Utilização de Técnicas de Regressão para Prever a Efetividade de Desempenho de Medicamentos

Pedro Wendling Hernandes

[1]Departamento de Computação - Universidade Estadual de Londrina (UEL)

deprao.pwhernandes@uel.br

Abstract. *This article describes an experimental work on a public dataset taken from Kaggle, which contains data about drug performance metrics for 37 common medical conditions. After an exploratory data analysis, the use of linear, K-Nearest Neighbors (K-NN), an artificial neural network (ANN) applied as a multi-layered perceptrons regressor (MLP) and Random Forest (RF) regression techniques is explored in order to verify prediction of future effectiveness values, then results are compared to conclude which method is better in this context.*

Resumo. *Este artigo descreve um trabalho experimental em um conjunto de dados público retirado do Kaggle, que diz respeito sobre avaliações de desempenho de aplicações de diversas drogas em 37 condições médicas comuns. Após uma análise exploratória nos dados, explora-se o uso das técnicas de regressão linear, K-Nearest Neighbors (K-NN), Random Forest (RF) e de regressão aplicada em uma rede neural artificial (ANN) como perceptrons em várias camadas (MLP) para verificar previsões de futuros valores de efetividade, então comparam-se os resultados para concluir o melhor método para uso neste contexto.*

1. Introdução

Poder realizar a predição de dados pode vir a otimizar diversas atuações profissionais - reduzindo tempo gasto e carga de trabalho. Isso se aplicaria também na avaliação de desempenho de medicamentos, ainda mais necessário visto que há a colaboração de pacientes, aos quais devemos atentar em ser quanto ágeis e precisos no processo de aplicação e avaliação de desempenho das drogas, em respeito à saúde e integridade deles. Dito isso, como poderíamos então garantir que a aplicação de um medicamento terá algum efeito positivo no paciente, a fim de agilizar a avaliação de aplicação?

Durante a análise exploratória do conjunto de dados que será trabalhado, retirado do Kaggle veremos que há uma variável que diz respeito à efetividade da droga aplicada. Então, devemos prever para novas aplicações os valores de efetividade. Como mencionado antes, será feita uma análise exploratória dos dados para identificar as variáveis que possuem maior influência na efetividade, aplicar regressão linear, K-NN, RF e MLPRegressor para então comparar a eficácia dos métodos. Anterior à análise, serão discutidos alguns trabalhos similares, que servem de inspiração para este. Após a análise, mostraremos como foram as aplicações dos métodos e então compararemos os resultados obtidos, utilizando como métricas o R quadrado (R^2) e a raiz do erro médio quadrático (RMSE).

2. Revisão da Literatura

2.1. Regressão Linear e K-NN

Como inspiração para regressão linear e K-NN, em [Fumo and Biswas 2015], os autores utilizam de regressão linear, simples e múltipla, para obter resultados quanto à predição do consumo de energia elétrica residencial. [Kohli et al. 2021] é um trabalho realizado bastante parecido com este, utilizando de um dataset de vendas também retirado do Kaggle e aplicando nele regressão linear e K-NN de maneira análoga a este trabalho. Em [Imandoust et al. 2013], temos uma análise quanto ao uso de regressão K-NN para prever eventos econômicos, como a falência de uma empresa.

2.2. Random Forest

Uma regressão Random Forest (RF) é modelada como um conjunto de diversas árvores de regressão. Cada árvore de regressão funciona da mesma maneira que uma árvore de decisão, porém ela é adaptada para decidir valores numéricos. Ao invés de decidir o split por Ganho de Informação e Entropia, decide-se a partir do conjunto de valores, entre as variáveis envolvidas na decisão, que tiverem o menor erro médio quadrático (MSE, Mean Squared Error).

Em [Smith et al. 2013], compara-se o uso de regressão linear múltipla e regressão RF para prever concentrações de 9 componentes neuroquímicos, tomando como base para aplicação dados expressos na região cerebral posterior de ratos. [Iannace et al. 2019] utiliza de RF para prever a intensidade sonora de turbinas eólicas a serem construídas.

2.3. Regressão Aplicada em uma Rede Neural de Perceptrons Multicamadas

Servindo como base para utilizar regressão aplicada em uma rede neural artificial (ANN) de perceptrons arranjados em diversas camadas (MLP), em [Hayati and Mohebi 2007] os autores estudam a aplicação de regressão MLP para modelar sistemas de previsão de temperatura à curto prazo (no artigo citado utiliza-se a sigla STTF, para Short-term Temperature Forecasting) para a cidade de Kermanshah, no Irã.

O estudo é realizado utilizando do erro médio absoluto (MAE, Mean Absolute Error) como métrica de erro da previsão da rede neural, função de ativação sigmóide para as duas camadas ocultas e função de ativação linear para a camada de saída, com o algoritmo do método gradiente conjugado escalado sendo o melhor definido para o treinamento, conseguindo obter resultados satisfatórios para a aplicação.

Em [Adebiyi et al. 2014], comparam-se a performance entre a aplicação de autoregressão integrada de médias móveis (ARIMA) e de ANN para a previsão de valores de ações do mercado, utilizando como dados de teste e treinamento valores de ações da Dell, obtidos da corretora New York Stock Exchange. A rede neural é aplicada como regressão em MLP utilizando o algoritmo de backpropagation para treinamento.

Para métricas de resultado, foram considerados os erros de previsão dos dados levantados utilizando ARIMA, calculados pela diferença entre cada valor atual e previsto, dividida por esse mesmo valor atual; para MLP foram considerados os MSE's dos dados resultantes. Mesmo que o algoritmo de Backpropagation seja pouco aplicável para problemas reais, chegam à conclusão de que os resultados obtidos pela rede neural apresentam performance geral melhor que os resultados por ARIMA.

Por fim, em [Maleki et al. 2019] os autores procuram medir a performance de ANN para prever valores de poluição do ar, utilizando como base os índices de qualidade do ar (AQI) e de qualidade do ar para a saúde (AQHI) de agosto de 2009 até agosto de 2010, para a cidade de Ahvaz, no Irã. Consideraram 6 compostos poluentes dentre quatro localizações diferentes para os índices; o coeficiente de correlação médio R e o RMSE como métricas de performance para o resultado.

O estudo conclui que modelos de rede neural podem ser aplicados efetivamente para o monitoramento da qualidade do ar. Porém, os autores recomendam que o estudo seja continuado com a comparação da eficiência modelos de ANN com outros modelos estatísticos, para que facilite a seleção de uma ferramenta apropriada por gerentes responsáveis pela tomada de decisões em campos que cuidam da qualidade do ar urbano.

3. Desenvolvimento

Nesta seção, mostraremos uma análise exploratória do conjunto de dados Drug_clean.csv, demonstrando relações entre as variáveis do conjunto. Para a implementação dos códigos, a fim de trazer uma análise ágil para este grande conjunto de dados, utilizamos da linguagem Python, versão 3.9.13. Com Python, conseguimos utilizar o módulo Pandas para converter o conjunto de dados em uma variável do tipo DataFrame, assim como fácil manipulação dela e o módulo matplotlib.pyplot para gerar gráficos e diagramas dos resultados obtidos.

3.1. Análise Exploratória

O conjunto de dados presente no arquivo Drug_clean.csv apresenta métricas de performance de desempenho de diversos medicamentos ou drogas em 37 condições médicas comuns. A seguir, listaremos as 10 variáveis pertencentes ao conjunto. Vale considerar que, ao serem transformadas em uma variável DataFrame via Pandas, por padrão os valores de tipo numérico são convertidos para ponto flutuante de 64 bits ('float64') e os valores de String são convertidos para tipo objeto ('object'), podendo sofrer conversões durante a implementação, caso convém.

- Condition (String): Nome da condição médica associada com a droga aplicada;
- Drug (String): Nome da droga aplicada;
- EaseOfUse (Numérica): Indicador de facilidade de uso da droga, com base em análises de consumidores. Varia de 1 a 5;
- Effective (Numérica): Indicador de efetividade da atuação da droga em combate à condição médica associada, com base em análises de consumidores. Varia de 1 a 5;
- Form (String): Formato de consumir a droga aplicada, podendo ser por cápsula, líquido (bebida ou injeção), tablete, creme, etc;
- Indication (String): Variável indicativa se a aplicação da droga para a condição médica aplicada é permitida pelas normas de saúde. Dois valores: On label/Off label. Possui entradas vazias preenchidas com espaços, serão listadas como N/A;
- Price (Numérica): Preço médio da droga, em dólares;
- Reviews (Numérica): Quantidade de análises de consumidores associadas ao uso da droga. Valores não-inteiros serão arredondados devido ao contexto da variável;
- Satisfaction (Numérica): Nível de satisfação médio das análises de consumidor. Varia de 1 a 5;

- Type (String): Indica se a droga deve ser adquirida apenas por receita prescrita ou pode ser adquirida em forma genérica. Três valores: RX – apenas por receita; OTC (Over-the-Counter) – receita não necessária, genérica ou ambos – RX/OTC. Possui entradas vazias preenchidas com espaços, serão listadas como N/A.

Para descobrir quais variáveis utilizar perante o problema proposto, devemos procurar relações entre elas levando os requisitos dos problemas em conta. Para saber como modelar para prever a efetividade de novos medicamentos, veremos a relação de cada variável com a variável Effective:

Começando com a variável Condition, não tem como ela por si só ser determinante para a efetividade de um medicamento, visto que é para ela que se aplica a droga. Logo, devemos estabelecer uma relação entre Condition, Drug e Effective para saber diferenças de efetividade de um medicamento em uma condição médica ou outra.

O autor pensou em representar essa relação por um diagrama de dispersão. Há muitos valores diferentes para a variável Drug, por isso serviria ser representada por um eixo do diagrama. No entanto, devido à possíveis limitações do módulo matplotlib, isso tornaria o diagrama completamente ilegível. Acaba-se por atribuir a variável Drug às cores do diagrama, porém o módulo não possui cores suficientes para diferenciar cada valor. abaixo o diagrama, que novamente devido à limitações quantitativas, não deve possuir legenda à favor da legibilidade.

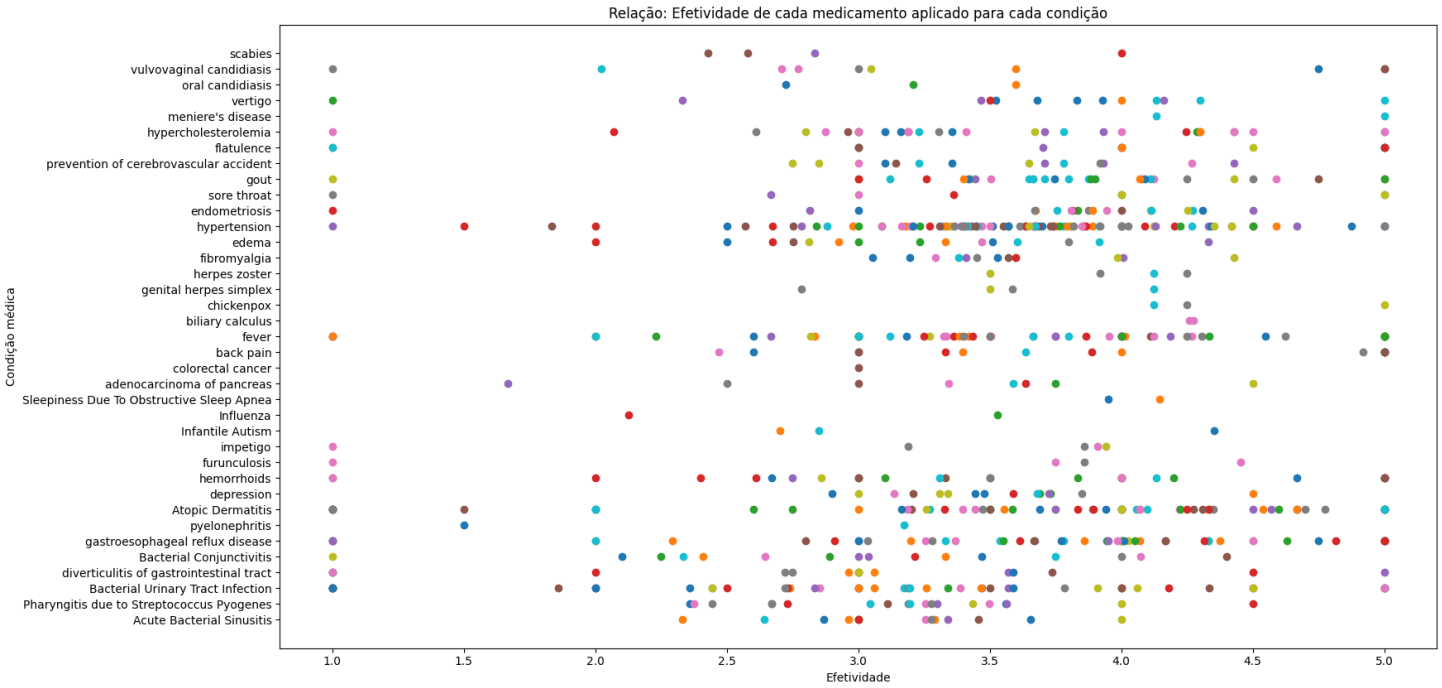

Figure 1. Gráfico de dispersão relacionando a aplicação de cada medicamento em cada condição médica com a métrica de efetividade. Não é possível mostrar uma legenda devido à imensa quantidade de medicamentos diferentes, fator este que causa também repetição de cores para os valores da variável Drug.

Mesmo assim, é possível inferir que em casos com muitas aplicações, como em hipertensão - valor ”hypertension” da variável Condition - há repetição do uso de medicamentos, trazendo à tona que nem sempre medicamentos aplicados em uma mesma condição médica serão eficientes do mesmo jeito.

Dito isso, dados que influenciam na efetividade em si ainda são inconclusivos, por isso relacionaremos com as demais variáveis, cujos valores servem para inferir conclusões sem um terceiro fator. Para as variáveis numéricas, utilizamos diagramas de dispersão e para as variáveis categóricas, utilizamos boxplots.

Como mencionado anteriormente ao listar as variáveis, consideramos N/A como valores indefinidos - não atribuídos. Felizmente, não é necessário realizar muitas alterações quanto aos valores indefinidos, pois ocupam pouco espaço do conjunto todo, possuindo pouca relevância. Observe:

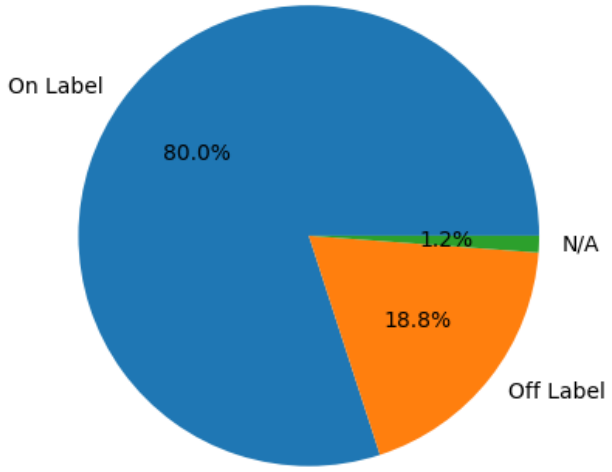

Figure 2. Gráfico de pizza da proporção amostral da variável Indication.

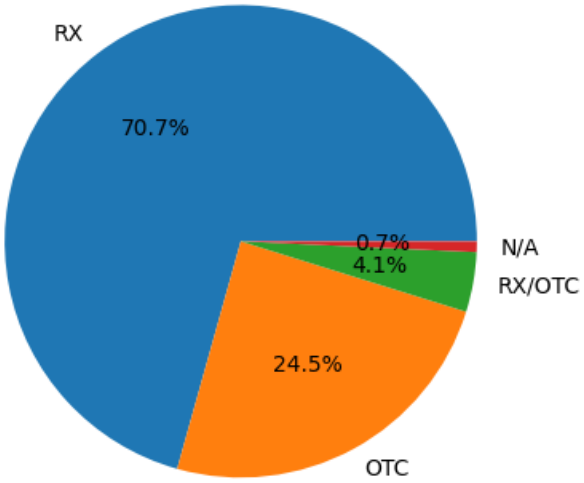

Figure 3. Gráfico de pizza da proporção amostral da variável Type.

Agora, observe os diagramas que dizem respeito às variáveis numéricas:

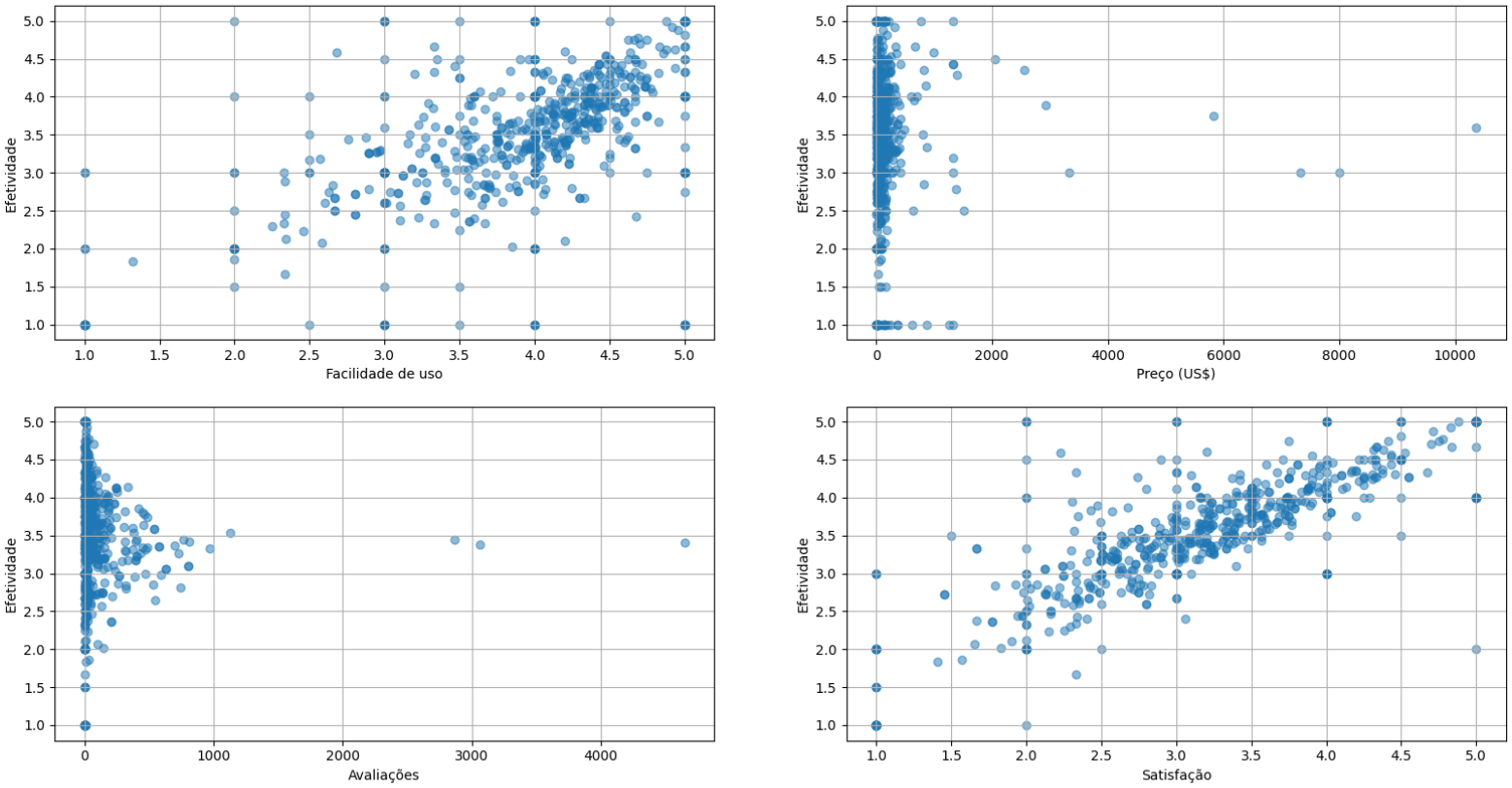

Figure 4. Conjunto de diagramas de dispersão relacionando a variável Effective com as demais variáveis contínuas.

Coeficientes de correlação de Pearson entre as variáveis numéricas:

	EaseOfUse	Effective	Price	Reviews	Satisfaction
EaseOfUse	1	0.66	-0.11	0.01	0.65
Effective	0.66	1	-0.02	-0.04	0.87
Price	-0.11	-0.02	1	-0.03	-0.03
Reviews	0.01	-0.04	-0.03	1	-0.08
Satisfaction	0.65	0.87	-0.03	-0.08	1

De acordo com os gráficos de dispersão, notamos que há bastante variação - tendendo a ser crescente - da variável Effective em relação às variáveis EaseOfUse e Satisfaction, enquanto as variáveis Price e Review não demonstram muita variação. Dito isso, apenas as variáveis EaseOfUse e Satisfaction possuem influência significativa na efetividade. Pelos coeficientes de Pearson, possuem também relação significativamente linear.

Observe os boxplots em relação às variáveis restantes:

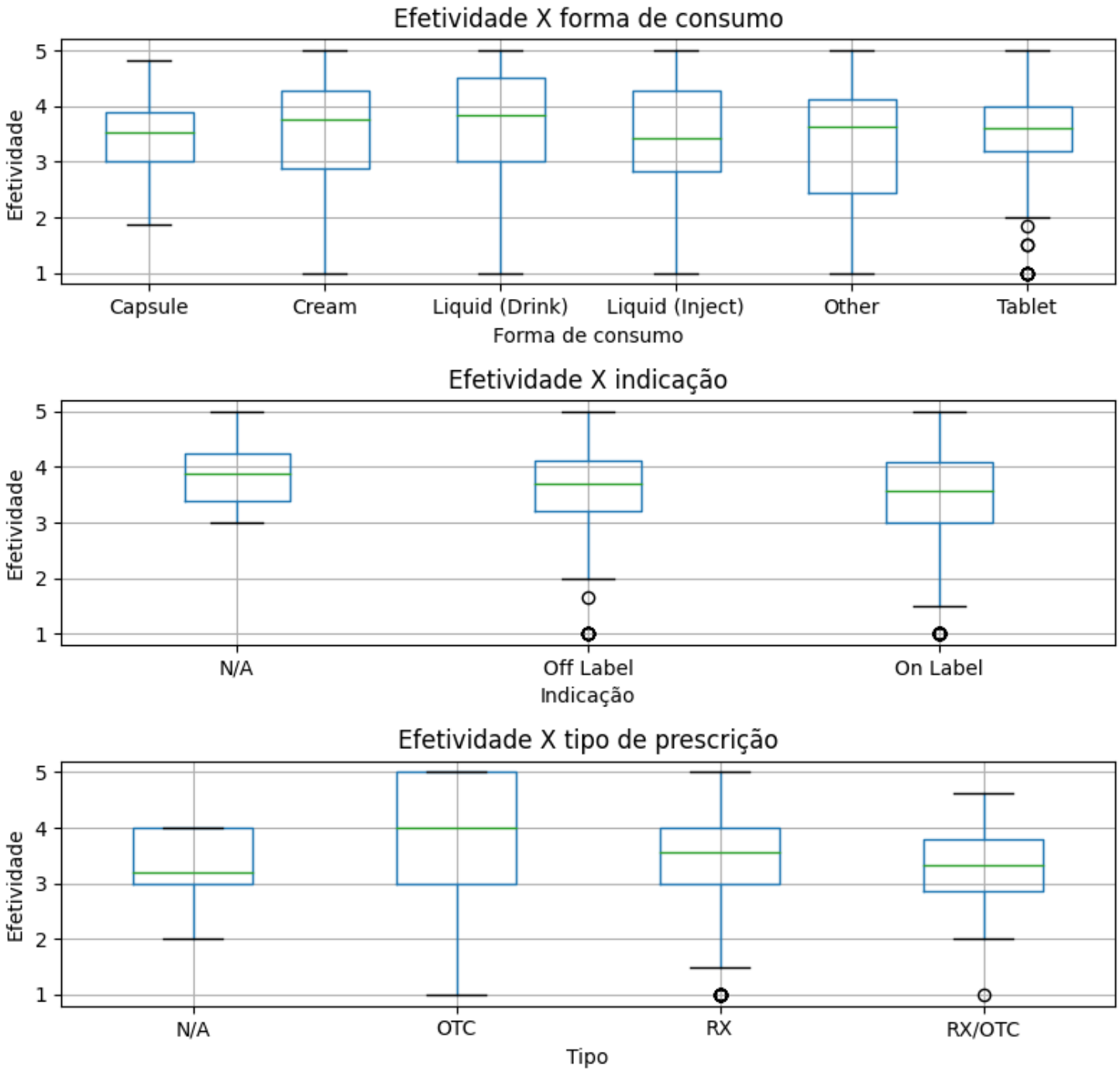

Figure 5. Conjunto de boxplots relacionando a variável Effective com as variáveis Form, Indication e Type.

De acordo com os boxplots, notamos que há apenas variação significativa entre os valores da variável Type. Não podemos estabelecer uma relação linear, por se tratar de uma variável discreta que não representa valores de intensidade. Portanto, para prever

a efetividade de novos medicamentos utilizaremos as variáveis Condition, EaseOfUse, Type e Satisfaction.

3.2. Técnicas de Regressão

Como mencionado anteriormente, para prever valores futuros da variável Effective, utilizaremos regressão linear, regressão K-NN e regressão Random Forest. Para o split de treinamento e teste utilizaremos 20% dos conjuntos de dados para teste e o 80% restante para treinamento.

Para fácil e agilizada aplicação das técnicas utilizamos a biblioteca scikit-learn, que possui diversos módulos contendo métodos de aprendizado de máquina. Dito isso, atribuímos o estado aleatório do split de treinamento e teste para todas as regressões arbitrariamente para 42, algo necessário para que resultados não variem para cara execução.

Para a regressão linear, consideramos apenas variáveis relacionadas linearmente. Então, não podemos utilizar as variáveis discretas nesta regressão, pois não há como definir linearidade para elas. Como visto pelos coeficientes de Pearson, utilizaremos então para ela apenas as variáveis EaseOfUse e Satisfaction.

Para as demais regressões, utilizaremos todos os valores que foram inferidos terem relação com Effective, de acordo com a análise anterior. Como uma regressão é feita apenas com valores numéricos, as variáveis discretas são codificadas para codificação One-Hot, que transforma cada valor da variável em uma variável distinta, podendo ter valor 0 ou 1 - se a amostra contém a variável, o valor dela será 1; será 0 para todas as demais variáveis referentes à variável original. Os resultados obtidos pelas aplicações dos métodos serão discutidos à frente na seção 4.

3.2.1. Regressão Linear

1. Montamos o conjunto das variáveis relevantes e aplicamos o split de test e treinamento;

linear-regression.py

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import OneHotEncoder

# Carregar o conjunto de dados original
df = pd.read_csv('Drug_clean.csv')

# Selecionar as variáveis relevantes
X = df[['EaseOfUse', 'Satisfaction']]

y = df['Effective']

# Dividir o conjunto de dados em treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figure 6. Implementação em Python da leitura, montagem e split em treinamento e teste do conjunto de dados a ser usado para regressão linear.

2. Assim, podemos criar o modelo, aplicar nele os dados de treinamento, fazer as previsões e obter o RMSE e o R^2

linear-regression.py

```python
# Criar o modelo de regressão linear e treiná-lo
model = LinearRegression()
model.fit(X_train, y_train)

# Fazer as predições utilizando o conjunto de teste
y_pred = model.predict(X_test)

# Calcular o erro médio quadrático (MSE) das predições
rmse = mean_squared_error(y_test, y_pred, squared=False)
r_2 = r2_score(y_test, y_pred)
```

Figure 7. Implementação em Python da previsão do modelo de regressão linear e aplicação dos cálculos de RMSE e R^2.

3.2.2. Regressão K-NN

1. Montamos o conjunto das variáveis relevantes e aplicamos treinamento e teste nele, desta vez utilizando de codificação One-Hot para as variáveis discretas;

knn.py

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import OneHotEncoder

# Carrega o conjunto de dados
df = pd.read_csv('Drug_clean.csv')

# Transformar as variáveis categóricas em codificação one-hot
encoder = OneHotEncoder()
data_encoded = pd.DataFrame(encoder.fit_transform(df[['Type', 'Condition']]).toarray(), columns=encoder.get_feature_names_out(['Type', 'Condition']))

# separando os dados em atributos descritores e atributo de classe
X = pd.concat([data_encoded, df[['EaseOfUse', 'Satisfaction']].reset_index(drop=True)], axis=1)
y = df['Effective']

# dividindo os dados em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figure 8. Implementação em Python da leitura, montagem e split em treinamento e teste do conjunto de dados a ser usado para regressão knn, com as variáveis categóricas codificadas para One-Hot.

2. Para aplicar a previsão do modelo, devemos decidir qual o número de vizinhos próximos (K) a ser aplicado no experimento. Considerando um máximo de 20 vizinhos, organizaremos as possibilidades em uma lista e então iteramos por ela, realizando a previsão do modelo para cada K. O K que tiver o menor RMSE é o melhor K, e será considerado como o determinante para a regressão aplicada;

```python
knn.py > ...
22
23  # definindo o número máximo de vizinhos a serem testados
24  max_neighbors = 20
25
26  # criando a lista de vizinhos para serem testados
27  neighbors = list(range(1, max_neighbors+1))
28
29  # Gráfico de linha do desempenho do modelo no conjunto de teste para cada valor de k
30  test_errors = []
31  for k in neighbors:
32      knn = KNeighborsRegressor(n_neighbors=k)
33      knn.fit(X_train, y_train)
34      y_pred = knn.predict(X_test)
35      test_errors.append(mean_squared_error(y_test, y_pred, squared=False))
36
37  plt.plot(neighbors, test_errors)
38  plt.xlabel('Número de vizinhos')
39  plt.ylabel('Erro de teste (RMSE)')
40  plt.title('Desempenho do modelo no conjunto de teste para diferentes valores de k')
41  plt.show()
42
43  best_k = neighbors[np.argmin(test_errors)]
```

Figure 9. Implementação em Python de como encontrar o melhor K a partir do RMSE, e construção de gráfico de desempenho a partir dos possíveis K's.

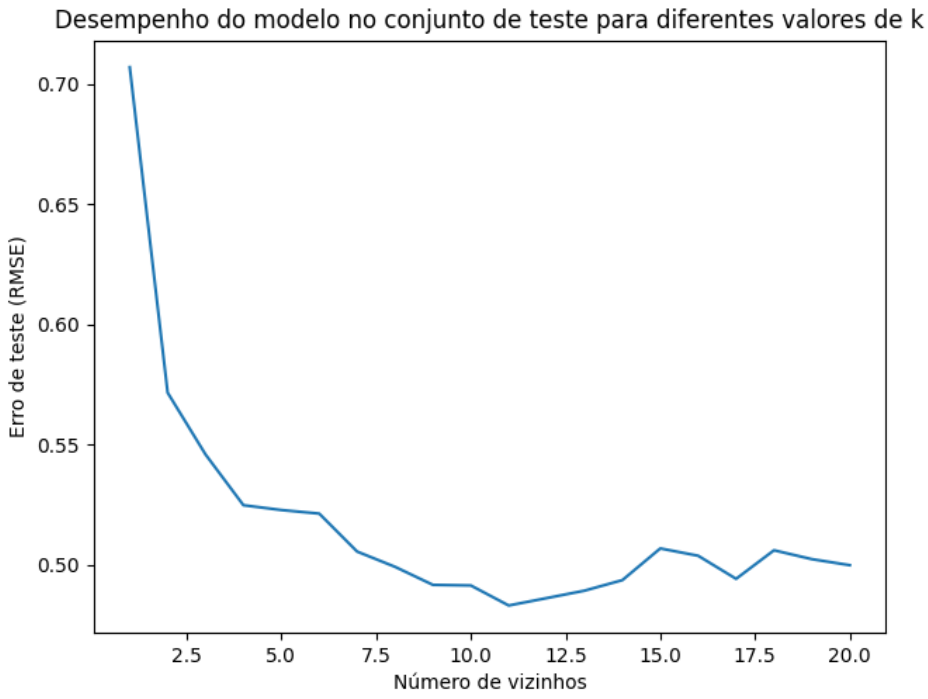

Figure 10. Gráfico de linha número de vizinhos X RMSE, para encontrar o melhor K.

3. A partir do gráfico, concluímos que o melhor K para a regressão é $K = 11$. treinando o modelo definitivo com o K escolhido, realizamos as previsões e calculamos o RMSE e o R^2.

```python
knn.py > ...
43    best_k = neighbors[np.argmin(test_errors)]
44
45    # treinando o modelo com o melhor valor de k
46    knn = KNeighborsRegressor(n_neighbors=best_k)
47    knn.fit(X_train, y_train)
48
49    # fazendo as previsões e calculando o MSE
50    y_pred = knn.predict(X_test)
51    rmse = mean_squared_error(y_test, y_pred, squared=False)
52    r_2 = r2_score(y_test, y_pred)
```

Figure 11. Implementação em Python da previsão do modelo de regressão K-NN com o melhor K encontrado e aplicação dos cálculos de RMSE e R^2.

3.2.3. Regressão Random Forest

1. Montamos o conjunto das variáveis relevantes com codificação One-Hot, aplicando o split de treinamento e teste do mesmo que jeito que em 8;

2. Para encontrar os melhores parâmetros para a regressão RF, utilizamos de Grid Search Cross-Validation (GSCV). Organizamos os parâmetros elegíveis em listas contidas em um dicionário e cada conjunto possível de parâmetros será testado em uma validação cruzada de 5 dobras - para cada conjunto de parâmetros, teremos 4 subconjuntos de dados para treinamento e 1 para teste. O conjunto de parâmetros terá duas variáveis: $n\_estimators$ e max_depth:

    - $n\_estimators$: número de árvores para a floresta, serão testadas de 40 a 200 árvores de 5 em 5;
    - max_depth: profundidade máxima de cada árvore, serão testados de 3 a 20 níveis máximos de 2 em 2;

    Utilizaremos da raiz do erro médio quadrático negativa (NRMSE) como métrica para descobrir os melhores parâmetros pela busca;

```python
param_grid = {
    'n_estimators': np.arange(40, 200, 5, dtype=list),
    'max_depth': np.arange(3, 20, 2, dtype=list)
}

grid_search = GridSearchCV(estimator=RandomForestRegressor(random_state=42), param_grid=param_grid, cv=5, n_jobs=-1, scoring='neg_root_mean_squared_error')
grid_search.fit(X_train, y_train)

print('Melhores parâmetros:', grid_search.best_params_)
```

Figure 12. Implementação em Python da aplicação de GridSearchCV, com os parâmetros especificados, validação cruzada com 5 dobras e medida de pontuação com a NRMSE. Feita a busca, encaixa as variáveis de treinamento no modelo.

- Feita a busca, encontramos os melhores parâmetros como $n\_estimators = 190$ e $max\_depth = 7$

```
Usuario@DESKTOP-8U73QV4 MINGW64 /d/COMPUEL/IA/trab-ml
$ python3 regression-forest.py
Melhores parâmetros: {'max_depth': 7, 'n_estimators': 190}
```

Figure 13. Saída do print dos melhores parâmetros encontrados por GridSearchCV.

3. Aplicado o fit das variáveis encontradas pela GSCV, realizamos as previsões e calculamos o RMSE e o R^2 da Random Forest.

```python
y_pred = grid_search.predict(X_test)

rmse = mean_squared_error(y_test, y_pred, squared=False)
print('RMSE:', rmse)

r_2 = r2_score(y_test, y_pred)
print("R\u00b2: ", r_2)
```

Figure 14. Implementação em Python da previsão do modelo de regressão Random Forest com os melhores parâmetros encontrados e aplicação dos cálculos de RMSE e R^2.

3.2.4. Regressão MLP

1. Montamos o conjunto das variáveis relevantes com codificação One-Hot, aplicando o split de treinamento e teste do mesmo que jeito que em 8;

2. Utilizar de GSCV para encontrar os melhores parâmetros não convém com aplicação de regressão MLP neste trabalho, pois há muitos hiperparâmetros para serem testados, demandando um tempo de execução muito acima de um teste deste método, diferentemente de Random Forest.
   Logo, foram feitos diversos testes empíricos para encontrar uma possível combinação de menor erro. O conjunto de parâmetros principais testados, considerando taxa de aprendizado inicial sempre 0.001 e sempre constante para os testes sujeitos ao algoritmo gradiente descendente, terá quatro variáveis: $hidden\_layer\_sizes$, $activation$, $solver$ e max_iter:

   - $hidden\_layer\_sizes$: Tupla da quantidade de neurônios em cada camada oculta, cada número inteiro na tupla será a quantidade de neurônios da camada oculta em sua posição, respectiva à sua posição na tupla;
   - $activation$: Função de ativação da camada oculta aplicada para a rede neural. Valores testados:
     - ”relu”: Função de unidade linear retificada, $f(x) = max(0, x)$;
     - ”tanh”: Função da tangente hiperbólica, $f(x) = tanh(x)$;
     - ”logistic”: Função sigmóide logística, $f(x) = 1/(1 + exp(-x))$;
   - $solver$: Algoritmo de otimização e atualização dos pesos da rede. Valores testados:
     - ”sgd”: Algoritmo gradiente descendente estocástico comum;
     - ”adam”: Versão estendida do algoritmo gradiente descendente comum, o otimizador Adam costuma trazer resultados melhores que o algortimo base, pois atualiza a taxa de aprendizado individualmente para cada peso da rede, podendo demorar mais;
   - max_iter: Número máximo de épocas que se aplica o algoritmo de otimização, enquanto não convergir.

3. Assim, montamos o modelo de regressão, encaixamos o modelo e assim calculamos o RMSE e o R^2 da regressão MLP, para cada teste. Abaixo um exemplo:

```
mlp = MLPRegressor(hidden_layer_sizes=(2), activation='relu', solver='adam',max_iter=1000, random_state=42, learning_rate='constant', learning_rate_init=0.001)

# Treinando o modelo com GridSearchCV
mlp.fit(X_train, y_train)

# Fazendo a predição
y_pred = mlp.predict(X_test)

# Avaliando o desempenho do modelo
rmse = mean_squared_error(y_test, y_pred, squared=False)
print('RMSE:',rmse)

r_2 = r2_score(y_test, y_pred)
print("R\u00b2: ", r_2)
```

Figure 15. Implementação em Python de exemplo de teste do modelo de regressão MLP e aplicação dos cálculos de RMSE e R^2.

4. Resultados

Executando as implementações das regressões, temos as seguintes saídas:

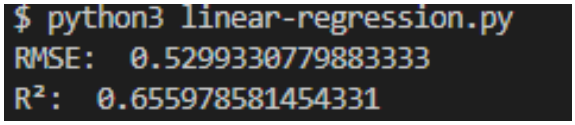

Figure 16. Saída no terminal de comando da execução do código contendo a implementação do modelo de regressão linear em Python, imprimindo a $RMSE$ e o R^2 calculados em 7

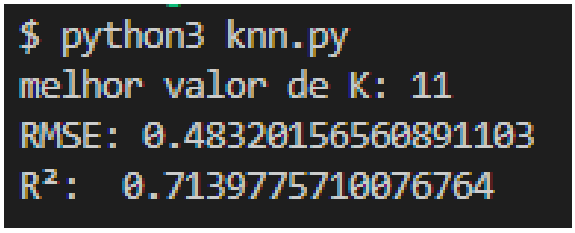

Figure 17. Saída no terminal de comando da execução do código contendo a implementação do modelo de regressão K-NN em Python, imprimindo o melhor valor de K, a $RMSE$ e o R^2 calculados em 9 e 11

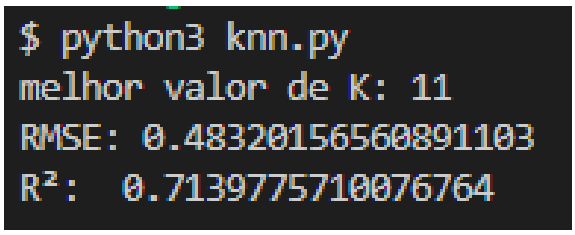

Figure 18. Saída no terminal de comando da execução do código contendo a implementação do modelo de regressão Random Forest em Python, imprimindo o melhor conjunto de parâmetros, a $RMSE$ e o R^2 calculados em 12 e 14

Tabela com os testes de maior relevância de regressão MLP:

ID do teste	$hidden\_layer\_sizes$	$activation$	$solver$	max_iter	Convergência	$RMSE$	R^2
T10	(1,1)	relu	adam	100	não	3.357	-12.807
T11	(1,1)	relu	adam	200	não	3.090	-10.697
T12	(1,1)	relu	adam	500	não	2.362	-5.835
T20	(2,1)	relu	adam	500	não	1.656	-2.358
T21	(2,1)	tanh	adam	500	não	0.879	0.054
T22	(2,1)	tanh	adam	1000	não	0.617	0.533
T23	(2,1)	logistic	adam	1000	não	0.828	0.161
T24	(2,1)	tanh	sgd	1000	sim	0.571	0.601
T30	(2,2)	tanh	sgd	1000	sim	0.544	0.637
T40	(3,2)	tanh	sgd	1000	sim	0.588	0.577
T50	(1)	tanh	sgd	1000	sim	0.569	0.603
T51	(1)	relu	sgd	1000	sim	0.562	0.613
T60	(2)	relu	sgd	1000	sim	0.917	-0.031
T61	(2)	relu	adam	1000	sim	0.480	0.718
T70	(3)	relu	adam	1000	não	1.767	-2.827
T80	(4)	relu	adam	1000	sim	0.489	0.707

Assim, dos testes de regressão MLP dados encontramos a rede do teste T61, com apenas uma camada oculta de dois neurônios, função de ativação linear retificada, algoritmo otimizador Adam, e limite de 1000 épocas para convergência como o melhor resultado, com $RMSE = 0.480$ e $R^2 = 0.718$.

Tabela de sumarização:

Técnica utilizada	$RMSE$	R^2
Regressão Linear	0.530	0.656
Regressão K-NN	0.483	0.714
Regressão RF	0.491	0.705
Regressão MLP	0.480	0.718

Conseguimos perceber que mesmo a Random Forest e regressão MLP sendo técnicas que geralmente apresentam melhor desempenho, de fato nem sempre acaba sendo ótimo, visto que nesse caso não foi melhor que K-NN. Porém, se não considerarmos a análise exploratória para K-NN, RF e MLP, teremos o seguinte resultado:

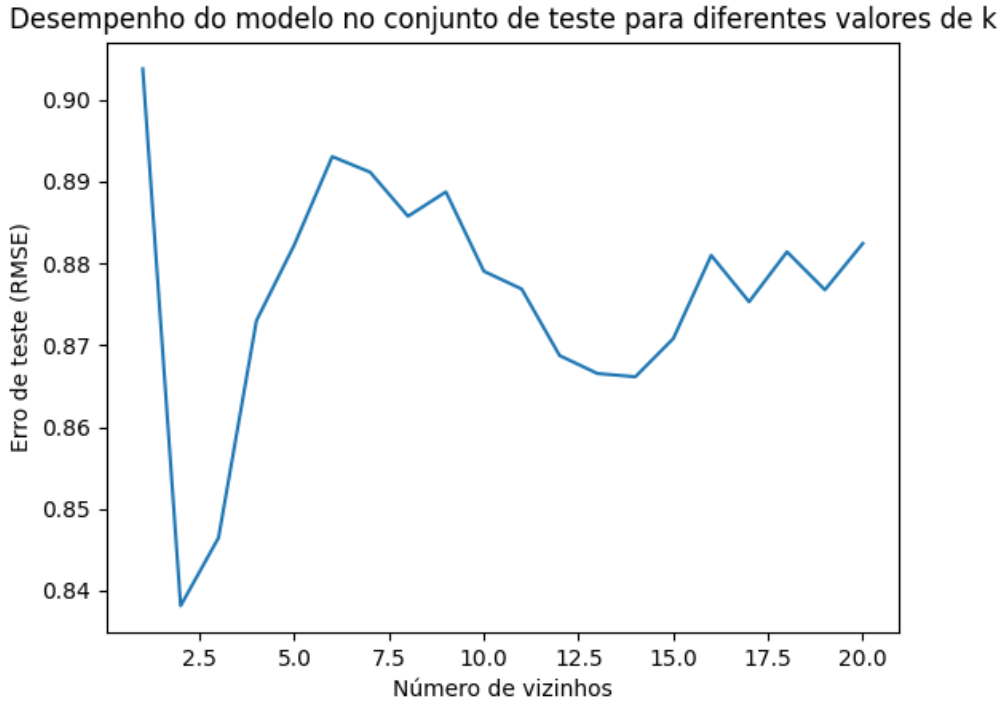

Figure 19. Gráfico de linha número de vizinhos X RMSE, para encontrar o melhor K. Neste caso, desconsidera-se a análise exploratória.

Para KNN, teremos $K = 2$ e para Random Forest, $n\_estimators = 140; max\_depth = 15$. Um teste exemplar de MLP para este caso foi similar ao T61, porém com duas camadas ocultas com 8 neurônios cada. As medidas calculadas serão:

Técnica utilizada	$RMSE$	R^2
Regressão K-NN	0.838	0.140
Regressão RF	0.495	0.700
Regressão MLP	0.490	0.706

5. Conclusão e Considerações Finais

Percebemos então a diferença que traz uma análise exploratória para a aplicação de regressão, o poder da Random Forest, visto que com uma grande mudança no escopo variou muito pouco de resultado e da rede neural, que consegue facilmente se adaptar com pouca mudança nos parâmetros. Dito isso, ainda convém considerarmos a rede neural com prioridade acima da Random Forest, pois mesmo sem a busca GSCV, uma regressão RF demanda muito mais tempo de execução para trazer resultados.

References

Adebiyi, A. A., Adewumi, A. O., Ayo, C. K., et al. (2014). Comparison of arima and artificial neural networks models for stock price prediction. *Journal of Applied Mathematics*, 2014.

Fumo, N. and Biswas, M. R. (2015). Regression analysis for prediction of residential energy consumption. *Renewable and sustainable energy reviews*, 47:332–343.

Hayati, M. and Mohebi, Z. (2007). Application of artificial neural networks for temperature forecasting. *International Journal of Electrical and Computer Engineering*, 1(4):662–666.

Iannace, G., Ciaburro, G., and Trematerra, A. (2019). Wind turbine noise prediction using random forest regression. *Machines*, 7(4):69.

Imandoust, S. B., Bolandraftar, M., et al. (2013). Application of k-nearest neighbor (knn) approach for predicting economic events: Theoretical background. *International journal of engineering research and applications*, 3(5):605–610.

Kohli, S., Godwin, G. T., and Urolagin, S. (2021). Sales prediction using linear and knn regression. In *Advances in Machine Learning and Computational Intelligence: Proceedings of ICMLCI 2019*, pages 321–329. Springer.

Maleki, H., Sorooshian, A., Goudarzi, G., Baboli, Z., Tahmasebi Birgani, Y., and Rahmati, M. (2019). Air pollution prediction by using an artificial neural network model. *Clean technologies and environmental policy*, 21:1341–1352.

Smith, P. F., Ganesh, S., and Liu, P. (2013). A comparison of random forest regression and multiple linear regression for prediction in neuroscience. *Journal of neuroscience methods*, 220(1):85–91.