TÉCNICO LISBOA

Relatório de projecto

Tree augmented naive Bayes classifier

Programação Orientada a Objectos

Grupo 13

André Silva	86945
Maria Frutuoso	84303
Tiago Pires	84349

2019/2020

1 Introdução

Este projecto tem como objectivo implementar um classificador de Bayes, utilizando o algoritmo TAN (*Tree augmented naive Bayes classifier*). A implementação proposta encontra-se dividida em pacotes por funcionalidade: leitura dos ficheiros de entrada, construção de grafos (e árvores), classificador, métricas finais e tradução das classes em inteiros. No pacote correspondente aos classificador, a interface `Classifier` é uma generalização que permite utilizar outros tipos de classificadores além do de Bayes, ainda que este seja o único implementado no projecto. No pacote associado às estruturas abstractas (`structuresPackage`), foi criada a interface `Node`, no qual são guardados e calculados os dados necessários à classificação: pesos dos nós (α's) e parâmetros θ_{jikc}. Desta forma, o treino dos dados e previsões são feitas dentro deste pacote, sendo os dados de entrada lidos dentro do classificador, invocando a classe `FileClass`, passados para as classes `GraphClass` e `TreeClass` que recorrem à interface `Node`.

2 Extensibilidade da solução

Durante o planeamento do projecto, o grupo procurou sempre por criar uma implementação o mais genérica possível, tendo em vista a sua extensibilidade. Com efeito, a abordagem ao problema consistiu em identificar padrões de abstracção – por exemplo, uma *árvore* é um *grafo*, logo a *árvore* estende o *grafo* – e esquematizar este tipo de relações no UML.

De facto, a solução proposta é modular, estando dividida em diferentes pacotes e contendo diversas interfaces definidas. Assim, qualquer programador pode criar uma implementação do serviço oferecido pela interface.

Numa fase embrionária do projecto, o grupo definiu praticamente todos métodos declarados nas interfaces sem quaisquer argumentos de entrada nem de retorno (isto é, *void*), visando a generalização e posterior modificação por outro programador. A dependência de argumentos foi, assim, contornada recorrendo aos construtores das classes. Posteriormente, na integração dos resultados obtidos pelo classificador com as métricas generalizadas, alguns métodos sofreram alterações neste aspecto, retornando dados.

Assim, da forma como foi construído este projecto, torna-se bastante acessível e exequível substituir ou adicionar implementações que ofereceram serviços do tipo dos definidos na interface. Nomeadamente, criar um novo classificador, um novo tipo de grafo, um nó que implemente outro tipo de *score*, uma métrica diferente ou um tradutor que efectue outra conversão.

3 Análise crítica da performance

O custo computacional da avaliação das métricas, pelo método `evaluate()`, com excepção da classe `Accuracy`, é um pouco mais elevado do que o esperado, tendo em conta a análise estatística subjacente. Contudo, este custo foi comprometido tendo em vista uma implementação o mais genérica possível. Com efeito, este método efectua uma leitura completa do vector que contém as classes esperadas, provenientes do ficheiro de teste, guardando numa *ArrayList*. De seguida estas são ordenadas, recorrendo à classe `TComparator`, que opera sobre todo o tipo de dados, números inteiros ou decimais, e caracteres ou palavras. Isto permite avaliar posteriormente cada classe, calcular a sua pontuação e apresentar os resultados obtidos no terminal de forma ordenada. Assim, não há restrição dos avaliadores das métricas a qualquer tipo de classes, apesar do aumento da complexidade.

Em relação às contagens necessárias para calcular os parâmetros α e θ, estas foram feitas apenas quando necessárias e não foram guardadas em nenhum lado. Assim o tempo da construção do classificador sofreu uma pequena penalização, e seria um aspecto que poderia ser alterado.

As estruturas de dados usadas foram todas bastantes simples. Para guardar a maioria dos parâmetros necessários foram usados *arrays* ou duplos *arrays*, pelo que o custo computacional associado ao acesso aos mesmos é bastante pequeno.

Além disso, é de notar, por exemplo, que a classe `Bayes` tem como atributo um vector de inteiros tridimensional, `matrix`. Contudo, este aspecto deve-se ao facto de se poder passar esta estrutura por referência nos construtores de outras classes, nomeadamente a `FileClass`, que depois modificam o valor dos seus elementos, sendo estas alterações reflectidas na classe `Bayes`. Para passar por valor, é especificada a primeira das três dimensões, como acontece com a `GraphClass`, correspondendo a uma matriz de duas dimensões.

4 Funcionalidades extra

A aplicação contém algumas funcionalidades não directamente pedidas no enunciado do projecto, que se explicitam nos parágrafos seguintes.

A primeira resulta da tradução do vector das classes presente nos conjuntos de treino e de teste num vector de inteiros, implementada pela classe `IDTranslator`. Deste modo, qualquer que seja o valor da classe – um número inteiro, ou decimal, um caractere, uma palavra, entre outros –, fica garantida a generalização do algoritmo implementado pelo classificador. Além disso, o classificador pode endereçar estruturas de dados directamente a partir do valor das classes, o que é aproveitado, por exemplo, pela contagem das ocorrências da variável N_{ijkc}.

A segunda, relativa à classe `MetricAbstract`, consiste no tratamento das classes dos conjuntos de treino e de teste como tipos genéricos. Adicionalmente, foi criado um comparador de objectos genéricos (`TComparator`), permitindo a sua ordenação no método `sortClasses`. Assim, é conseguida a apresentação das classes, e respectivos resultados (*scores*), com ordenação numérica e/ou alfabética. Isto significa que, se as classes forem constituídas por números ou palavras, estas serão apresentadas correctamente ordenadas. Se um conjunto de teste contiver quer números, quer palavras, as classes numéricas serão apresentadas em primeiro lugar, seguidas das alfabéticas.

A terceira traduz-se na robustez no processamento da chamada do programa na linha de comandos, protegendo o mesmo de invocações incorrectas por parte do utilizador. Inclui-se a ausência de parâmetros de entrada, ou a passagem insuficiente ou excedente de argumentos, a indicação de um critério de pontuação (*score*) não previsto e a referência a ficheiros de entrada inexistentes. Foi criada a *flag* `-h` que explicita no terminal a forma como um utilizador deve efectuar a chamada do programa. Estas funcionalidades estão implementadas no método `validUI()` da classe `Main`.

A quarta é a presença da função de *debug* que pode ser usada através da invocação da aplicação com quatro argumentos em vez de três. O quarto argumento não tem que ser nada em específico, apenas tem de existir. Quando especificado permite ver os valores dos parâmetros *alpha* e *theta* para cada nó em conjunto com o pai do nó correspondente quando se encontra na árvore.

5 Conclusões

Após a implementação do classificador de Bayes, foram realizados testes com dois pares de ficheiros, cada um com um conjunto de treino e um conjunto de teste. Em ambas as execuções foram obtidos os resultados previstos em termos de classificação e de avaliação das métricas.

Este projecto foi desenvolvido seguindo a metodologia de programação orientada por objectos (OO). Começando pela sua estruturação mais abstracta, foram pensados em primeira instância os módulos essenciais e definidas as interfaces genéricas. Posteriormente, foi feita a concretização desses módulos em pacotes e das interfaces em classes.

Após a conclusão do projecto, é possível afirmar que a utilização de uma linguagem OO foi vantajosa no seu contexto, uma vez que este, para além da implementação do algoritmo do treino e da classificação, era composto por outras componentes, tais como as métricas e leitura de ficheiros, independentes e externas ao classificador. A separação destas componentes em diferentes pacotes permitiu isolar melhor o desenvolvimento de cada uma, incluindo a própria divisão de trabalho. Além disto, a divisão em classes beneficia, do ponto de vista do cliente, a utilização do código em outros projectos, mas também a sua modificação, dada a existência de interfaces que permite criar novas implementações e garantir a extensibilidade.