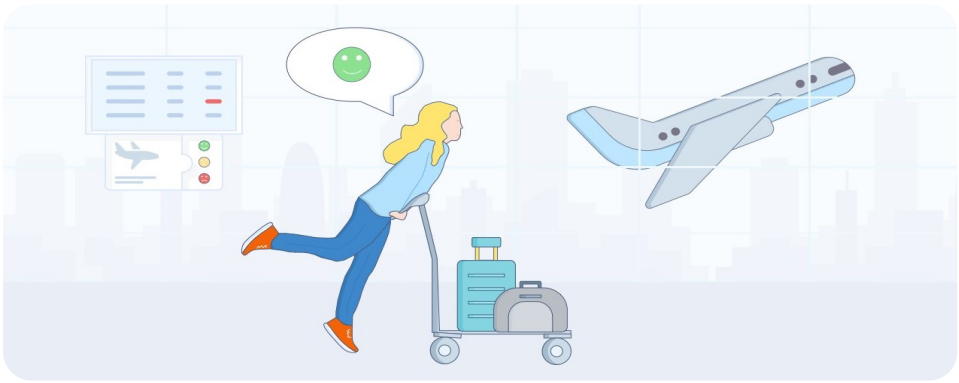
Instituto Politécnico de Coimbra

Instituto Superior de Engenharia de Coimbra

Mestrado em Engenharia Informática

Machine Learning

Predição do nível de satisfação dos passageiros de uma companhia aérea: Meta 2



Autores:

Miguel Fernandes

Nº aluno: 2016014470

Stephanie Batista

Nº aluno: 2019114900

Orientador:

Professor Doutor Simão Paredes

Coimbra, 9 de janeiro de 2023

Índice

Lista de Figuras

Lista de Tabelas

Resumo

Neste trabalho, apresenta-se a última fase (meta 2) relativa a um problema de *Machine Learning* (ML) do tipo de classificação binária (identificação do nível de satisfação ou insatisfação de passageiros de uma companhia aérea) no qual se efetua a submissão a algoritmos de aprendizagem supervisionada (*Supervised Learning,* SL) e não supervisionada (*Unsupervised Learning*, UL) dos dados obtidos e preparados na meta 1.

Palavras-chave: *Machine Learning*, *Dataset*, *Supervised Learning*, *Unsupervised Learning*.

1. Introdução

Nesta secção apresentam-se os objetivos e a contextualização do problema a resolver.

1.1 Objetivos

A meta 2 do presente trabalho tem como objetivo a submissão do *dataset* previamente apresentado na meta 1 a algoritmos de SL e UL.

1.2 Descrição do trabalho

Na meta 1, foi escolhido um *dataset* caracterizado por um total de 22 atributos compostos por dados reais e 10000 instâncias, e o mesmo foi analisado e preparado para a fase apresentada neste relatório.

Na meta 2, a principal biblioteca utilizada foi a *scikit-learn*, sendo uma das mais usadas para a aplicação de diversos algoritmos em ML. Para esta fase, o trabalho divide-se em duas partes:

- **Aprendizagem Supervisionada**

Criação de uma *short-list* de algoritmos de SL para avaliar o desempenho dos diversos classificadores.

- **Aprendizagem não supervisionada**

Aplicação de algoritmos de *clustering*, quer de forma isolada quer como forma de pré-processamento dos dados, para avaliar a possibilidade de melhorar o desempenho obtido com os modelos de SL.

1.3 Arquitetura do trabalho

Para a realização do trabalho, foi considerada a arquitetura apresentada no Apêndice I.

2. Aprendizagem supervisionada

Neste capítulo descreve-se a abordagem implementada para a aplicação de modelos de SL.

2.1 Definição

A aprendizagem supervisionada (SL) implica a aprendizagem de um mapeamento entre um conjunto de variáveis de entrada X e uma variável de saída Y e a posterior aplicação deste mapeamento para prever as saídas de dados nunca vistos anteriormente. A SL é a metodologia mais importante em ML e tem uma importância central no processamento de todo o tipo de dados [1].

Nesta ilustração, apresenta-se um exemplo de um problema de SL de classificação, no qual a partir dos dados de treino *input-output* (xn, tn), com n = 1, ..., N, pretende-se predizer o *output* t para um *input* não observado x.

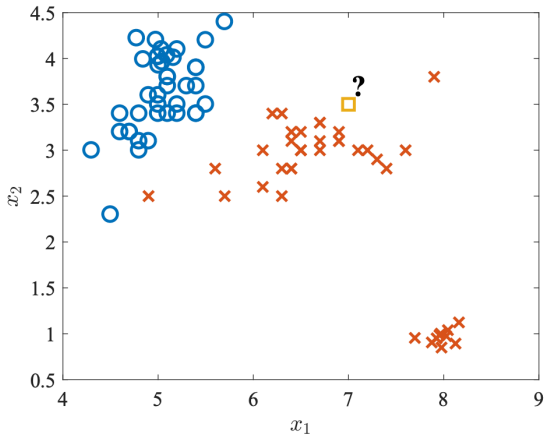


Figura 1. Ilustração de um problema de SL de classificação.

2.2 Métricas

Para esta fase do projeto, procedeu-se à análise dos algoritmos de SL a partir das seguintes métricas [2][3]:

Accuracy representa um percentual total de acertos do modelo.

Precision é a capacidade do modelo de não prever uma instância negativa como positiva. Para todas as instâncias classificadas como positivas, qual é o percentual de acerto.

Recall é a capacidade do modelo de encontrar todas as instâncias positivas. Para todas as instâncias que são de fato positivas, qual é o percentual de acerto.

A métrica F1 conjuga as duas anteriores como uma média harmônica entre ambas. Ela deve sempre ser priorizada para comparar modelos de classificação em relação à *accuracy*.

Uma excelente alternativa é fazer a curva ROC e calcular o AUC (área debaixo da curva).

A curva ROC (*Receiver Operating Characteristic Curve*) tem em conta a TPR (*True Positive Rate* ou *Recall* ou *Sensitity*) e a FPR (*False Positive Rate* ou *Specificity*). A curva ROC traça esses dois parâmetros. O AUC (*Area Under the Curve*) é um valor que sintetiza a informação da curva ROC. Ela varia de 0.5 a 1. Em suma, essa métrica diz o quanto o modelo é capaz de distinguir as duas classes.

Sob este paradigma, ter uma alta precisão será mais importante para o problema em questão. Para identificar corretamente os fatores cruciais que levam à satisfação do cliente, o modelo de previsão da classe positiva, 'Satisfeito', precisa de ser muito fiável. No entanto, o valor da métrica F1 também é considerado.

2.3 Short list de algoritmos

Para a aplicação dos algoritmos de SL no *dataset*, foram considerados a maior parte dos modelos de SL aprendidos durante as aulas. Destes todos, o algoritmo de *Linear Regression* não foi considerado pois, este algoritmo assume uma relação linear entre os atributos e o alvo, e o algoritmo de *Linear Regression* apenas lida com valores contínuos. Posteriormente foi conduzida a avaliação do desempenho de cada um deles no *dataset* através das métricas mencionadas anteriormente, tendo em consideração a natureza do tipo de problema (classificação binária) e dos atributos do *dataset.*

O procedimento implementado consistiu na utilização de ***Column Transformer*** (chamado '*preprocesser'*) para albergar 2 transformações: *StandardScaler*() para os atributos numéricos, e *OneHotEncod*() para os categóricos, o ***Pipeline*** que tem como passos '*preprocesser'* e o modelo que será aplicado. Posteriormente, criou-se uma **grelha com hiperparâmetros** possíveis e adequados para cada algoritmo. Finalmente, implementou-se uma instância do *GridSearchCV* que tem como parâmetros a *pipeline* do modelo, a grelha de parâmetros, e o número de pastas durante a *CrossValidation*).

2.3.1 Logistic Regression

O algoritmo *Logistic Regression* (LR) consiste num método estatístico utilizado para prever um resultado binário com base numa ou mais variáveis independentes. Pode ser utilizado para prever a probabilidade de diferentes eventos. A LR simplifica o processo de análise do impacto de múltiplas variáveis sobre um determinado resultado (Apêndice I).

2.3.1.1 Parâmetros utilizados

```
param_grid = {
    "classifier__C": [0.1, 1.0, 10, 100],
    "classifier__penalty":["elasticnet", "l1", "l2", None],
    "classifier__solver": ["lbfgs","newton-cg","saga"]
}
```

Os parâmetros *penalty* e *solver* são utilizados para especificar como o modelo deve ser treinado, enquanto o C é utilizado para controlar a força da regularização aplicada ao modelo [4].

2.3.1.2 Melhores parâmetros

Com base nas possíveis combinações definidas na grelha dos parâmetros, obteve-se que a melhor combinação para determinar o desempenho da LR é:

```
{'classifier__C': 0.1,
 'classifier__penalty': 'l1',
 'classifier__solver': 'saga'}
```

2.3.1.3 Gráfico ROC

A partir do gráfico ROC apresentado na Figura 2 verifica-se uma representação que indica um bom desempenho do algoritmo aplicado.

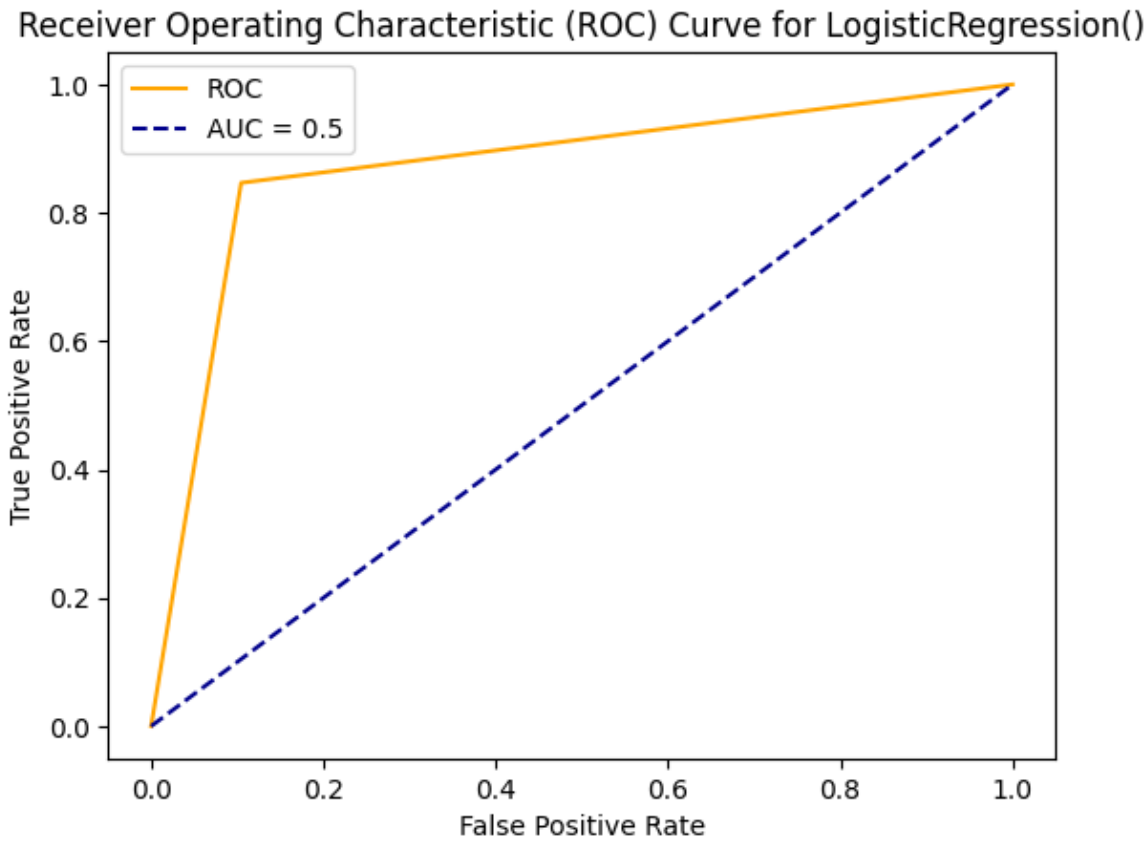


Figura 2. ROC para LR.

2.3.2 Support Vector Machine

O algoritmo *Support Vector Machine* (SVM) situa o hiperplano num espaço de alta dimensão que separa ao máximo as classes. A distância entre o hiperplano e os pontos de dados mais próximos é conhecida como a margem. O objetivo é encontrar um hiperplano com a margem máxima, uma vez que isto levará a um modelo com o menor erro de generalização (Apêndice I).

2.3.2.1 Parâmetros utilizados

```
param_grid = {
    "classifier__C": [0.1, 1.0, 10, 100],
    "classifier__penalty":["l1", "l2"],
    "classifier__loss" : ["hinge","squared_hinge"]
}
```

Os parâmetros *penalty* e *loss* são utilizados para especificar o tipo de regularização e função de perda a utilizar na formação do modelo, enquanto o C é utilizado para controlar a força da regularização aplicada ao modelo [5].

2.3.2.2 Melhores parâmetros

Com base nas possíveis combinações definidas na grelha dos parâmetros, obteve-se que a melhor combinação para determinar o desempenho do modelo SVM é:

```
{'classifier__C': 1.0,
 'classifier__loss': 'hinge',
 'classifier__penalty': 'l2'}
```

2.3.2.3 Gráfico ROC

A partir do gráfico ROC apresentado na Figura 3 verifica-se uma representação que indica um bom desempenho do algoritmo aplicado.

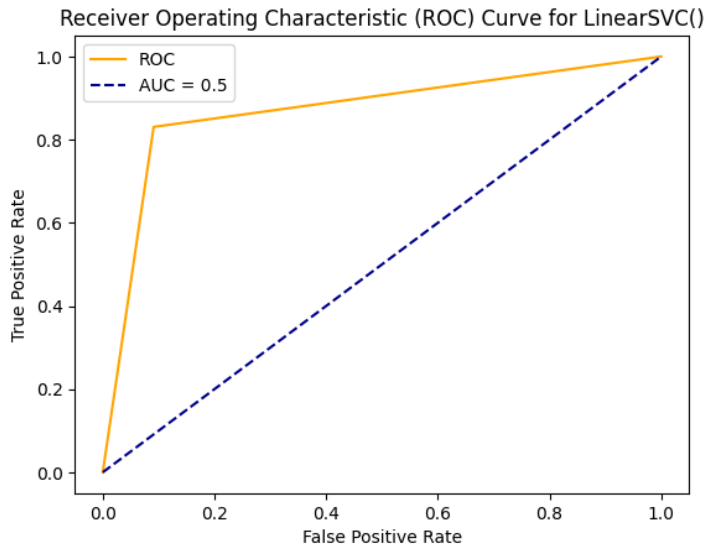


Figura 3. ROC para SVM.

2.3.3 K-Nearest Neighbors

O algoritmo *K-Nearest Neighbors* (KNN) prevê o atributo alvo para uma nova amostra, encontrando os vizinhos K mais próximos da amostra nos dados de formação e fazendo a maioria dos votos (para classificação) ou fazendo a média dos alvos (para regressão) (Apêndice I).

2.3.3.1 Parâmetros utilizados

```
param_grid = {
    "classifier__weights": ["uniform","distance"],
    "classifier__n_neighbors":list(range(1, 40)),
    "classifier__leaf_size": [10,20,30,40,50]
}
```

Os parâmetros *weights* e *n_neighbors* são usados para especificar como o modelo deve fazer previsões, enquanto o parâmetro de *leaf_size* é usado para controlar o desempenho do modelo. Todos estes são fatores importantes para determinar o desempenho do modelo KNN [6].

2.3.3.2 Melhores parâmetros

Com base nas possíveis combinações definidas na grelha dos parâmetros, obteve-se que a melhor combinação para determinar o desempenho do modelo KNN é:

```
{'classifier__leaf_size': 10,
 'classifier__n_neighbors': 11,
 'classifier__weights': 'distance'}
```

2.3.3.3 Gráfico ROC

A partir do gráfico ROC apresentado na Figura 4 verifica-se uma representação que indica um bom desempenho do algoritmo aplicado.

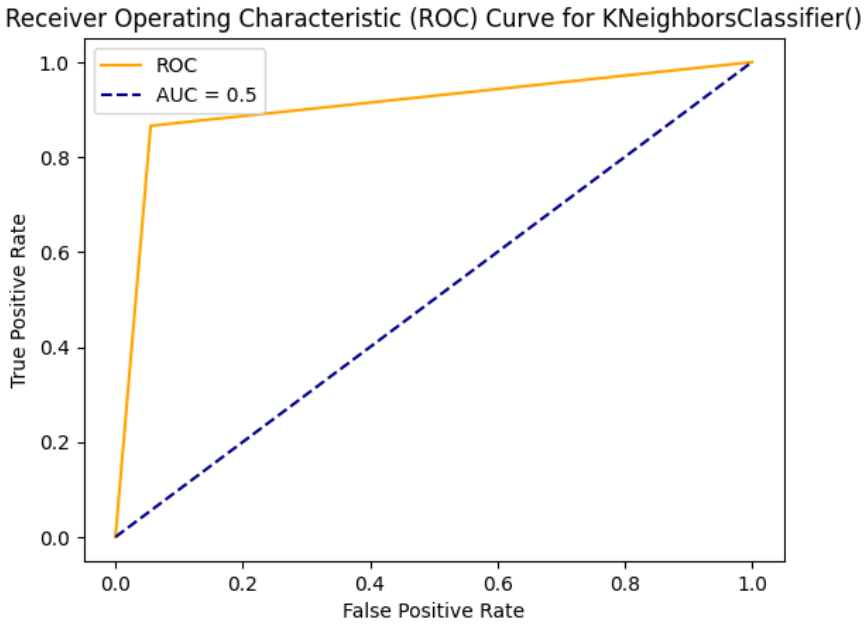


Figura 4. ROC para KNN.

2.3.4 Decision Tree

O algoritmo *Decision Tree* (DT) cria um modelo de decisões em forma de árvore que pode ser utilizado para prever o valor de um atributo alvo através da aprendizagem de regras de decisão simples inferidas a partir das características dos dados.

2.3.4.1 Parâmetros utilizados

```
param_grid = {
    "classifier__criterion": ["gini", "entropy"],
    "classifier__max_depth":list(range(1, 10)),
    "classifier__max_features": ["sqrt", "log2", "auto", 22]
}
```

Os parâmetros *criterion*, *max_depth* e *max_features* são fatores importantes para determinar o desempenho do modelo DT [7].

2.3.4.2 Melhores parâmetros

Com base nas possíveis combinações definidas na grelha dos parâmetros, obteve-se que a melhor combinação para determinar o desempenho do modelo DT é:

```
{'classifier__criterion': 'entropy',
 'classifier__max_depth': 9,
 'classifier__max_features': 22}
```

2.3.4.3 Gráfico ROC

A partir do gráfico ROC apresentado na Figura 5 verifica-se uma representação que indica um bom desempenho do algoritmo aplicado.

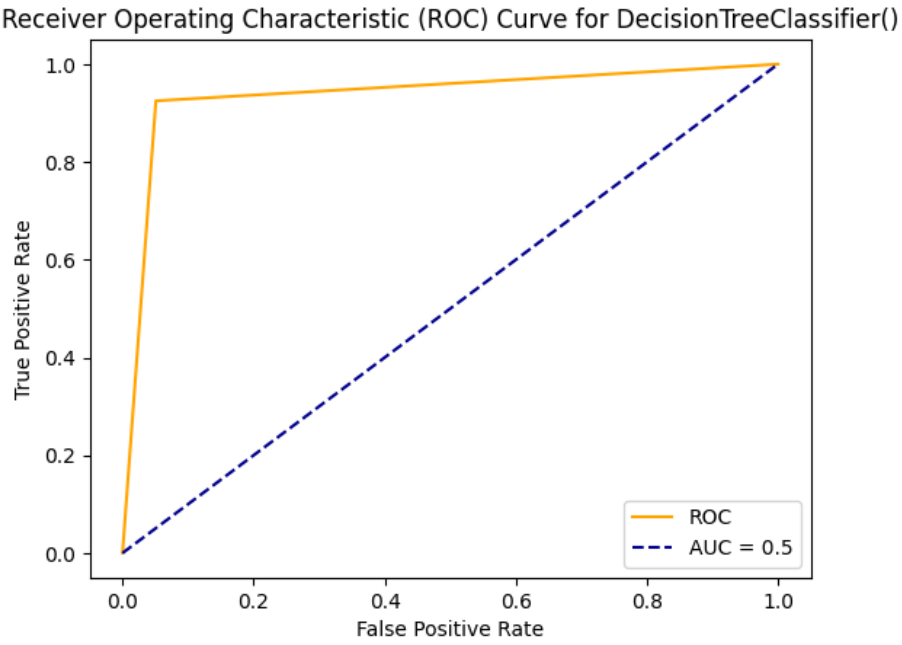


Figura 5. ROC para DT.

2.3.5 Naive Bayes

O algoritmo *Naive Bayes* (NB) prevê a *label* de classe de uma nova amostra utilizando as probabilidades das *labels* de classe e as probabilidades dos atributos dados as *labels* de classe.

2.3.5.1 Parâmetros utilizados

```
param_grid = {
    "classifier__alpha":list(range(0,40)),
    "classifier__fit_prior":[True,False]
}
```

Os parâmetros *alpha* e *fit_prior* fazem parte da classe *NaiveBayes* em *scikit-learn,* que fornece implementações de vários tipos de classificadores NB, incluindo *GaussianNB*, *MultinomialNB* e *BernoulliNB*. Estes parâmetros são importantes porque controlam o comportamento do modelo NB e podem ter um impacto significativo no seu desempenho [8].

2.3.5.2 Melhores parâmetros

Com base nas possíveis combinações definidas na grelha dos parâmetros, obteve-se que a melhor combinação para determinar o desempenho do modelo NB é:

Modelo: *Multinomial NB*

```
{'classifier__alpha': 39, 'classifier__fit_prior': False}
```

2.3.6 Random Forest

O algoritmo *Random Forest* (RF) prevê através da agregação das previsões dos modelos de árvores de decisão individuais. A ideia é que ao treinar múltiplos modelos e ao combinar as suas previsões, a floresta aleatória pode fazer previsões mais precisas e estáveis do que uma única *decision tree* (Apêndice I).

2.3.6.1 Parâmetros utilizados

```
param_grid = {
    "classifier__criterion": ["gini", "entropy"],
    "classifier__n_estimators":[100,120,140],
    "classifier__max_depth": [30,50],
    "classifier__min_samples_split": [2,3,4,5],
    "classifier__min_samples_leaf": [2,3,4,5],
    "classifier__bootstrap": [True,False],
    "classifier__class_weight": [{0:1,1:1},{0:1,1:5},{0:1,1:3},"balanced"],
}
```

Os principais hiperparâmetros utilizados foram *criterion*, *n_estimators*, *max_depth*, *min_samples_split*, *min_samples_leaf*, *bootstrap* e *class_weight* [9].

2.3.6.2 Melhores parâmetros

Com base nas possíveis combinações definidas na grelha dos parâmetros, obteve-se que a melhor combinação para determinar o desempenho do modelo RF é:

```
{'classifier__bootstrap': False,
 'classifier__class_weight': {0: 1, 1: 1},
 'classifier__criterion': 'gini',
 'classifier__max_depth': 50,
 'classifier__min_samples_leaf': 2,
 'classifier__min_samples_split': 5,
 'classifier__n_estimators': 140}
```

2.3.7 Artificial Neural Network

O algoritmo *Artificial Neural Network* (ANN) prevê, aprendendo a reconhecer padrões nos dados de treino através da utilização de múltiplas camadas ocultas de neurónios interligados.

2.3.7.1 Parâmetros utilizados

```
param_grid = {
    "classifier__hidden_layer_sizes": [50,100,150],
    "classifier__activation":["logistic", "tanh", "relu"],
    "classifier__solver":["lbfgs", "sgd", "adam"],
    "classifier__learning_rate": ["constant", "adaptive"],
    "classifier__max_iter": [200,500,1000],

}
```

Os parâmetros utilizados foram *hidden_layer_sizes*, *activation*, *solver*, *learning_rate* e *max_iter* [10].

2.3.7.2 Melhores parâmetros

Com base nas possíveis combinações definidas na grelha dos parâmetros, obteve-se que a melhor combinação para determinar o desempenho do modelo ANN é:

```
{'classifier__activation': 'logistic',
 'classifier__hidden_layer_sizes': 50,
 'classifier__learning_rate': 'adaptive',
 'classifier__max_iter': 500,
 'classifier__solver': 'adam'}
```

2.4 Voting Classifier

O objetivo do classificador *VotingClassifier* (VL) é combinar classificadores de ML conceptualmente diferentes e utilizar um voto maioritário (*hard vote*) ou as probabilidades médias previstas (*soft vote*) para prever os *labels* das classes. Este classificador pode ser útil para um conjunto de modelos com o mesmo desempenho, a fim de equilibrar as suas fraquezas individuais.

2.5 Resultados

Com base nos algoritmos de SL previamente apresentados, foram obtidos os resultados apresentados na Tabela 1, observa-se que através do ensemble do classificador VC não se obtêm resultados mais elevados do que certos classificadores utilizados individualmente, nomeadamente RF e ANN.

Tabela 1. Resultados das métricas dos algoritmos de SL.

LR	SVM	KNN
Accuracy: 0.86 Precision: 0.87 Recall-Score: 0.86 F1-score: 0.85	Accuracy: 0.88 Precision: 0.88 Recall-Score: 0.84 F1-score: 0.86	Accuracy: 0.92 Precision: 0.91 Recall-Score: 0.87 F1-score: 0.89
DT	Multinomial NB	Gaussian NB
Accuracy: 0.93 Precision: 0.93 Recall-Score: 0.92 F1-score: 0.92	Accuracy: 0.82 Precision: 0.78 Recall-Score: 0.82 F1-score: 0.80	Accuracy: 0.81 Precision: 0.84 Recall-Score: 0.83 F1-score: 0.82
RF	ANN	VC
Accuracy: 0.95 Precision: 0.96 Recall-Score: 0.95 F1-score: 0.95	Accuracy: 0.94 Precision: 0.95 Recall-Score: 0.94 F1-score: 0.94	Accuracy: 0.95 Precision: 0.92 Recall-Score: 0.91 F1-score: 0.87

Para se visualizar estes resultados graficamente, elaborou-se o seguinte gráfico (Figura 6):

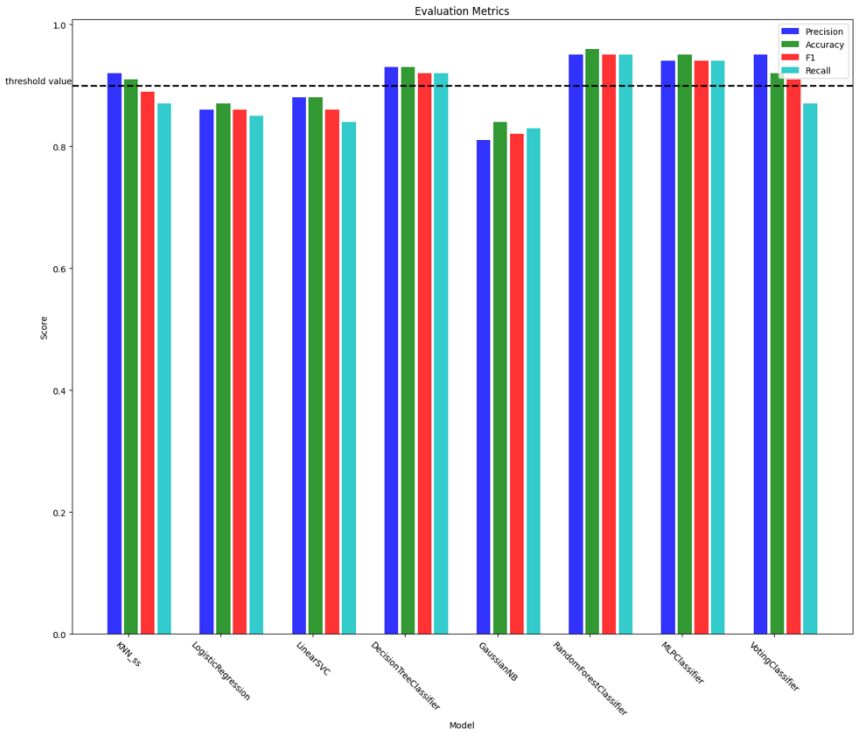


Figura 6. Métricas dos algoritmos de SL.

Foi estabelecido um valor (manual e aleatório) de *threshold* = 0.9. O intuito deste valor consistiu em verificar quais dos modelos tinham métricas inferiores a esse limite, e a partir dessa observação, selecionar esses modelos para utilizar na fase posterior (apresentada na secção 3.2.1).

3. Aprendizagem não supervisionada

Neste capítulo faz-se uso de algoritmos de UL para avaliar a possibilidade de melhorar as classificações obtidas na secção anterior com os modelos de SL. Posteriormente, estes algoritmos são também utilizados isoladamente para avaliar as performances obtidas.

3.1 Definição

Os algoritmos de UL baseiam o seu processo de treino num conjunto de dados sem *labels* previamente definidas.

No UL, o conjunto de treino consiste em *inputs* sem *labels*, ou seja, de *inputs* sem qualquer *output* desejado atribuído. Por exemplo, na Figura 7, os *inputs* são novamente pontos no plano bidimensional, mas não é proporcionada nenhuma indicação relativamente ao *output* correspondente. A UL visa geralmente descobrir as propriedades do mecanismo gerador dos dados. No exemplo dessa ilustração, o objetivo da UL consiste em agrupar 2 pontos de *input* que estão próximos uns dos outros, atribuindo assim uma *label* - o índice de *clustering* (agrupamento) - a cada ponto de *input* (os *clusters* são delimitados por linhas tracejadas) [11].

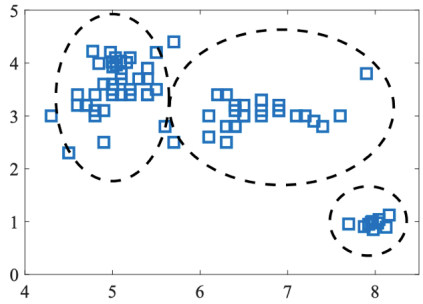


Figura 7. Ilustração de um problema de UL.

3.2 Clustering

O *clustering* consiste o processo de divisão de um grupo de pontos de dados em vários subgrupos, ou *clusters*, com base nos padrões dos dados. É usado para dividir os dados em grupos, de modo que os pontos dentro de cada agrupamento sejam mais semelhantes uns aos outros do que os pontos de outros agrupamentos (Figura 8) [12] [13].

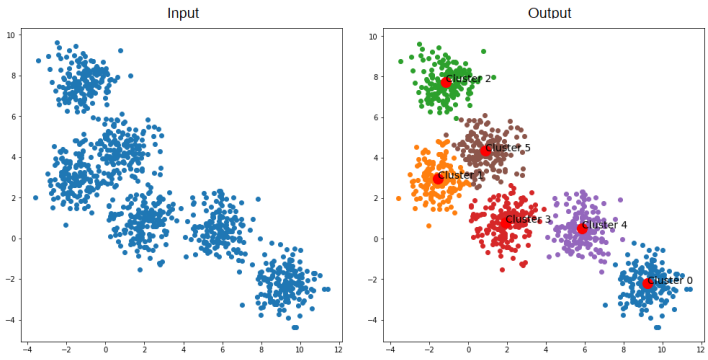


Figura 8. Representação gráfica do *clustering*.

3.2.1 Aplicação no pré-processamento da aprendizagem supervisionada

Com base nos resultados previamente obtidos ao longo da fase de SL, procedeu-se à aplicação de *clustering* no pré-processamento dos dados, com o objetivo de validar se o desempenho melhorava. Os modelos deSL utilizados nesta etapa, foram os modelos escolhidos anteriormente que obtiveram resultados inferiores ao *threshold* estabelecido, tendo sido obtidas as métricas apresentadas na Tabela 2:

Tabela 2 Resultados das métricas dos algoritmos de UL.

Kmeans + LR	Kmeans + SVM	Kmeans + KNN	Kmeans + Gaussian NB
Accuracy: 0.90 Precision: 0.90 Recall-Score: 0.87 F1-score: 0.89	Accuracy: 0.92 Precision: 0.91 Recall-Score: 0.87 F1-score: 0.80	Accuracy: 0.88 Precision: 0.90 Recall-Score: 0.9 F1-score: 0.89	Accuracy: 0.76 Precision: 0.80 Recall-Score: 0.80 F1-score: 0.78

Com base nos resultados da Tabela 2, observa-se que através do pré-processamento dos dados com algoritmos de UL (*KMEANS*) e posterior classificação com algoritmos de SL se obtêm resultados mais elevados para os modelos de LR e SVM do que previamente, sem pré-processamento. Relativamente aos restantes modelos os resultados obtidos não melhoraram. Na figura … observa-se os valores obtidos graficamente:

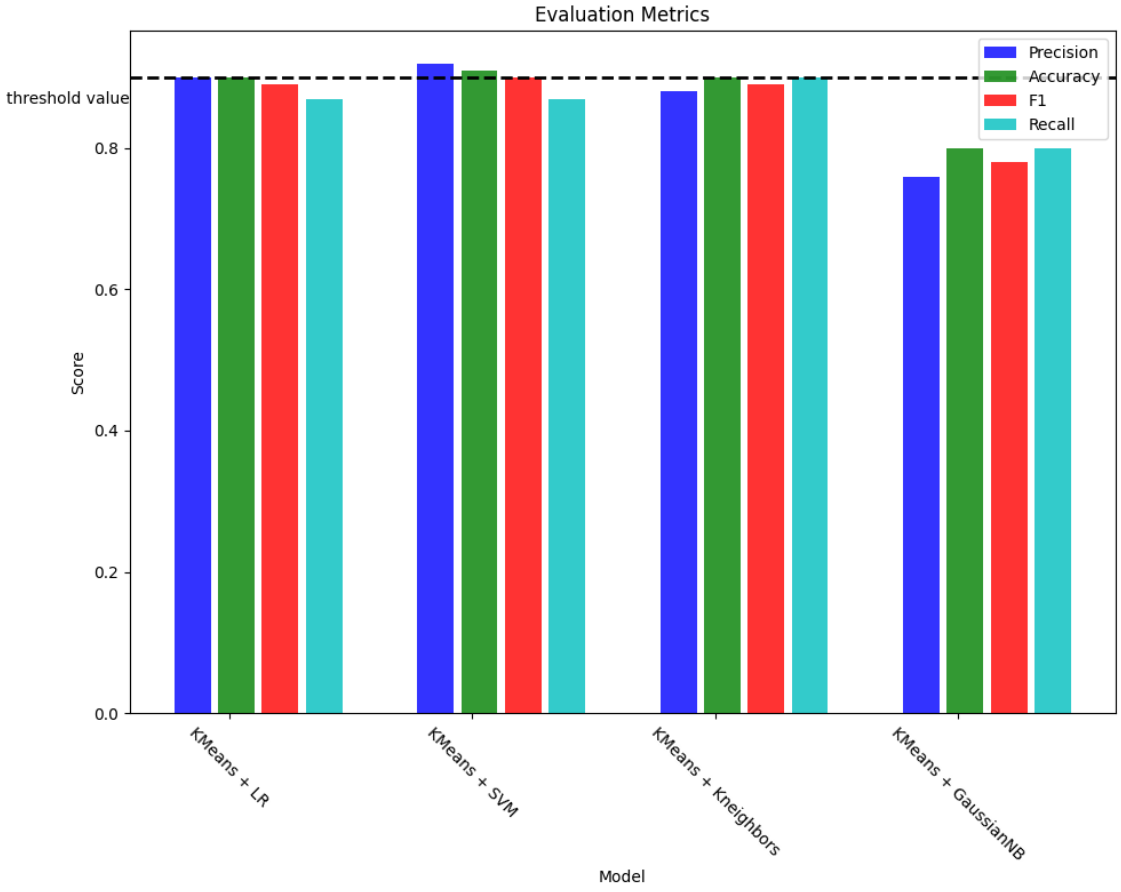


Figura 9. Métricas de SL após aplicação de *clustering* no pré-processamento.

3.2.2 BIRCH

BIRCH (*Balanced Iterative Reduction and Clustering using Hierarchies*) é um algoritmo de *clustering* que pode ser utilizado para dividir um conjunto de dados num número especificado de *clusters*. Funciona através da construção de uma representação dos dados baseada em árvores e utilizando esta representação para realizar o agrupamento hierárquico. Ao utilizar este algoritmo obtiveram-se os resultados apresentados na Figura 10.

```
# BIRCH isolado
birch=Birch(n_clusters=3)

trainModel(birch, False, True, False)
```

	Accuracy	Precision	Recall	F1	Cross Validation	Silhouette	Homogeneity	Completeness	ARI	NMI
0	0.555	0.555	0.555	0.555	NaN	0.048834	0.183009	0.128742	0.114498	0.151152

Figura 10. Resultados da aplicação do BIRCH.

Observa-se que os resultados obtidos a partir do algoritmo de UL não foram superiores aos obtidos anteriormente, quer com algoritmos de SL individualmente, quer com algoritmos de UL como pré-processamento dos dados.

3.2.3 Redução de dimensionalidade

Nesta etapa, utilizou-se o algoritmo *BIRCH* precedido do algoritmo *t-SNE* como uma forma de analisar o conjunto de dados com múltiplas características para identificar padrões ou grupos dentro dos dados. O algoritmo *t-SNE* é utilizado para reduzir o número de dimensões nos dados, o que facilita a sua visualização e compreensão. O algoritmo *BIRCH* é então utilizado para identificar clusters ou grupos dentro dos dados reduzidos, o que pode ajudar a descobrir padrões ou tendências que podem não ter sido óbvios nos dados originais de alta dimensão. Após ser efetuada esta *pipeline,* obteve-se a visualização Figura 11:

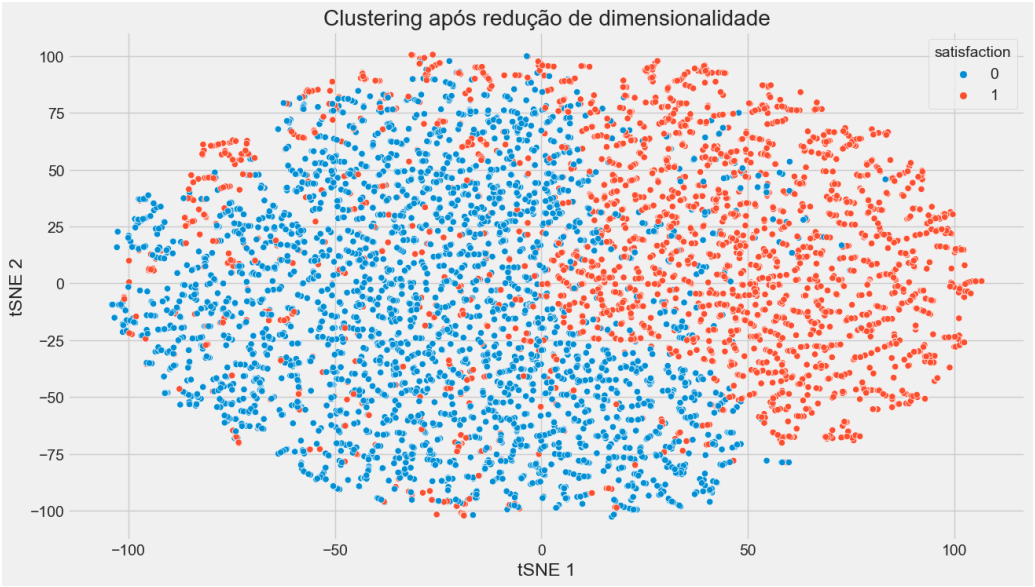


Figura 11. Redução de dimensionalidade.

Aquí, é bastante clara a utilidade do algoritmo *t-SNE,* pois após a redução da dimensionalidade do *dataset*, o *clustering* final efetuado pelo algoritmo *BIRCH* possibilita uma divisão claro dos dados em dois grupos: passageiros satisfeitos e não satisfeitos.

4. Conclusões

Neste trabalho, foram abordados vários algoritmos de SL e UL, sendo descrito o processo desde a preparação dos dados para os algoritmos, até à seleção e aplicação destes ao contexto do *dataset* em estudo.

De uma forma geral, os algoritmos de SL aplicados apresentaram resultados bastante positivos, relembrando o facto do algoritmo *Linear Regression* não ser utilizado, devido à natureza do atributo alvo do *dataset* e do número de atributos, respetivamente.

Verificou-se que o pré-processamento com a integração de algoritmos de UL melhorou o desempenho do algoritmo de SL, na maioria das situações. Por outro lado, o uso de algoritmos de UL de forma isolada obteve valores bastante insatisfatórios, muito abaixo dos obtidos anteriormente. Por outro lado, o uso de algoritmos de UL para efetuar redução da dimensionalidade do nosso *dataset* (reduzir o número de *features*) com a posterior aplicação do algoritmo de clustering mostrou-se bastante efetivo, permitindo a identificação clara do nosso atributo alvo.

Em resumo, o objetivo do trabalho foi concluído com sucesso, sendo possível pôr em prática os conhecimentos adquiridos acerca de várias abordagens a tomar na classificação do *dataset* e da implementação dos respetivos algoritmos, quer de SL quer de UL.

5. Bibliografia

[1] Cunningham, P., Cord, M., Delany, S. (2008). *Supervised learning. In Machine learning techniques for multimedia* (pp. 21-49). Springer, Berlin, Heidelberg.

[2] Paredes, S. (2022). *Aulas Práticas* [Slides de PowerPoint]. Departamento de Engenharia Informática e de Sistemas, Instituto Superior de Engenharia de Coimbra

[3] Paredes, S. (2022). *Aulas Teóricas* [Slides de PowerPoint]. Departamento de Engenharia Informática e de Sistemas, Instituto Superior de Engenharia de Coimbra

[4] Scikit-learn (s.d). *Logistic Regression*. Acedido em 10 de dezembro de 2022. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[5] Scikit-learn (s.d). *SVC*. Acedido em 10 de dezembro de 2022. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[6] Scikit-learn (s.d). *K-Neighbors Classifier.* Acedido em 13 de dezembro de 2022. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[7] Scikit-learn (s.d). *Decision Tree Classifier.* Acedido em 13 de dezembro de 2022. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[8] Scikit-learn (s.d). *Naive Bayes.* Acedido em 13 de dezembro de 2022. Disponível em: https://scikit-learn.org/stable/modules/classes.html#module-sklearn.naive_bayes

[9] Scikit-learn (s.d). *Random Forest Classifier.* Acedido em 13 de dezembro de 2022. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[10] Scikit-learn (s.d). *Neural Networks Supervised.* Acedido em 13 de dezembro de 2022. Disponível em: https://scikit-learn.org/stable/modules/neural_networks_supervised.html

[11] Goodfellow, I., Bengio, Y. (2016). *Deep learning.* MIT press Cambridge, vol. 1.

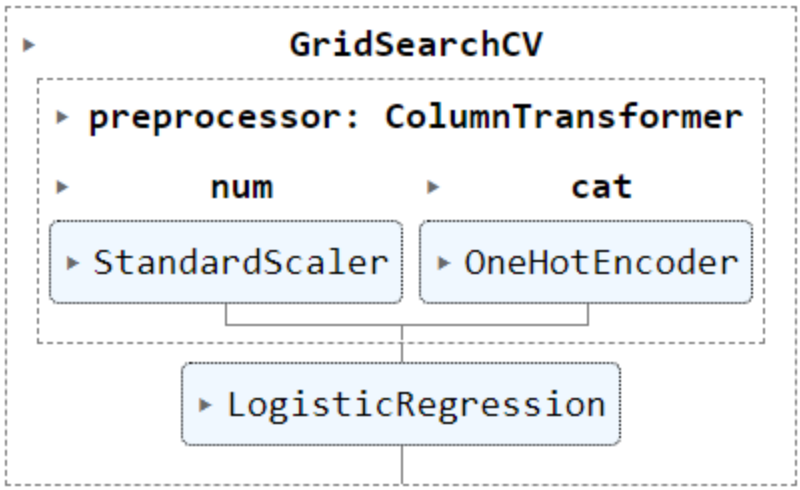
[12] Toushik, A. (2022). *Introduction to Clustering: Discussion about different types of Clustering.* Acedido em 20 de dezembro de 2022. Disponível em: https://medium.com/@azmine_wasi/introduction-to-clustering-discussion-about-different-types-of-clustering-dd6af9fbfc21

[13] Dongkuan, X., Yingjie, T. (2015). *A Comprehensive Survey of Clustering Algorithms*. Springer-Verlag Berlin Heidelberg. Ann. Data. Sci. (2015) 2(2):165–193

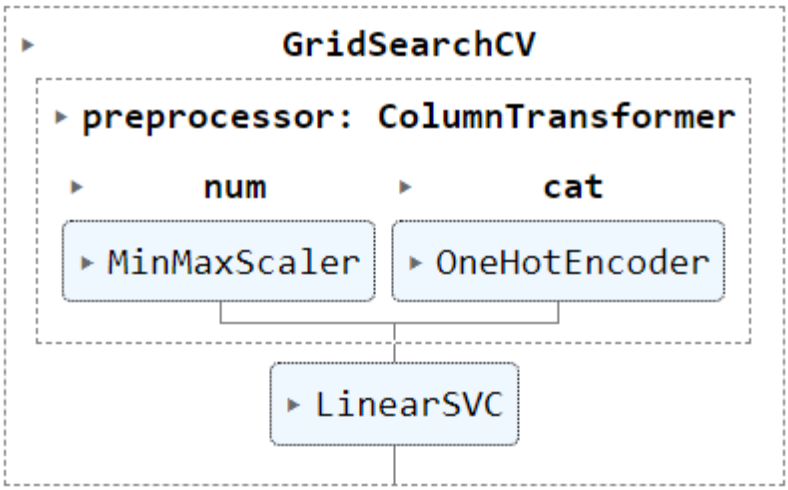
APÊNDICES

Apêndice I Arquitetura dos modelos aplicado.

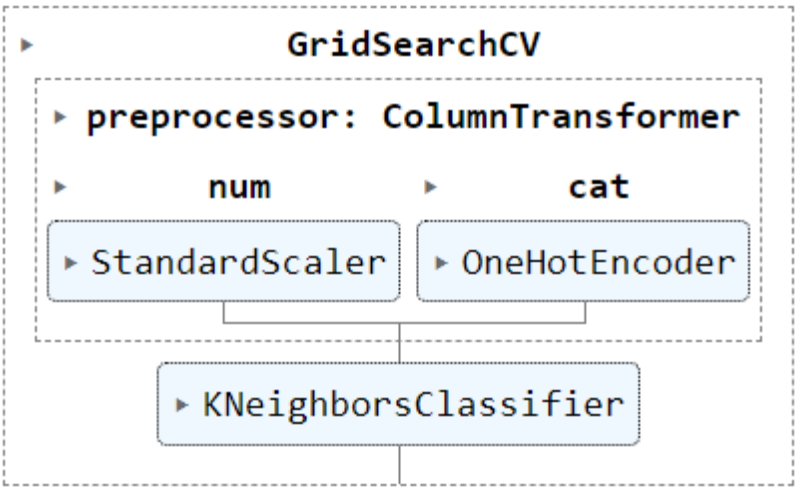
- *Logistic Regression*



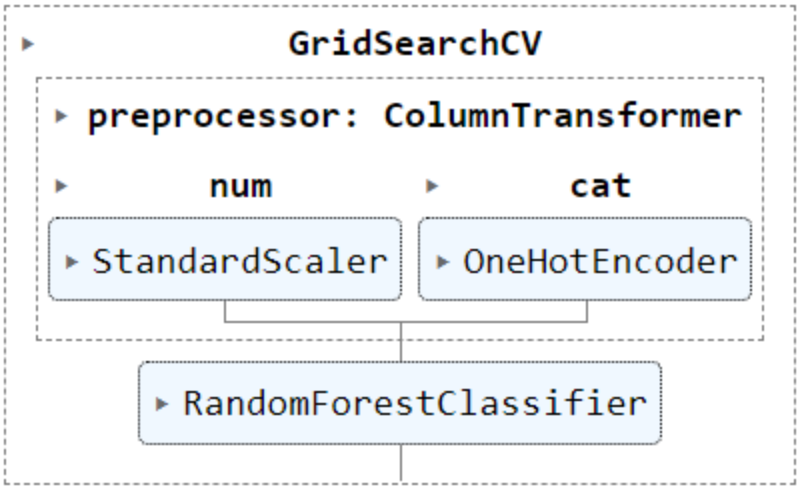
- *Support Vector Machines*



- *K-Nearest Neighbors*



- *Random Forest*



- *Artificial Neural Network*