Trabalho 2 Aprendizagem Automática

Por: Carlos Palma 46520

Bruno Ascensão 45460

Introdução

Foi pedido pelo professor Luís Rato, responsável pela cadeira de Aprendizagem Automática do curso de Engenharia Informática da Universidade de Évora a construção de um modelo preditivo que responda à questão "Quais os alunos em risco de abandonar os estudos?".

O trabalho será desenvolvido com base num conjunto de dados a que todos os alunos têm acesso, cada grupo poderá utilizar os métodos que preferir para a elaboração do modelo, o trabalho será desenvolvido na linguagem Python com as bibliotecas sklearn e pandas. Além destas podem ser incluídas outras bibliotecas tais como (Matplotlib, Seaborn, etc). As métricas de desempenho a serem utilizadas deverão ser a precisão e a cobertura, do ponto de vista de classificação binária em que a classe positiva é a deteção de "insucesso académico" (indicado como 1 na coluna "Failure" do dataset). O objetivo principal é maximizar a deteção de casos de insucesso (ou seja, que todos alunos em risco de insucesso sejam classificados na classe positiva) minimizando os Falsos Negativos. Pretende-se portanto maximizar a cobertura, no entanto, para evitar que se lance a suspeita de insucesso demasiadas vezes sobre casos que não estão em risco, pretende-se garantir um nível mínimo de 70 % de precisão. Como objetivo secundário deve ser proposto um modelo alternativo simplificado que tenha no máximo 2 atributos, os quais poderão ser alguns dos atributos indicados no dataset, ou transformação dos atributos existentes no dataset. Por exemplo, poderia criar um atributo que fosse média-global calculada a partir das médias em cada semestre. Mas também poderia selecionar diretamente dois dos atributos. Podemos portanto criar esses dois atributos do modo que quisermos, desde que usemos apenas a informação disponível nos atributos do dataset.

Apresentação e análise do conjunto de dados

De maneira a iniciar a exploração do conjunto de dados que visa entender melhor a estrutura e possíveis padrões ou relações que possam existir começamos por carregar os mesmos num dataframe com o auxilio do módulo pandas do Python. Posteriormente, com o auxilio do módulo pandas/matplotlib iniciamos a nossa análise:

- df.head()/ df.info() - Começamos por imprimir no terminal as primeiras linhas do dataframe, assim conseguimos ter uma ideia geral da estrutura dos dados e do tipo de informação incluída. Com a informação facultada pelo professor no enunciado do trabalho conseguimos então recolher e listar os seguintes atributos: **Id** - é um identificador único do aluno representado como um inteiro; **Program** - tem valores inteiros de 0 a 3 e indica a licenciatura a que o aluno pertence; **YNsX_enrol** – número de ECTS inscritos, há N anos atrás, no semestre X; **YNsX_grade** - classificação média há N anos atrás, no semestre X; **YNsX_complete** - número de ECTS em UCs com aprovação, há N anos atrás, no semestre X; **Rest_enrol** - ECTS inscritos há mais de 4 anos atrás; **Rest_ complete –** ECTS aprovados há mais de 4 anos atrás; **Rest_grade –** Classificação média há mais de 4 anos atrás; **Failure**- classe binária, indicação da classe 0 para sucesso e 1 para Insucesso (classe positiva). Graças

ao df.info que mostra informações relativas ao dataframe conseguimos perceber que os atributos Id, Program e Failure são representados com números inteiros, no entanto todos os outros atributos são representados com floats. Todas as entradas da tabela são não nulas, pelo que ,não existirão problemas relativos a nulidades.

- df.describe() - De seguida imprimimos algumas estatísticas relativas ao dataframe tais como média, desvio padrão, valor mínimo, valor máximo e quartis. O nosso objetivo era detetar possíveis problemas, como valores ausentes ou outliers no entanto não conseguimos constatar qualquer tipo de problema.

1. df.corr() - Através deste método conseguimos obter uma matriz de correlação entre os atributos do dataframe. Esta mesma matriz deu-nos a entender quais os atributos que estão relacionados, valores próximos de: 1 indicam que se um atributo aumenta o outro tende a aumentar; -1 que se um aumenta o outro tende a diminuir; de 0 que não há relação entre os atributos. Foi muito aquilo que descobrimos, no entanto devido há extensão do dataframe vou apenas exemplificar algumas relações: Quanto maior o valor de Rest_enrol , maior o valor de Rest_complete (maior numero de ECTS inscritos há mais de 4 anos, maior o numero de ECTS aprovados) e quanto maior Rest_grade maior o valor de Rest_complete (maior classificação média há 4 anos atrás, maior numero de ECTS aprovados).

Modelo principal

Atributos escolhidos e bibliotecas

Na elaboração do modelo principal são utilizados todos os atributos para gerar o modelo. Requer algumas bibliotecas Python para funcionar corretamente. A primeira é a "pandas", que é utilizada para ler e manipular dados em formato de tabela, como o ficheiro "dropout-trabalho2.csv". A segunda biblioteca é a "sklearn", que fornece várias ferramentas para tarefas de aprendizagem automática, como seleção de modelo e otimização de hiperparâmetros. Neste caso, são utilizadas as classes "GridSearchCV" e "train_test_split", para otimização do modelo e divisão dos dados em conjuntos de treino e teste, respectivamente. Finalmente, o modelo de random forest é importado da classe "RandomForestClassifier" da biblioteca "sklearn".

Em resumo, as bibliotecas "pandas" e "sklearn" são necessárias para ler e preparar os dados, bem como para treinar e otimizar o modelo de árvore de decisão.

Experiências realizadas

1. Árvore de decisão

No presente trabalho, foi realizado um teste de modelo de aprendizagem baseado em árvore de decisão, visando maximizar a cobertura enquanto se mantinha uma precisão superior a 70%. Para otimizar o modelo, foi empregado o método GridSearch.

Os resultados obtidos foram bastante positivos, com uma precisão de **86,73%** e uma cobertura de **87,5%**. Os melhores parâmetros encontrados foram 'max_depth' igual a 6 e 'min_samples_split' igual a 12.

Observou-se que aumentar o 'min_samples_split' e diminuir o 'max_depth' resultavam em melhores resultados em termos de cobertura, no entanto é importante lembrar que a escolha do

modelo e do otimizador deve ser feita de forma cuidadosa e levando em consideração o contexto específico de cada problema. Além disso, é importante realizar a validação do modelo com conjuntos de dados distintos do conjunto de treino, de forma a obter uma avaliação mais precisa do desempenho do modelo.

2. Naive Bayes

Foi realizado um segundo teste de modelo de aprendizagem baseado em Gaussian Naive Bayes, utilizando o método GridSearch para otimizar o modelo.

Os resultados obtidos foram bastante satisfatórios, com uma precisão de **71,014%** e uma cobertura de **87,5%** com o parâmetro var_smoothing igual a 1e-13. Quando o parâmetro foi ajustado para 1e-12, observou-se uma melhora na cobertura, atingindo 89,29%, mas uma queda na precisão, que ficou em 69% que ficou aquém dos 70% pedidos no enunciado. Isso demonstra que, mesmo que se alcance um aumento na cobertura, é importante ter em consideração o impacto que isso pode ter na precisão do modelo.

3. Knn(vizinhos-mais-próximos)

Foi realizado um terceiro teste de modelo de aprendizagem baseado em Knn, utilizando o método GridSearch para otimizar o modelo.

Infelizmente, os resultados obtidos neste teste não foram os mais satisfatórios, com uma precisão de **88,77%** e uma cobertura de **77,67%** com os seguintes parâmetros {'metric': 'manhattan', 'n_neighbors': 3, 'weights': 'distance'}. Embora a precisão tenha sido bastante alta, a cobertura ficou aquém do esperado.

4. Random Forest

No quarto teste realizado, foi utilizado um modelo de aprendizagem baseado em random forest. O random forest é um modelo de aprendizagem que se baseia em múltiplas árvores de decisão e é amplamente utilizado em diversas aplicações. O facto de utilizar múltiplas árvores de decisão permite que o modelo capture diferentes padrões presentes nos dados, o que pode resultar em um desempenho melhor em comparação com um modelo baseado em uma única árvore de decisão.

Os resultados obtidos foram bastante satisfatórios, com uma precisão de **95,23%** e uma cobertura de **89,28%**. Os parâmetros utilizados foram {'max_depth': 12, 'min_samples_split': 3, 'n_estimators': 200}.

5. Mlp

Decidimos realizar o quinto teste com um algoritmo de redes neuronais, especificamente uma rede neural de múltiplas camadas (MLP). Uma das principais vantagens das redes neuronais é a sua capacidade de modelar relações não lineares entre os atributos e a classe alvo.

Foram inúmeros os resultados obtidos com este algoritmo. Existiram tentativas que retornaram 100% de precisão e tentativas que retornaram 100% de cobertura. A otimização de parâmetros neste algoritmo foi sem duvida a que demorou mais tempo, ficamos inclusive mais de 15 minutos à espera de algumas soluções. Das inúmeras tentativas aquela que retornou melhores

resultados para aquilo que é o nosso objetivo neste trabalho foi com os parâmetros {'alpha': 0.001, 'hidden_layer_sizes': (50, 200), 'max_iter': 1000} com **Precisão: 24%** e **Cobertura: 100 %.**

Conclusão

Tendo em conta os resultados dos mais diferentes testes chegamos a conclusão que a melhor solução para aquilo que é o nosso problema, seria implementar um modelo baseado em Random Forest com os parâmetros {'max_depth': 12, 'min_samples_split': 3, 'n_estimators': 200}. Neste teste obtemos uma cobertura de praticamente 90% mantendo uma grande precisão perto dos 96%. Além disso, o Random Forest possui uma estrutura intuitiva e fácil de implementar, o que nos permitiu facilmente otimizar os seus hiperparâmetros através do uso do GridSearch.

Em resumo, optamos pelo uso do Random Forest devido ao seu bom desempenho e facilidade de implementação, permitindo-nos atingir os objetivos propostos de maximizar a cobertura enquanto mantemos uma precisão mínima de 70%.

Modelo secundário

Atributos escolhidos e bibliotecas

A escolha de utilizar a média global de grades e enrol como atributos para o nosso modelo foi baseada na consideração de que esses dois fatores podem ter um impacto significativo na probabilidade de um aluno abandonar os estudos. As grades, ou notas, obtidas por um aluno ao longo do seu percurso académico podem ser um indicador da sua capacidade de compreensão e assimilação da matéria, bem como do seu esforço e dedicação. Por outro lado, o número de ECTS inscritos por um aluno pode ser um indicador do seu nível de envolvimento e compromisso com o seu percurso académico. Consideramos que a combinação desses dois fatores em médias globais pode fornecer uma medida mais completa da situação académica de um aluno e, portanto, ser um bom predictor da sua probabilidade de abandonar os estudos. Além disso, a utilização de apenas dois atributos simplifica o modelo, tornando-o mais fácil de interpretar e utilizar.

O trabalho proposto requer algumas bibliotecas Python para funcionar corretamente. A primeira é a "pandas", que é utilizada para ler e manipular dados em formato de tabela, como o ficheiro "dropout-trabalho2.csv". A segunda biblioteca é a "sklearn", que fornece várias ferramentas para tarefas de aprendizagem automática, como seleção de modelo e otimização de hiperparâmetros. Neste caso, são utilizadas as classes "GridSearchCV" e "train_test_split", para otimização do modelo e divisão dos dados em conjuntos de treino e teste, respetivamente. Finalmente, o modelo de random forest é importado da classe "RandomForestClassifier" da biblioteca "sklearn".

Em suma, as bibliotecas "pandas" e "sklearn" são necessárias para ler e preparar os dados, bem como para treinar e otimizar o modelo de árvore de decisão.

Experiências realizadas

1. **Árvore de decisão**

Para avaliar o desempenho do modelo na tarefa de identificar os alunos em perigo de abandonar os estudos, optamos por realizar uma avaliação inicial utilizando uma árvore de decisão como algoritmo de aprendizagem. Este tipo de algoritmo é amplamente utilizado para resolver problemas de classificação binária, como o nosso, e possui uma estrutura intuitiva e de fácil implementação. Depois de treinar e testar o modelo de árvore de decisão, utilizando GridSearch onde os melhores parâmetros encontrados foram {'max_depth': 4, 'min_samples_split': 8}, obtivemos os seguintes resultados: **Precisão: 95.60%** e **Cobertura: 77.67%**. A precisão mede a razão de instâncias classificadas corretamente pela classe positiva (alunos em risco de abandonar os estudos) sobre o total de instâncias classificadas pela classe positiva. Já a cobertura mede a razão de instâncias classificadas corretamente pela classe positiva sobre o total de instâncias da classe positiva. Em geral, ficamos satisfeitos com os resultados obtidos pelo modelo de árvore de decisão. A precisão e a cobertura ambas atingem níveis superiores a 70%, cumprindo assim os requisitos pedidos pelo professor de maximizar a cobertura enquanto garantimos uma precisão mínima. Além disso, os resultados são bastante encorajadores, o que nos dá confiança de que o modelo pode ser útil para identificar os alunos em perigo de abandonar os estudos com um nível aceitável de precisão e cobertura.

2. Naive Bayes

No nosso segundo teste, decidimos utilizar o algoritmo Gaussian Naive Bayes para prever quais alunos estariam em risco de abandonar os estudos. Carregámos os dados em um dataframe e separámos os atributos (X) e a coluna alvo (y). Em seguida, dividimos os dados em conjuntos de treino e teste, com uma proporção de 80/20 e, com o auxílio do otimizador de parâmetros GridSearch, procedemos à otimização dos parâmetros.

Posteriormente, treinámos o modelo com os dados de treino e avaliámos o seu desempenho nos dados de teste. Os resultados foram positivos tendo em conta a **Precisão: 72.22 %** e **Cobertura: 81.25 %** com o melhor parâmetro sendo {'var_smoothing': 1e-09}.

3. Knn(vizinhos-mais-próximos)

No terceiro teste, decidimos utilizar o algoritmo KNN (K Nearest Neighbors) para prever quais alunos estariam em risco de abandonar os estudos. Carregámos os dados em um dataframe e separámos os atributos (X) e a coluna alvo (y). Em seguida, dividimos os dados em conjuntos de treino e teste, com uma proporção de 80/20 e com o auxilio do otimizador de parâmetros GridSearch procedemos a otimização dos parâmetros.

Posteriormente treinámos o modelo com os dados de treino e avaliámos o seu desempenho nos dados de teste. Os resultados não foram os mais animadores sendo que obtemos **Precisão: 94.56 %** e **Cobertura: 77.67 %** com os parâmetros {'metric': 'manhattan', 'n_neighbors': 9, 'weights': 'uniform'}.

Embora a precisão seja bastante boa, a cobertura é bastante baixa em comparação com os resultados dos testes anteriores.

4. Random Forest

Decidimos realizar o quarto teste com o algoritmo Random Forest. Os resultados obtidos foram de **Precisão: 95.78 %** e **Cobertura: 81.25 %** utilizando o GridSearch como otimizador de

parâmetros sendo o melhor parâmetro {'max_depth': 8, 'min_samples_split': 16, 'n_estimators': 800}.

Ficamos extremamente satisfeitos com os resultados obtidos, uma vez que conseguimos atingir uma precisão acima da mínima exigida, mantendo uma cobertura elevada. Quer isto dizer que o nosso modelo é capaz de detetar com precisão os casos de insucesso académico sem, no entanto, dar muitos falsos positivos.

5. Mlp

Decidimos realizar o quinto teste com um algoritmo de redes neuronais, especificamente uma rede neural de múltiplas camadas (MLP). Uma das principais vantagens das redes neuronais é a sua capacidade de modelar relações não lineares entre os atributos e a classe alvo.

Foram inúmeros os resultados obtidos com este algoritmo, existiram tentativas que retornaram 100% de precisão e tentativas que retornaram 100% de cobertura, a otimização de parâmetros neste algoritmo demorou algum tempo.

Após várias tentativas, aquela que retornou melhores resultados para o nosso objetivo foi com os parâmetros {'alpha': 0.001, 'hidden_layer_sizes': (50, 200), 'max_iter': 2000} com **Precisão: 98.80 %** e **Cobertura: 74.10 %.**

Conclusões

Após a avaliação dos resultados obtidos, pelos diferentes algoritmos utilizados, decidimos optar pelo uso do Random Forest para o nosso modelo secundário. Este algoritmo foi capaz de alcançar uma precisão de 95,78% e uma cobertura de 81,25%, superando assim os requisitos mínimos de precisão e maximizando a cobertura. Além disso, o Random Forest possui uma estrutura intuitiva e fácil de implementar, o que nos permitiu facilmente otimizar os seus hiperparâmetros através do uso do GridSearch.

Em resumo, optamos pelo uso do Random Forest devido ao seu bom desempenho e facilidade de implementação, permitindo-nos atingir os objetivos propostos de maximizar a cobertura enquanto mantemos uma precisão mínima de 70%.