TÉCNICO LISBOA

# Aprendizagem Automática

## Relatório do projeto


| | | |
|---|---|---|
| Nº 93243 | Duarte Ribeiro da Silva | duarte.leao@tecnico.ulisboa.pt |
| Nº 105737 | João Lourenço de Almeida e Paiva | joaolourencopaiva@tecnico.ulisboa.pt |

Docente responsável: Maria Margarida Campos da Silveira


# Parte 1 - Regressão

## Primeiro problema

Para resolver o primeiro problema relacionado com regressão linear, o grupo decidiu implementar 7 modelos de regressão, sendo estes: *Ordinary Least Squares*, *Ridge Regression*, *Lasso*, *Elastic-Net*, *Least Angle Regression*, *LARS Lasso* e *Orthogonal matching pursuit*.

Após escolher os modelos a implementar, foi preciso compará-los para ver qual é que obteria um menor resultado para *MSE (Mean Square Error)*.

Foi usado o método *cross-validation* em todos os modelos, de forma a testar qual dos modelos se adaptava melhor aos dados de treino facultados. O método *cross-validation* consiste em dividir os dados de treino, no nosso caso, em 10 parcelas (K = 10), sendo 9 dessas parcelas para treino e a restante para validar os modelos. Este processo é depois iterado até que todas as 10 parcelas tenham sido usadas como dados de validação. Após realizar cada iteração é registado o valor de *MSE* e depois de serem realizadas as 10 iterações, é feita a média dos *MSE's* obtidos em cada iteração. Deste modo, considera-se como o modelo que melhor performance obeteve, aquele que tiver a média mais baixa de *MSE*.

Para os modelos *Ordinary Least Squares* e *Least Angle Regression* foi utilizada a função *cross_validate* da biblioteca *Sklearn*. No entanto, para os restantes modelos foram usadas funções especificas de *cross-validation*, para que se pudesse obter os melhores hiperparâmetros para cada modelo. Assim, para o modelo *Orthogonal matching pursuit* foi utilizada a função *OrthogonalmatchingpursuitCV*, de maneira a escolher o número de *non zero coefficients* que se adapta melhor aos dados de treino facultados. Para o modelo *LARS Lasso* foi utilizada a função *LassoLarsCV*, de maneira a escolher o valor de *alpha* que melhor se adapta aos dados de treino. Os restantes modelos seguiram um processo análogo. Para cada um destes casos foi criada uma lista de possíveis valores dos hiperparâmetros, associados a cada modelo, de modo a obter os que melhores resultados tinham quando testados em *cross-validation*.

Após se implementarem todos os modelos selecionados e se obterem todas as médias dos *MSE's* para cada modelo foi possível comparar os resultados e verificar que com o modelo *Orthogonal matching pursuit* obeteve o menor valor para *MSE*, como é possível observar na tabela 1.

A partir da análise da tabela 1 verifica-se que os modelos submetidos têm desempenhos muito semelhantes. No entanto, observa-se que o *Orthogonal matching pursuit* é o modelo mais indicado para este conjunto de dados, podendo-se concluir que há *features* mais relevantes que

outras, dado que o número de *non zero coefficients* obtido foi 7 e número de *features* é 10.

| Modelo | Média dos *MSE*'s |
|---|---|
| *Ordinary Least Squares* | 0.01229486 |
| *Ridge Regression* | 0.01227688 |
| *Lasso* | 0.01222462 |
| *Elastic-Net* | 0.01222501 |
| *Least angle Regression* | 0.01229486 |
| *LARS Lasso* | 0.01222416 |
| *Orthogonal Matching Pursuit* | 0.01198525 |

Tabela 1: Média dos valores de *MSE* para cada modelo de regresssão.

## Segundo problema

Para o segundo problema de regressão linear, sabe-se que os dados de treino fornecidos contém cerca de 20% de *outliers*. Começou-se por treinar com os seguintes modelos robustos: *KNN Regressor*, *Huber Regressor*, *RANSAC Regressor*, *TheilSen Regressor* e *Random Forest Regressor*, utilizando os dados de treino na sua íntegra(*data-set* com os *outliers*). Após se verificar que os resultados de *MSE*, obtidos com a utilização dos modelos robustos(tabela 2), foram muito altos em comparação com os valores de *MSE* da tabela 1, o grupo decidiu fazer a identificação e remoção dos *outliers* e, de seguida, aplicar outros modelos de regressão linear.

| Modelo | Média dos *MSE*'s |
|---|---|
| *KNN* | 6.92950078 |
| *Huber* | 6.98441759 |
| *RANSAC* | 7.41513744 |
| *TheilSen* | 6.85594328 |
| *Random Forest* | 7.35399641 |

Tabela 2: Média dos valores de *MSE* para cada modelo robusto de regressão.

Para identificação e remoção dos *outliers* foram utilizados os seguintes métodos: *Interquartile Range(IQR)*, *Leave one out(LOO)*, *Local outlier factor(LOF)*, *Elliptic Envelope* e *Isolation Forest*. Foi escolhido um parâmetro de *contamination* $= 0.2$, visto que, cerca de 20% dos dados de treino são *outliers*. Após serem removidos os *outliers* dos dados de treino é aplicado o modelo *Ordinary Least Squares* conjugado com um *5-fold Cross-Validation* para que se possa avaliar a perfomance de cada um dos métodos usados (tabela 3).

| Método | Nº de *outliers* removidos | Média dos *MSE's* |
|---|---|---|
| *IQR* | 10 | 5.60935978 |
| *LOO* | 20 | 0.01142294 |
| *LOF* | 20 | 0.8628965 |
| *Elliptic Envelope* | 20 | 1.08178245 |
| *Isolation Forest* | 20 | 1.0330641 |

Tabela 3: Média dos valores de *MSE* para cada método de remoção de *outliers*.

Com o modelo de remoção de *outliers* baseado no *IQR* identificou-se primeiro *outliers* para todas as *features* do conjunto de dados de treino na matriz dos *X's*, e, de seguida, os *outliers* no vector dos *y's*. Por último os dados removidos foram os que resultavam da interceção entre os dois processos de identificação de *outliers*, isto é se um dos pontos fosse identificado como outlier nos *X's* e não fosse nos *y's* não era removido e vice versa. A má performance deste método deve-se em grande parte há elevada percentagem de *outliers* existentes.

A partir da tabela 3, o grupo retirou uma conclusão fundamental. Esta, foi que quantos mais elementos se retirar, melhor é a performance dos modelos de regressão utilizados. Isto deve-se ao facto de se se conseguir retirar os *outliers* e por engano se remover um ponto ou outro que não o seja, o modelo irá obter, por norma, melhores resultados do que se forem retirados pontos a menos(i.e. deixar *outliers* nos dados de treino). No entanto, é necessário ter cuidado, pois não se quer retirar muitos dados a mais, assim estar-se-ia a remover dados que seriam úteis para a aprendizagem dos nossos modelos. Daí a necessidade de definir um critério de paragem na remoção de *outliers*(parâmetro de $contamination = 0.2$).

Por análise da tabela 3, verifica-se que o método que melhores resultados produz é o baseado no *Leave one out cross-validation*. A ideia essencial deste algoritmo é apresentada no algoritmo 1. Para uma representação mais ilustrativa pode-se observar a figura 1 onde o primeiro ponto a ser removido por este método, devido ao elevado erro residual, seria o que está classificado como *outlier* na figura.

**Algorithm 1** *Outlier removal via LOO CV* (*X*,*y*)

**while** *True* **do**
  **for** $i$¸$1, ..., X\ length$ **do**
    $X\_train \leftarrow (X[1], ..., X[i-1], X[i+1], ..., X[X\ length])$
    $y\_train \leftarrow (y[1], ..., y[i-1], y[i+1], ..., y[X\ length])$
    $X\_validate \leftarrow X[i]$
    $y\_predicted \leftarrow model.fit(X\_train, y\_train).predict(X\_validate)$
    $error[i] \leftarrow (y[i] - y\_predicted)^2$
  **end for**
  **if** $\max(error) > 1$ **then**
    $j \leftarrow \arg\max(error)$
    Delete element j from X and y
  **else**
    *break*
  **end if**
**end while**

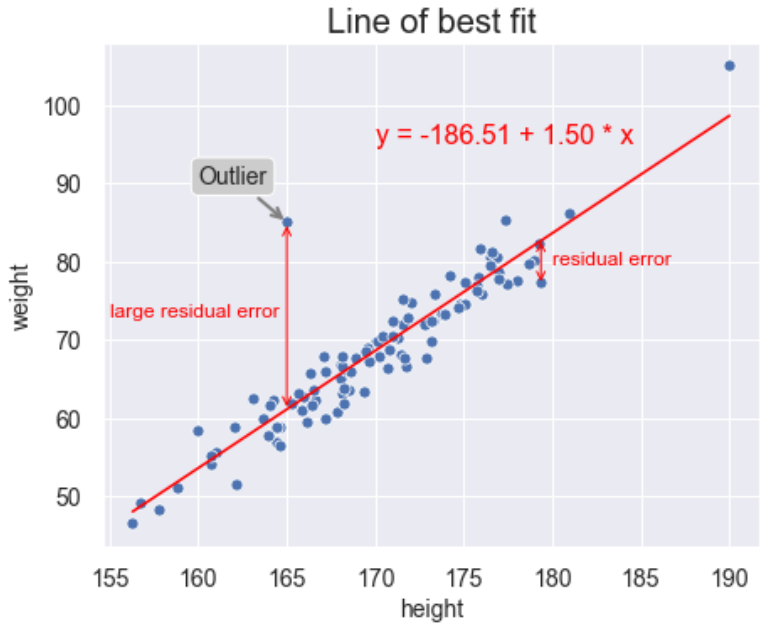



Figura 1: Exemplo da distância de um outlier a um hiperplano

De forma a implementar este algoritmo, o grupo optou por usar o *ordinary least squares* como modelo e o *MSE* como métrica para calcular a distância dos dados de treino ao hiperplano gerado(como se verifica no algoritmo 1). Isto deve-se ao facto de o erro quadrático ajudar a identificar os *outliers*, visto que se o valor tiver muito distante do hiperplano o termo quadrático irá amplificar essa distância.

Após se ter obtido o método para a remoção dos *outliers* que melhor se aplica aos dados de treino fornecidos, foi realizado a optimização dos hiperparâmetros, de cada modelo, através da utilização de um *30-fold Cross-Validation*, de modo, a obter os parâmetros que melhor se adaptam ao problema em questão. Os modelos implementados para treino foram os mesmos que no pirmeiro problema:*Ordinary Least Squares*, *Ridge*, *Lasso*, *Elastic-Net*, *Least Angle*, *Lasso LARS* e *Orthogonal Matching Pursuit*.

Por último é possível verificar na tabela 4 que o modelo que se melhor adaptou aos dados de treino fornecidos, já com os *outliers* removidos através do método *LOO* foi outra vez *Orthogonal Matching Pursuit*.

| Modelo | Média dos *MSE*'s |
|---|---|
| *Ordinary Least Squares* | 0.01132085 |
| *Ridge* | 0.01131438 |
| *Lasso* | 0.01086731 |
| *Elastic-Net* | 0.01088645 |
| *Least Angle* | 0.01132085 |
| *Lasso LARS* | 0.01086755 |
| *Orthogonal Matching Pursuit* | 0.01036877 |

Tabela 4: Média dos valores de *MSE* para cada modelo de regressão.

# Parte 2 - Classificação

Nesta segunda parte, ambos os problemas estão relacionados com classificação, um primeiro que trata de classificação binária de imagem e um segundo que aborda classificação multiclasse de imagens segmentadas onde o objetivo é classificação de píxeis.

## Primeiro problema

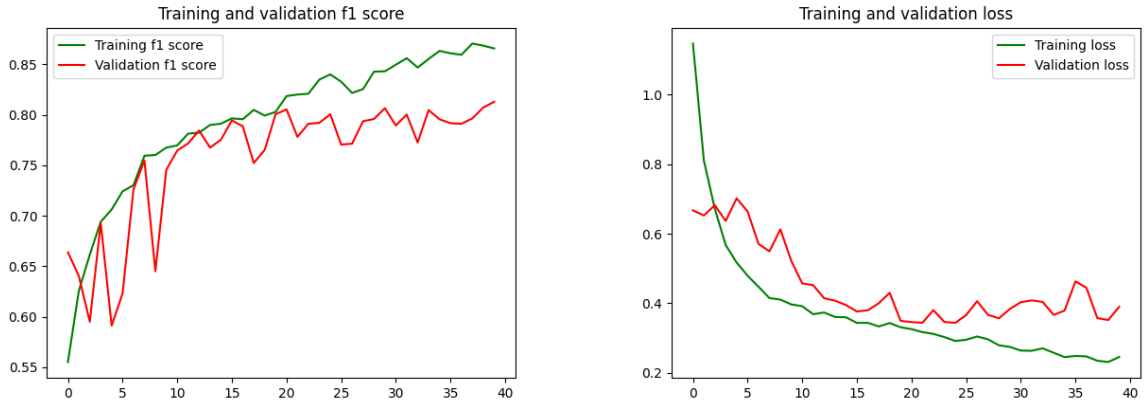
No primeiro problema, pretendia-se treinar um modelo para classificar imagens que continham padrões em asas de borboletas, onde o objetivo final era que este modelo conseguisse distinguir entre os dois padrões existentes.

Para a resolução deste problema foram implementados os seguintes modelos: *Convolutional Neural Networks* (CNN); *Gaussian Naive Bayes*; *K-Nearest Neighbors* (KNN); *Decision Tree*; *Logistic Regression*; *Support Vector Machine* (SVM). Numa primeira iteração do programa desenvolvido, começou-se por dividir os dados fornecidos para treino em dois conjuntos de dados distintos: dados de treino (80 %) e dados de validação (20 %). Após esta divisão, os modelos acima referidos foram treinados com os dados de treino e de seguida testados com os dados de validação, onde a performance do teste foi baseado na métrica do *F1-score*. Após esta primeira iteração, observou-se uma performance superior por parte das *Convolutional Neural Networks*.

Dado isto, optou-se por implementar outro modelo de CNN numa tentativa de obter um modelo com uns hiperparâmetros melhores. Os modelos de CNN implementados foram concebidos de forma a perceber o impacto das diversas características dos modelos (número de *layers*, existência ou não de *dropout*, número de filtros nas *layers* de convulsão, número de neurónios nas *dense layers*, entre outros) na eficácia da classificação de imagens. Tendo-se verificado que o modelo com melhores resultados incluía um maior número de *layers* de convulsão e maior número de neurónios, no entanto, usava também *dropouts* de modo a prevenir *overfitting*. Estes resultados podem ser verificados nas figuras 2 e 3, onde se observa que o segundo modelo reduz o *overffiting* presente no primeiro modelo (figs. 2(b) 3(b)) ao mesmo tempo que obtém uma melhor performance (figs. 2(a) e 3(a)).


(a) (b)

Figura 2: CNN1: *F1-score* no treino e validação (a). Perda no treino e validação (b)




(a) (b)

Figura 3: CNN2: *F1-score* na validação (a). Perda na validação (b)

No enunciado é referido que os dados fornecidos não estão balanceados, pelo que foi também necessário arranjar métodos para contraria este desequilíbrio entre classes. Assim, fizeram-se as seguintes 4 implementações: *data augmentation* com o intuito de reduzir o possível *overfitting*, pelo facto de gerar diferentes versões da imagem original(fazendo rotações, zoom, reflexões,etc.), tornando assim o modelo mais robusto; balanceamento dos dados fazendo *random oversampling* da classe minoritária (que como o nome indica seleciona imagens da classe minoritária de forma aleatória e repete-as nos dados de treino); uma implementação que combina as duas primeiras (*random oversampling* seguido de *data augmentation*); uma última implementação onde é feito um *re-weighting* das classes, atribuindo um peso maior à classe minoritária.

De seguida, com o intuito de obter o melhor método para o pré-processamento dos dados, avaliou-se, ao longo de 30 iterações, a performance de um modelo de CNN, baseada nas estatísticas de duas métricas (*F1-score* e *Balanced accuracy*(BACC)), de cada um dos métodos referidos acima, sempre com os mesmos dados de treino e validação. A título comparativo, também se testou a mesma performance quando se usava os dados de treino sem se aplicar qualquer tipo de pré-processamento. Assim, obtiveram-se os seguintes resultados:

| Método | F1 *Std* | F1 *Mean* | BACC *Std* | BACC *Mean* |
|---|---|---|---|---|
| *Data augmentation* | 0.01043 | 0.83066 | 0.00859 | 0.86169 |
| *Random oversampling* | 0.01041 | 0.82847 | 0.00920 | 0.86093 |
| Dados originais | 0.00856 | 0.82742 | 0.00754 | 0.85894 |
| Combinação de métodos | 0.00955 | 0.82997 | 0.00788 | 0.86215 |
| *Re-weighting* | 0.00832 | 0.82922 | 0.00710 | 0.86094 |

Tabela 5: Performance dos métodos de pré-processamento

Pelos resultados apresentados na tabela 5, verifica-se que os métodos produzem todos resultados muito semelhantes, isto deve-se ao facto, de apesar de haver uma classe minoritária, o desiquilíbrio entre classes é pequeno(as classes têm uma proporção de 60% para 40%). No entanto, o método de *re-weighting* obeteve desvios padrões menores face a valores médios idênticos. Deste modo, este foi o método utilizado ao treinar-se os modelos.

Por último, treinaram-se todos os modelos referidos, que quando testados obtiveram os seguintes resultados:

| Modelo | *CNN 1* | *CNN 2* | *Naive Bayes* | *KNN* | *Decision Tree* | *Logistic Regression* | *SVM* |
|---|---|---|---|---|---|---|---|
| *F1-score* | 0.79107 | 0.82964 | 0.53064 | 0.70640 | 0.61304 | 0.65776 | 0.74592 |

Tabela 6: *F1-score*

De facto, como se pode observar na tabela 6, confirma-se que os modelos de convulsão atingem melhores resultados em comparação com os restantes modelos.

Este resultado já era de esperar. Os modelos de convulsão são compostos por *layers* de convulsão onde são aplicados vários filtros sobre as diferentes imagens que passam em cada *layer*. Estes filtros têm intuito de identificar padrões nas imagens a partir da geração de diferentes mapas de *features*. Em que, em diferentes mapas estão realçadas *features* diferentes como, por exemplo, os contornos das imagens ou certas cores. Assim, os modelos de CNN são mais eficazes na classificação de imagens, uma vez que não interpretam a imagem como um todo, mas sim em porções das mesmas que, permitem a indentificação de padrões. É também vantajoso aos modelos CNN o facto de, ao terem múltiplas *layers* de convulsão, poderem reduzir a quantidade de dados a serem processados em cada uma das *layers* sem grande perda de informação em relação à imagem inicial.

O modelo *Gaussian Naive Bayes* assume à partida que todas as *features* são independentes, o que no problema em questão pode não fazer sentido, dado que, cada 6 features correspondem, por norma, a dois píxeis vizinhos. Assim, não se verifica uma total independência de pixel para pixel, visto que, existe uma certa continuidade no tipo de cores em píxeis vizinhos. Dado que, esses píxeis correspondem à cor da asa de uma borboleta. Como tal, este modelo demonstra-se como não adequado para este tipo de problema.

O modelo de *KNN* tem uma performance razoável, no entanto, o facto de não conseguir fazer uma melhor classificação dos dados de validação, pode-se dever às classes em questão estarem muito misturadas no *feature space*. Isto causa uma grande incerteza quando se está a classificar pela distância aos k vizinhos mais próximos, dado que o número de vizinhos de cada classe pode ser quase indêntico, estando assim mais suscetível a uma classificação errónea.

O modelo de *Decision Tree* apresenta uma má performance devido ao provável *overfit*, dado que nao se impôs limite no comprimento máximo da árvore, este ter-se-á tornado demasiado adaptada aos dados de treino. Dificultando assim, a classificação correta de novas imagens.

O modelo *Logistic Regression* é relativamente usado na resolução de problemas de classificação binária dado que se baseia na função *sigmoid*. No entanto, a sua baixa performance nesta tarefa de classificação pode ser explicada pela dificuldade em garantir que os dados são linearmente separáveis. Sendo os dados os píxeis de cada imagem, e a separação linear o que separa um dado conjunto de píxeis de uma certa classe de outro conjunto pertencente há classe oposta.

Apesar de a performance do modelo *SVM* ser razoável, o facto de não conseguir alcançar melhores resultados ao longo dos testes realizados, deve-se à dificuldade em seperar exatamente a training data, sem fazer *overfitting* da mesma.

## Segundo problema

O segundo problema, diz respeito a classificação múltipla de imagens segmentadas. Deste modo, pretendia-se treinar um modelo para classificar píxeis, a partir de segmentações de 5×5 píxeis, havendo três classes diferentes.

Inicialmente começou-se por verificar que, ao contrário do primeiro problema, os dados de treino fornecidos, encontravam-se bastante desequilibrados, havendo uma das classes bastante predominante. Deste modo, para obter uns dados de treino, que permitissem treinar os modelos, de modo a conseguir classificar bem as classes minoritárias implementaram-se dois métodos de *oversampling* para balancear os dados: *SMOTE* e o *Random oversampling*. Para testar qual dos métodos faculta uns dados de treino melhores testaram-se os dois métodos da seguinte forma, primeiro obeteve-se um *test set* correspondente a 20% dos dados de treino, de seguida treinaram-se dois modelos diferentes(um de *KNN* e outro de *Random forest*), com os dados balanceados por cada método e por último testou-se a performance destes modelos no *test set*. A métrica usada para avaliar a performance dos modelos, tendo em conta o método de *oversampling* usado, foi a *balanced accuracy*(BACC).

| Modelo | *KNN* | *Random Forest* |
|---|---|---|
| *Random oversampling* | 0.7872 | 0.8656 |
| *SMOTE* | 0.7605 | 0.8222 |

Tabela 7: *BACC score* para diferentes métodos de balanceamento

Como se pode observar, na tabela 7 o método de *Random oversampling* obteve melhores resultados, deste modo, optou-se pelo uso deste no resto do programa desenvolvido.

Escolhido o método de balanceamento, implementaram-se de seguida, os seguintes modelos: *Multi-layer perceptron (MLP)*; *K-Nearest Neighbors (KNN)*; *Support Vector Machine (SVM)*; *Decision Tree*; *Random forest*; *Logistic Regression*. Numa primeira iteração do problema desenvolvido ponderou-se usar *CNN's*, no entanto, devido à pequena dimensão das imagens em questão o grupo concordou que não seria um modelo adequado a este problema.

De maneira, a encontrar os melhores hiperparâmetros para cada classificador usou-se a função *gridsearchCV* da biblioteca *Sklearn*. Esta função permite fazer *cross validation* com diferentes hiperparâmetros, retornando por último os que obtiveram melhores resultados, baseado na *BACC*. Conjugou-se, ainda, o uso desta função, com uma função, da biblioteca *imblearn*, que permitia criar um *pipeline* em que a cada *fold* da *cross validation* os correspondentes dados treino eram balanceados sem afetar os dados de validação.

De seguida, treinaram-se os modelos usando apenas 80% dos dados de treino, com o intuito de os testar nos restantes 20% (dados de validação) e avaliar a performance de cada um deles. A partir destes testes obtiveram-se os seguintes resultados(tabela 8):

| Modelo | *MLP* | *KNN* | *SVM* | *Decision tree* | *Random forest* | *Logistic Regression* |
|---|---|---|---|---|---|---|
| *BACC* | 0.8736 | 0.9062 | 0.8621 | 0.7366 | 0.8656 | 0.7620 |

Tabela 8: *BACC* nos dados de validação (1ª implementação)

Deste modo, verifica-se que *KNN* obteve a melhor performance. No entanto, para verificar a qualidade deste resultado, desenvolveu-se um implementação em que se começou por dividir

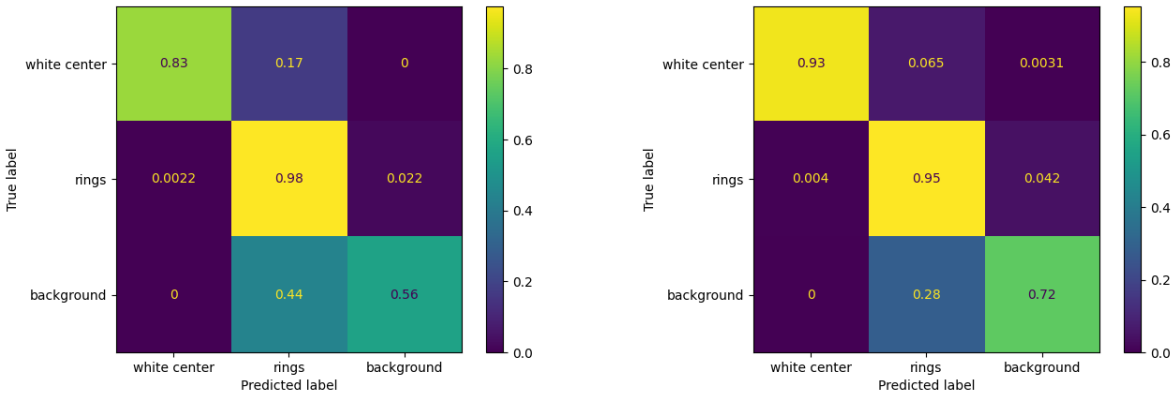
os dados de treino em 70% para treino e 30% para teste. De seguida realizou-se *5-fold cross validation*. E por último, treinaram-se os modelos com os 70% para treino e testaram-se com o *test set* Esta implementação tinha como intuito simular todo o segundo problema. Dado isto, obtiveram-se o seguintes resultados:

| Modelo | *MLP* | *KNN* | *SVM* | *Decision tree* | *Random forest* | *Logistic Regression* |
|---|---|---|---|---|---|---|
| *Cross valid. BACC* | 0.8738 | 0.8933 | 0.8709 | 0.7460 | 0.8184 | 0.7458 |
| *Test BACC* | 0.7681 | 0.7754 | 0.6633 | 0.7472 | 0.8601 | 0.5889 |

Tabela 9: *BACC* nos dados de validação (2ª implementação)

Os resultados apresentados na tabela 9 suscitaram alguma surpresa, porque apesar dos resultados de *cross validation* estarem em linha com os obtidos na implementação anterior, houve um queda drástica nos resultados de teste face aos de validação, excepto para *decision tree* e *random forest*. Estes resultados contrariam-nos, de certa forma, a intuição dado que demonstram que o classificador que melhor generalizou foi o *random forest* e que os resultados de *cross validation* não se mostram, aqui, particularmente indicativos do melhor classificador. Com isto, o grupo optou por usar *random forest* como classificador, para, primeiro, treinar com os dados de treino originais e depois fazer a previsão a partir dos dados de teste.

Apesar dos resultados algo contraditórios, verifica-se que os classificadores têm de uma forma geral uma boa performance. É importante também pereber como é que o modelo escolhido (*Random forest*) se desempenha na classificação de cada classe. Para tal recorremos a uma matriz de confusão.




(a) (b)

Figura 4: Matriz de confusão para dados: desequilibrados (a) balanceados(b)

Pode-se verificar na figura 4 que a performance, do modelo desenvolvido, na identificação da classe *background* é relativamente baixa em comparação com a performance nas restantes classes. A título comparativo, é também apresentada a matriz de confusão para quando os dados não são balanceados. Neste último caso torna-se evidente a dificuldade em classificar bem o *background*, devido ao número reduzido de padrões disponíveis para esta classe nos dados de treino originais. Evidenciando-se assim, a necessidade de balancear os dados de treino.