Segundo Trabalho de Inteligência Artificial CC2006

Pedro Miguel Ribeiro Carvalho, up201805068

João Paulo Macedo Sampaio, up201805112

Resumo

Num mundo cada vez mais abundante em termos de dados, Machine Learning é sem dúvida uma das áreas de investigação mais importantes nos dias que correm, principalmente através de Árvores de Decisão ou Naive Bayes. Um dos exemplos é de um estudo realizado pela Columbia University sobre speed dating em que podemos inferir através de um conjunto de atributos se há match entre duas pessoas ou não. Neste trabalho podemos perceber como estratégias diferentes de tratamentos dos dados pode influenciar no outcome das previsões destes modelos, bem como a forma como devemos dividir os dados entre conjuntos de treinos e conjuntos de testes. Assim sendo, tal como neste contexto Machine Learning é capaz de prever com alguma robustez o resultado previsto, dado que quanto maiores forem os conjuntos de dados maiores as chances de tais resultados estare corretos.

Introdução

Este segundo trabalho foi proposto na cadeira de Inteligência Artificial CC2006 com o objetivo de criar e avaliar modelos de classificação para um conjunto de dados que nós são fornecidos, recorrendo a dois algoritmos de *Machine Learning:* Árvores de Decisão (*CART* ou *ID3*) e *Naive Bayes*. O conjunto de dados que nós foram facultados é referente a um estudo realizado pela Columbia University entre 2002 e 2004, dados esses que são sobre participantes em eventos de speed dating.

De seguida iremos explicar de forma sucinta a forma como foi resolvido o problema, porém mais à frente toda esta parte será explicada em detalhe. Numa primeira fase lidamos com a fase de pré-processamento, desde lidar com valores nulos a uma referência à forma como os dados não estão balanceados e como isso pode prejudicar o desempenho dos algoritmos. Passando pela dita execução dos algoritmos, porém usando duas estratégias diferentes para obter estimativas confiáveis: *Método Holdout* e *Método Validação Cruzada* (*Cross-Validation*). E finalmente obter as métricas quer para cada algoritmo usado quer para cada estrategia diferente usada a *Accuracy, Precision, Recall* e *F-measure.*

A implementação dos algoritmos bem como o pré-processamento dos dados e a avaliação dos desemepnhos encontram-se no diretório: /src/AI.py. Usamos a linguagem Python e as seguintes packages: *pandas, sklearn, pydotplus, matplotlib e Ipython*. A package *pandas* foi utilizada para ler e manipular os dados, a *sklearn* para implementar os algoritmos pretendidos neste trabalho, enquanto que as restantes packages foram aplicadas para a criação de gráficos para avaliação de desempenho.

Algoritmos

Uma vez que neste trabalho tínhamos a opção de usar ou o algoritmo *ID3* ou o *CART*, optamos pelo *CART* dado que de uma forma geral é o mais usado. Assim sendo, é importante explicar não só o algoritmo *CART* mas também algumas noções essenciais, tais como, árvores de decisão.

Uma árvore de decisão representa uma função que recebe como argumento um vetor de valores de atributos e devolve um valor de decisão como resultado. Neste sentido, recebe a árvore de decisão recebe os atributos já mencionados e devolve um valor entre 0 e 1. Cada nó interno da árvore corresponde a um teste ao valor de determinado atributo e os ramos que partem desse nó estão etiquetados com cada possível valor do atributo. Cada nó folha corresponde ao valor retornado pela árvore de decisão.

O algoritmo *CART* pode ser explicado pelo seguinte pseudocódigo:

```
function DECISION-TREE-LEARNING(examples, attributes, parent_examples) returns
a tree
if examples is empty then return PLURALITY_VALUE(parent_examples)
else if examples have the same classification then return the classification
else if attributes is empty then return PLURALITY_VALUE(parent_examples)
else
        A <- argmax a ∈ attributes IMPORTANCE(a, examples)
        tree <- a new decision tree with root test A
        for each value v_k of A do
                exs <- {e : e ∈ examples and e.A = v_k}
                subtree <- DECISION-TREE-LEARNING(exs, attributes - A, examples)
                add a branch to tree with label (A = v_k) and subtree subtree
        return tree
```

A função PLURALITY-VALUE seleciona o valor mais comum no conjunto dos exemplos, caso haja empate a escolha é aleatória.

A função IMPORTANCE escolhe o próximo atributo a seguir.

Esta função deve selecionar o atributo que mais exemplos classifica exatamente, de forma a minimizar a altura da árvore.

O algoritmo *CART* implementa na função IMPORTANCE a Impureza de Gini que de um conjunto de dados, onde cada exemplo pertence a uma classe é dado por:

$$Gini(D) = 1 - \sum_{i=1}^{c} p_i^2 \times p_i,$$

D – conjunto de dados

c – classe do conjunto de dados

p_i - probabilidade da classe i estimada pela frequência nos dados.

Em cada nó interno, o atributo escolhido é aquele que maximiza a redução da Impureza de Gini do conjunto de dados.

Agora iremos passar a uma breve explicação do algoritmo *Naive Bayes*

O algoritmo *Naive Bayes* é baseado no teorema de Bayes que consiste na probabilidade de acontecer um evento com informação relacionada. Este algoritmo usa então as probabilidade conhecidas à priori para filtrar e classificar a informação dada assumindo que as probabilidades são independentes entre si. Usando a assunção do Naive Bayes, ou seja, as probabilidades serem independentes entre si faz com que a quantidade de probabilidades a estimar seja de um número muito menor. De referir que neste trabalho usaremos a versão Gaussian do Naive Bayes que basicamente um dado atributo numérico, a probabilidade de observar um valor da uma classe, assumindo uma distrubuição normal é dada por:

$$P(x_k|y_j) = \frac{1}{\sigma_{kj}\sqrt{2\pi}} e^{-\frac{(x_k - \mu_{kj})^2}{2\sigma_{kj}^2}}$$

Análise exploratória dos dados

Neste trabalho inicialmente iremos dividir os atributos do conjunto em dados na variável X e a variável objetivo em y.

O conjunto de dados contém os seguintes atributos:

- **id**: número de identificação do participante;
- **partner**: número de identificação do par;
- **age**: idade do participante;
- **age_o**: idade do par;

- **goal**: qual é o seu objetivo principal ao participar neste evento? (Passar uma noite divertida = 1, Conhecer novas pessoas = 2, Conseguir um encontro = 3, Procurar um relacionamento sério = 4, Dizer que consegui = 5, Outro = 6);
- **date**: em geral, quão frequentemente sai para encontros? (Várias vezes por semana = 1, Duas vezes por semana =2, Uma vez por semana = 3, Duas vezes por mês = 4, Uma vez por mês = 5, Várias vezes por ano = 6, Quase nunca = 7);
- **go_out**: com que frequência sai (não necessariamente para encontros)? (Várias vezes por semana = 1, Duas vezes por semana =2, Uma vez por semana = 3, Duas vezes por mês = 4, Uma vez por mês = 5, Várias vezes por ano = 6, Quase nunca = 7);
- **int_corr**: correlação entre os ratings de interesses (desporto, museus, caminhadas, música, filmes, livros, etc.) do participante e do seu par ([-1,1]);
- **length**: a duração de 4 minutos para o encontro é: (1 = Demasiado curta , 2 = Demasiado longa, 3 = Adequada);
- **met**: já conhecia o seu par anteriormente? (0/1);
- **like**: quão gostou do seu par? (escala 1-10; nada = 1; muito =10);
- prob: probabilidade do seu par ter gostado de si? (escala 1-10; pouco provável = 1; muito provável =10).

Finalmente, temos uma variável objetivo:

- **match:** há match? (0/1).

Primeiramente deparámo-nos com o problema de existência de valores nulos no conjunto de dados e usamos como referência — “A Comprehensive guide on handling Missing Values” [1], como tal para resolver o problema tinha duas diferentes abordagens ou remover linhas e/ou colunas ou substituir esses valores nulos por alguma métrica (média, moda ou mediana). Seria útil usar a moda se tivéssemos variáveis categóricas, como não temos não faz sentido usar. Uma vez que, os dados que temos seguem uma distribuição normal é recomendado o uso da média em detrimento da mediana.

Desta forma, faria sentido remover colunas caso existissem valores nulos em mais de 70% das entradas para cada coluna. Neste conjunto de dados temos 8378 linhas, correndo o programa é possível verificar que obtemos o valor falso para todas as colunas do conjunto de dados, uma vez que verifica a condição se 70% ou mais das entradas das colunas são valores nulos. Assim sendo, não faz sentido remover colunas. Em relação à remoção de linhas, faria sentido remover qualquer linha que tivesse pelo menos um valor nulo. Verificamos que isso acontece em 1498 linhas, como tal ficaramos com apenas 6878 linhas, o que entendemos ser um valor muito baixo tendo em conta o tipo de problema que estamos a lidar. Resta assim a solução de substituir os valores nulos pela media para cada coluna. Portanto, do nosso ver pareceu a melhor solução de entre as possíveis e foi a que utilizamos.

Por outro lado, é importante mencionar que fazer previsões e testar no mesmo conjunto de dados é um erro metodológico, uma vez que isso levaria a métricas de desempenho perfeitas, pois seriam feitas previsões em dados já observados. Assim sendo, é interessante introduzir dois métodos para resolver esse problema. O primeiro *Método Holdout,* que divide aleatoriamente o conjunto de dados em treino/teste (tipicamente 70%/30%) e o *Método Validação Cruzada (Cross-Validation)*, que divide o conjunto de dados em k partições (folds) e usa alternadamente, cada partição como conjunto teste e as restantes k-1 partições como conjunto de treino. No final é feita a média da avaliação obtida em cada uma das k iterações.

Finalmente, é interessante verificar que os dados não estão balanceados, uma vez que temos 1380 com valor de 1 na variável objetivo e 6998 com o valor de 0. Desde já podemos concluir que as métricas de avaliação de desempenho dos algoritmos poderão obter resultados piores por causa desta condicionante.

Experiências e resultados

Nesta secção iremos realizar uma experiência e avaliar os resultados encontrados. Primeiramente, submetemos os dados do conjunto ao pré-processamento que já explicamos anteriormente. De seguida, damos inicio à execução dos algoritmos. Iremos para cada algoritmo usar as duas estrategias já referidas e comparar. No Método Cross-Validation iremos usar 5 como o número de splits.

Neste trabalho no Método Holdout usaremos sempre como parâmetro o *test_size* = 0,3 para obtermos um conjunto de teste com 30% do tamanho do conjunto inicial, como dito anteriormente. Também usaremos o parâmetro o *random_state* = 42 para assim garantir que para efeitos de comparação temos conjuntos divididos de forma igual. Por outro lado, no Método Cross-Validation usaremos o *Kfold cross-validator* para dividir o conjunto em k *folds*, usaremos também sempre o parâmetro o *random_state* = 42 e por último o parâmetro *shuffle = True* para dividir de forma aleatória.

Para o primeiro algortimo (*CART)* utilizamos as duas diferentes estratégias já vistas, *Holdout* e *Cross-Validation*. No *Método Holdout* dividimos o conjunto de dados em 4 conjuntos (X_train, X_test, y_train, y_test) usando a função *train_test_split* com parâmetros X, y e 0,3 para o tamanho do conjunto de teste e o 42 para o random_state. Posteriomente usamos a função fit para treinar o conjunto de dados como parâmetros os conjuntos de treino já obtidos, X_train e y_train. De seguida, usamos a função predict para prever os resultados do conjunto X_test. Por outro lado, no *Método Cross Validation* usamos a função kfold com os parâmetros já referidos anteriormente e para o número de split usaremos 5. assim obtemos os conjuntos divididos. Posteriormente, para todos os splits que temos iremos treinar os conjuntos e prever com o restante conjunto de teste.

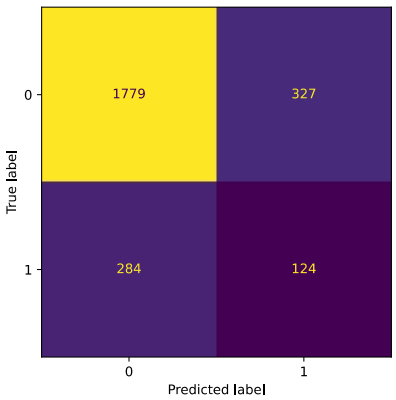
No segundo algoritmo (*Naive Bayes*) utilizamos da mesma forma as duas estratégias já vistas. Usaremos a função GaussianNB para estimar as probabilidaes e de forma igual ao primeiro algoritmo obtemos os conjuntos de testes e treinos adequados para as duas estratégias e treinamos e prevemos com o conjunto de teste e assim obtemos os resultados pretendidos.

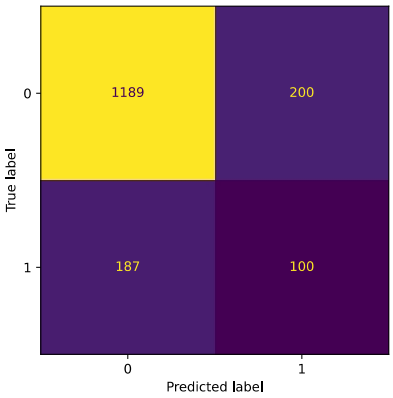
Após isto, daremos inicio à avaliação de desempenho com os resultados obtidos para ambos os algoritmos e para ambas as estrategias obtemos as matrizes de confusão, sendo que no *Método Holdout* obtemos apenas uma matriz e no *Método Cross-Validation* obtemos tantas matrizes quanto o número de splits. Em que as entradas da matriz se referem aos seguintes termos:

- (0,0) – verdadeiro negativo;
- (0,1) – falso positivo;
- (1,0) - falso negativo;
- (1,1) – verdadeiro positivo.

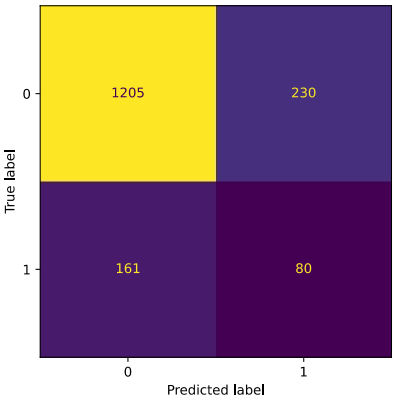
Obtemos assim as seguintes matrizes de confusão:

CART - Holdout

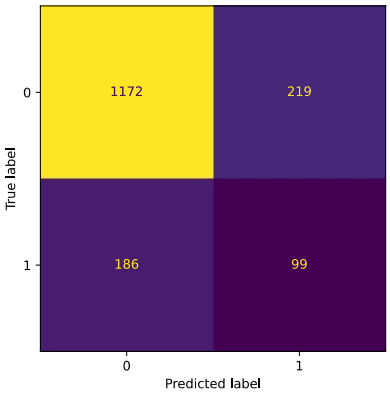




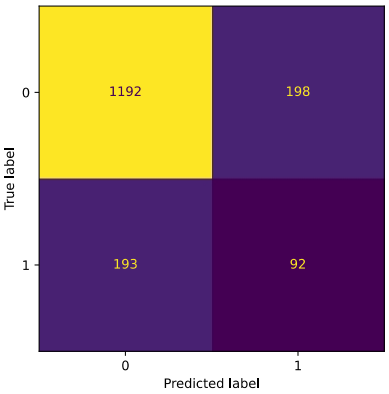
CART – CV 1



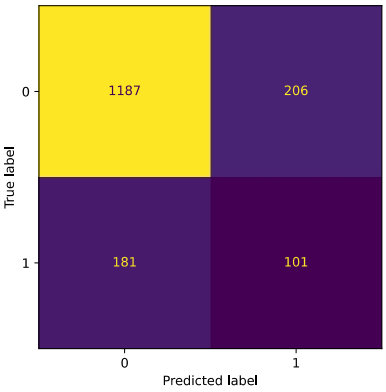
CART – CV 2



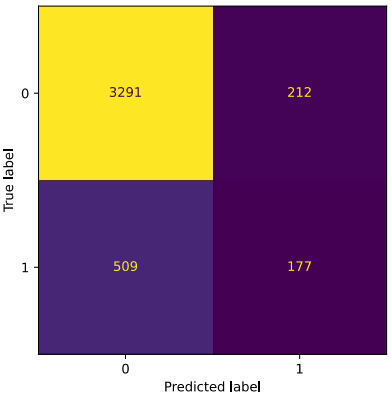
CART – CV 3



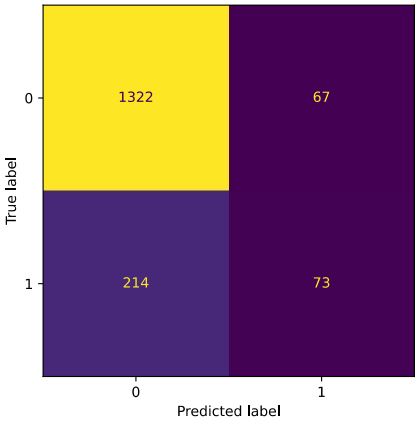
CART – CV 4



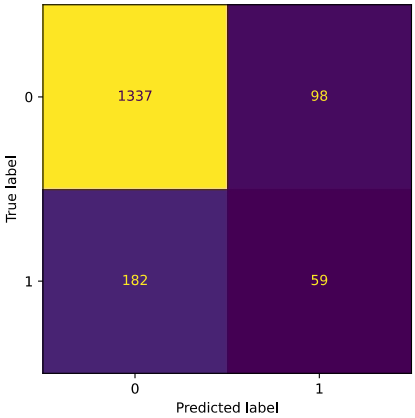
CART – CV 5



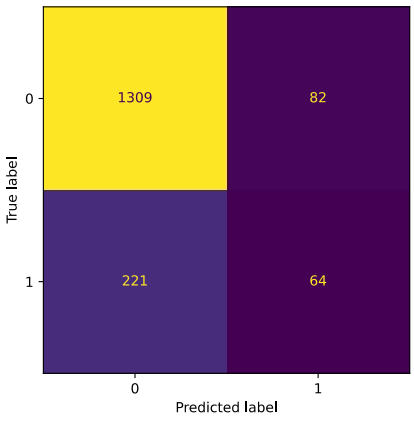
GNB - Holdout



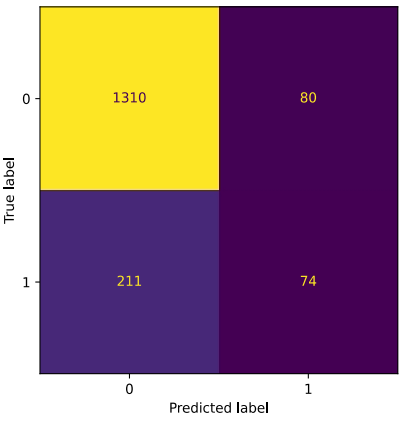
GNB – CV 1



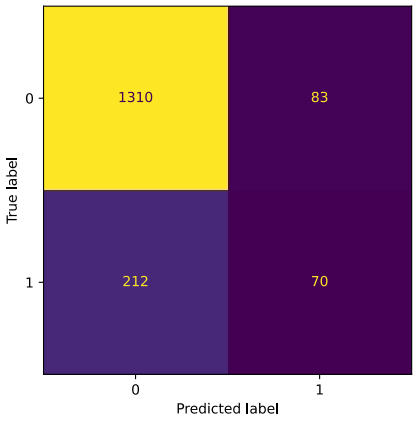
GNB – CV 2



GNB – CV 3



GNB – CV 4



GNB – CV 5

Importante mencionar que para o cálculo das métricas no *Holdout* basta usar os valores dessa matriz, enquanto que no *Cross-Validation* temos de fazer a soma de cada entrada de todas as matrizes e assim obter os resultados de desempenho. Finalmente, usando as métricas accuracy, precision, recall, f-measure obtemos os seguintes resultados representados na tabela que se segue:

Value	CART - Holdout	CART - Cross Validation	Naive Bayes - Holdout	Naive Bayes - Cross Validation
Accuracy	0.757	0.766	0.828	0.827
Precision	0.275	0.310	0.455	0.453
Recall	0.304	0.342	0.258	0.246
F1	0.289	0.325	0.329	0.319

Com base na tabela anterior podemos concluir, através da métrica Accuracy de uma forma geral nos 4 casos é possível prever com alguma exatidão se existe match ou não com base nos atributos obtidos. Porém, podemos realçar que o Naive Bayes obteve melhor classificação nesse sentido. Por outro lado, através da métrica Precision que o algoritmo Naive Bayes é melhor para garantir que não prevê casos em que não há match quando na realidade existe match, importante ressalvar que entre usar Holdout ou Cross-Validation não tem qualquer peso nesta métrica. De uma outra forma, usando a métrica Recall podemos verificar que o algoritmo *CART* é o melhor a encontrar casos em

que existe match e usarmos Cross-Validation nesse algoritmo será ainda melhor. Por último através da métrica F1 uma vez que para todos os casos toma valores baixos podemos concluir quer que a Precision e a Recall não obtiveram valores muito bons.

Podemos inferir que uma limitação do trabalho é o facto de a quantidade de dados não ser muito extensa, na realidade a quantidade não é muito má, mas como podemos verificar quantos mais exemplos tivermos maior é a probabilidade de serem feitas previsões corretas.

Conclusões

Nesta ultima parte iremos fazer conclusões finais sobre este trabalho, assim sendo de uma forma geral podemos concluir que o tamanho de conjunto de dados apesar de não ser muito pequeno claramente se fosse maior obteríamos métricas melhores. Por outro lado, podemos constatar que o conjunto de dados não se encontrava balanceado, uma vez que existem mais exemplos com a variável objetivo a 0 do que a 1, o que torna a tarefa de previsão ainda mais complicada. Também de mencionar com as métricas usadas podemos obter melhores resultados utilizando o algoritmo *Naive Bayes* e que no algoritmo *CART* a melhor estratégia é usar *Cross-Validation* enquanto que no *Naive Bayes* é melhor usar *Holdout*, porém a diferença é mínima. Por último em relação a tempos de execução podemos constatar que para os parâmetros usados e com este conjunto de dados a execução é sem dúvidas bastante rápida.

Referências

[1] A Comprehensive guide on handling Missing Values - https://medium.com/bycodegarage/a-comprehensive-guide-on-handling-missing-values-b1257a4866d1