Laboratório 3 - Manipulação de Instâncias

Nuno Gomes
DEETC - MEIC
Instituto Superior Engenharia de Lisboa
Lisboa, Portugal
A18364@alunos.isel.ipl.pt

Ricardo Ramos
DEETC - MEIC
Instituto Superior Engenharia de Lisboa
Lisboa, Portugal
A46638@alunos.isel.ipl.pt

Rafael Carvalho
DEETC - MEIC
Instituto Superior Engenharia de Lisboa
Lisboa, Portugal
A47663@alunos.isel.ipl.pt

***Resumo*—Este laboratório prático tem por objetivo compreender o conceito de seleção e manipulação de instâncias bem como compreender os diferentes métodos de amostragem e de que forma influenciam a performance de um modelo de aprendizagem baseado no algoritmo *Random Forest*. Em simultâneo, pretende-se desenvolver alguma aptidão com a ferramenta de processamento de dados em larga escala Apache Spark [1]**

Index Terms*—Megadados, Amostragem, Aprendizagem Automática, *Random Forest

I. Introdução

Num contexto de informação, maiores *datasets* não representam necessariamente melhores dados. Grandes *datasets* são complexos, e por vezes esparsos, o que produzem em grande maioria má *performance* em algoritmos de mineração de dados. Desta forma surge a necessidade de fazer a manipulação dos *datasets*, tipicamente reduzindo o número de instâncias, ou aplicar técnicas de amostragem que permitam melhorar a *performance* dos algoritmos de aprendizagem, e obter modelos mais robustos.

O laboratório 3 tem por objetivo estudar as técnicas de amostragem lecionadas em aula, nomeadamente o *oversampling*, *undersampling* e técnicas não lineares de geração de instâncias, como o *Synthetic Minority Oversampling Technique - SMOTE*. Pretende-se comparar a *performance* do algoritmo *Random Forest* para diferentes conjuntos de treino obtidos das diferentes técnicas mencionadas. Em simultâneo, pretende-se desenvolver competências com a ferramenta de processamento de dados em larga Apache Spark no treino de algoritmos de *data mining*.

II. Visualização do conjunto de dados

O conjunto de dados fornecido denominado *Influenza* apresenta 545 *features*, 2 *class labels* e 2190 instâncias, onde o problema proposto é realizar a classificação com base no conjunto de dados não balanceado. Isto é, uma *class label* apresenta um número de instâncias significativamente maior que outra.

A. Schema e conteúdo

Pela observação do *schema*, compreendemos que à exceção da *class label*, todas as *features* são do tipo *Integer Type*. A Figura 1 demonstra os nomes e tipos das primeiras cinco *features* do conjunto de dados:

	name	type
CLASS	CLASS	IntegerType
V1	V1	IntegerType
V2	V2	IntegerType
V3	V3	IntegerType
V4	V4	IntegerType

Figura 1: Schema df original

Selecionando os primeiros 10 elementos da *dataframe* notamos que as instâncias estão dispostas de forma crescente do valor das várias *features*.

	CLASS	V1	V2	V3	V4
	<int>	<int>	<int>	<int>	<int>
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0
6	0	0	0	0	0
7	0	0	0	0	0
8	0	0	0	0	0
9	0	0	0	0	0
10	0	0	0	0	0

Figura 2: Primeiros 10 elementos do dataframe df

Para confirmar que o número de instâncias e *features* na *dataframe* está correto, utilizamos o método *stopifnot* onde determinamos se o conjunto de linhas e colunas da *dataframe* local corresponde às da *dataframe Spark* criada, para tal, utilizaram-se, respetivamente os métodos *nrow*/*ncol* e *sdf_nrow*/*sdf_ncol*. O método *stopifnot* não retorna um erro, pelo que conclui-se que as dimensões são idênticas.

III. Feature Selection

De relembrar a importância da redução de dimensionalidade em *datasets* e do problema da *Curse of Dimensionality*. Esta dita que para um algoritmo de *machine learning* ter a capacidade de generalizar e corresponder a um resultado correto para novos dados é necessário reduzir ao máximo a redundância e remover *features* desnecessárias ao problema. Um maior conjunto de características implica um maior número de instâncias, que como consequência torna o algoritmo de aprendizagem mais lento e mais pesado computacionalmente. Em simultâneo, quanto maiores as dimensões dos conjunto, mais difícil se torna para o algoritmo compreender padrões nos dados, que resulta numa *performance* menor.

Do *dataset* fornecido, são destacadas apenas um conjunto de *features*, onde se pretende fazer a redução de dimensionalidade utilizando o operador de *pipe* de *magrittr* denotado pelo conjunto de símbolos "%>%". Dadas as *features* a manter, o resultado obtido demonstra-se abaixo

	CLASS	V1	V4	V5	V8
	<int>	<int>	<int>	<int>	<int>
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0
6	0	0	0	0	0

Figura 3: Primeiros elementos do dataframe reduzido

IV. Técnicas genéricas para amostragem

A *Feature Selection* pretende atenuar o problema causado pela elevada dimensionalidade de *features* em grandes conjuntos de dados. No entanto, existem outros problemas que podem levar a que o algoritmo seja demasiado pesado computacionalmente, e que leve demasiado tempo a treinar. Uma vez que os recursos computacionais são finitos, deve ser tomada a decisão de fazer a amostragem do conjunto de dados previamente pré-processado. Esta técnica visa fazer a seleção de instâncias e criar um novo conjunto de dados com menos exemplos, porém, que representem da melhor forma possível a população total.

Um método de amostragem eficiente e que tende a mostrar bons resultados é a amostragem aleatória do conjunto de dados. Desta forma, dividiu-se o *dataset* em duas partições utilizando o método *sdf_random_split*, com 2/3 reservado a um conjunto de dados de treino e 1/3 para teste. Como desvantagem, este método de amostragem pode resultar num conjunto de dados de treino não balanceado, isto é, uma *class label* é majorante no conjunto de dados, o que pode levar a um maior *bias*.

Criados os conjuntos de treino e teste, para podermos determinar o número de instâncias de cada classe, é necessário converter a *dataframe Spark* em uma *dataframe R* utilizando o método *collect()*, e só depois utilizar o método nativo do R, *table()*.

	CLASS	#Instances
1	0	32963
2	1	1899

Figura 4: Número de instâncias do conjunto de treino para cada classe

V. Técnicas de amostragem não equilibrada

Como visto anteriormente, o *conjunto* de dados de treino tinha um número consideravelmente desbalanceado de amostras de uma classe face a outra. Atendendo aos problemas que possam surgir dado este problema, existem técnicas de amostragem não equilibrada/balanceada, como é o exemplo do *undersampling*, onde se faz a sub-amostragem da classe majorante, procurando ao máximo garantir o mesmo número de instâncias que a classe minoritária; *oversampling*, em que são feitas cópias de instâncias da classe minoritária para equilibrar o conjunto de dados; ou mesmo técnicas de geração de amostras da classe minoritária como é o caso do *Synthetic Minority Oversampling Technique* ou SMOTE. Neste último exemplo, as instâncias são geradas segundo uma amostra do espaço de *features* e os vizinhos mais próximos da classe que se pretende amostrar (*K-Nearest Neighbors*).

Utilizou-se então o conjunto de treino anterior para realizar o *undersampling*, relembrando que pretende-se sub-amostrar a classe majorante, pelo que se calculou a fração de amostragem dada pela razão entre o número de instâncias da classe positiva (*CLASS* = 1) e o número de instâncias da classe negativa (*CLASS* = 0).

$$undersampling = \frac{\#Minorante}{\#Majorante}$$

O problema inerente neste tipo de amostragem trata-se da sub-representação da classe majorante, que pode introduzir erros de generalização dada a perda de informação relevante.

Para o *oversampling* pretende-se amostrar a classe minorante, onde a fração de amostragem é o inverso da descrita anteriormente.

$$oversampling = \frac{\#Majorante}{\#Minorante}$$

Ao copiar amostras da classe minorante introduz um maior *bias* em relação aos valores das *features* das instâncias copiadas e consequentemente tornar o modelo *overfit*, pelo que se torna ineficaz a prever novos resultados. Outra desvantagem é o aumento no número de instâncias, que torna o conjunto de treino maior e mais pesado computacionalmente. Ao replicar exemplos, também pode acontecer um cenário onde são criadas relações artificiais entre instâncias, e produzir um modelo menos robusto.

Observando os novos conjuntos de treino após *undersample* e *oversample*, respetivamente:

	CLASS	#Instances
1	0	1914
2	1	1899

Figura 5: Conjunto de amostras depois de *undersample*

	CLASS	#Instances
1	0	32963
2	1	32774

Figura 6: Conjunto de amostras depois de *oversample*

A técnica SMOTE gera novas instâncias sintéticas da classe minoritária com base nos *K-Nearest Neighbors*, é particularmente vantajosa pois permite evitir o *bias* introduzido pelo *oversampling*, uma vez que as novas instâncias não são cópias exatas de outras já existentes no conjunto de dados. Como desvantagem tem a sensibilidade ao ruído, uma vez que pode gerar novas instâncias com base num vizinho que é considerado ruído no conjunto de dados, e consequentemente aumentá-lo, o que reduz a *performance* do modelo.

A função BLSMOTE disponivel no R Studio permite configurar os seguintes parâmetros do algoritmo:

- K: Número de vizinhos durante o processo de *sampling*.
- C: Número de vizinhos durante o processo de cálculo do nível de segurança.
- Method: Método utilizado para *oversampling*

VI. Treino e comparação de modelos

Após preparação dos conjuntos de dados com as diversas técnicas de *random sampling*, *oversampling*, *undersampling* e SMOTE, pretende-se comparar o desempenho de um modelo baseado em *Random Forest* para classificação, onde a

fórmula que se pretende treinar, e prever é a *class label* CLASS. As métricas mais importantes a ter em conta na avaliação da *performance* dos modelos são:

- **Taxa de falsos positivos**: Mede a percentagem de falsos positivos face a todas as classificações positivas (soma de falsos positivos com verdadeiros positivos). Esta taxa é baseada em quantos verdadeiros negativos foram previstos incorretamente.
- **Accuracy**: Percentagem de previsões que o modelo acertou
- **Kappa**: Métrica que compara a *accuracy* observada com a esperada ao medir a proximidade entre as instâncias classificadas pelo modelo e os dados rotulados manualmente. Um valor mais elevado representa um modelo com melhor *accuracy* [2]. Esta métrica é diretamente comparável com outros modelos para a mesma tarefa de classificação.
- **Pos Pred Value**: Percentagem de resultados classificados como positivos, que são realmente positivos
- **Neg Pred Value**: Percentagem de resultados classificados como negativos, que são realmente negativos

Elaborou-se então a tabela I, que se encontra em anexo, onde o modelo *baseline* representa a amostragem aleatória, que resultou num conjunto de dados não equilibrado:

Olhando aos resultados do modelo *baseline*, podemos comprovar a influência de um *dataset* não balanceado na *performance* do modelo. Embora tenha uma *accuracy* elevada e não errar na previsão de valores negativo, isto deve-se ao *dataset* de teste ser igualmente não balanceado, pelo que tem muitas mais amostras negativas do que positivas. Em simultâneo, a taxa de identificação correta de casos positivos é extremamente baixa, pelo que é problemático caso o foco do modelo seja classificar corretamente novos casos positivos.

Para os *modelos* treinados com *undersampling* e *oversampling*, os resultados obtidos são semelhantes, no entanto, o segundo apresenta melhor *performance* na deteção de casos positivos, o que pode ser justificado pela replicação de casos positivos durante o *oversampling* do conjunto de treino.

O último modelo treinado utilizou o *dataset oversampled* segundo o método SMOTE e para além de ter a maior *accuracy* de todos e melhor *kappa*, peca na deteção dos casos positivos comparativamente ao modelo treinado com *oversampling*. No entanto, o parâmetro *kappa* dita que a *accuracy* obtida, comparativamente à esperada foi superior, o que mostra que o modelo tem uma *performance* melhor.

Conclui-se que o modelo utilizado deve ser tipicamente amostrado de forma balanceada e preferencialmente com a geração de novas instâncias de forma sintética, para evitar o *bias* do modelo face às novas instâncias, no entanto, deve ser tido em conta o número de medições erradas (*outliers*) antes de aplicar esta amostragem. A escolha do modelo deve também depender do caso de estudo, por exemplo, se o objetivo for identificar casos positivos de forma correta com baixa taxa de erro, então a escolha do modelo deve incidir sobre o que foi treinado com *oversampling* ou com SMOTE.

Para a escolha dos parâmetros K e C do *Borderline-SMOTE* foi efetuado um teste para verificar os valores dos parâmetros que obtinha melhores resultados. Os resultados encontram-se apresentados nas tabelas II e III que se encontram em anexo.

De acordo com os testes realizados, os melhores parâmetros são K=7 e C=4.

VII. Conclusão

A elaboração do terceiro laboratório prático tinha por objetivo contactar com técnicas de manipulação de instâncias e técnicas de amostragem de *datasets*, bem como avaliar de que forma estes *datasets* de treino influenciam a *performance* do modelo final. Após conclusão do trabalho, o grupo compreendeu o erro associado a modelos cujo *dataset* de treino é desbalanceado, e as vantagens e desvantagens associadas às técnicas de *random oversampling* e *undersampling* e do algoritmo SMOTE. Adquiriram-se também conhecimentos acerca das métricas de avaliação, nomeadamente a métrica *Kappa*, que complementa a medida da *accuracy* de forma mais robusta e permite comparar os diversos modelos treinados.

Referências

[1] https://apache.spark.org
[2] https://stats.stackexchange.com/a/82187

VIII. ANEXOS

	False Positive Rate	Accuracy	Kappa	Pos Pred Value	Neg Pred Value
Baseline	0.995	0.943	0.009	0.005	1
Undersampled	0.367	0.730	0.128	0.633	0.736
Oversampled	**0.355**	0.719	0.123	**0.645**	0.723
Borderline-SMOTE K=7 C=4	0.503	**0.820**	**0.167**	0.497	**0.840**

Tabela I: Desempenho do modelo de *Random Forest* para variações do mesmo *dataset*

	False Positive Rate	Accuracy	Kappa	Pos Pred Value	Neg Pred Value
K = 3	0.452	0.794	0.157	0.548	0.809
K = 4	0.475	0.811	0.167	0.525	0.829
K = 5	0.484	0.813	0.166	0.516	0.831
K = 6	**0.444**	0.790	0.155	**0.556**	0.804
K = 7	0.480	**0.818**	**0.173**	0.520	**0.836**

Tabela II: Teste dos valores de K com C=5 e method=c("type1","type2")

	False Positive Rate	Accuracy	Kappa	Pos Pred Value	Neg Pred Value
C = 3	0.503	0.812	0.157	0.497	0.831
C = 4	0.497	**0.828**	**0.179**	0.503	**0.847**
C = 5	0.478	0.812	0.167	0.522	0.829
C = 6	0.505	0.816	0.162	0.495	0.836
C = 7	**0.476**	0.807	0.162	**0.524**	0.824

Tabela III: Teste dos valores de C com K=5 e method=c("type1","type2")