Aprendizagem Automática 2

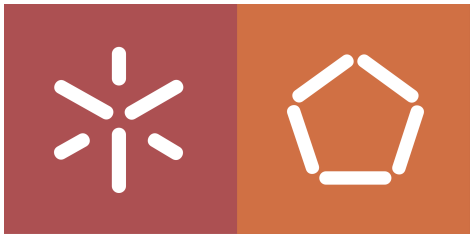
Trabalho Prático

-

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

12 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 da 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*.

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.

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.

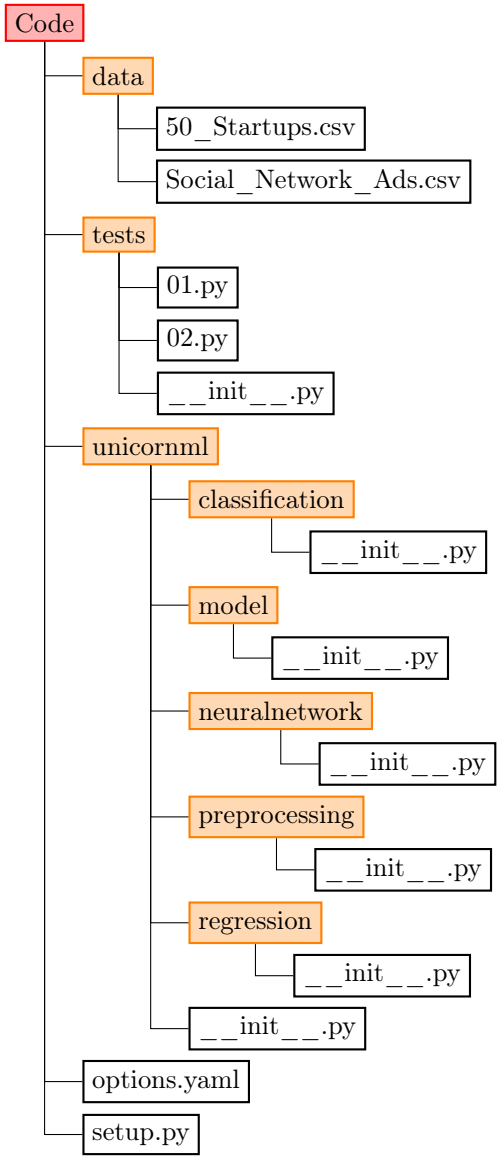


Figura 3.1: Estrutura da *framework*

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** - métricas a avaliar (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:

```yaml
Problem:
    Classification:
        algorithms:
            - logistic
            - knn
            - svm
            - kernelSVM
            - naiveBayes
            - decisionTree
            - randomForest
            - neuralNetwork
        metrics:
            - accuracy
            - f1
            - precision
            - recall

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

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 *supervided learning* escolhido, procedendo-se, em última instância, à computação do melhor modelo, ou seja, o que se ajusta de forma adequada ao conjunto de dados em causa.

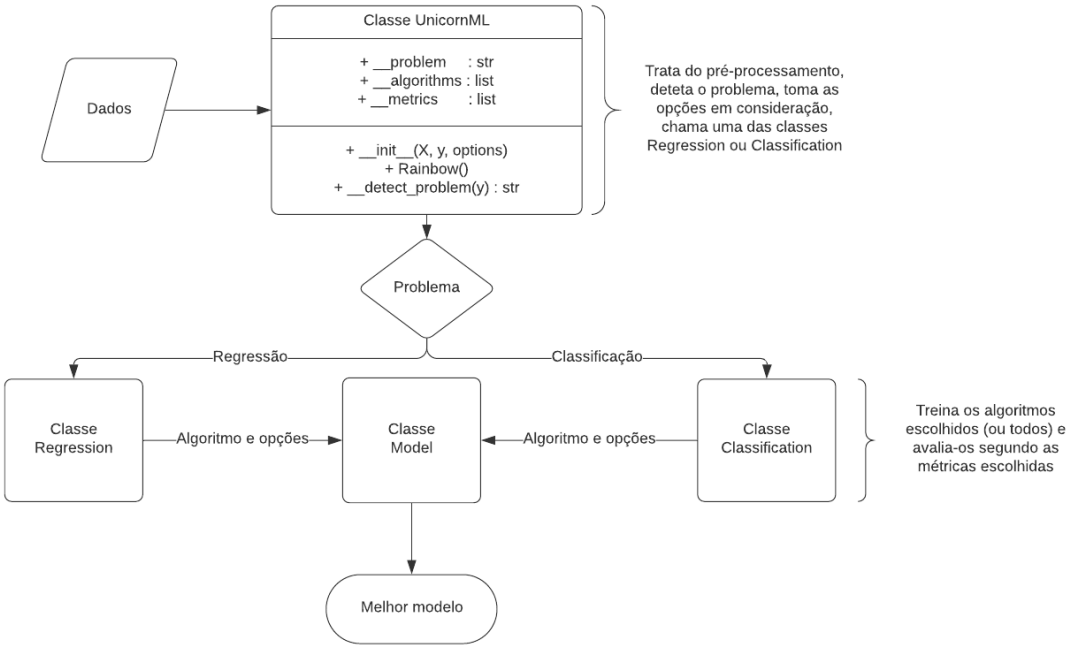


Figura 3.3: Fluxo de execução da *framework*

3.1.1 Pré-processamento

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 quatro algoritmos diferentes: *Gaussian*, *Multinomial*, *Bernoulli* e *Complement*;
- Árvore de decisão;
- *Random Forest* - operam construindo uma multitude de árvores de decisão.

Estes algoritmos encontram-se na classe `Regression`, em `unicornML/regression/__init__.py`.

3.1.3.2 Regressão

Os algoritmos disponíveis para problemas de classificação são os seguintes:

- Regressão linear;
- Regressão polinomial - os polinómios testados variam entre grau 2 e grau igual ao número de colunas da base de dados;
- *Support Vector Regressor* (SVR) - uma SVM para regressão;
- Árvore de decisão;
- *Random Forest* - operam construindo uma multitude de árvores de decisão.

Estes algoritmos encontram-se na classe `Classification`, em `unicornML/classification/__init__.py`.

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*;
- *F1*;
- *Precision*;
- *Recall*.

3.1.4.2 Regressão

As métricas disponíveis para problemas de regressão são as seguintes:

- *R-squared* (R^2);
- *Mean Square Error* (MSE).

3.1.5 Otimização de hiperparâmetros

A classe `Model`, em `unicornML/model/__init__.py`, é o coração de toda a *framework*. A mesma foi pensada de forma a simplificar o restante código e reduzir duplicações do mesmo. É, ainda, onde é feita a procura do melhor modelo, segundo as opções escolhidas pelo utilizador. Esta classe é utilizada pelas classes `Regression` e `Classification`.

Encontramos aqui mais uma opção que será futuramente disponibilizada ao utilizador: a escolha do método de otimização - *grid search* (*randomizedSearch*) ou otimização bayesiana (Bayes). Por defeito, o método de otimização utilizado será `randomizedSearch`. Futuramente será também possível optar pela otimização bayesiana, embora esta ainda não esteja implementada.

3.1.6 Modelo

Capítulo 4

Testes e análise de resultados

Capítulo 5

Conclusão

Será um pouco prematuro tentar tirar ilações de um trabalho incompleto, pelo que esta conclusão permite apenas refletir sobre as perspetivas existentes, tendo em conta o trabalho desenvolvido até agora. No entanto, é possível perceber que a *UnicornML* está no bom caminho, tendo em conta os resultados obtidos para os métodos já implementados e a ideia do que será possível ainda desenvolver.

Apêndice A

Observações

- Documentação *Python* 3:
  https://docs.python.org/3/
- 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