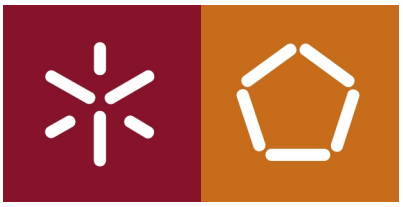

Universidade do Minho
mestrado integrado em engenharia informática

Aprendizagem e Extração de Conhecimento

Sistemas Inteligentes

(1º Semestre / 4º Ano)

TP2 - Aprendizagem e Extração de Conhecimento

Diogo Braga (a82547)
Pedro Ferreira (a81135)
Ricardo Caçador (a81064)

Conteúdo

1 Introdução

O presente relatório é referente à segunda fase do trabalho prático da unidade curricular de Aprendizagem e Extração de Conhecimento, integrada no perfil de especialização em Sistemas Inteligentes.

Este projeto tem como ponto principal a preparação e análise de um dataset relativo às condições de funcionários de uma empresa, como forma de prever a sua ausência.

O produto final será um modelo de classificação para a situação agora referida, através da utilização de um ambiente de desenvolvimento Python/Sklearn.

Após a introdução, na secção 2 será realizada uma abordagem aos dados em estudo, com visualização dos mesmos, e posterior tratamento destes, incluindo transformação e balanceamento.

De seguida, na secção 3, serão detalhados métodos para realizar seleção de *features*, e consequente diminuição do *dataset*, que provoca melhorias no desempenho da análise.

Na secção 4 são apresentados os modelos utilizados nos testes, e analisados os melhores resultados obtidos.

Por fim, conclui-se referindo os pontos principais atingidos, assim como as dificuldades e limitações encontradas.

2 *Data Preparation*

2.1 *Data visualization*

Inicialmente, com o objetivo de analisar melhor os dados com que iríamos trabalhar e perceber as relações existentes entre os diversos atributos e o trabalhador não aparecer ao trabalho. Primeiro é importante perceber por que características e tipos de dados é constituído o nosso dataset.

2.1.1 Informação das características

- **ID:** identificador do funcionário
- **Reason for absence:** número associado à razão de ausência, associado ao CID (Código Internacional da Doenças)
- **Month of absence:** mês no qual se deu a ausência. Os meses correspondem aos números de 1 a 12.
- **Day of the week:** dia da semana do caso de estudo. Segunda corresponde ao número 2, terça ao número 3 e assim sucessivamente até sexta, número 6.
- **Seasons:** estação do ano relativa ao caso de estudo, identificadas pelos números de 1 a 4.
- **Transportation expense:** são os gastos financeiros associados ao transporte do trabalhador até ao local de trabalho.
- **Distance from Residence to Work:** distância, em quilómetros desde o local de trabalho até à residência do funcionário.
- **Service time:** tempo de serviço, em anos, do funcionário do caso de estudo.
- **Age:** idade do funcionário do caso de estudo.
- **Work load average/day:** corresponde à carga de trabalho média que um funcionário tem diariamente.
- **Hit target:** carga de trabalho percentual concluída pelo funcionário.
- **Disciplinary failure:** valor que pode ser 0 ou 1, caso o funcionário já apresente uma falta disciplinar na empresa.
- **Education:** grau de escolaridade do funcionário do caso de estudo, que pode variar de 1 a 4 pela ordem: ensino médio, licenciatura, pós-graduação e mestrado/douturado.
- **Son:** número que corresponde aos filhos do funcionário.
- **Social drinker:** booleano que indica se o funcionário tem por hábito beber álcool socialmente.
- **Social smoker:** booleano que indica se o funcionaro tem por hábito fumar socialmente.
- **Pet:** número de animais de estimação que o funcionário do caso de estudo possui.
- **Weight:** peso do funcionário, em kilogramas.

- **Height:** altura do funcionário, em centímetros.
- **Body mass index:** índice de massa corporal do funcionário.
- **Absent:** é a target variable que indica se um trabalhador se irá ausenter (1) ou não (0).

2.1.2 *Heat Map*

Primeiro, construímos um heat map, de forma a perceber a correlação existentes entre as várias características do dataset.

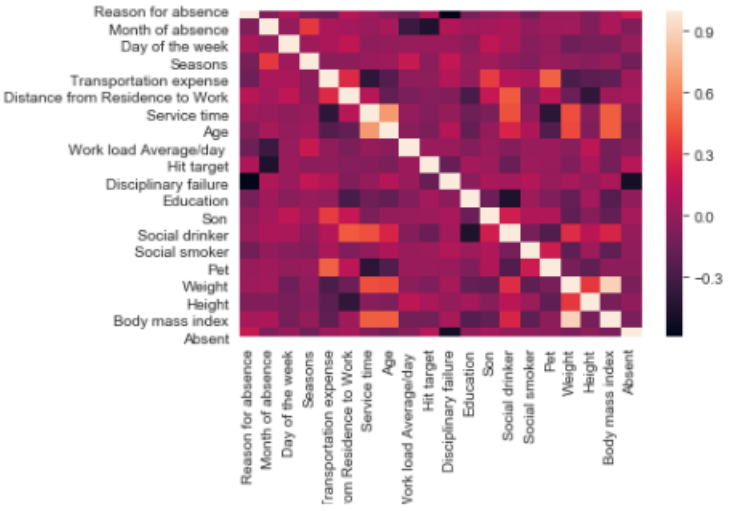

Figura 1: Heat map com os todos os atributos

2.1.3 *Abstenção por dia da semana*

Para percebermos se os trabalhadores não apareciam mais vezes em dias específicos como ínicio ou final da semana, fizemos um histograma que agrupava o número de casos de abstenção por dia da semana. Assim comprovou-se que as pessoas tendem a faltar mais às segundas e sextas do que a meio da semana.

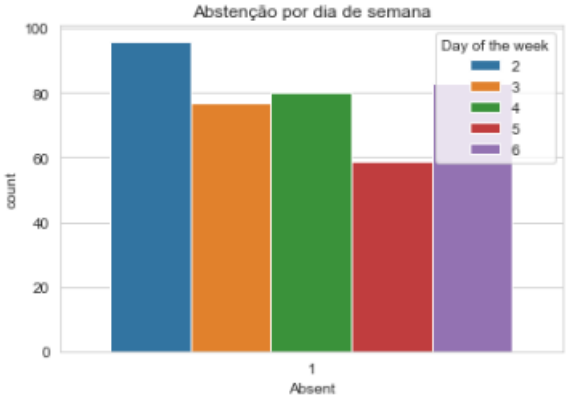

Figura 2: Histograma - Abstenção/Dia da semana

2.1.4 *Abstenção relacionada com hábitos sociais*

Os hábitos sociais, por norma, têm vários efeitos na vida de um ser humano. Por essa razão, queríamos perceber se o mesmo se aplicava ao facto de uma pessoa não aparecer ao trabalho. Pela observação do gráfico percebe-se que, por exemplo, mais de metade dos casos em que o trabalhador não apareceu, o mesmo tinha por hábito beber socialmente.

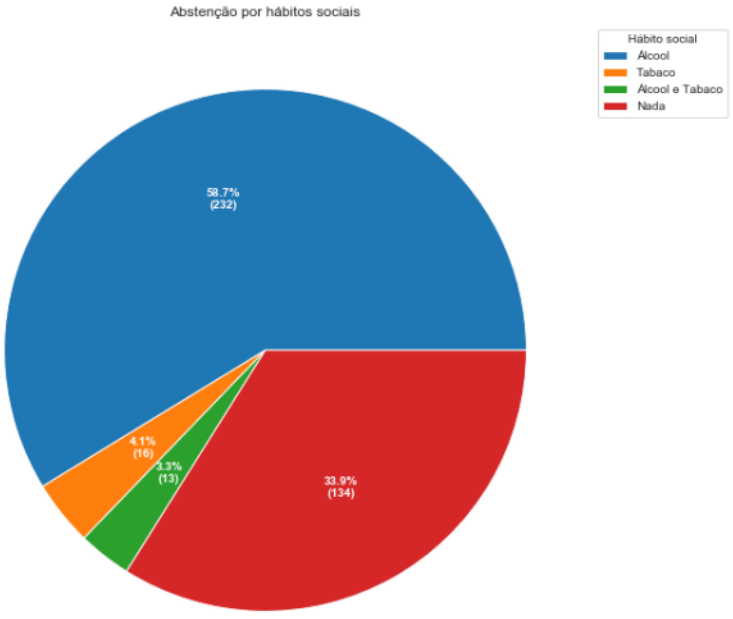

Figura 3: Gráfico circular - Abstenção e hábitos sociais

2.1.5 *Abstenção na relação idade e tempo de serviço*

A idade de um trabalhador e o tempo de serviço do mesmo na empresa podem ter influência. Através do gráfico percebe-se que há mais casos de abstenção em pessoas com pouco tempo de serviço e mais jovens do que pessoas mais experientes e com mais tempo de serviço.

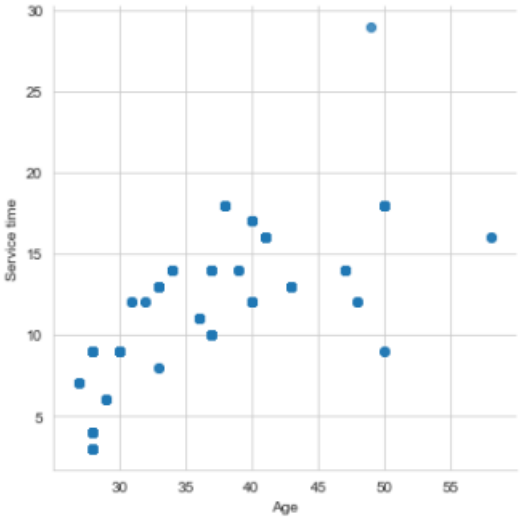

Figura 4: Gráfico de pontos de relação Idade e Tempo de Serviço

2.1.6 *Motivos para abstenção*

Para identificar por que motivos as pessoas mais faltavam ao trabalho fizemos um histograma que apresentava a contagem pelo número de razões para abstenção em todos os casos de abstenção.

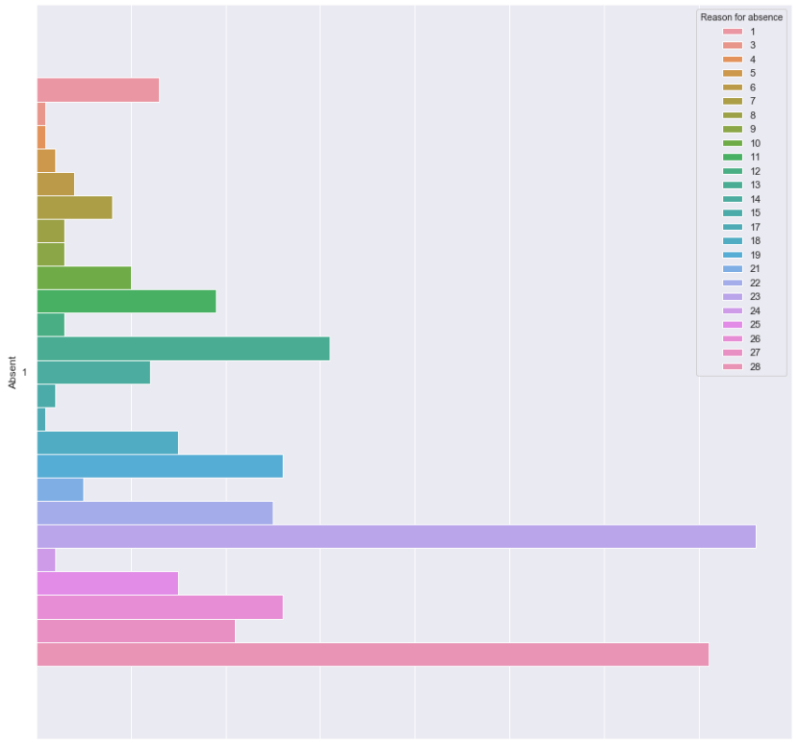

Figura 5: Histograma com os motivos de abstenção

2.1.7 *Abstenção e número de filhos*

Por fim, para perceber se o número de filhos de um trabalhador influenciava a abstenção de um trabalhador, fizemos um gráfico com o número de casos de abstenção em relação ao número de filhos.

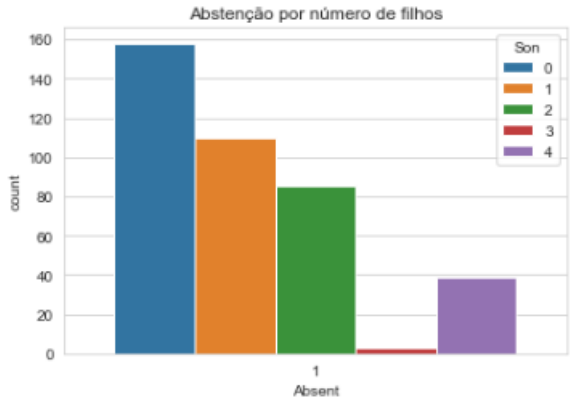

Figura 6: Histograma - Abstenção por número de filhos

2.2 *Data transformation*

De forma a adequar a informação contida no *dataset* aos modelos de extração de conhecimento que foram utilizados, foi necessário proceder ao seu pré-processamento.

Muitas vezes, os dados recolhidos do mundo real apresentam valores em falta ou contêm inconsistências. Posto isto, começamos por verificar se existiam valores em falta:

```
print(AbsenteeismAtWork.isnull().sum(axis=0))
```

```
Reason for absence                0
Month of absence                  0
Day of the week                   0
Seasons                           0
Transportation expense            0
Distance from Residence to Work   0
Service time                      0
Age                               0
Work load Average/day             0
Hit target                        0
Disciplinary failure              0
Education                         0
Son                               0
Social drinker                    0
Social smoker                     0
Pet                               0
Weight                            0
Height                            0
Body mass index                   0
Absent                            0
```

Figura 7: Verificação da existência de valores em falta

Através da figura 7, podemos concluir que todos os campos de dados se encontram preenchidos.

O próximo passo foi proceder à transformação dos dados, analisando e avaliando a aplicação de três técnicas diferentes: discretização, standardização e normalização. Utiliza-se a discretização para reduzir o número de valores de um atributo contínuo, dividindo-o em intervalos. A standardização corresponde ao processo de aproximar os dados de uma distribuição normal. Este processo é um pré-requisito de muitos classificadores, uma vez que certos elementos da função objetivo destes pressupõem que todas as caraterísticas estarão centradas perto de 0, com variância na mesma ordem. Se uma caraterística tiver uma variação maior que as outras, ela poderá dominar a função objetivo e tornar o classificador incapaz de aprender com as restantes caraterísticas, conforme seria esperado [4]. Por fim, a normalização corresponde a escalar individualmente cada um dos registo, de forma a que estes passem a ter norma unitária.

De seguida, apresentam-se os resultados da aplicação das técnicas descritas. Estes foram obtidos através de *cross validation (5-fold)*, com o objetivo obter várias métricas e não apenas uma, como se obteria se a aplicação das técnicas fosse avaliada através da precisão da previsão do *dataset* de teste. Além disso, foram testados diferentes modelos, sem modificações relevantes sobre os seus parâmetros por defeito. Como estamos perante um *dataset* desbalanceado, avaliamos também a métrica AUROC.

Os resultados da aplicação de standardização encontram-se representados na tabela 1. Recorremos ao `StandardScaler` e ao `RobustScaler`, sendo este segundo método mais resistente a

outliers.

	Standard Scaling		**Robust Scaling**	
	Accuracy	AUROC	Accuracy	AUROC
Logistic regression	0.822 (+/- 0.126)	0.708	0.834 (+/- 0.082)	0.670
SGDClassifier	0.736 (+/- 0.193)	0.676	0.784 (+/- 0.137)	0.714
KNearest Neighbors	0.812 (+/- 0.113)	0.672	0.774 (+/- 0.088)	0.594
SVM - RBF	0.850 (+/- 0.082)	0.650	0.834 (+/- 0.093)	0.615
SVM - Linear	0.840 (+/- 0.093)	0.707	0.844 (+/- 0.093)	0.704
Gaussian Naive Bayes	0.854 (+/- 0.084)	0.653	0.854 (+/- 0.084)	0.653
Gaussian Process	0.774 (+/- 0.120)	0.589	0.772 (+/- 0.082)	0.624
Decision Tree	0.756 (+/- 0.206)	0.680	0.764 (+/- 0.209)	0.685
Multi-layer Perceptron	0.766 (+/- 0.113)	0.661	0.774 (+/- 0.154)	0.665
AdaBoost	0.780 (+/- 0.244)	0.737	0.780 (+/- 0.244)	0.738
Random Forest	0.816 (+/- 0.087)	0.692	0.816(+/- 0.098)	0.714

Tabela 1: Resultados óbtidos com *standardização* das caraterísticas

Analisando a tabela 1, podemos concluir que os resultados obtidos pelos dois métodos são semelhantes. Além disso, os valores de *accuracy* foram bastante elevados, sendo acompanhados de um valor de AUROC igualmente satisfatório, o que nos indica que os modelos não estão a ser tendenciosos.

Uma vez que o *dataset* apresentava diferentes características cujos valores eram contínuos, a discretização poderia ser utilizada para reduzir o número de valores existentes, reduzindo o tamanho dos dados. A definição dos intervalos onde os dados serão enquadrados pode ser feita segundo duas abordagens: discretização de igual altura, ou discretização de igual largura. Na primeira, cada intervalo contém aproximadamente o mesmo número de valores, enquanto que na segunda, cada intervalo corresponde a uma porção mesma da gama de valores que contém todos os registos. Neste trabalho, analisamos a utilização de ambas as abordagens e os resultados era muito idênticos. As caraterísticas que decidmos discretizar foram: *Transportation expense*, *Distance from Residence to Work*, *Service time*, *Age*, *Work load Average/day*, *Hit target*, *Weight*, *Height* e *Body mass index*. Posto isto, na tabela 2 encontram-se os dados obtidos através da aplicação de discretização de igual largura, às caraterísticas mencionadas.

	Discretization	
	Accuracy	AUROC
Logistic regression	0.838 (+/- 0.063)—	0.662
SGDClassifier	0.690 (+/- 0.455)	0.591
KNearest Neighbors	0.782 (+/- 0.081)	0.653
SVM - RBF	0.826 (+/- 0.065)	0.670
SVM - Linear	0.844 (+/- 0.071)—	0.713
Gaussian Naive Bayes	0.854 (+/- 0.084)	0.650
Gaussian Process	0.716 (+/- 0.114)	0.546
Decision Tree	0.722 (+/- 0.204)	0.655
Multi-layer Perceptron	0.780 (+/- 0.104)	0.699
AdaBoost	0.752 (+/- 0.216)	0.722
Random Forest	0.816 (+/- 0.082)	0.687

Tabela 2: Resultados óbtidos com *discretização* das caraterísticas

Podemos concluir que os resultados são ligeiramente piores que os obtidos através da standardização. No entanto, esta técnica não foi descartada uma vez que os resultados continuam a ser satisfatórios e os seus princípios parecem perfeitamente aplicáveis ao problema em concreto.

Por fim, aplicamos a técnica de normalização de registos. Os resultados obtidos podem ser obersevados na tabela 3-

	Normalization	
	Accuracy	AUROC
Logistic regression	0.790 (+/- 0.020)	0.569
SGDClassifier	0.558 (+/- 0.569)	0.569
KNearest Neighbors	0.756 (+/- 0.111)	0.588
SVM - RBF	0.790 (+/- 0.020)	0.582
SVM - Linear	0.790 (+/- 0.020)	0.569
Gaussian Naive Bayes	0.848 (+/- 0.072)	0.617
Gaussian Process	0.790 (+/- 0.020)	0.568
Decision Tree	0.714 (+/- 0.170)	0.634
Multi-layer Perceptron	0.790 (+/- 0.020)	0.589
AdaBoost	0.786 (+/- 0.170)	0.673
Random Forest	0.786 (+/- 0.161)	0.683

Tabela 3: Resultados óbtidos com *normalização* dos registos

Ao contrário das restantes, esta técnica não parecia ser apropriada para o problema pois como é aplicada registo a registo, não temos a garantia que os valores de cada característica sejam coerentes entre registos após a transformação.

Aplicadas e analisadas as diferentes técnicas, decidimos utilizar a discretização e a standardização. É de referir que os dados apresentados nas diferentes tabelas podem ser ilusórios, uma vez que resultam de *cross-validation*. Validando as diferentes técnicas contra o conjunto de dados de teste, o resultados são substancialmente piores, principalmente no que diz respeito à métrica AUROC. Isto pode ser combatido através do balanceamento do conjunto de dados de treino, tema abordado na secção seguinte.

2.3 *Dataset Balancing*

O conjunto de dados de treino disponibilizado encontra-se desbalanceado, conforme se pode observar na figura 8:

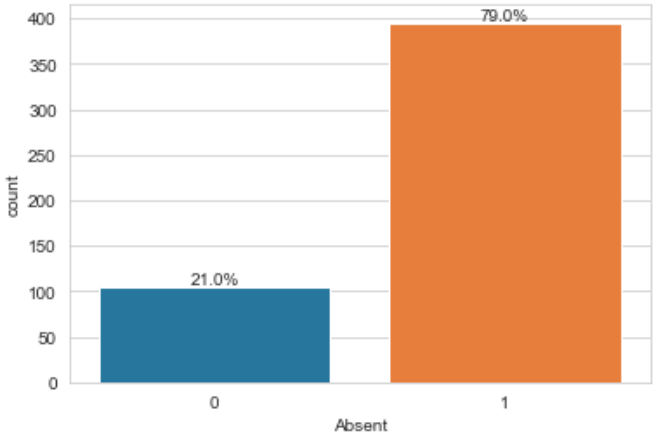

Figura 8: Número de registos por valor da característica *Absent*

Nas 500 observações que compõem o *dataset* de treino, apenas 105 representam situações em que uma determinada pessoa não faltou ao trabalho. Ora, se nada for feito em contrário, ao treinar um modelo classificador com este conjunto de dados, este tenderá a classificar todas as observações futuras como pertencendo à classe maioritária. No entanto, a *accuracy* deste classificador pode ser elevada (quando o conjunto de teste também é desbalanceado), o que poderá induzir em erro. Deste modo, convém identificar outras métricas que nos permitam avaliar se a *performance* de um classificador é adequada ao problema. Com base na figura 9, podemos estabelecer que valores pretendemos melhorar.

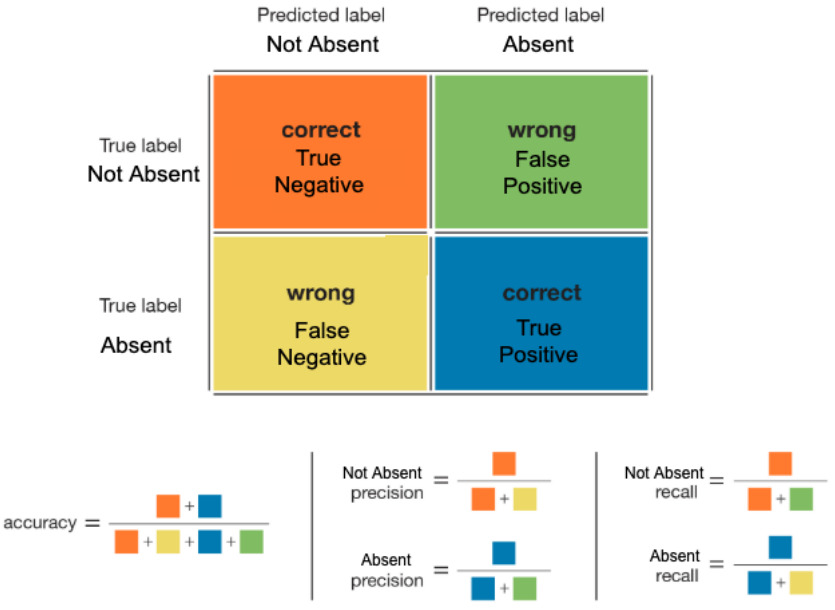

Figura 9: Matriz de confusão, adaptado de [5]

Pela figura 9, podemos concluir que a métrica *recall* representa quantos registos de uma

determinada classe conseguimos classificar corretamente, enquanto que a métrica *precision* representa a taxa de acerto das previsões relativas a uma classe. Ou seja, enquanto que a primeira quantifica a capacidade que o modelo apresenta para identificar uma classe, a segunda indica quão confiável são as previsões de uma classe.

Quando o modelo é tendencioso para a classe maioritária ('Absent'), a maioria das suas classificações será *true positive* ou *false positive*, quando prevê que uma pessoa vai faltar, mas esta não falta. Por sua vez, a medida *recall* para a classe minoritária será muito baixa, assim como a precisão com que classificamos estas observações. Assim, o objetivo de balancear o modelo passa por conseguir melhores valores estas métricas. No entanto, e transpondo o problema para o mundo real, é preferível prever que uma pessoa faltará e tal não se suceder, do que o contrário, isto é, prever que uma pessoa estará presente e esta faltar. Em suma, pretendemos reduzir o número de falsos positivos, sem que isso aumente de forma significativa o número de falsos negativos.

2.3.1 Estratégias para balancear o conjunto de dados

As estratégias que foram utilizadas para proceder ao balanceamento do conjunto de dados de treino encontram-se representadas na figura 10:

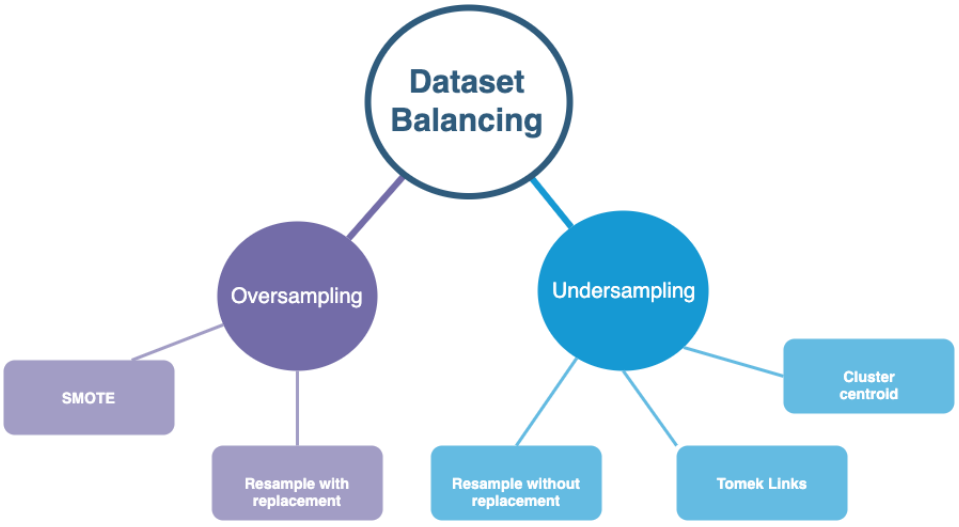

Figura 10: Procedimentos para balancear conjuntos de dados

Estas dividem-se entre duas abordagens distintas: igualar o número de observações da classe minoritária ao número de observações da classe maioritária (*oversampling*) ou reduzir o número de observações da classe maioritária ao número de observações da classe minoritária (*undersampling*).

No contexto de *oversampling*, a forma mais simples para balancear um conjunto de dados consiste em, simplesmente, repetir múltiplas vezes as observações da classe minoritária (*resample with replacement*). Outra forma, *Synthetic Minority Oversampling Technique* (SMOTE), passa por considerar pontos vizinhos e interpolar a sua informação de forma a que uma nova observação seja gerada. O funcionamento desta técnica encontra-se representado na figura 11:

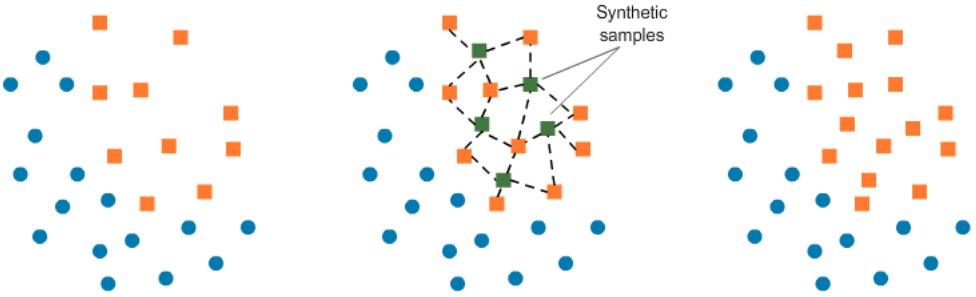

Figura 11: Aplicação de SMOTE no balanceamento de *datasets* [6]

No contexto de *undersampling*, podemos simplesmente selecionar aleatoriamente exemplos da classe maioritária (*resample without replacement*), ou então recorrer a técnicas como *Tomek Links* ou *Cluster Centroid*. Com *Cluster Centroid* são gerados centroides através da aplicação de algoritmos de *clustering* (K-means). A técnica *Tomek Links*, ao contrário das restantes, não procura igualar o número de observações de ambas as classes. Um *Tomek link* é um par de observações muito próximas, mas que pertencem a classes opostas. Esta técnica procura identificar este tipo de ligações e remover a observação que pertença à classa maioritária. Desta forma, aumentamos a separação entre as duas classes promovendo um processo de classificação mais simples. O funcionamento desta técnica encontra-se representado na figura 12

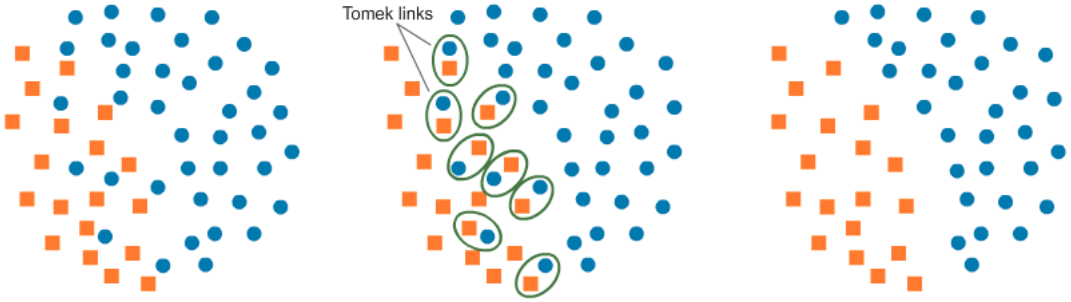

Figura 12: Aplicação de *Tomek Links* no balanceamento de *datasets* [6]

Na tabela 10 encontra-se um sumário dos melhores resultados obtidos para cada um dos métodos supramencionados.

	Técnica	**Accuracy**	**AUROC**	**'Not Absent' (Recall, Precision)**	**'Absent' (Recall, Precision)**
Multi-layer Perceptron	Resample with replacement	0.738	0.540	(0.227, 0.256)	(0.852, 0.831)
SGDClassifier	Resample with replacement	0.717	0.553	(0.295, 0.260)	(0.811, 0.837)
SVM linear	SMOTE	0.667	0.567	(0.409, 0.250)	(0.724, 0.845)
Random Forest	Resample without replacement	0.629	0.553	(0.432, 0.229)	(0.673, 0.841)
Logistic Regression	Cluster Centroid	0.567	0.567	(0.568, 0.227)	(0.566, 0.854)
Multi-layer Perceptron	Cluster Centroid	0.604	0.529	(0.409, 0.207)	(0.648, 0.830)
Logistic Regression	Tomek Links	0.646	0.545	(0.386, 0.227)	(0.704, 0.836)

Tabela 4: Melhores resultados alcançadas por técnica de balanceamento

Podemos concluir que, com o balanceamento do conjunto de dados, podemos aumentar a eficácia na classificação da classe minoritária. No entanto, estamos a prejudicar substancialmente a classificação feita sobre a classe maioritário. Nenhum dos métodos aplicados revelou resultados dominantes, embora seja possível reverificar que os métodos de *oversampling* conseguem alcançar melhores resultados. Ao reduzir uma conjunto de 395 registos a 105 registos, estamos a perder muita informação útil pelo que os modelos deixam de conseguir identificar tão acertadamente as ocorrências da classe maioritária.

3 *Feature Selection*

Feature selection é o processo de, manual ou automaticamente, selecionar as Feature Selection que mais influenciam a variável de previsão em causa. *Features* irrelevantes, ou parcialmente relevantes, podem ter um impacto muito negativo no desempenho de um modelo. [1]

Os principais benefícios de realizar *feature selection* antes de executar a modelação dos dados são:

- **Redução do *overfitting***: dados menos redundantes significam menos oportunidade de tomar decisões baseadas no ruído.
- **Melhoria na precisão**: dados menos enganosos significam uma melhoria na precisão da modelação.
- **Redução do tempo de treino**: menos dados reduzem a complexidade dos algoritmos, e desta forma, estes realizam os treinos mais rapidamente.

3.1 Tipos de *Feature Selection*

Tal como referido anteriormente, um número elevado de *features* nos dados aumenta o risco de *overfitting* no modelo. Os métodos de *feature selection* ajudam, desta forma, a reduzir a dimensão dos dados sem perder informação fulcral. [2]

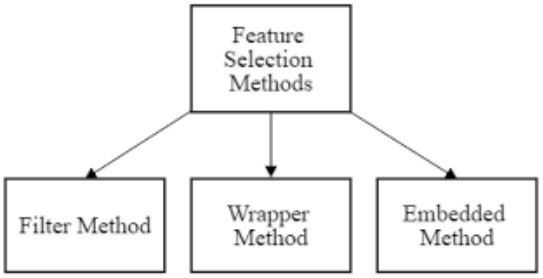

Figura 13: Tipos de métodos para *feature selection*

Filter Method

Utilizando este método, a capacidade de previsão de cada variável é avaliada independentemente do modelo. Inúmeros métodos estatísticos podem ser utilizados para cálculo desse valor. Uma forma possível é calcular a correlação da *feature* com o *target* que se tenta prever. As *features* com melhor correlação passam à fase seguinte do processo, enquanto as restantes menos interessantes são retiradas.

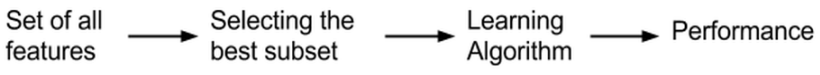

Figura 14: *Filter Method*

Wrapper Method

Os métodos de *wrapper* usam combinações de variáveis para determinar o poder preditivo, procurando encontrar a melhor combinação de variáveis. Este método testa cada *feature* em relação aos modelos de teste criados com eles, de forma a avaliar os resultados.

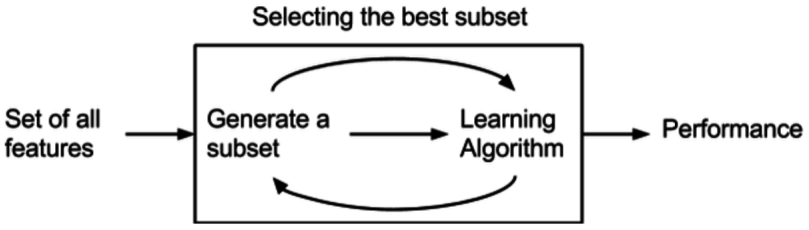

Figura 15: *Wrapper Method*

Embedded Method

Este método tenta combinar as vantagens dos dois métodos anteriores. Trata-se de um algoritmo de aprendizagem que tira proveito de seu próprio processo de seleção de variáveis e executa a seleção e classificação de recursos simultaneamente. De certa forma, não são selecionados nem rejeitados variáveis neste método. Isso controla o valor dos parâmetros, ou seja, os valores menos importantes recebem um peso mais baixo.

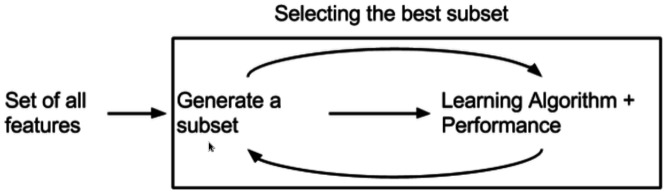

Figura 16: *Embedded Method*

3.2 Métodos aplicados

Nesta secção são abordados vários métodos que podem ser usados para *feature selection*, e consequente redução da dimensão dos dados. [3]

3.2.1 Remoção de *features* com baixa variância

Variance Threshold

VarianceThreshold é uma abordagem simples para a seleção de *features*. Resumidamente, este método remove todos os recursos cuja variação não atinge um limite estipulado. Por padrão, são removidas todas as *features* de variação zero.

Como no caso de estudo não se encontram variações nulas entre *features*, este método mantêm tudo igual, não realizando qualquer efeito sobre o processo em causa.

3.2.2 Seleção univariada de *features*

Estes métodos selecionam as melhores *features* com base em testes estatísticos univariados. Podem ser vistos como uma etapa de pré-processamento para um estimador.

Os objetos destes métodos tomam como entrada uma função de pontuação que retorna resultados univariados. Como o problema em causa se caracteriza como de classificação, as funções de pontuação utilizadas são o qui-quadrado (chi2) e o resultado dum cálculo dum valor através duma tabela ANOVA (f_classif).

SelectKBest

O *SelectKBest* remove tudo, exceto os k recursos com pontuação mais elevada.

Neste caso, foi realizada uma pesquisa pelo k que poderia retribuir os melhores resultados, neste caso, ao nível da *accuracy*.

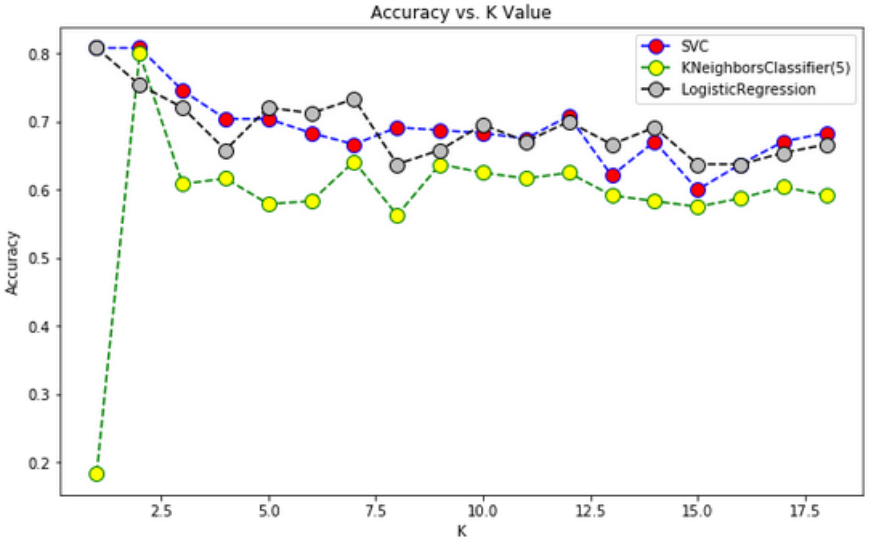

Figura 17: Pesquisa pelo *KBest*

No gráfico é possível verificar três modelos e quais os melhores valores de k no qual a *accuracy* atinge os maiores pontos. Neste caso, é possível concluir que k=2 será, em princípio, o valor no qual se tem obtêm os melhores resultados.

Não se encontram apresentados todos os modelos aplicados no projeto, no entanto, a ideia principal foi representar a possibilidade de tal poder ser realizado.

Aplicado ao projeto em causa, este método manteve as *features 'Reason for absence'* e *'Disciplinary failure'*.

SelectPercentile

O *SelectPercentile* remove tudo, exceto uma percentagem de pontuação mais alta especificada pelo utilizador.

Tal como realizado no método anterior, também seria possível realizar uma pesquisa para o melhor *Percentile* a utilizar.

Aplicado ao projeto em causa, com um valor *Percentile*=10, este método manteve as mesmas *features* que o KBest (k=2) apresentou, com classificações semelhantes.

GenericUnivariateSelect

O *GenericUnivariateSelect* permite executar a seleção univariada de recursos com uma estratégia configurável. Isso permite selecionar a melhor estratégia de seleção univariada com o estimador de pesquisa com hiper-parâmetros.

3.2.3 Eliminação recursiva de *features*

Dado um estimador externo que atribui pesos a *features* (por exemplo, os coeficientes de um modelo linear), a eliminação recursiva de *features* (RFE) seleciona recursivamente, conjuntos cada vez menores.

Primeiro, o estimador é treinado no conjunto inicial de *features* e a importância de cada uma é obtida através de um atributo coef_ ou através de um atributo feature_importances_.

Em seguida, as *features* menos importantes são removidos do conjunto atual. Esse procedimento é repetido recursivamente no conjunto removido, até que o número desejado de *features* a serem selecionadas seja atingido.

Existe também RFECV (RFE com *Cross Validation)*, que executa o RFE num *loop* de validação cruzada para encontrar o número ideal de *features*.

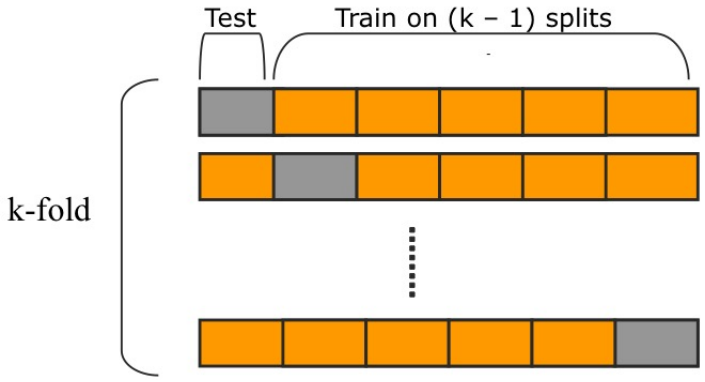

Figura 18: *Cross Validation*

Este método foi aplicado à técnicas de *Logistic Regression* e *SVC*. Exemplificando o primeiro caso:

Sem *Cross Validation*, foram mantidas doze *features*, sendo elas:

- Seasons
- Service time
- Hit target
- Disciplinary failure
- Education
- Son

- Social drinker
- Pet
- Weight
- Body mass index

Os valores atingidos foram os seguintes:

- Accuracy: 0.792
- AUROC: 0.511
- Recall 'Not Absent': 0.068
- Precision 'Not Absent': 0.250
- Recall 'Absent': 0.954
- Precision 'Not Absent': 0.820

Com *Cross Validation*, o modelo manteve todas as features, melhorando alguns valores:

- Accuracy: 0.804
- AUROC: 0.545
- Recall 'Not Absent': 0.136
- Precision 'Not Absent': 0.400
- Recall 'Absent': 0.954
- Precision 'Not Absent': 0.831

3.2.4 Utilização do *SelectFromModel*

Este modo inclui métodos meta-transformadores que podem ser usados junto com qualquer outro estimador que possua um atributo coef_ ou feature_importances_. Os recursos são considerados sem importância e removidos, se esses atributos correspondentes estiverem abaixo do parâmetro de limite fornecido. Neste caso, utilizamos um método baseado em árvores (*Tree-based*).

3.2.5 *PCA*

A análise de componentes principais (ou PCA) utiliza álgebra linear para transformar o conjunto de dados num conjunto mais pequeno. Uma propriedade do PCA é que é possível escolher o número de dimensões, assim como o componente principal, no resultado transformado.

Teoricamente, o PCA é um método de criação de novas variáveis (conhecidas como componentes principais, PCs), que são composições lineares das variáveis originais. Os valores dos PCs criados pelo PCA são conhecidos como pontuação dos componentes principais (PCS). O número máximo de novas variáveis é equivalente ao número de variáveis originais.

Os valores de VFs maiores que 0,75 são considerados "fortes", os valores que variam de 0,50-0,75 são considerados "moderados"e os valores que variam de 0,30-0,49 são considerados carga fatorial "fraca".

No projeto em causa, foi possível concluir que os valores fortes presentes foram *'Transportation expense'* e *'Work load Average/day'*.

4 *Model Evaluation & Parameter Fine-tuning*

Após o estudo de diferentes técnicas para pré-processar os dados, tanto a nível da sua transformação como no que diz respeito à seleção das caraterísticas mais importantes, procedemos à avaliação de diferentes combinações de técnicas e modelos, tendo como o objetivo obter o modelo que conseguisse atingir uma maior *accuracy*, sem comprometer a classificação da classe 'Not Absent'.

Ao longo das fases precedentes, fomos utilizando diferentes modelos para avaliar a eficácia das técnicas a serem estudadas, entre eles:

- Logistic regression;
- SGDClassifier;
- KNearest Neighbors (5);
- SVM-rbf;
- SMV-linear;
- Gaussian naive bayes;
- Gaussian Process;
- Decision Tree;
- Multi-layer Perceptron;
- AdaBoost;
- Random Forest.

No decorrer deste processo, alguns dos modelos enumerados foram apresentando, regularmente, resultados que se destacavam. Por esse motivo, tentamos proceder à sua *hyper-parameterization*, de forma a maximizar a performance dos mesmos.

Primeiro, aplicamos `GridSearch` sobre um `Support Vector Classifier (SVC)`. Os parâmetros sobre os quais essa procura incidiu foram os seguintes:

```
param_grid_svc = {
    'class_weight': ['balanced', None],
    'C': [0.1,1, 10, 100],
    'gamma': [1,0.1,0.01,0.001],
    'kernel': ['rbf', 'linear']
}
```

Com este tipo de classificador, o melhor resultado que conseguimos atingir foi:

- Accuracy: 0.721
- AUROC: 0.529
- Recall 'Not Absent': 0.227
- Precision 'Not Absent': 0.233
- Recall 'Absent': 0.832

- Precision 'Not Absent': 0.827

O parâmetros selecionados foras os seguintes: C = 10, class_weight = 'balanced', gamma = 1, kernel = 'rbf'. Na obtenção destes resultados, os dados foram discretizados, as caraterísticas selecionadas através do método `selectPercentile` (percentil 25) e procedeu-se ao balanceamento dos dados de treino com *random over sampling*.

Outro classificador que vinha a apresentar bons resultados era `K-Nearest Neighbors`. Com objetivo de identificar o melhor valor de k e de outros parâmetros a utilizar, efetuamos `GridSearch` sobre os seguintes parâmetros:

```
grid_params_knn = {
    'n_neighbors' : [3,5,7,11,13, 15, 17, 25, 30, 50],
    'weights' : ['uniform', 'distance'],
    'metric' : ['euclidean', 'manhattan']
}
```

O melhor resultado obtido deu-se com os mesmo pré-processamento que no caso anterior, usando a métrica euclidiana, 15 vizinhos, prioritizando vizinhos que se encontrem mais próximos (weights = distance). Foram registados os seguintes valores:

- Accuracy: 0.667
- AUROC: 0.540
- Recall 'Not Absent': 0.341
- Precision 'Not Absent': 0.227
- Recall 'Absent': 0.740
- Precision 'Not Absent': 0.833

Tentamos ainda efetuar `GridSearch` sobre *ensemble methods*, mais especcífico, sobre um `RandomForestClassifier`. Este teste passou por modificar o número de estimadores utilizados e a profundidade que estes poderiam ter. O melhor resultado deu-se com 100 estimadores com um profundidade máxima de 20 unidades. As métricas registadas foram:

- Accuracy: 0.704
- AUROC: 0.528
- Recall 'Not Absent': 0.250
- Precision 'Not Absent': 0.224
- Recall 'Absent': 0.806
- Precision 'Not Absent': 0.827

Como os modelos avaliados estavam a apresentar melhores resultados quando os dados eram discretizados, procuramos modelos que estivessem predispostos a lidar com este tipo de dados. Encontramos uma variante do modelo classificador `Naive Bayes`, o modelo `Categorical Naive Bayes`. Os valores alcançados por este classificador foram:

- Accuracy: 0.729
- AUROC: 0.570
- Recall 'Not Absent': 0.318
- Precision 'Not Absent': 0.286
- Recall 'Absent': 0.821
- Precision 'Not Absent': 0.843

Os valores registados por este modelo são bastante satisfatórios, na medida que em quase um terço das observações pertencentes à classe 'Not Absent' estão a ser bem classficadas, sem prejudicar as previsões da classe maioritária. O pré-processamento realizado para a obtenção deste resultados foi o mesmo que nos casos anteriores. Os valores registados por este modelo apenas foram igualados por um classificador de àrvores de decisão. Ao contrário dos restantes modelos, o `DecisionTreeClassifier` não precisou de pré-processamento de dados, sendo que a sua performance diminuía se tal fosse feito. Os valores registados por este classificador foram:

- Accuracy: 0.733
- AUROC: 0.564
- Recall 'Not Absent': 0.295
- Precision 'Not Absent': 0.283
- Recall 'Absent': 0.832
- Precision 'Not Absent': 0.840

Analisando os resultados obtidos até então, podemos verificar que sempre que se verifica um aumento na métrica AUROC, a *accuracy* do modelo baixa. No entanto, quanto maior for a AUROC, mais capaz é um modelo de distinguir entre as classes de dados. Tal acontece pois o aumento na métrica AUROC é acompanhada por um aumento do *recall* e *precision* na classificação da classe minoritário. No entanto, a classificação da classe maioritária é ligeiramente prejudicada, baixando substancialmente a *accuracy* do modelo. Com objetivo de inverter esta tendência, recorremos a um **VotingClassifier** composto por 5 modelos cujo valor da AUROC era satisfatório.

```
clf1 = LogisticRegression()
clf2 = SVC(C= 100, class_weight = 'balanced', gamma= 0.1, kernel= 'rbf')
clf3 = KNeighborsClassifier(metric= 'manhattan', n_neighbors = 17, weights = 'distance')
clf4 = CategoricalNB()
clf5 = RandomForestClassifier(bootstrap= False, max_depth= 10, n_estimators= 2000)

eclf1 = VotingClassifier(estimators=[
    ('lr', clf1), ('svc', clf2), ('knn', clf3), ('cnb', clf4), ('rfc', clf5)], voting='hard')
```

Os valores registados por este sistema foram os seguintes:

- Accuracy: 0.721
- AUROC: 0.529

- Recall 'Not Absent': 0.227
- Precision 'Not Absent': 0.233
- Recall 'Absent': 0.832
- Precision 'Not Absent': 0.827

Analisando os valores apresentados, podemos concluir que este sistema não superou as métricas atingidas por outros classificadores.

5 Conclusão

A realização deste trabalho permitiu a análise detalhada das diferentes fases de desenvolvimento de um modelo para previsão sobre um conjunto de dados, desde o pré-processamento ao *fine tuning* dos parâmetros.

A tarefa que nos foi proposta foi especialmente dificultada pelo facto do conjunto de dados ser desbalanceado. Embora a métrica que pretendíamos maximizar fosse a *accuracy*, como o modelo era desbalanceado, uma *accuracy* alta podia ser ilusória. De facto, o conjunto de dados de teste era composto por 240 amostras, das quais 196 pertencem à classe 'Absent', um modelo que classificasse todos os registos como pertencentes a esta classe teria uma *accuracy* de 0.816. Posto isto, foi necessário estudar outras métricas, tais como a AUROC (*Area Under the Receiver Operating Characteristics*) e explorar diversas formas de balancear o conjunto de dados. Das formas encontradas, nenhuma pareceu ser totalmente apropriada. Estas não tinham a expressividade necessária, principalmente as técnicas de *undersampling* uma vez que ao transformar 395 registos em 105, se perde muita informação útil. Mesmo assim, conseguimos aumentar a precisão na identificação dos casos da classe minoritária.

A necessidade de selecionar as caraterísticas mais relevantes também nos permitiu explorar diversas técnicas, sendo que nos surpreendeu a capacidade que alguns modelos exibiram para fazer classificações corretas com base em poucas caraterísticas.

Por fim, tarefa de *fine tuning* dos modelos foi dificultada pelo facto dos métodos existentes serem orientados à maximização da *accuracy*. No nosso caso, nem sempre alcançamos valores satisfatórios com estes métodos.

Em suma, fomos capazes de atingir um modelo que consegue classificar corretamente 31.8% dos casos da classe minoritária, com uma precisão de 28.6% (2 previsões corretas por cada 5 erradas). Em relação à previsão de faltas ao trabalho, este modelo consegue identificar 82.1% dos casos, com uma precisão de 84.3%. O resultado atingido é satisfatório, no entanto a previsão da classe minoritária pode ser melhorada.

Referências

[1] Raheel Shaikh. "Feature Selection Techniques in Machine Learning with Python"Medium, Towards Data Science, 28 Oct. 2018, https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e. Accessed 13 Dez. 2019.

[2] Sagar Rawale. "Feature Selection Methods in Machine Learning."Medium, 1 Aug. 2018, https://medium.com/@sagar.rawale3/feature-selection-methods-in-machine-learning-eaeef12019cc. Accessed 13 Dez. 2019.

[3] "1.13. Feature selection - Scikit-Learn 0.22 Documentation."Scikit-Learn.Org, 2019, https://scikit-learn.org/stable/modules/feature_selection.html.

[4] "6.3. Preprocessing Data — Scikit-Learn 0.22 Documentation." Scikit-Learn.Org, 2019, scikit-learn.org/stable/modules/preprocessing.html.

[5] Baptiste Rocca. "Handling Imbalanced Datasets in Machine Learning." Medium, Towards Data Science, 27 Jan. 2019, towardsdatascience.com/handling-imbalanced-datasets-in-machine-learning-7a0e84220f28.

[6] "Resampling Strategies for Imbalanced Datasets." Kaggle.Com, Kaggle, 15 Nov. 2017, www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets.