Aprendizagem Automática 2

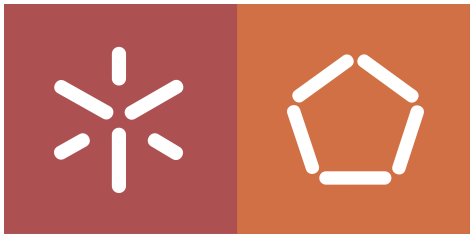
Trabalho Prático

-

Mestrado em Engenharia Informática
Universidade do Minho

Grupo nº 11	
PG41080	João Ribeiro Imperadeiro
PG41081	José Alberto Martins Boticas
PG41091	Nelson José Dias Teixeira
PG41851	Rui Miguel da Costa Meira

23 de junho de 2020



Conteúdo

Lista de Figuras

Capítulo 1

Introdução

No âmbito da unidade curricular (UC) *Aprendizagem Automática II* (AA2), foi requerida a realização de um trabalho prático para avaliação. Tal como foi proposto a 23 de abril, o grupo escolheu a opção relativa ao desenvolvimento de algoritmos/*software* no âmbito de aprendizagem máquina. Mais especificamente, optou-se pelo desenvolvimento de uma *framework* de *AutoML*, com o objetivo de obter o melhor modelo para problemas de *supervised learning* e *unsupervised learning*, de forma automática e com a menor intervenção possível por parte do programador. À *framework* idealizada foi atribuído o nome *UnicornML*.

Atualmente existem já algumas *frameworks* que abordam este tema de forma mais profunda e complexa. Destas soluções destacam-se a *Lex*, desenvolvida pela *Amazon* e que disponibiliza funcionalidades de *deep learning* relacionadas com texto e voz, o *AutoKeras*, um sistema de *AutoML* baseado em `keras` e, ainda, a *Google AutoML*, que vai ao encontro com o que o grupo deste trabalho pretende realizar. Esta última *framework* permite desenvolver modelos para utilizadores que não possuem qualquer conhecimento de *machine learning*. Consequentemente, esta acaba por ser transparente para o cliente na obtenção do resultado obtido.

Por sugestão do docente da UC, foram postos de parte os problemas de *unsupervised learning*, pela sua complexidade e menor atenção dada durante as aulas. Assim, sobram apenas os problemas de *supervised learning* que podem ser divididos em duas categorias: classificação e regressão. Mais à frente serão abordadas as duas categorias em pormenor. Para proceder à avaliação dos modelos disponibilizados pela *framework*, adotaram-se algumas métricas para cada um dos tipos de problemas mencionados acima. De salientar que o grupo teve o cuidado de avaliar situações relacionadas com o *underfitting* e *overfitting* dos modelos gerados. Para tal foram utilizadas, na generalidade, metodologias intrínsecas à otimização de hiperparâmetros.

Relativamente à estrutura deste documento, será, de seguida, exibida a planificação deste projeto, definindo alguns dos objetivos a serem alcançados. Posteriormente, parte-se para a implementação da *framework* proposta, evidenciando-se questões relacionadas com o pré-processamento de dados, os problemas e algoritmos suportados pela aplicação e também as métricas utilizadas no momento da avaliação dos modelos. Por fim, é demonstrada uma análise sobre os vários testes realizados sobre a *framework*, sumariando os objetivos atingidos neste projeto.

Capítulo 2

Planificação

Uma vez feita a escolha acerca do tema que este grupo de trabalho se propôs a efetuar, segue-se a planificação do que vai ser realizado.

- Criação do *package UnicornML* com diversas classes;
- Implementação de testes unitários para validar as funcionalidades;
- Utilização das bibliotecas *scikit-learn*, *tensorflow* e *kerastuner*;
- Código *open source* disponível para todos os utilizadores de `Python` no *PyPI*;
- Desenvolver uma *framework* que seja capaz de encontrar um modelo com uma exatidão alta;
- Implementar uma aplicação simples de utilizar, sendo apenas necessário fornecer os dados;
- Garantir a busca do modelo ótimo para um determinado conjunto de dados de forma rápida, eficiente e robusta;
- Servir esta *framework* como uma excelente base para um projeto de maiores dimensões.

Capítulo 3

Implementação

3.1 Estrutura

A *UnicornML* apresenta um estrutura simples e clara, tal como se pode observar no seguinte diagrama.

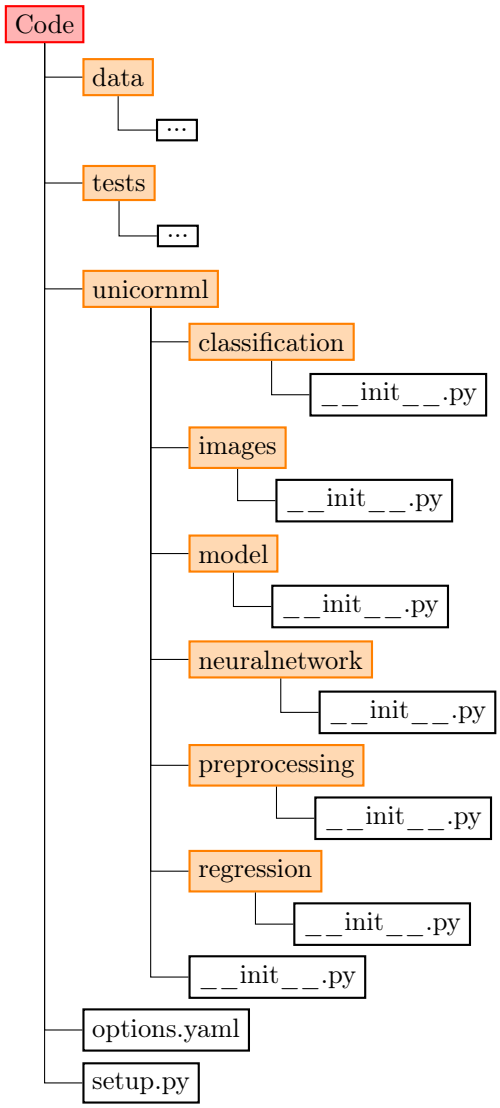


Figura 3.1: Estrutura da *framework*

Neste esquema evidenciam-se as pastas *data* e *tests*. Na primeira podem ser colocados os *datasets* para os quais se querem desenvolver modelos. Na segunda devem

ser colocados os testes a realizar, ou seja, uma classe que faça uso da *framework* e lhe passe os argumentos necessários, nomeadamente, os dados de *input* e outras opções.

Para além destas, existe uma classe principal, em `unicornML/__init__.py`, que deve ser usada para definir a computação a realizar. Assim, esta classe deve ser instanciada pela classe referida anteriormente e recebe os dados - que podem ser passados já separados ou como um ficheiro *csv* - e todas as opções. Entre as opções, o utilizador pode referir qual o tipo de problema (classificação ou regressão), quais os algoritmos que quer que sejam testados e qual a métrica a ser usada. Estas opções são facultativas, sendo que, se não forem fornecidas, a *framework* tratará de identificar o problema em causa e usará todos os algoritmos que tem disponíveis para o mesmo bem como uma métrica definida por defeito.

Nesta classe principal, é ainda realizado o pré-processamento dos dados, explicado mais à frente.

Todas as opções indicadas pelo utilizador desta *framework* são verificadas e validadas através de um ficheiro previamente definido, isto é, ***options.yaml***. Neste encontram-se todos os parâmetros suportados pela aplicação desenvolvida, ou seja, o tipo de problemas, os algoritmos e, ainda, as respetivas métricas. Eis o conteúdo do ficheiro em causa:

```yaml
Problem:
    Classification:
        algorithms:
            - logistic
            - knn
            - svm
            - kernelSVM
            - gaussianNB
            - bernoulliNB
            - decisionTree
            - randomForest
            - neuralNetwork
        metrics:
            - accuracy
            - auc
            - precision
            - recall

    Regression:
        algorithms:
            - linear
            - svr
            - decisionTree
            - randomForest
            - neuralNetwork
        metrics:
            - mse
            - mae
            - r2
```

Figura 3.2: Configuração - *options.yaml*

Numa fase final, é corretamente exibido, para o utilizador, não só o tipo do problema em questão como também os algoritmos a testar e a métrica selecionada. Consequentemente, é invocada a classe relativa ao tipo de problema de *supervised learning* escolhido, procedendo-se, também à procura do melhor modelo, ou seja, o que se ajusta de forma mais adequada ao conjunto de dados em causa.

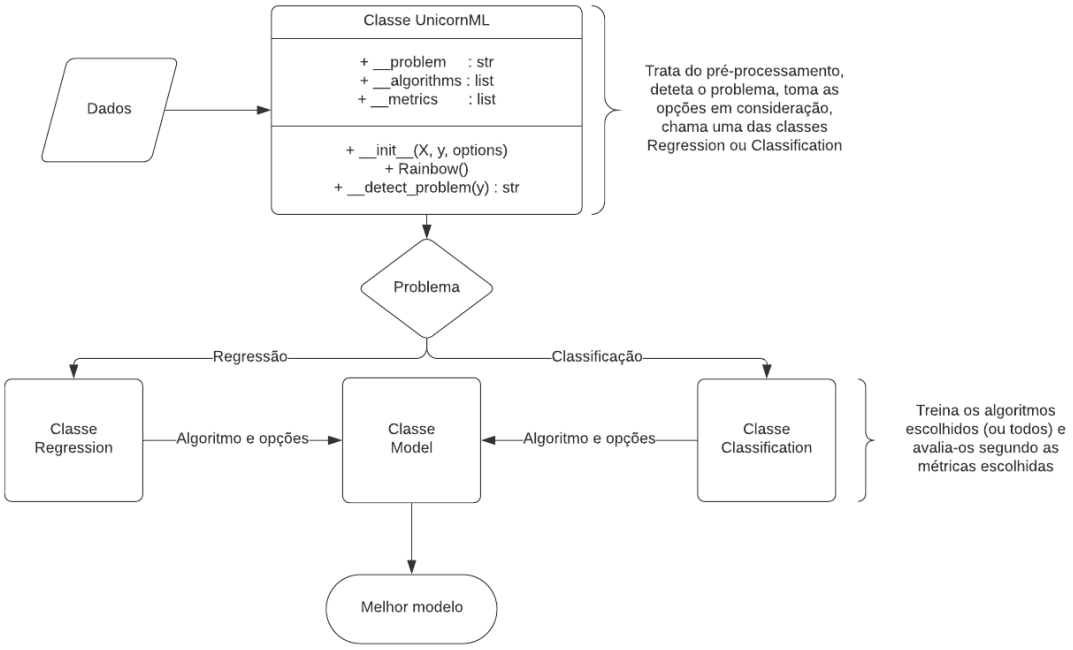


Figura 3.3: Fluxo de execução da *framework*

Posto isto, existe ainda um outro problema, tratado de forma diferente do que é exposto acima. Para o caso de imagens, o utilizador deve indicar, ao instanciar a classe principal, que o *dataset* é de imagens, passando nas opções o formato das imagens - altura, largura e profundidade. A profundidade da imagem, entenda-se, corresponde ao facto de a imagem ter ou não cor, sendo igual a 3 no caso de ter (RGB).

3.1.1 Pré-processamento

Na classe de pré-processamento realizam-se várias tarefas sobre os conjuntos de dados disponibilizados. Exibem-se de seguida as mesmas:

- Transformação dos dados:
  - Separação entre os dados relativos às variáveis independentes dos dados associados à variável de interesse ou de saída;
  - Tratamento de valores omissos: colunas com mais de 40% de valores nulos são eliminadas. Os valores numéricos são substituídos pela média dos valores da coluna enquanto que os categóricos são alterados para a moda;
  - Redimensionamento e normalização: utilização de mecanismos de *scaling* e *label encoding* da biblioteca `scikit-learn`.
- Seleção de *features*:
  - Utilização do *PCA* para obter os principais componentes das *features* com maior correlação e, por isso, com maior relevância.

3.1.2 Problema

Os problemas de aprendizagem supervisionada podem ser divididos em dois conjuntos: problemas de regressão e problemas de classificação. Como tal, é importante

perceber-se qual dos dois problemas enfrentamos, de forma a que se possa poupar tempo e recursos de computação na procura do melhor modelo. Para isso, foi pensada uma forma de identificar o tipo do problema. No entanto, tal como referido na proposta já entregue, esta não é uma prioridade, pelo que o método para já utilizado é simples e identifica apenas a presença de inteiros ou *floats* para fazer esta distinção.

No entanto, este processamento é evitado se o utilizador indicar qual dos problemas os seus dados representam. Esta indicação é dada através de uma opção, sendo passada uma de duas *strings*: `Regression` ou `Classification`.

3.1.3 Algoritmos

A *UnicornML* oferece diversos algoritmos para cada um dos tipos de problemas. O utilizador pode escolher, dentro dos algoritmos disponíveis, quais os que quer que sejam testados. No entanto, os algoritmos só serão testados se estiverem disponíveis para o tipo de problema identificado pela *framework* ou indicado pelo mesmo.

Caso o utilizador não indique quais os algoritmos que prefere que sejam testados, a *framework* testará todos os algoritmos disponíveis para o tipo de problema identificado pela mesma ou indicado pelo utilizador.

3.1.3.1 Classificação

Os algoritmos disponíveis para problemas de classificação são os seguintes:

- Regressão logística;
- *K-Nearest Neighbors* (KNN);
- *Support Vector Classifier* (SVC) - uma *Support Vector Machine* (SVM) para classificação;
- *kernel* SVM - uma SVM com uma função *kernel*, que permite a classificação em espaços de dimensão superiores;
- Classificadores Bayesianos - família de classificadores baseados na teoria de Bayes. Foram implementados dois algoritmos distintos: *Gaussian* e *Bernoulli*;
- Árvores de decisão;
- *Random Forest* - operam construindo uma multitude de árvores de decisão.
- Redes neuronais artificiais.

Estes algoritmos encontram-se na classe `Classification`, tal como se pode observar na figura 1.

3.1.3.2 Regressão

Os algoritmos disponíveis para problemas de regressão são os seguintes:

- Regressão linear;
- *Support Vector Regressor* (SVR) - uma SVM para regressão;
- Árvores de decisão;
- *Random Forest* - operam construindo uma multitude de árvores de decisão;
- Redes neuronais artificiais.

Estes algoritmos encontram-se na classe `Regression`, presente na estrutura da *framework* desta aplicação (consultar a figura 1).

3.1.4 Métricas

As métricas permitem avaliar o desempenho de um certo modelo. O utilizador também pode escolher as métricas que irão ser tomadas em consideração e, posteriormente, apresentadas. Mais uma vez, isso está limitado às métricas disponíveis para cada tipo de problema. De realçar que nem todas as métricas podem estar disponíveis num determinado momento.

3.1.4.1 Classificação

As métricas disponíveis para problemas de classificação são as seguintes:

- *Accuracy* - Percentagem de exemplos corretamente classificados (PECC). Esta métrica foi definida por defeito;
- *Auc*;
- *Precision* - Valor preditivo positivo;
- *Recall* - Sensibilidade.

3.1.4.2 Regressão

As métricas disponíveis para problemas de regressão são as seguintes:

- *Mean Square Error* (MSE) - métrica definida por defeito;
- *Mean Absolute Error* (MAE);
- *R-squared* (R^2).

3.1.5 Modelo

A classe `Model`, em `unicornML/model/__init__.py`, é o coração de toda a *framework*. Esta classe é utilizada tanto pela classe `Regression` como pela classe `Classification`. A mesma foi pensada de forma a simplificar o restante código e reduzir duplicações do mesmo. Nesta são incorporados vários tipos de informação, nomeadamente o conjunto de dados (treino e teste), a métrica selecionada para realizar a estimação do erro associada ao modelo escolhido e, ainda, os resultados obtidos após efetuar a otimização dos hiperparâmetros.

3.1.5.1 Otimização de hiperparâmetros

Tal como foi referido anteriormente, é nesta classe onde é realizada a procura do melhor modelo, segundo as opções escolhidas pelo utilizador. Para desempenhar a otimização de hiperparâmetros dos modelos considerados, o utilizador desta *framework* de *AutoML* pode indicar qual a otimização que pretende realizar. Nesta tomada de decisão este pode explicitamente indicar que não quer realizar tal tarefa. Por omissão, é realizada uma procura aleatória dos melhores parâmetros (*RandomizedSearch*). No caso das redes neuronais, é utilizado o método de otimização bayesiana com recurso à biblioteca `keras`.

3.1.6 Imagens

A classe `Images`, em `unicornML/images/__init__.py`, é responsável pelo desenvolvimento de modelos para imagens.

Capítulo 4

Testes e análise de resultados

Para validar todos os aspetos considerados durante a implementação deste trabalho, foram incorporados nesta *framework* vários conjuntos de dados que servem de teste à mesma. Como tal, foram adicionados no total 12 conjuntos de dados distintos entre si, havendo tanto problemas de regressão como de classificação com múltiplas classes a serem modelados. Estes conjuntos de dados foram obtidos em https://machinelearningmastery.com/standard-machine-learning-datasets, sendo que alguns deles contêm informação sobre os valores esperados da precisão do modelo gerado. Assim, faremos uma comparação dos resultados obtidos com os esperados para alguns destes dados.

De forma a exibir apenas alguns dos resultados obtidos, expõem-se agora os conjuntos de dados considerados para o efeito. Para cada um, é indicado o valor mínimo esperado (e os melhores resultados, no caso de existirem). São ainda apresentados os valores obtidos pela nossa framework, tanto na fase da optimização dos hiperparâmetros como o treino com todos os dados disponíveis:

- *Pima Indians Diabetes Dataset* Desempenho mínimo esperado: 65%. Melhores resultados previstos: 77%. Resultado: 80.5% (accuracy 76.5%), Gaussian Nayve Bayes
- *Sonar Dataset* Desempenho mínimo esperado: 53%. Melhores resultados previstos: 88%. Resultado: 83% (accuracy 96.6%), KNN
- *Banknote Dataset* Desempenho mínimo esperado: 50%. Resultado: 97.8% (accuracy 98%), Logistic Regression
- *Iris Flowers Dataset* Desempenho mínimo esperado: 26%. Resultado: 93.3% (accuracy 98.7%), Decision Tree Classification
- *Abalone Dataset* Desempenho mínimo esperado: 16%. Resultado: 27.5% (accuracy 28.4%), Kernel Support Vector Machine
- *Ionosphere Dataset* Desempenho mínimo esperado: 64%. Melhores resultados previstos: 94%. Resultado: 93.3% (accuracy 98.7%), Decision Tree Classifier
- *Wheat Seeds Dataset* Desempenho mínimo esperado: 28%. Resultado: 95.2% (accuracy 91.4%), Neural Network
- *Imagens Gatos e Cães* Desempenho mínimo esperado: 95%. Resultado: 87.5%, CNN

Como pode ser notado, quase todos os casos obtiveram valores muito superiores ao esperado, o que demonstra as vantagens da nossa framework quando comparada com a otimização manual de hiperparâmetros. A única exceção foi o conjunto de

imagens dos gatos e cães, onde o resultado obtivo foi um pouco inferior ao esperado. Isto deve-se a ter sido utilizado um subconjunto parcial dos dados totais e a rede ter sido treinada durante apenas uma época. Imposemos esta restrição para este caso específico para acelerar o processo de treino, mas note-se que não é um problema da framework em si.

Capítulo 5

Conclusão

- Falar dos objetivos inicialmente traçados;
- Falar da realização das tarefas do trabalho futuro;
- Falar levemente das ilações retiradas da análise dos resultados;
- Sumariar o trabalho prático como um todo.

Apêndice A

Documentação

- *Python* 3:
  https://docs.python.org/3/
- *Pandas*:
  https://pandas.pydata.org/docs/
- *Numpy*:
  https://numpy.org/doc/
- *Scipy*:
  https://docs.scipy.org/doc/scipy/reference/
- *Scikit-learn - supervised learning*:
  https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
- *Scikit-learn (API)*:
  https://scikit-learn.org/stable/modules/classes.html
- *Tensorflow* - `keras`:
  https://www.tensorflow.org/api_docs/python/tf/keras
- *Kerastuner*:
  https://keras-team.github.io/keras-tuner/