PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Inteligência Artificial e Aprendizado de Máquina

Gustavo Vinicius De Morais Pereira

Classificação de Leads para o Mercado

Goiânia
2020

Gustavo Vinicius De Morais Pereira

Classificação de Leads para o Mercado

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Inteligência Artificial e Aprendizado de Máquina como requisito parcial à obtenção do título de especialista.

Goiânia
2020

SUMÁRIO

1. Introdução

1.1. Contextualização

Os leads são clientes em potencial para as empresas, pessoas interessadas em comparar o produto, mas que ainda não fecharam um acordo ou compraram o produto de fato.

A classificação de um lead para determinar se ele é um cliente que vai fechar a venda com a empresa é uma ferramenta valiosa para o mercado.

1.2. O problema proposto

O poder de classificar um lead para determinar se ele é um potencial cliente ou não pode trazer benefícios para a empresa, como por exemplo, redução do custo de campanhas de marketing para clientes que vão ter baixo interesse, redução nas ligações telefônicas do departamento da empresa que entra em contato com clientes. A classificação de leads traz assertividade, acurácia, para investimento de orçamento, tempo e recursos humanos da empresa.

Os dados analisados são de um desafio da plataforma Kaggle, sobre leads interessados em fazer um curso com determinada escola.

O objetivo dessa análise é classificar leads para determinar se eles tem um alto, médio ou baixo interesse em fazer o curso.

Os leads são de origem norte americana.

Não está sendo analisado um período especifico neste modelo, mas sim o interesse de um lead em fazer um curso.

2. Coleta de Dados

Os dados foram obtidos da plataforma Kaggle no formato CSV, no link https://www.kaggle.com/rockbottom73/leads-dataset no dia 23 de Julho de 2020. Segue a

descrição das colunas utilizadas para a criação do modelo. Foi aplicado Feature Engineering para a seleção dos melhores atributos para a construção do modelo.

Nome da coluna/campo	Descrição	Tipo
lead_stage	Indica se o lead está interessado no curso, se é um lead qualificado ou se foi possível ou não entrar em contato com esse lead	Texto
engagement_score	Informa a pontuação de engajamento do lead	Numérico
lead_quality	Indica a qualidade do lead, por exemplo, sinaliza a relevância do lead ou incerteza quanto ao lead	Texto
tags	Indica se o lead está interessado em outro curso, ou se o lead vai mudar de ideia após ler o e-mail enviado.	Texto
lead_profile	Indica se é um lead em potencial	Texto
If_you_are_a_working_professional_please_mention	Indica a profissão do lead caso ele seja um profissional de alguma área no mercado	Texto

3. Processamento/Tratamento de Dados

Nessa etapa foi convertido o nome de cada coluna do data set em um formato padrão, exemplo: lead_type.

```
1. df = pd.read_csv('SampleData.csv', sep=',', encoding='cp1252')
2. df.head()
3.
4. df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_').str.rep
   lace('(', '').str.replace(')', '')
```

As colunas de índices do dataset foram removidas para evitar processamento desnecessário.

```
# drop ID's columns
df = df.drop('prospect_id' , axis='columns')
df = df.drop('lead_number' , axis='columns')
```

Foi criada uma coluna nova no dataset para informar a classificação do lead para o aprendizado supervisionado utilizado. Para preencher o valor da coluna Class no data set foi utilizado o valor da coluna Lead Stage que aponta o interesse do lead com relação ao curso, e a partir do valor dessa coluna os leads foram classificados com auto, médio e baixo interesse.

```
# create a column class based on a column value
df['Class'] = pd.np.where(df.lead_stage.str.contains("Not Interested"), "LOW"
,
               pd.np.where(df.lead_stage.str.contains("Lost"), "LOW",
               pd.np.where(df.lead_stage.str.contains("Unreachable"), "LOW",
               pd.np.where(df.lead_stage.str.contains("Junk Lead"), "LOW",
               pd.np.where(df.lead_stage.str.contains("Not Eligible"), "LOW",
               pd.np.where(df.lead_stage.str.contains("Qualified"), "MEDIUM",
               pd.np.where(df.lead_stage.str.contains("Not Called"), "MEDIUM",

               pd.np.where(df.lead_stage.str.contains("Interested"), "HIGH",
               pd.np.where(df.lead_stage.str.contains("Closed"), "HIGH", "null
"
))))))))))
```

Cada coluna do dataset foi convertida para o tipo String para que fosse possível utilizar o método de Label Encoder que transforma valores textuais em valores numéricos para facilitar o treinamento do algoritmo.

```
# convert type of column's values to string to apply label encoder
df = df.astype(str)

# label encoder all at once
new_df = df.apply(LabelEncoder().fit_transform)
```

Devido a utilização do Label Encoder não foi necessário tratar os valores NaN (valores omissos) do dataset, porque esses valores foram rotulados com dígitos para representar esse valor.

A quantidade de registros utilizados é 9240 com 121 atributos utilizados como variáveis independentes para prever a variável dependente Calss.

4. Análise e Exploração dos Dados

Para a análise do problema de criação da classe para classificação dos leads, foram listados os valores únicos da coluna Lead Stage do dataset para que a partir do seu valor fosse criada uma coluna de classe com o rótulo que deveria ser aprendido pelo algoritmo.

Os leads estavam classificados em nove tipos, e a partir dessa classificação anterior esses tipos foram distribuídos em três classes como mostrado na tabela abaixo:

HIGH	MEDIUM	LOW
Interested	Qualified	Not Interested
Closed	Not Called	Lost
		Unreachable
		Junk Lead
		Not Eligible

5. Criação de Modelos de Machine Learning

Para a criação do modelo foi utilizada a linguagem Python e o modelo SVM da biblioteca Sklearn com o kernel linear.

O modelo SVM é um classificador binário muito utilizado para resolver problemas que envolvem classificação de instancias em classes. O motivo de sua popularidade é o seu “kernel.method”, que pode ser ajustado para problemas de regressão linear, distância euclidiana e outros.

O SVM encontra uma linha de separação entre duas classes (hiperplano), que busca maximizar a distância entre os pontos mais próximos em relação a cada uma das classes. No modelo foi utilizado o kernel linear que retorna um hiperplano mais bem ajustado aos dados passados para o modelo, não causando overfitting mas sim melhorando a acurácia do modelo.

Foram importadas as bibliotecas python mostradas no trecho de código abaixo para trabalhar com dataframes, dividir os dados de treinamento e teste, transformar os dados textuais para rótulos numéricos para melhor aprendizado do modelo, calcular a acurácia do modelo, treinar o modelo SVM, ajustar os valores das

colunas para uma escala padrão de valores, selecionar os melhores atributos para o modelo e exibir os resultados para mostrar o quanto o modelo acertou ou errou ao prever cada classe.

```
#imports
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
import seaborn as sn
import matplotlib.pyplot as plt
```

A escala dos valores foram ajustadas para aumentar a acurácia do modelo, como mostrado no código abaixo:

```
# Adjust the scale of the values in the columns
#StandardScaler
df_scaled = pd.DataFrame(StandardScaler().fit_transform(new_df), columns=new_
df.columns)
```

As colunas do dataset foram divididas em atributos previsores e classe.

```
# get independent and dependent variables
previsores = df_scaled.iloc[:,0:120].values
classe = new_df.iloc[:,120].values
```

Foi aplicado o processo de seleção de atributos para a construção do modelo.

```
# choose best attributes

# Create an SelectKBest object to select features with two best ANOVA F-
Values
selector = SelectKBest(f_classif, k=10)

# Choose the best attributes to the model
selector.fit(previsores, classe)

# Show the name of the columns in the data set that are the best attributes t
o the model
cols = selector.get_support(indices=True)
features_df_new = df.iloc[:,cols]

# show the columns that best contribute to the model
features_df_new.head()
```

A partir dos melhores atributos selecionados, foi criado um dataset contendo apenas os melhores atributos para o modelo.

```
# build dataframe with the features that best contributes to the model
df_selcted_features_1 = df_scaled.filter(['lead_stage', 'engagement_score', '
lead_quality', 'tags','lead_profile', 'if_you_are_a_working_professional_plea
se_mention'])
df_selcted_features_1.head()

# get best independent variables
best_features = df_selcted_features_1.values
```

Os dados foram divididos em treinamento e teste, e dados para avaliação da acurácia do modelo.

```
# get train test data
X_treinamento2, X_teste2, y_treinamento2, y_teste2 = train_test_split(best_fe
atures,
                                                                      classe,
                                                                      test_size=0
.3,
                                                                      random_stat
e=0)
```

O modelo SVM foi treinado com os melhores atributos para a resolução do problema utilizando o kernel linear que é muito utilizado para problemas com uma grande quantidade de variáveis independentes.

```
# train model
# I am going to use the best attributes to build the model
svm = SVC(kernel='linear')
svm.fit(X_treinamento2, y_treinamento2)
```

Após o modelo ter sido treinado, foi realizada a predição da classe para os dados de teste e o calculo de acurácia do modelo.

```
# show model results
p_y = svm.predict(X_teste2)

taxa_acerto = accuracy_score(y_teste2, p_y)
taxa_acerto # 0.8892496392496393
```

6. Apresentação dos Resultados

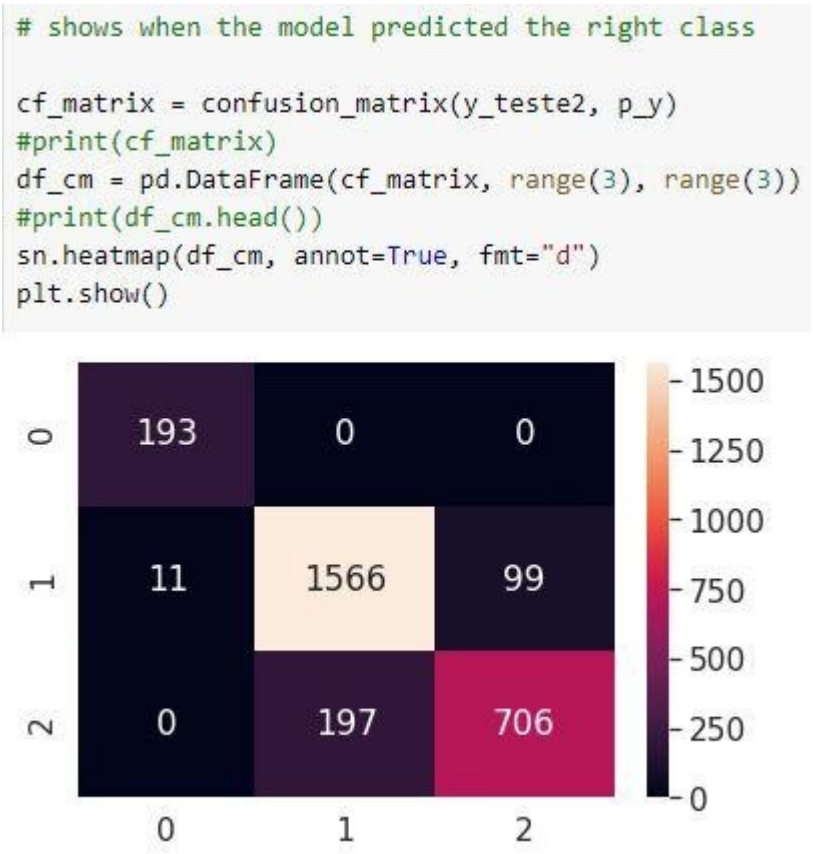
O modelo conseguiu atingir uma acurácia de 88% com os dados de teste como mostrado na imagem abaixo. Para chegar a essa acurácia foi feito um trabalho de feature engineering para utilizar apenas as melhores variáveis independentes que mais contribuíssem para o modelo. A escolha do kernel linear também aumentou a acurácia do modelo, pois esse tipo de kernel é a melhor escolha para problema que envolvem um grande número de variáveis independentes.

Segue abaixo o trecho de código utilizado para gerar a matriz de confusão mostrando o quanto o modelo acertou em cada classe.

```
# shows when the model predicted the right class

cf_matrix = confusion_matrix(y_teste2, p_y)
#print(cf_matrix)
df_cm = pd.DataFrame(cf_matrix, range(3), range(3))
#print(df_cm.head())
sn.heatmap(df_cm, annot=True, fmt="d")
plt.show()
```

A matriz de confusão foi criada utilizando a biblioteca Seaborn, pode-se ver o quanto o modelo acertou e errou para cada classe. Nos resultados pode-se ter uma leitura de que o modelo acertou 100% das vezes ao classificar leads de alto interesse, para leads de médio interesse ele acertou 93% das vezes, e para leads de baixo interesse ele acertou 78% das vezes.



7. Considerações Finais

A classificação de leads para o mercado imobiliário traz vários benefícios, dentre eles a economia de recursos e tempo, pois ao saber a probabilidade de fechamento de um lead é possível evitar o desperdício de tempo ao entrar em contato com o lead e o desperdício de recursos como tarifa telefônica ou quantidade de envio de e-mails feitos por uma API de processamento de requisições como por exemplo o Horizon do framework Laravel.

É muito importante destacar que a tarefa de feature engineering é essencial na construção de modelos de machine learning para que o tempo de treinamento do modelo seja reduzido, sua acurácia seja maior, e que haja um maior entendimento do problema proposto a se resolver.

7. Links

Link da apresentação:
https://www.loom.com/share/361a785946af41babf4ee37d7989419e
Link para o repositório no Github:
https://github.com/GustavoViniciusdeMorais/TCC_MachineLearning

REFERÊNCIAS

SAMPAIO, Cleiton. **Data Science para Profissionais Utilizando R. 1 ed.** Rio de Janeiro: Ciência Moderna Ltda, 2018.