Sistemas de Aprendizagem

André Figueiredo, Luís Ferreira, Pedro Machado e Rafael Lourenço
e-mail: {a84807,a86265,a83719,a86266}@alunos.uminho.pt

Universidade do Minho, Departamento de Informática, 4710-057 Braga, Portugal
Aprendizagem e Extração de Conhecimento, Trabalho prático Nº2

10 de Janeiro de 2021

Resumo Neste relatório serão abordadas as várias formas de tratamento de dados realizadas, bem como os modelos preditivos utilizados e a análise de resultados consequente.
O *dataset* trabalhado tem como objetivo classificar os vários indivíduos conforme os seus salários, sendo que estes serão diferenciados consoante recebam um valor anual superior ou inferior/igual a 50 mil unidades monetárias.
Desta forma, desenvolveram-se não só vários modelos preditivos mais tradicionais e simples, como *Decision Tree* e *Naive Bayes*, mas também modelos mais complexos e resultantes de técnicas de *ensemble learning*, como *Random Forest* e *AdaBoost*.
Por fim, iremos avaliar os resultados dos modelos e tratamentos usados, decidindo qual o melhor modelo para este problema.

Keywords: Machine Learning · SEMMA · Naive Bayes · Support Vector Machines · K Nearest Neighbors · Decision Tree · Random Forest · AdaBoost · Stacking · Imbalanced Data · Python

1 Introdução

No âmbito da Unidade Curricular de Análise e Extração do Conhecimento, foi-nos proposta a realização de um trabalho prático cujo principal objetivo passa por preparar um *dataset* relativo às características dos funcionários de múltiplas empresas, posteriormente trabalhado através de modelos preditivos, de tal forma que seja possível antever o nível salarial anual de um dado conjunto de indivíduos.

Para a realização deste projeto, seguimos a metodologia `SEMMA` [1], desenvolvida pelo *SAS*, que divide o processo em cinco etapas: *Amostragem*, *Exploração*, *Modificação*, *Modelação* e *Avaliação*, dando, assim, a conhecer todos os processos efetuados no desenvolvimento. Uma das razões para o uso deste paradigma, prende-se com a ausência do estudo de negócio, presente no paradigma `CRISP-DM`. [2]

Numa primeira fase, pretende-se preparar o *dataset*, eliminando informação desnecessária para um *target* em concreto (que, neste caso, consiste na classificação do salário de um funcionário ser superior ou inferior a 50 mil unidades monetárias por ano), normalizar os dados, categorizar os atributos necessários, verificar *outliers*, etc.
Posteriormente, serão usados modelos preditivos de *Machine Learning*, uma vez que tiram partido de algoritmos estatísticos para antecipar acontecimentos futuros, sendo que estes são expostos a dados de treino antes de processarem dados novos.
Desta forma, serão aplicados vários modelos de classificação (como, por exemplo, *K Nearest Neighbors* e *Decision Tree*), com o intuito de obter o algoritmo ótimo.
Por fim, será feita a análise de resultados onde é, então, comparada a *performance* dos modelos utilizados através de algumas métricas de avaliação, nomeadamente a *accuracy*, precisão de escolha, entre outras.

Este projeto foi desenvolvido através da plataforma *Jupyter Notebook*, uma aplicação *open-source*, que possibilita o tratamento dos dados e a aplicação dos modelos, em *Python*.

2 Análise e Pré-Processamento do *dataset*

Como referido anteriormente, o caso em estudo consiste na análise do salário dos funcionários de várias empresas. Para isso, foi-nos fornecido dois *datasets*, um para treino e outro para teste, sendo que ambos possuem funcionários classificados em quinze atributos distintos (colunas do *dataset*). Desta forma, passamos a descrever as *features*:

1. **age** - Idade de um funcionário;
2. **workclass** - Tipo de entidade à qual o funcionário presta serviço;
3. **fnlwgt** - Peso representativo do funcionário no *dataset*;
4. **education** - Grau académico;
5. **education-num** - Grau académico numérico (atributo ordinal);
6. **marital-status** - Estado civil;
7. **occupation** - Profissão;
8. **relationship** - Representação familiar;
9. **race** - Raça;
10. **sex** - Género;
11. **capital-gain** - Ganho de capital anual;
12. **capital-loss** - Perda de capital anual;
13. **hours-per-week** - Número de horas de serviço por semana;
14. **native-country** - Nacionalidade;
15. **salary-classification** - Classificação salarial anual (mais ou menor/igual que 50 mil unidades monetárias).

No que toca aos atributos do *dataset*, tivemos alguma dificuldade em associar um significado às variáveis **fnlwgt**, **capital-loss** e **capital-gain**.
No caso da variável **fnlwgt**, este valor corresponde ao número de indivíduos que aquela instância representa. Por exemplo, a instância `[43, black, 1000, ...]` corresponde a 1000 indivíduos com raça negra e 43 anos.

Posto isto, é necessário examinar a informação, de forma a eliminar e/ou modificar *features*, para que possamos otimizar a *performance* dos modelos.
Um dos primeiros aspetos a realçar prende-se no facto deste conjunto de dados conter uma percentagem insignificante de valores nulos (cerca de 5% na *feature* **workclass**, p.e.), como se pode verificar na figura 1. Estes registos não foram retirados, levando ao preenchimento dos *missing values* com um código específico aquando da transformação de *features* nominais para numéricas[1].

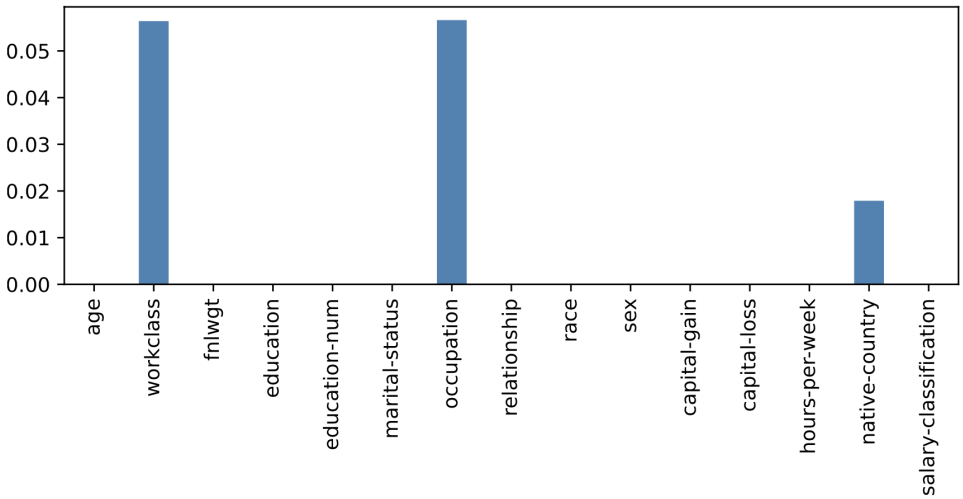


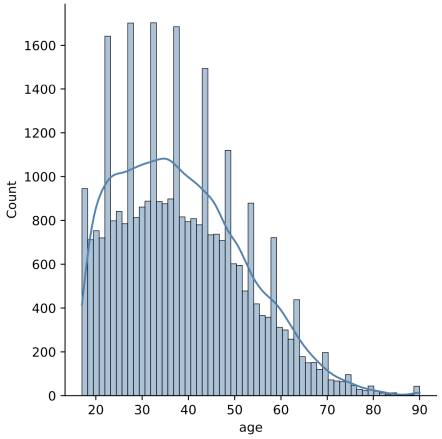
Figura 1: Percentagem de valores nulos para cada variável

[1] Também foi experimentada a substituição destes valores em falta pela moda, cujos resultados se encontram na secção de análise de resultados (6).

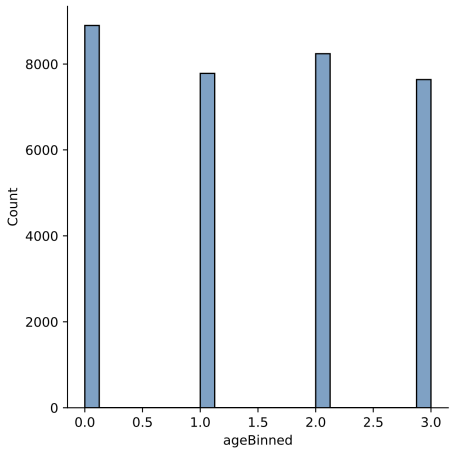
Um dos motivos para a linha em questão do *dataset* não ter sido retirada, deve-se ao facto de apenas três *features* possuírem *missing values*, o que, no nosso entender, não justifica a perda de dados das restantes *features*. Outro ponto a ter em conta, é o facto dos dados de teste também possuírem valores semelhantes em falta, todavia não faz sentido a sua remoção, visto que estamos a querer prevê-los.

Posto isto, começámos por analisar a idade dos funcionários [figura 2a] e categorizá-la em diversos intervalos [figura 2c].
Através de múltiplas iterações com diferentes modelos preditivos, bem como com a distribuição das idades dos funcionários, chegou-se à conclusão de que a discretização em quatro intervalos com igual frequência seria um bom tratamento para esta *feature*, pelo que considerámos os seguintes intervalos de idade e os seus códigos:

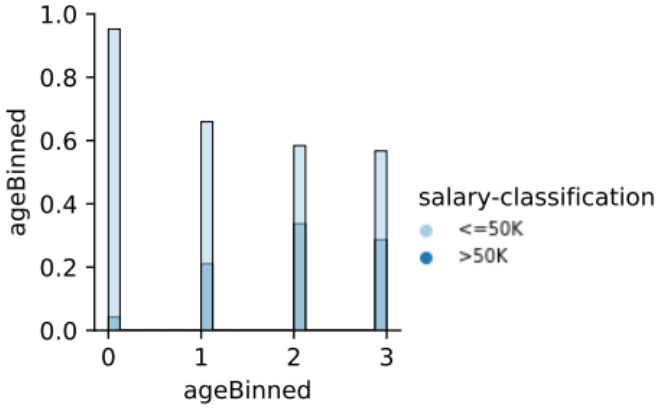
$$0 \rightarrow [17, 28],\ 1 \rightarrow\ ]28, 37],\ 2 \rightarrow\ ]37, 48] \text{ e } 3 \rightarrow\ ]48, 90]$$



(a) Distribuição inicial das idades



(b) Distribuição categorizada das idades

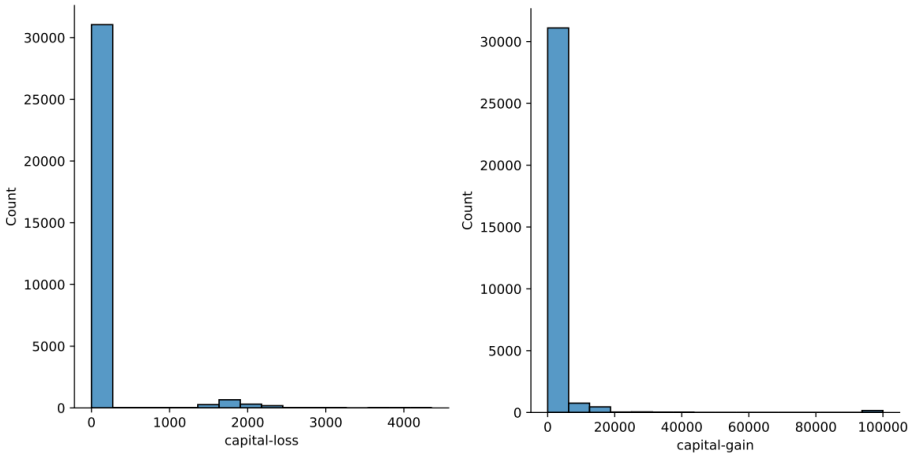


(c) Salário anual por grupo de idade

Figura 2: Tratamento realizado ao atributo **age** do *dataset*

Na figura 2c verifica-se que a probabilidade de um indivíduo receber um salário superior é mais acentuada nos grupos etários 2 e 3, que representam as faixas etárias mais velhas, visto que o gráfico de barras possui uma maior representação dessa classe para esses intervalos de idade. Analogamente, os indivíduos mais jovens tendem a receber menos.

De seguida, foram executadas várias tentativas de tratamento que, após a realização de modelos preditivos e da sua otimização através de um pré-processamento mais adequado, se revelaram irrelevantes ou até prejudiciais. Como exemplo disso, temos as *features* **capital-loss** e **capital-gain** que, após a realização da baixa dispersão dos dados para ambos os atributos numéricos, foram executadas discretizações e normalizações que não obtiveram os resultados esperados no que diz respeito à *performance* dos modelos. Assim, estes atributos não sofreram estes tratamentos, sendo apenas agregados através da subtração do **capital-gain** pelo **capital-loss** (*feature* **capital**).

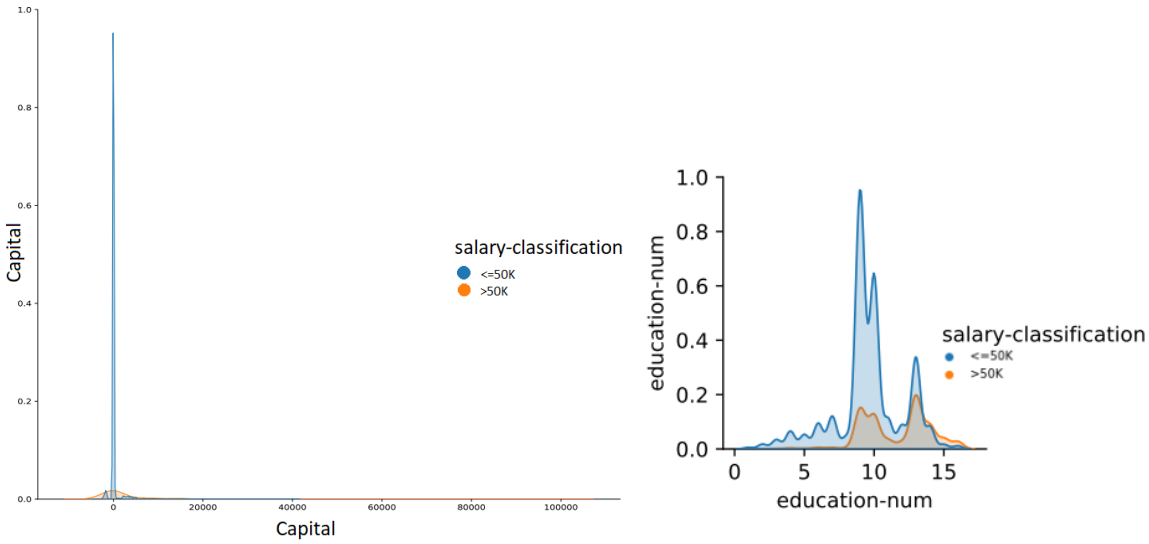


(a) Distribuição do **capital-loss** (b) Distribuição do **capital-gain**

Figura 3

```
training['capital'] = training['capital-gain'] -
    training['capital-loss']

test['capital'] = test['capital-gain'] - test['capital-loss']
```



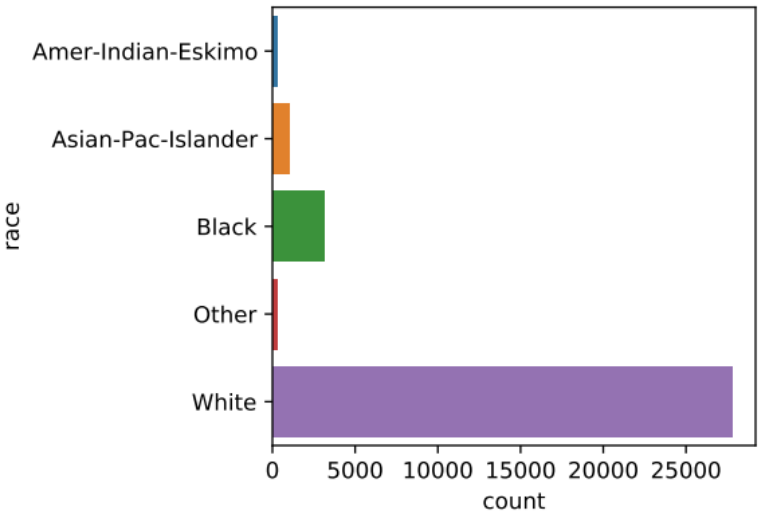
(a) Capital pelo salário (b) Grau de educação pelo salário

Figura 4

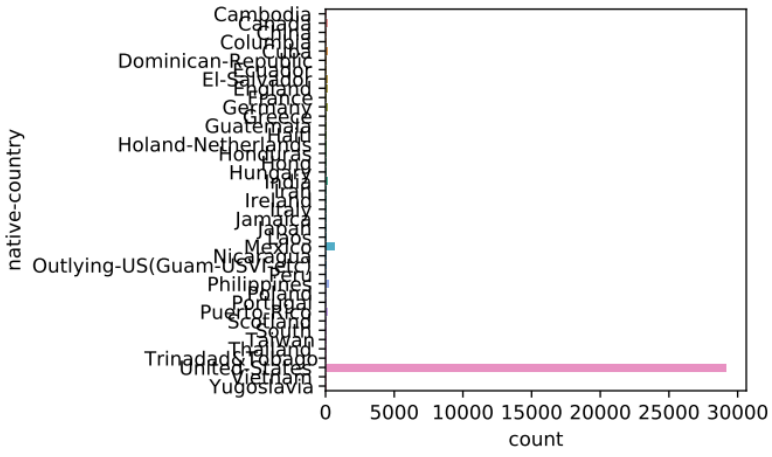
A partir destas imagens, conseguimos depreender que valores de capital muito elevados correspondem a indivíduos com um salário superior e que um grau de educação maior

também contribui significativamente para tal. É de realçar que, para a educação, utilizámos um gráfico de densidade que, apesar de estar incorreto, pois considera a *feature* **education-num** como contínua, é aquele em que a análise se compreende com maior facilidade.

Com o tratamento do **capital** efetuado, os atributos nominais foram visualizados através de gráficos de barras de forma a compreender a sua distribuição, sendo de destacar os seguintes:



(a) Distribuição das raças no *dataset*



(b) Distribuição das nacionalidades no *dataset*

Figura 5

Nestas figuras verifica-se que existe uma baixa dispersão dos dados, sendo que poderão ser alvo de remoção, pois podem não transmitir valor informativo relevante para os modelos preditivos (efetivamente, após a realização dos modelos preditivos, estas *features* acabaram mesmo por ser removidas do *dataset*).

Para além das duas *features* representadas, foram analisadas todas as outras *features* nominais, pelo que optámos por apenas destacar o **sex**. Assim, sabemos que existe uma maioria masculina no *dataset*[2].

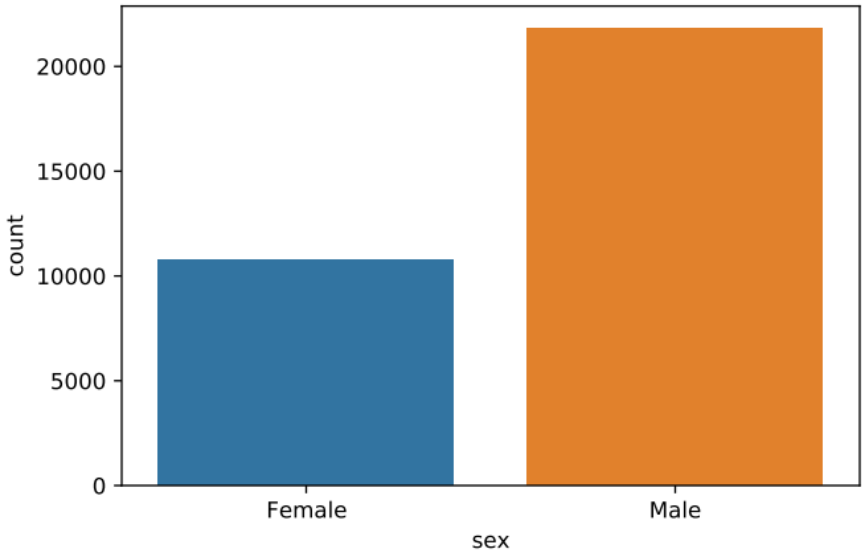


Figura 6: Distribuição do género dos indivíduos no *dataset*

De seguida e com vista a verificar a correlação entre os atributos e a *feature target* (**salary-classification**), foi necessário transformar as variáveis nominais em numéricas, através da atribuição de códigos para cada valor.

```
trainingNBGaussian['workclass'] =
    trainingNBGaussian['workclass'].astype('category')
                                   .cat.codes

trainingNBGaussian['sex'] =
    trainingNBGaussian['sex'].astype('category')
                             .cat.codes
```

É importante realçar que as *features* qualitativas não deveriam ter sido analisadas através desta forma de correlação, mas sim por associação através de um teste de *qui-quadrado*. Apesar disto, optámos por proceder com esta resolução, pois facilitou a análise e compreensão.

[2] Todas as outras visualizações de *features* podem ser observadas no *notebook* em anexo.

Assim, foi obtida a seguinte matriz correlação:

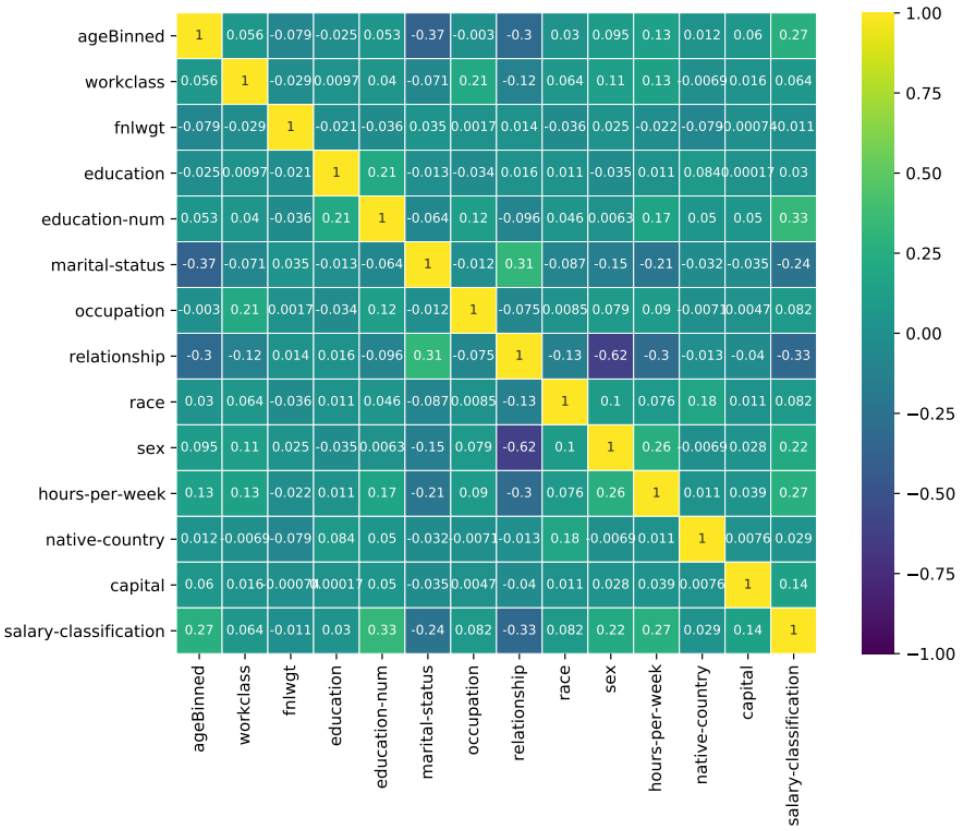


Figura 7: Matriz correlação do *dataset*

A matriz de correlação indica a correlação de atributos do *dataset*, sendo que se existir dois atributos com uma correlação muito próxima de 1 ou -1, podemos concluir que ambas transmitem uma informação semelhante para o futuro modelo preditivo, ou seja, é útil a permanência de *features* com muita correlação com a coluna *target*.
Através desta matriz, é possível verificar que *features* como a **fnlwgt**, **race**, **native-country**, **workclass**, **education** e **occupation** possuem correlações muito baixas com o *target*, pelo que poderão ser alvo de remoção.
Por outro lado, a *feature* **relationship** e **education-num** possuem as maiores correlações, pelo que traduzem melhor a variação do **salary-classification**. Neste caso quando a *feature* **education-num** aumenta, o **salary-classification** tende também a aumentar.

Após a análise das correlações das *features* presentes no *dataset*, podemos começar por verificar quais as *features* com um possível valor informativo menor para os modelos preditivos. Esta análise foi executada mediantes os valores de correlação, a dispersão dos valores dos atributos, a análise dos resultados dos modelos e a *feature importance* destes.

Desta forma, as *features* **education**, **native-country**, **fnlwgt** e **race** foram removidas, devido à dispersão dos seus valores e à correlação associada ao *target*.

Após a construção dos modelos e com o propósito de decidir, em conjunto com os valores de correlação, quais os atributos que têm menor valor informativo, analisámos a *feature importance* de todos atributos do *dataset* com o modelo *AdaBoost*:

```
Feature: 0, Score: 0.06000
Feature: 1, Score: 0.02000
Feature: 2, Score: 0.04000
Feature: 3, Score: 0.04000
Feature: 4, Score: 0.12000
Feature: 5, Score: 0.04000
Feature: 6, Score: 0.14000
Feature: 7, Score: 0.12000
Feature: 8, Score: 0.02000
Feature: 9, Score: 0.04000
Feature: 10, Score: 0.08000
Feature: 11, Score: 0.22000
Feature: 12, Score: 0.06000
Feature: 13, Score: 0.00000
```

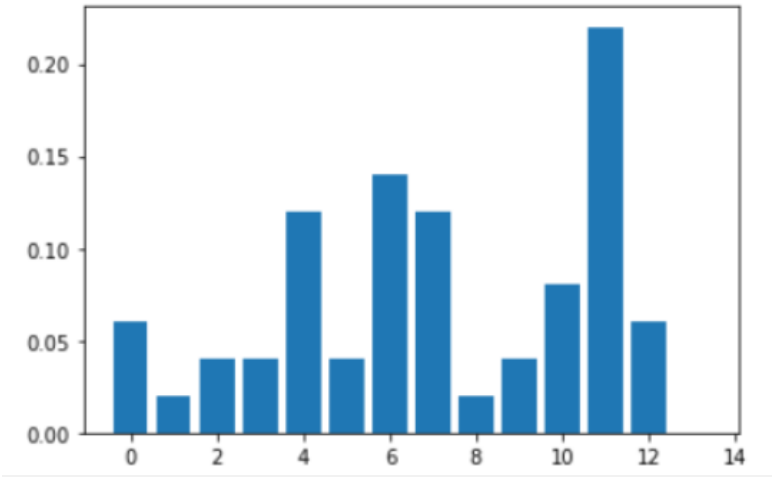


Figura 8: *Feature importance* através de *AdaBoost* com *Decision Tree*

Como é observável, as *features* **marital-status** (5) e **sex** (9) encontram-se nas *features* com menor valor informativo do modelo *AdaBoost* com *Decision Tree*. Contudo, a remoção destas duas *features* não teve bons resultados, pelo que se optou pela sua permanência no *dataset*.

Assim sendo, apenas existiu a remoção das seguintes *features* do *dataset* aquando do tratamento[3]:

- **education**;
- **native-country**;
- **fnlwgt**;
- **race**.

[3] Nas restantes *features* apenas foi discretizado o atributo da idade e a agregação do capital, como foi previamente referido.

Portanto, o tratamento dos dados foi concluído e encontrou-se, no nosso entender, num estado próximo do ideal para a posterior fase de atuação de modelos preditivos.

Em suma, o formato do *dataset* de treino após o tratamento foi o seguinte:

	ageBinned	workclass	education-num	marital-status	occupation	relationship	sex	hours-per-week	capital
0	2	6	13	4	0	1	1	40	2174
1	3	5	13	2	3	0	1	13	0
2	2	3	9	0	5	1	1	40	0
3	3	3	7	2	5	0	1	40	0
4	0	3	13	2	9	5	0	40	0
...	...	...	...	...	...	...	...	...	...
32556	0	3	12	2	12	5	0	38	0
32557	2	3	9	2	6	0	1	40	0
32558	3	3	9	6	0	4	0	40	0
32559	0	3	9	4	0	3	1	20	0
32560	3	4	9	2	3	5	0	40	15024

Figura 9: Dados após o tratamento

É necessário referir que a *feature target* (**salary-classification**) tornou-se num atributo binário, sendo que 0 corresponde a "$\leq 50k$" e 1 a "$> 50k$".

Poderá ser interessante visualizar os dados do *dataset* de treino apenas com duas dimensões, pelo que decidimos utilizar o *PCA*, com vista a reduzir as dimensões dos dados:

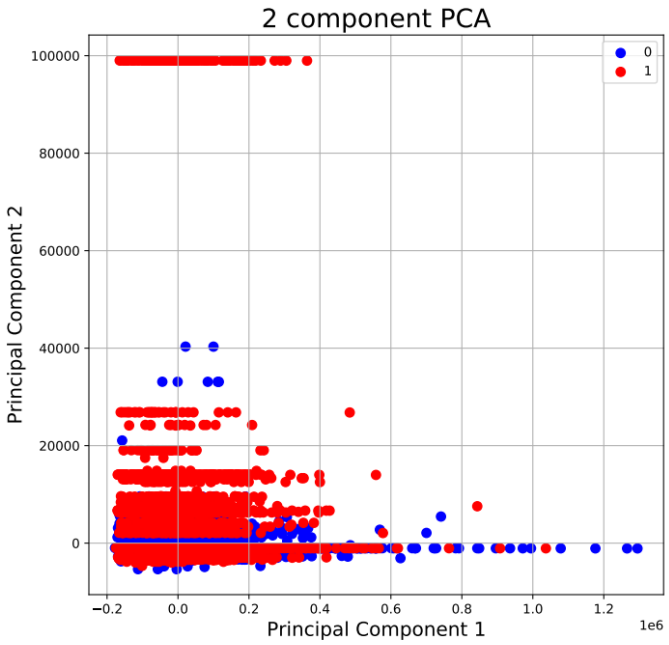
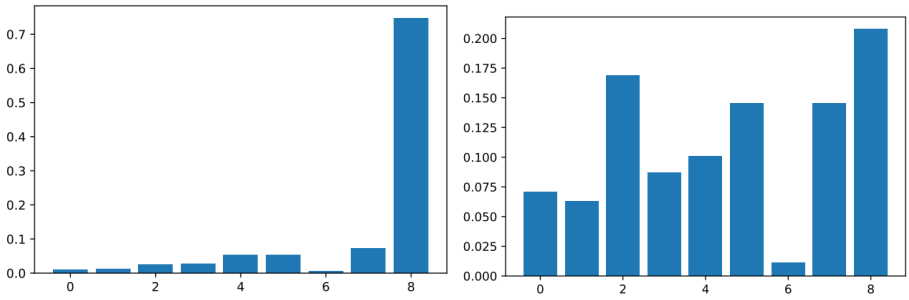


Figura 10: Dados em duas dimensões

É de notar uma concentração de dados bem visível da classe "$> 50k$" (1) no lado superior, enquanto que, no resto dos dados, tem-se uma perceção menor para a distinção das duas classes devido à alta concentração.
Deste modo, a separação das duas classes, através de um discriminante p.e., não teria bons resultados. Outro dado importante é que também não existem *clusters* óbvios à primeira vista, se não contarmos com os casos da classe 1 no canto superior esquerdo (visto que estes poderiam formam um *cluster*).

Segue-se, ainda, a *feature importance* obtida com o *dataset* após tratamento:



(a) Importância de cada *feature* no *AdaBoost* com *Decision tree*

(b) Importância de cada *feature* no *Random Forest*

Figura 11

As figuras prévias comprovam a nossa interpretação em relação ao **capital** (8) da figura 4b, visto ser das *features* com maior importância para estes dois modelos preditivos. Também é percetível que a sexta *feature* não traz um valor muito relevante para os modelos, contudo, a remoção deste atributo não se revelou a melhor solução, visto que influenciava negativamente a *performance* do modelo.

3 Modelos Preditivos

Antes de iniciarmos a enumeração dos modelos desenvolvidos, é necessário referir que todos os modelos usaram o mesmo *dataset* após tratamento, de forma a podermos comparar os resultados entre eles, isto é, todos os modelos utilizaram um *dataset* apenas com *features* numéricas[4].

3.1 *Naive Bayes*

O modelo de *Naive Bayes* baseia-se no teorema de *Bayes*, equação que descreve relações de probabilidades condicionais de quartis estatísticos. Neste teorema assume-se a independência das *features* do *dataset* de forma *naive*.
Este modelo de *Machine Learning* é altamente escalável, rápido e simples. Devido à sua rapidez e escassez de parâmetros, acaba por ser muito usado para previsões num curto espaço de tempo, mas com pouco grau de confiança. [3]

Para este problema, foi utilizado o *Naive Bayes Gaussiano*, que assume que cada atributo possui uma distribuição gaussiana (normal) nos dados.

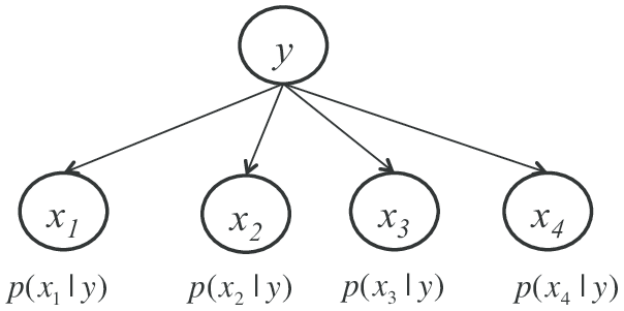


Figura 12: Exemplo de *Naive Bayes*

Segue-se um excerto da aplicação deste modelo em *Python*:

```
pipeline = make_pipeline(
    QuantileTransformer(
        output_distribution = 'normal'),
        GaussianNB())
pipeline.fit(train_x, train_y)
```

```
[[10475  1960]
 [ 1201  2645]]
              precision    recall  f1-score   support

           0      0.897     0.842     0.869     12435
           1      0.574     0.688     0.626      3846

    accuracy                          0.806     16281
   macro avg      0.736     0.765     0.747     16281
weighted avg      0.821     0.806     0.812     16281
```

Figura 13: Resultado do modelo *Naive Bayes*

[4] Com a transformação de atributos nominais em numéricos, previamente referido.

3.2 *K Nearest Neighbors*

O modelo de *K Nearest Neighbors* consiste no uso dos *K* objetos mais próximos para a classificação de um outro objeto, isto é, este objeto é classificado por uma certa classe que está na maioria dos *K* objetos mais próximos. Se $k = 1$, então o objeto é sempre classificado pelo outro objeto mais próximo dele.
Este modelo necessita do uso de métricas de distância (como a euclidiana, Manhattan, ...) com vista a calcular os objetos mais próximos. [4]

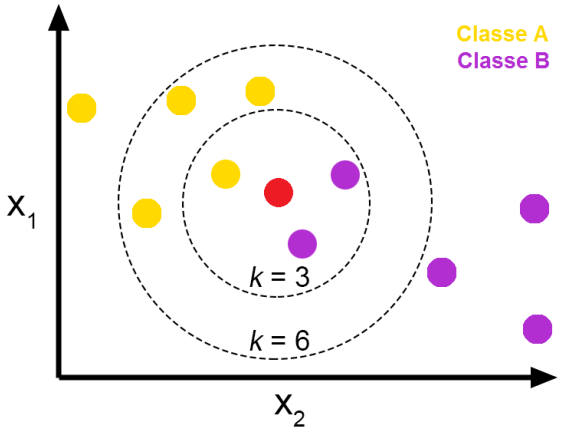


Figura 14: Exemplo de *K Nearest Neighbors*

É de ressalvar que, neste modelo, não foram executadas uniformizações dos dados, visto que essa uniformização descia, consideravelmente, a *accuracy* do modelo, pelo que o modelo foi desenvolvido da seguinte forma:

```
knn = KNeighborsClassifier(n_neighbors = 1)
knn.fit(train_x, train_y)
```

Com vista a otimizar o modelo, realizou-se a visualização da média dos erros do mesmo:

Figura 15: Escolha do *K* ótimo através do *error rate* mínimo

Através desta figura, verificou-se que $k = 16$ seria a melhor escolha para minimizar o erro e, consequentemente, aumentar a *accuracy*:

```
[[10947  1488]
 [ 1367  2479]]
```

	precision	recall	f1-score	support
0	0.889	0.880	0.885	12435
1	0.625	0.645	0.635	3846
accuracy			0.825	16281
macro avg	0.757	0.762	0.760	16281
weighted avg	0.827	0.825	0.826	16281

Figura 16: Matriz de classificação com 1 vizinho

```
[[11684   751]
 [ 1493  2353]]
```

	precision	recall	f1-score	support
0	0.887	0.940	0.912	12435
1	0.758	0.612	0.677	3846
accuracy			0.862	16281
macro avg	0.822	0.776	0.795	16281
weighted avg	0.856	0.862	0.857	16281

Figura 17: Matriz de classificação com 16 vizinhos (*K* ótimo)

3.3 *Decision Tree*

Uma *Decision Tree* é um modelo em formato de árvore em que cada folha representa uma classe da *feature target* e os ramos são as observações, isto é, conjuntos de atributos que levam a essas folhas.
Este modelo é muito usado na área de *Machine Learning* devido à sua simplicidade e facilidade de compreensão do modelo gerado (neste caso, as ramificações e folhas da árvore). [5]

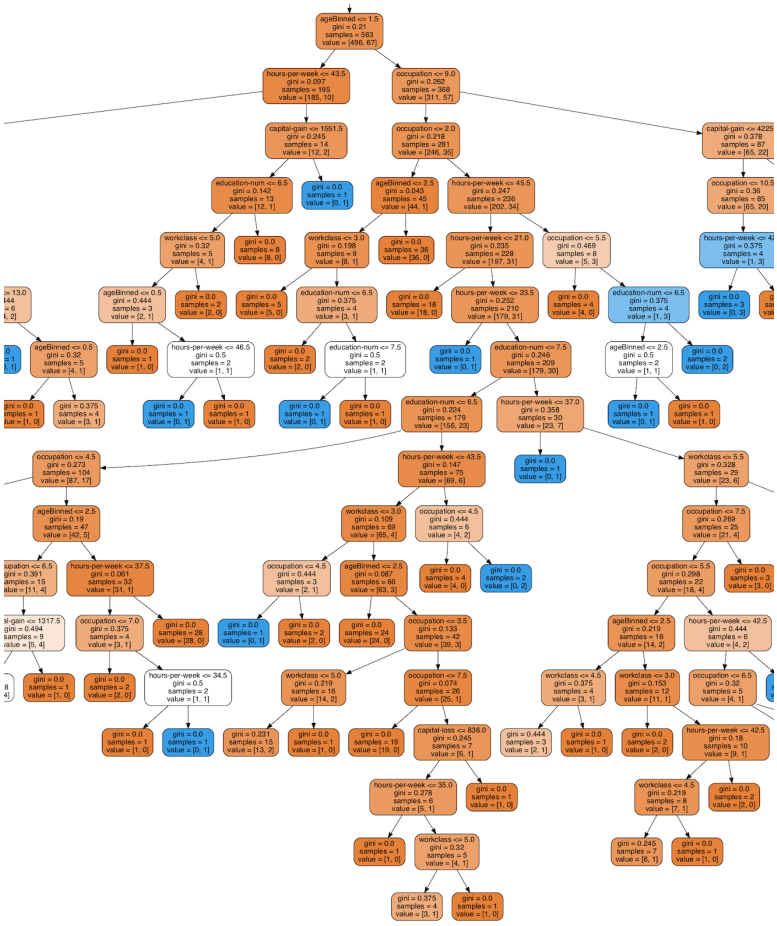


Figura 18: Excerto da árvore de decisão do *dataset* deste problema

Segue-se o modelo desenvolvido através da *Decision Tree* da *package sklearn*. É de destacar que os hiper-parâmetros do modelo sofreram *tuning*.

```
decisionTree = DecisionTreeClassifier(random_state = 42,
                    criterion = 'gini',
                    max_depth = 11,
                    min_samples_split = 15,
                    min_samples_leaf = 4)

decisionTree.fit(train_x, train_y)
```

A partir deste conseguiu-se a seguinte *accuracy*:

	precision	recall	f1-score	support
0	0.877	0.944	0.909	12435
1	0.758	0.572	0.652	3846
accuracy			0.856	16281
macro avg	0.818	0.758	0.781	16281
weighted avg	0.849	0.856	0.848	16281

Figura 19: Resultado do modelo *Decision Tree*

3.4 *AdaBoost* com *Decision Tree*

O algoritmo *AdaBoost* (*Adaptive Boosting*) é um método de *ensemble learning* que aumenta a *performance* de outros algoritmos de aprendizagem. [6]
Este modelo é adaptável, pois as classificações feitas são ajustadas a favor das instâncias classificadas negativamente por classificações anteriores.
Deste modo, ao aplicarmos *AdaBoost* à *Decision Tree*, proporcionamos uma otimização em relação ao modelo anterior, uma vez que este executa n iterações desse algoritmo, levando à consequente diminuição da *variance* e do coeficiente de *bias*. Esta diminuição, permite uma maior *accuracy* por parte do modelo construído. [7]

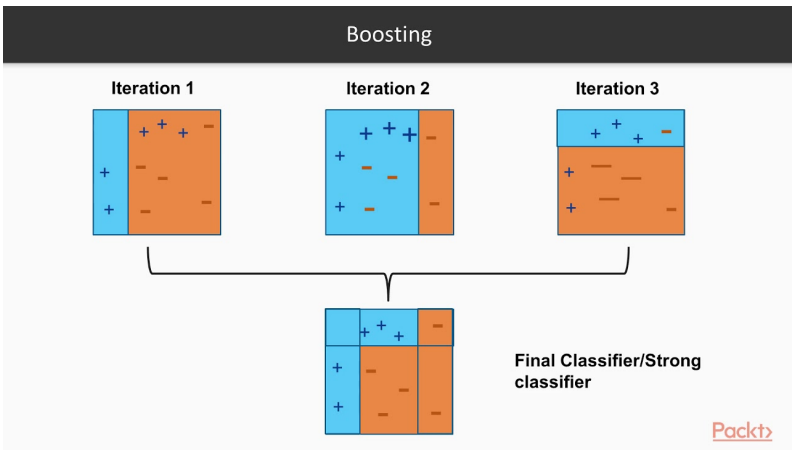


Figura 20: Otimização efetuada por *Boosting (AdaBoost)*

A variância (*variance*) de um modelo mede as diferentes previsões indicadas pelo modelo, caso os dados de treino variem.
Já o *bias* é o erro das previsões do modelo, ou seja, um modelo com *bias* elevado irá ignorar relações importantes nas *features* e nas previsões indicadas (*underfitting*).

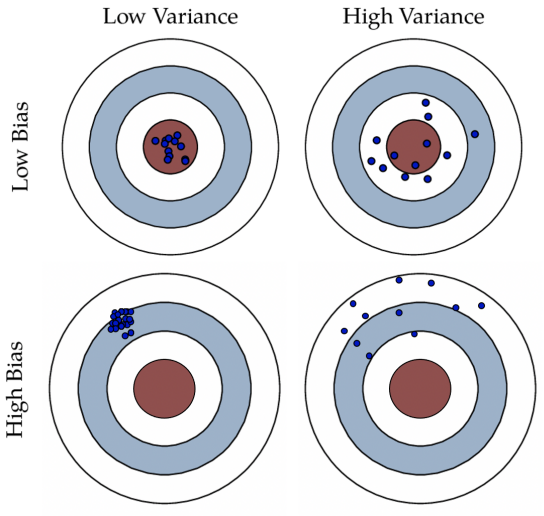


Figura 21: *Variance* e *bias* num modelo preditivo

Segue-se o desenvolvimento do *AdaBoost* com 800 iterações e um *learning rate* de 1.7 (obtido através de *tuning*), através do *sklearn*.

```
abc = AdaBoostClassifier(n_estimators = 800,
            learning_rate = 1.7,
            random_state = 42)

model = abc.fit(train_x, train_y)
```

Seguem-se, ainda, os resultados obtidos neste modelo:

	precision	recall	f1-score	support
0	0.896	0.943	0.919	12435
1	0.777	0.645	0.705	3846
accuracy			0.872	16281
macro avg	0.836	0.794	0.812	16281
weighted avg	0.868	0.872	0.868	16281

Figura 22: Resultado do modelo *Adaboost* com *Decision Tree*

3.5 *Random Forest*

As *Random Forest* são também modelos de *ensemble learning* que se baseiam na construção de várias árvores de decisão, existindo uma diversidade no conjunto de árvores. A partir destas, é determinada a previsão mais frequente, que se torna na previsão final do modelo. [8]
O uso de *Random Forest* permite um menor *overfitting* quando comparado com o uso de *Decision Trees*.

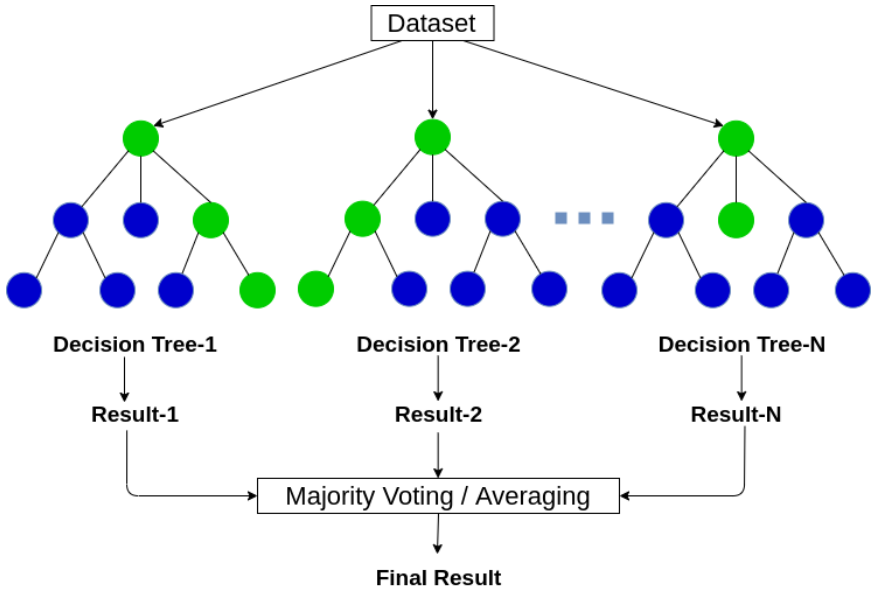


Figura 23: Exemplo de *Random Forest*

Segue-se um excerto de código para a aplicação deste modelo[5]:

```
rf = RandomForestClassifier(n_estimators = 1000,
                 random_state = 42,
                 criterion = 'gini',
                 max_depth = 9,
                 min_samples_leaf = 3,
                 min_samples_split = 11)
rf.fit(train_x, train_y);
```

Seguem-se, ainda, os resultados obtidos com este modelo:

	precision	recall	f1-score	support
0	0.869	0.958	0.911	12435
1	0.797	0.533	0.638	3846
accuracy			0.858	16281
macro avg	0.833	0.745	0.775	16281
weighted avg	0.852	0.858	0.847	16281

Figura 24: Resultado do modelo *Random Forest*

[5] Foram utilizadas 1000 árvores para o aumento de diversidade e consequente aumento da *performance* do modelo.

3.6 *Support Vector Machine*

Os *Support Vector Machine* são modelos robustos de previsão para problemas de classificação binários. O objetivo deste modelo é criar a melhor linha para dividir o espaço multidimensional em duas classes para, depois, efetuar a classificação. [9]

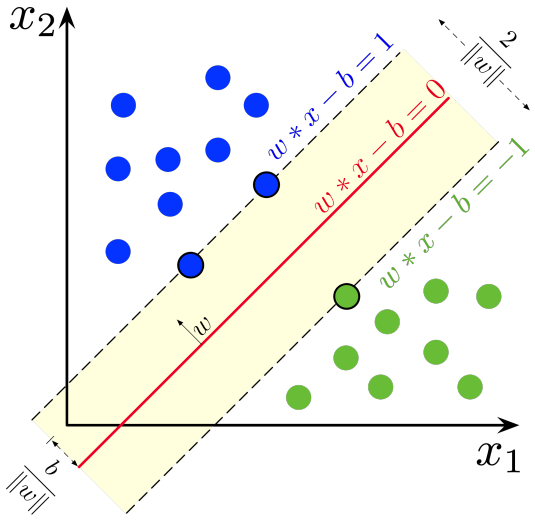


Figura 25: Exemplo de *Support Vector Machine*

Tendo em conta este problema, a *package SVC* do *sklearn* foi utilizada para classificação[6]:

```
svc_model = SVC()
svc_model.fit(train_x, train_y)
```

Através deste modelo chegou-se à seguinte *accuracy*:

	precision	recall	f1-score	support
0	0.810	0.968	0.882	12435
1	0.719	0.269	0.391	3846
accuracy			0.802	16281
macro avg	0.765	0.618	0.637	16281
weighted avg	0.789	0.802	0.766	16281

Figura 26: Resultado do modelo *Support Vector Machines*

[6] Neste modelo o *kernel* especificado foi o *rbf*.

3.7 *Stacking*

Para além do *Boosting* e *Bagging*, é possível otimizar a *performance* de modelos preditivos através de *Stacking* (outro algoritmo de *ensemble learning*). [10]
Stacking é uma técnica que utiliza a previsão de vários modelos para a construção de um novo. Desta forma, consegue-se captar as mais valias de vários modelos diferentes, visto que conseguimos desenvolver um modelo mais complexo (o que leva a *performance* superior na teoria), ao contrário de um único modelo, que tende a ser mais simples.
Esta técnica de *ensemble learning* difere do *Bagging* e *Boosting* na medida em que utiliza diferentes modelos para o mesmo *dataset* (ao contrário do *Bagging*) e um único modelo para a combinação das previsões dos diferentes modelos[7] (ao contrário do *Boosting*).

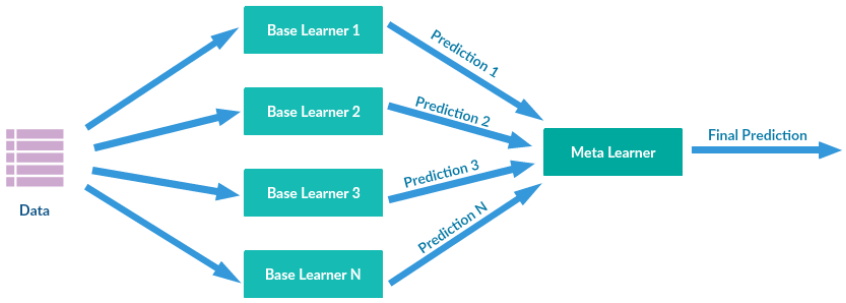


Figura 27: Exemplo de *Stacking*

Desta forma, utilizámos o *Stacking* disponível na package *sklearn*, como foi habitual em todos os modelos.
Os modelos escolhidos para o *Stacking* foram os três com melhor *performance*, ou seja, *K Nearest Neighbors, AdaBoost de Decision Tree* e *Random Forest*. Por fim, a regressão logística foi utilizada com vista a prever os resultados finais.

```
estimators = [
    ('knn', KNeighborsClassifier(
        n_neighbors = 16)),
    ('rf', RandomForestClassifier(
        n_estimators = 200,
        random_state = 42)),
    ('ada', AdaBoostClassifier(
        n_estimators = 200,
        learning_rate = 1,
        random_state = 42))
    ]

clf = StackingClassifier(
    estimators = estimators,
```

[7] Pode ser utilizada a regressão linear para o efeito, por exemplo.

```
final_estimator = LogisticRegression(),
cv = 10)
```

Seguem-se os resultados obtidos com *Stacking*:

	precision	recall	f1-score	support
0	0.888	0.938	0.912	12435
1	0.756	0.617	0.680	3846
accuracy			0.863	16281
macro avg	0.822	0.778	0.796	16281
weighted avg	0.857	0.863	0.857	16281

Figura 28: Matriz de classificação da técnica de *Stacking*

4 Tratamento do Desbalanceamento da Classe *target*

Após o uso dos diversos modelos de *Machine Learning* com respetiva otimização e técnicas de *ensemble learning*, o foco passa, agora, para a *feature target*, **salary-classification**.

Figura 29: Desbalanceamento do *target*

Nesta figura, deparámo-nos com uma quantidade bastante superior de casos '$\leq 50k$'. Para além desta discrepância, os dados de teste possuem 76.4% de casos com salário menor ou igual a 50 mil unidades monetárias. Devido a isto, torna-se fulcral focarmo-nos na classe desbalanceada ('$> 50k$'), conseguindo modelos com uma *performance* superior no geral, mas, sobretudo, nessa classe[8].
Assim e devido ao desbalancemanto verificado nos valores de treino, considerámos que seria necessário aplicar técnicas de *Imbalanced Data* de forma a realizar *Over* e *Under Sampling* do *target* e otimizar a *performance* obtida na classe menos representada. Com isto em mente, foram aplicadas técnicas como *Random Under Sample*, *Random Over Sample*, *SMOTE* e *ADASYN*. [11]
É de ressalvar que, para todas estas técnicas, era pretendido que o rácio do *target* fosse de 40%, ou seja, que houvesse um rácio de 40% entre '$> 50k$' (correspondente ao 1 no *dataset*) e '$\leq 50k$' (correspondente ao 0 no *dataset*).[9]
Este valor derivou de várias tentativas efetuadas e da consequente análise da *performance* dos modelos (presente na próxima secção).

Segue-se um excerto da aplicação das técnicas:

```
oversample = RandomOverSampler(sampling_strategy = 0.4)
undersample = RandomUnderSampler(sampling_strategy = 0.4)
smote = SMOTE(sampling_strategy = 0.4)
adasyn = ADASYN(sampling_strategy = 0.4)

over_train_x, over_train_y = oversample.fit_resample(
    train_x,train_y)
under_train_x, under_train_y = undersample.fit_resample(
    train_x,train_y)
smote_train_x, smote_train_y = smote.fit_resample(
    train_x, train_y)
```

[8] Um modelo que indicasse sempre 0 ('$\leq 50k$') teria uma *accuracy* de 76.4%.
[9] Ou seja, um valor superior ao rácio existente sem estas técnicas ($\frac{23.6}{76.4} \times 100 = 30.1\%$, sendo o numerador a percentagem da classe em minoria e o denominador a percentagem da classe em maioria).

```
adasyn_train_x, adasyn_train_y = adasyn.fit_resample(
    train_x, train_y)
```

4.1 *Random Under Sample*

O *Random Under Sample* torna a distribuição do *target* mais equilibrada, eliminando, de forma aleatória, instâncias da classe mais representada.
Esta metodologia tem como vantagem a redução do tempo computacional dos modelos, devido à redução do *dataset*, contudo, pode provocar perda de informação importante para os modelos.

4.2 *Random Over Sample*

O *Random Over Sample* aumenta o número de instâncias da classe em minoria, neste caso '$> 50k$', com replicação de forma aleatória.
Esta metodologia não tem qualquer perda de informação, ao contrário da anterior, e costuma ter uma melhor *performance*. Apesar disso, tende a ocorrer *overfitting*, visto que replica casos da minoria.

4.3 *SMOTE*

O *SMOTE (**S**ynthetic **M**inority **O**ver-sampling **TE**chnique for imbalanced data)* tende a evitar o *overfit* que ocorre quando se replica a ocorrência de casos da classe em minoria para o *dataset*. Nesta técnica, um subconjunto do *dataset* é retirado da classe em minoria e são criadas instâncias sintéticas deste subconjunto que são, posteriormente, adicionadas ao *dataset* original.
Desta forma, esta técnica evita o *overfitting* associado à técnica anterior, visto que cria instâncias novas ao invés de replicar as já existentes. Aqui, também não ocorre perda de informação.
Apesar disso, o *SMOTE* não tem em consideração os vizinhos da amostra que são de outras classes, logo pode aumentar o ruído (*noise*) dos dados.

4.4 *ADASYN*

Por fim, o *ADASYN* comporta-se de maneira muito semelhante ao *SMOTE*, apenas com a diferença de adicionar pequenos valores aleatórios aos pontos. Com esta pequena diferença é pretendido que os pontos pareçam mais reais, ou seja, existir uma maior variância nos dados sintéticos.

A análise de resultados de todos os modelos e da diferença com e sem o uso destas técnicas de *sampling* encontra-se na secção 6.

5 Otimização de Modelos

Com vista a otimizar a *performance* dos modelos anteriormente apresentados, para além do pré-processamento de dados, procedemos ao *tuning* dos hiper-parâmetros dos modelos. Através disto, conseguimos obter resultados superiores, o que nos levou a uma configuração próxima da ideal nos modelos desenvolvidos para este *dataset*.
Assim, utilizou-se o *GridSearchCV*, disponível no *sklearn*, de forma a procurar os melhores hiper-parâmetros.
Segue-se um exemplo de *tuning* para o modelo *Decision Tree*:

```
parameters = {
    'criterion': ['gini', 'entropy'],
    'max_depth': range(1, 12),
    'min_samples_split': range(2, 17),
    'min_samples_leaf': range(2, 8)
}

tuning = GridSearchCV(decisionTree,
                      parameters,
                      cv = StratifiedKFold(
                          n_splits = 10,
                          random_state = 42,
                          shuffle = True),
                      verbose = 1,
                      n_jobs = 14,
                      scoring = 'accuracy')

tuning.fit(train_x, train_y)
```

É necessário destacar que no *tuning* foi usado o *cross-validation* para conseguir resultados mais "reais", visto que o modelo foi testado com várias divisões diferentes. Desta forma, o uso de *cross-validation* para a otimização do modelo também permitiu ter uma noção da *performance* dos vários modelos, sem a previsão com valores de teste.

Assim, esta técnica permite-nos identificar um valor aproximado da *performance* dos modelos para múltiplos dados de teste, para além do usado, num caso futuro.

Por último, também é de realçar que a otimização de modelos através de *tuning* não foi efetuada para todos os modelos preditivos, sendo que apenas o *AdaBoost, Decision Tree* e *Random Forest* sofreram esta otimização.

6 Análise de Resultados

Primeiramente, iremos analisar os resultados obtidos por alguns modelos sem o tratamento de dados realizado:

Modelos	Accuracy [%]
Naive Bayes Gaussiano	80.2
KNN (k = 16)	80.2
Decision Tree	80.9
AdaBoost	87.2
Random Forest	85.6
SVM	79.9
Stacking	85.6

Tabela 1: Resultados sem tratamento de dados e sem *tuning* dos modelos

Em segundo lugar e de forma a comprovarmos a escolha do tratamento de dados referido na secção 2, iremos analisar a *accuracy* do modelo *Decision Tree* com várias mudanças que foram experimentadas durante o desenvolvimento deste trabalho:

Modelo Decision Tree	Accuracy [%]
Sem a remoção de *features* com baixo valor informativo (*race*, *education*, *native-country*, ...)	79.9
Com a remoção da *feature* '*capital*'	79.9
Com normalização da '*capital-loss*' e '*capital-gain*'	81.5
'*age*' com 6 intervalos de igual frequência	82.8
Com a substituição dos *missing values* pela moda	82.9
Com o tratamento referido anteriormente	83.0

Tabela 2: Resultados do modelo *Decision Tree* com diferentes tipos de pré-processamento de dados

Assim, o tratamento explicitado na secção 2 revela-se como o mais eficiente e que otimiza a *performance* dos modelos usados na sua generalidade.

De forma a analisarmos qual o melhor modelo para este problema, é necessário compararmos não só os valores de *accuracy*, como outras métricas para as classes do *target* (*Recall, Precision e f1-score*).
O *Recall* indica a quantidade de instâncias da classe que o modelo captou, enquanto que a *Precision* é a qualidade de acerto do modelo para essa mesma classe. Neste sentido, um modelo com elevada *Precision* não é necessariamente um bom modelo, pois pode ter indicado muito poucas instâncias como sendo daquela classe, deixando "escapar" outras. O mesmo pode ser dito para o *Recall* que, por si só, não garante a boa *performance* do modelo, visto que um modelo pode captar uma percentagem elevada de uma classe, devido à alta frequência que indica essa mesma classe.
Deste modo, o *f1-score* faz uso destas duas métricas através da seguinte fórmula:

$$f1 - score = 2 \cdot \frac{Recall \cdot Precision}{Recall + Precision}$$

Posto isto, podemos, então, analisar os resultados obtidos pelos modelos sem ter em consideração as técnicas de *imbalanced data*, mas já com todo o tratamento de dados efetuado, bem como a otimização dos modelos.

Modelo	**Accuracy [%]**	**Recall (classe 0) [%]**	**Precision (classe 0) [%]**	***f1-score* (classe 0) [%]**	**Recall (classe 1) [%]**	**Precision (classe 1) [%]**	***f1-score* (classe 1) [%]**
Naive Bayes Gaussiano	80.6	84.2	89.7	86.9	68.8	57.4	62.6
KNN (k = 16)	86.2	94.0	88.7	91.2	61.2	75.8	67.7
Decision Tree	85.6	94.4	87.7	90.9	57.2	75.8	65.2
AdaBoost	87.2	94.3	89.6	91.9	64.5	77.7	70.5
Random Forest	85.8	95.8	86.9	91.1	53.3	79.7	63.8
SVM	80.2	96.8	81.0	88.2	26.9	71.9	39.1
Stacking	86.3	93.8	88.8	91.3	61.7	75.6	68.0

Tabela 3: Resultados com pré-processamento de dados e otimização de alguns dos modelos

Com o tratamento de dados realizado, bem como com a otimização dos modelos, é claramente observável que os modelos sofreram uma melhoria da sua *performance* em comparação à obtida na tabela 1.
O melhor exemplo disto é o modelo *K Nearest Neighbors* que conseguiu incrementar a sua *accuracy* em 6%, uma subida considerável que representa bem a diferença de aplicação dos modelos com e sem pré-processamento do *dataset*. Contudo, esta subida não é geral, visto que o modelo de *AdaBoost* teve uma ligeira descida. Outro dado relevante mostra-se no facto do *Random Forest* apenas ter subido na *performance* devido ao *tuning* efetuado e não devido ao tratamento de dados.
Apesar desta descida ser inesperada, considerámos que se deveu aos próprios modelos utilizados, visto que são bastante complexos (e tratam-se de *ensemble learning* de *Decision Tree*). Assim, esta descida pode dever-se, essencialmente, à remoção de *features* consideradas menos relevantes, pelo que, devido à grande quantidade de árvores diferentes entre si (como existe no *Random Forest*), poderá provocar uma diminuição da *accuracy* do modelo, quando comparada com o uso de todos os atributos do *dataset*.
Mesmo assim, esta descida não influência a escolha do pré-processamento de dados, visto que o escolhido garante a melhor *performance* dos modelos de maneira geral.

O modelo *AdaBoost de Decision Tree* é aquele que apresenta uma maior *performance* devido à sua maior *accuracy* de 87.1%. Outro facto a ter em conta é que também é o modelo com melhor *f1-score* para as duas classes.[10]
A melhor *performance* deste modelo, muito possivelmente, deve-se à diminuição do *bias*, que levou a uma melhoria da *performance* relativamente ao *Decision Tree*. É possível verificar que o *boosting* melhorou a *performance* em todos os campos para a classe 0 e 1 do modelo original.

Em relação aos outros modelos, o *SVM* e *Naive Bayes* mostram-se como aqueles que possuem a pior *performance*. Tal, muito possivelmente, deve-se à distribuição dos dados (possível de observar na figura 10) e à simplicidade do modelo, respetivamente.
No caso *SVM*, é possível perceber que a definição da linha passou por quase todas instâncias da classe 0, visto que se encontram mais concentradas, perdendo-se, assim, muitas instâncias da outra classe.[11]

[10] *Stacking* surge como o segundo melhor modelo.
[11] Este facto é muito percetível pela imagem dada do *PCA* na figura 10.

O *Decision Tree* apresenta uma *accuracy* razoável que, após o *ensemble learning*, incrementa de maneira considerável e o *KNN*, após otimização do número de vizinhos, demonstra ter uma boa *performance*, muito ligada à concentração de dados nas classes (isto é, quando existe um dado da classe 1, há uma concentração elevada de outros dados da mesma classe, o que possibilita uma *accuracy* elevada).

Resta-nos agora comparar a *performance* dos modelos com as técnicas de *imbalanced data*. Neste relatório iremos apenas comparar os resultados no *AdaBoost*, visto ser o nosso melhor modelo.

Rácio para as técnicas de sampling [Accuracy | *f1-score* (classe 1)]	**Random Over Sampling [%]**		**Random Under Sampling [%]**		**SMOTE [%]**		**ADASYN [%]**	
40 %	86.9	71.4	87.1	71.6	87.0	71.3	87.0	71.6
45 %	86.7	71.8	86.6	71.6	86.6	71.5	86.3	71.7
50 %	86.5	72.0	86.4	71.9	86.5	71.9	85.7	71.8
60 %	86.0	72.1	86.0	72.1	85.8	71.9	84.2	71.2

Tabela 4: Resultados do modelo *AdaBoost* com diferentes técnicas de *sampling* para diferentes rácios

Na tabela acima, apresentámos os resultados dos modelos com diferentes técnicas de sampling e com vários rácios entre as duas classes do *target*. Através do par da *accuracy* e do *f1-score* da classe em minoria que pretendemos incrementar, teremos agora de escolher uma das técnicas com melhor resultado.

Como é percetível, quanto maior o rácio, maior o *f1-score* obtido[12], tal como era esperado, pois possuímos uma maior representatividade dessa classe no *dataset*. Contudo, associado a isto, temos uma diminuição da *accuracy*, uma vez que o *f1-score* da classe em maioria diminui.

Desta forma, foi escolhido o rácio de 40% entre a classe em minoria ('$> 50k$' ou 1) e a classe em maioria da *feature* **salary-classification**, pois considerámos que possui os resultados mais adequados ao problema.

Teremos, agora, de escolher a melhor técnica de *sampling*.

Modelo	**Accuracy [%]**	**Recall (classe 0) [%]**	**Precision (classe 0) [%]**	***f1-score* (classe 0) [%]**	**Recall (classe 1) [%]**	**Precision (classe 1) [%]**	***f1-score* (classe 1) [%]**
AdaBoost	87.1	94.2	89.5	91.8	64.2	77.3	70.2
AdaBoost (Random Over Sampling)	87.0	92.6	90.6	91.6	69.0	74.1	71.6
AdaBoost (Random Under Sampling	87.1	92.6	90.6	91.6	69.0	74.3	71.6
AdaBoost (SMOTE)	86.9	92.6	90.5	91.6	68.7	74.2	71.3
AdaBoost (ADASYN)	87.0	92.4	90.7	91.6	69.5	73.8	71.6

Tabela 5: Resultados dos modelos com técnicas de desbalanceamento

Primeiramente, o uso destas técnicas provoca uma ligeira redução da *performance* em comparação com o modelo original na classe maioritária, visto que o *f1-score* da classe 0 é su-

[12] O aumento não é linear.

perior no primeiro modelo. Isto deve-se a uma descida da *recall*, embora que acompanhada por uma ligeira subida na *precision*.
Por outro lado, o cenário na classe 1 é muito mais favorável para os modelos com estas técnicas, devido ao aumento do rácio de instâncias desta classe em comparação com a outra.

Assim, a técnica de *Random Under Sampling* revela-se como a mais indicada para uma *performance* do modelo, conseguindo um melhor equilíbrio de previsão entre as duas classes. Esta técnica possui a mesma *accuracy* que o modelo original e consegue prever mais 5% de instâncias da classe em minoria (*recall* com 69.5%) e apresenta um *f1-score* de 71.6%.

É de realçar que podia ter sido desenvolvida uma matriz de custos onde fosse tido em conta que a previsão de uma certa classe poderia ser mais benéfica que outra. Todavia, para o problema em causa, decidimos que tal abordagem não seria necessária.

A escolha do melhor modelo depende, essencialmente, do pretendido no que toca à previsão das duas classes. Caso a previsão da classe 0 seja o principal foco, dever-se-à optar por um modelo com melhor *f1-score* para essa mesma, que, nesse caso, seria o *AdaBoost* sem a técnica de *imbalanced data*.

No nosso entender, o modelo com melhor *performance*, tendo em conta as duas classes, e melhor *accuracy* é o *AdaBoost* com uso de *Random Under Sampling*. Esta escolha deve-se ao facto de estar entre os dois melhores modelos no que toca à *accuracy* (87.1%) e por apresentar um dos melhores *f1-score* da classe minoritária (classe 1 ou '$> 50k$').

7 Conclusões e Trabalho Futuro

A realização deste projeto permitiu-nos aplicar os conhecimentos aprendidos nas aulas teóricas e práticas da cadeira de Análise e Extração do Conhecimento.
O principal desafio durante a resolução deste projeto prendeu-se com a pesquisa e assimilação de técnicas populares da área de *Machine Learning*, bem como os modelos que permitissem a obtenção de resultados satisfatórios.

O facto de os dados trabalhados se encontrarem desbalanceados revelou-se como um obstáculo à boa *performance* dos nossos modelos e análise destes, todavia, o uso de técnicas de *imbalanced data* possibilitaram uma ligeira melhoria na *performance*.

Após uma análise cuidada do projeto desenvolvido, considerámos que a previsão para este *dataset* poderia ter sido melhorada, recorrendo a um maior número de *feature selections* disponíveis no *sklearn*, bem como uma otimização mais exaustiva dos modelos.
Outra solução passaria por transformar a *feature* **fnlwgt**, de forma a termos esse número de indivíduos com as mesmas características, em várias instâncias no *dataset*.

Por outro lado, o tratamento do *dataset* poderia ter sido explorado de outras formas para tentar melhorar a *performance* (uma vez que existem vários tipos de tratamento das *features* que não foram aplicados no nosso projeto).
É de ressalvar que, no futuro, seria bastante interessante a aplicação de mais modelos preditivos e técnicas de *Machine Learning* neste *dataset*, com vista a explorar novas ideias, tendo sempre em vista o aumento da *performance*.

Relativamente ao modelo ótimo, concluímos que o `AdaBoost`, após a técnica de `Random Under Sampling`, é o melhor modelo, conseguindo uma *accuracy* de 87.1% e um bom equilíbrio na previsão das duas classes.

Em suma, considerámos que o trabalho desenvolvido se encontra adequado para o pretendido pela equipa docente no contexto da problemática em questão e que foram desenvolvidos vários algoritmos que apresentam uma boa capacidade de predição.

Referências

1. **SAS Enterprise Miner**. Disponível em: https://www.sas.com/en_us/software/enterprise-miner.html

2. CHAPMAN, Pete, CLINTON, Julian, KERBER, Randy, KHABAZA, Thomas, REINARTZ, Thomas, SHEARER, Colin & WIRTH, Rüdiger. (1999); **'CRISP-DM 1.0: Step-by-step data miningguide'**. Disponível em: https://www.kde.cs.uni-kassel.de/wp-content/uploads/lehre/ws2012-13/kdd/files/CRISPWP-0800.pdf

3. RISH, Irina. (2001); **An empirical study of the naive Bayes classifier**. Disponível em: https://www.cc.gatech.edu/~isbell/reading/papers/Rish.pdf

4. CUNNINGHAM, Padraig & DELANY, Sarah Jane. (2007); **k-Nearest Neighbour Classifiers**. Disponível em: https://www.researchgate.net/publication/228686398_k-Nearest_neighbour_classifiers

5. PATEL, Harsh & PRAJAPATI, Purvi. (2018); **Study and Analysis of Decision Tree Based Classification Algorithms**. Disponível em: https://www.researchgate.net/publication/330138092_Study_and_Analysis_of_Decision_Tree_Based_Classification_Algorithms

6. SINGH, Aishwarya. (2018); **A Comprehensive Guide to Ensemble Learning (with Python Codes)** Disponível em: https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/

7. HU, Weiming, HU, Wei & MAYBANK, Steve. (2008); **AdaBoost-Based Algorithm for Network Intrusion Detection**. Disponível em: https://ieeexplore.ieee.org/document/4454220

8. BREIMAN, Leo. (2001); **Random Forests**. Disponível em: https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf

9. EVGENIOU, Theodoros & PONTIL, Massimiliano. (2001); **Workshop on Support Vector Machines: Theory and Application**. Disponível em: https://www.researchgate.net/publication/221621494_Support_Vector_Machines_Theory_and_Applications

10. ZHOU, Aolong, REN, Kaijun, LI, Xiaoyong & ZHANG, Wen. (2019); **MMSE: A Multi-Model Stacking Ensemble Learning Algorithm for Purchase Prediction**, 2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Chongqing, China, 2019, pp. 96-102, doi: 10.1109/ITAIC.2019.8785711. Disponível em: https://ieeexplore.ieee.org/abstract/document/8785711

11. MUKHERJEE, Upasana. (2017); **Imbalanced Data: How to handle Imbalanced Classification Problems**. Disponível em: https://www.analyticsvidhya.com/blog/2017/03/imbalanced-data-classification/