, e

que foram criados dois modelos de aprendizagem de máquina supervisionada capazes de prever o grupo de novos clientes com acurácia média de 92%, o objetivo de negócio foi alcançado. Os métodos de agrupamento e predição de grupos de novos clientes foram desenvolvidos e avaliados internamente como satisfatórios, sendo considerado um produto relevante dentro do contexto de negócio elucidado neste projeto.

4 Limitações do projeto

Para que o projeto tivesse maior performance e relevância, seria necessário que obtivéssemos uma base de dados mais robusta, com o objetivo de tornar os modelos mais confiáveis e utilizáveis em larga escala, considerando que a probabilidade de haver *overfitting* e/ou *underfitting* (casos em que o modelo é relevante apenas para aquele conjunto de dados específico ou até mesmo nem para este conjunto) é razoável. Além disso, a análise de cada grupo obtido pelo modelo de *clusterização* poderia ser aprofundada em uma futura versão do projeto, com o objetivo de torná-lo ainda mais completo e atraente para captação de investimentos.

5 Conclusão

O objetivo proposto foi alcançado, tendo em consideração que a segmentação de clientes em grupos com índices interessantes de similares foi desenvolvida com sucesso e pode ser facilmente modificada de acordo com estratégias de negócio distintas, a combinar com especialistas que venham a aproveitar o produto aqui desenvolvido. A metodologia usada foi satisfatória, já que sua complexidade não foi um empecilho e permitiu o desenvolvimento das aplicações de maneira descomplicada. Os modelos de aprendizagem de máquina são relevantes ao contexto de negócio explorado, entretanto, alcançaria maiores índices de performance em bases de dados com mais instâncias e as análises de cada grupo gerado são custosas, onde seriam geradas a partir de uma definição da quantidade de segmentos pelo cliente do serviço aqui desenvolvido.

Referências

ESTATÍSTICA PRÁTICA PARA CIÊNCIA DE DADOS - Peter Bruce & Andrew Bruce

MÃOS À OBRA: APRENDIZADO DE MÁQUINA COM *SCI-KIT LEARN & TENSORFLOW* - Aurelien Geron

STORYTELLING COM DADOS - Cole Nussbaumer

https://www.kaggle.com/imakash3011/customer-personality-analysis?select=marketing_campaign.csv

https://minerandodados.com.br/entenda-o-algoritmo-k-means/

https://www.ibm.com/docs/es/spss-modeler/SaaS?topic=trees-decision-tree-models

https://github.com/slundberg/shap

https://www.ibm.com/docs/es/spss-modeler/SaaS?topic=models-how-svm-works

https://dataml.com.br/validacao-cruzada-aninhada-com-scikit-learn/