

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

1	Introdução	3
2	Planificação	4
3	Implementação	5
3.1	Estrutura	5
3.1.1	Pré-processamento	7
3.1.2	Problema	7
3.1.3	Algoritmos	8
3.1.3.1	Classificação	8
3.1.3.2	Regressão	8
3.1.4	Métricas	9
3.1.4.1	Classificação	9
3.1.4.2	Regressão	9
3.1.5	Modelo	9
3.1.5.1	Otimização de hiperparâmetros	9
4	Testes e análise de resultados	10
5	Conclusão	11
A	Observações	12

Lista de Figuras

3.1	Estrutura da <i>framework</i>	5
3.2	Configuração - <i>options.yaml</i>	6
3.3	Fluxo de execução da <i>framework</i>	7

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.

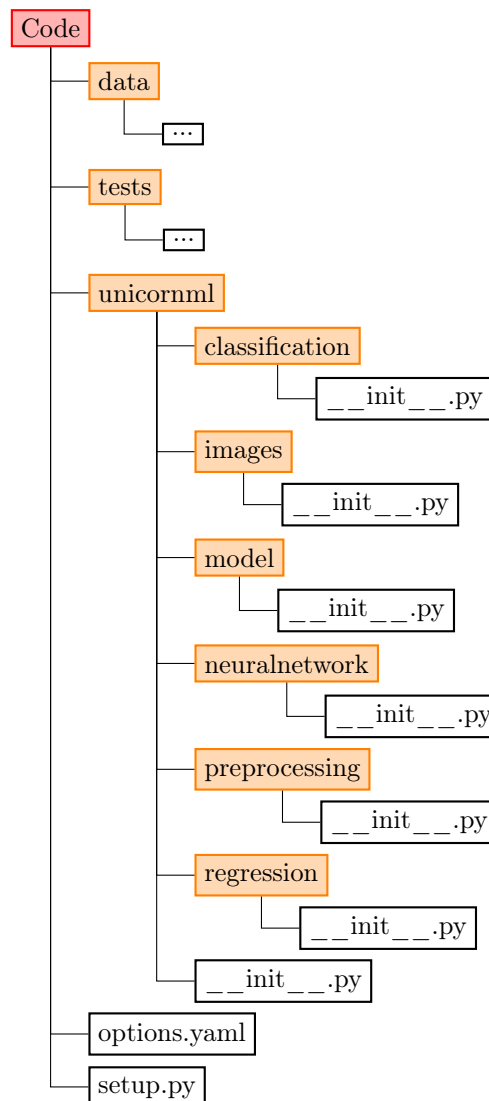


Figura 3.1: Estrutura da *framework*

Neste esquema evidenciam-se as pastas *data* e *tests*. Na primeira encontram-se disponíveis todos os *datasets* que servem de *input* à *framework* implementada. Na

segunda localizam-se os testes unitários associados a cada um dos *datasets* mencionados anteriormente. Para além disso, existe uma classe principal, com o mesmo nome da *framework*, que permite treinar dados supervisionados para problemas de classificação e regressão. Numa fase inicial, é despoletado o mecanismo associado ao pré-processamento dos dados fornecidos, etapa essa que se encontra implementada na respetiva classe. Posteriormente, são processadas todas as informações iniciais, que podem incluir:

- **Problema** - tratamento de um problema de regressão ou classificação; caso não seja fornecida essa informação, é identificado o tipo do problema em questão durante a etapa de pré-processamento;
- **Algoritmos** - quais os algoritmos, dentro dos disponibilizados, que o utilizador pretende que sejam testados (passado em forma de lista);
- **Métricas** - quais as métricas utilizadas para proceder à avaliação de modelos (passado em forma de lista).

Todas estas informações são passadas pelo terminal e são opcionais. Caso não sejam fornecidas, serão testadas todas as hipóteses oferecidas pela *framework*. O utilizador pode indicar o problema, os algoritmos e as métricas que pretender, no entanto pode sempre optar por escolher apenas alguns destes campos. A decisão tomada deve, obviamente, ter sempre em consideração as limitações e consequências da própria.

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:

```
1 Problem:
2   Classification:
3     algorithms:
4       - logistic
5       - knn
6       - svm
7       - kernelSVM
8       - gaussianNB
9       - bernoulliNB
10      - decisionTree
11      - randomForest
12      - neuralNetwork
13     metrics:
14       - accuracy
15       - auc
16       - precision
17       - recall
18
19   Regression:
20     algorithms:
21       - linear
22       - svr
23       - decisionTree
24       - randomForest
25       - neuralNetwork
26     metrics:
27       - mse
28       - mae
29       - 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 e as métricas selecionadas. Consequentemente, é invocada a classe relativa ao tipo de problema de *supervised learning* escolhido, procedendo-se, em última instância, à computação do melhor modelo, ou seja, o que se ajusta de forma mais adequada ao conjunto de dados em causa.

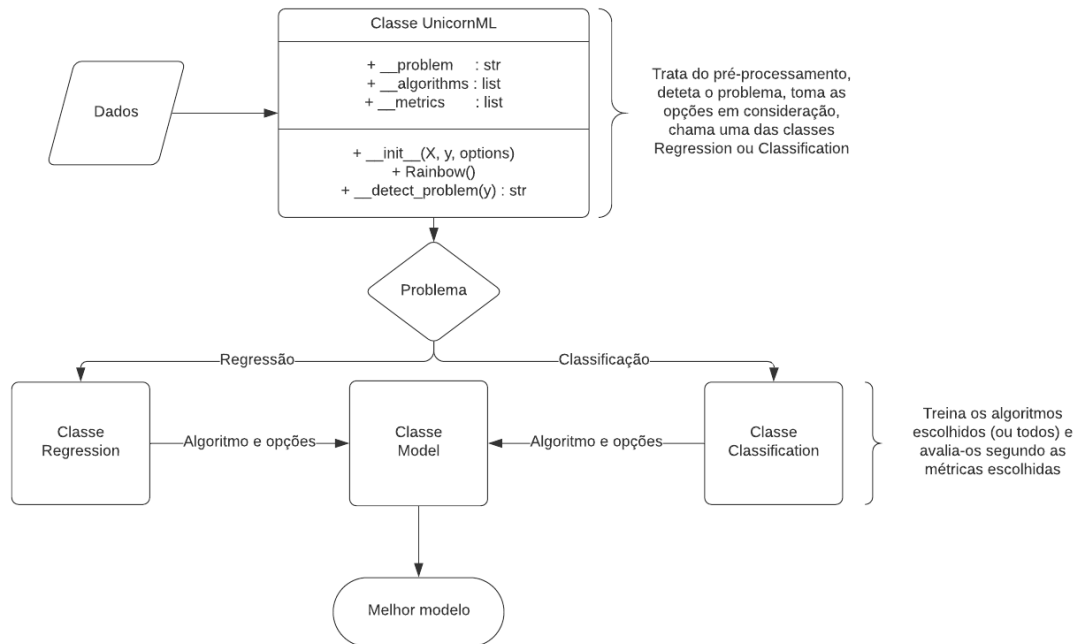


Figura 3.3: Fluxo de execução da *framework*

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 omissão;
- *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 omissão;
- *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`.

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 a serem modelados. Para estes conjuntos de dados é possível indicar a precisão de todos os modelos tomados em consideração, sendo que no fim de cada teste é apontado o melhor.

De forma a exibir apenas alguns dos resultados obtidos, expõem-se agora os conjuntos de dados considerados para o efeito:

- *50_Startups.csv*;
- *Social_Network_Ads.csv*;
- ...

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

Observações

- Documentação *Python* 3:
`https://docs.python.org/3/`
- Documentação *Pandas*:
`https://pandas.pydata.org/docs/`
- Documentação *Numpy*:
`https://numpy.org/doc/`
- Documentação *scikit-learn* - API:
`https://scikit-learn.org/stable/modules/classes.html`
- Documentação *scikit-learn* - supervised learning:
`https://scikit-learn.org/stable/supervised_learning.html#supervised-learning`