Classificadores para atribuição de nível de dificuldade para problemas de competições de programação

Mauro Mascarenhas de Araújo[1]

[1]Centro de Matemática, Computação e Cognição
Universidade Federal do ABC (UFABC) – Santo André – SP – Brasil

mauro.mascarenhas@aluno.ufabc.edu.br

Abstract. *The purpose of this work is to analyse the performance of classifiers which should infer the difficulty level of a programing contest challenges based on their respective statements. In order to do that, the performance of two different classifiers are compared: KNN (using either euclidean or cossine distance) and random forest.*
The text vectorization task is also performed in two different manners: performing a TF-IDF vectorization and by using a state-of-art SBERT sentence transformer [Reimers and Gurevych 2019].
Furthermore, it was observed that, in spite of the classifiers not being capable of performing well with the given dataset, some data preparation steps could be successfully applied to similar contexts.

Resumo. *A proposta deste trabalho é verificar a possibilidade de construir classificadores capazes de determinar o nível de dificuldade de problemas de competições de programação com base no enunciado do desafio. Para tal, são comparados dois tipos de classificadores: KNN (utilizando a distância euclidiana e do cosseno) e floresta aleatória.*
A vetorização dos textos também é realizada de duas formas: utilizando a clássica vetorização TF-IDF e utilizando um modelo transformador de sentença, estado da arte, SBERT [Reimers and Gurevych 2019].
Por fim, foi observado que, embora o desempenho dos classificadores gerados para a base de dados analisada não tenham sido satisfatórios, alguns passos de preparação dos dados propostas podem ser reutilizados de forma exitosa em casos similares.

1. Introdução

Considerando que, dentre as diversas linhas de pesquisa de recuperação de informação, há as que focam em descobrir atributos que compõem uma resposta aceitável para determinados tipos de questões [Meurers et al. 2011], ou ainda a detecção de plágio em respostas de atividades avaliativas [Chong et al. 2010] [Paul and Jamal 2015] [Gupta et al. 2014], as prováveis aplicações no contexto educacional de ciência da computação destacam-se as que envolvem análise de banco de questões de atividades, provas, ou ainda, de desafios típicos de maratonas de programação, que são eventos destinados a alunos de graduação ou que estejam iniciando a pós-graduação na área de tecnologia [de Oliveira 2019].

Tendo em vista que as informações fornecidas como entrada para processos de análise de questionários tendem a ser puramente textuais, a aplicação de técnicas de

vetorização de texto torna-se indispensável. Portanto, para atingir o objetivo deste trabalho, que é analisar o desempenho de classificadores de nível de dificuldade de problemas com base no conjunto de dados obtidos da *Beecrowd*, antigamente denominada *URI Online Judge*[1], será necessário vetorizar os textos a fim de obter uma matriz TF-IDF ou uma matriz resultante da aplicação de modelos de aprendizagem de máquina SBERT, atual estado da arte[Reimers and Gurevych 2019], além de aplicar técnicas para redução de *outliers* da variável objetivo.

2. Problemas de competições de programação

O conjunto de dados analisados consiste em problemas típicos de competições de programação, que podem ser encontrados em diversas plataformas *online*, como a *HackerRank* e a *Beecrowd*, onde normalmente há a descrição do problema junto às especificações de entrada e de saída e, em alguns casos, exemplos de execução.

Dado que já existem conjuntos de dados da plataforma *Beecrowd* publicamente disponíveis[de Araújo 2019], foi utilizada a base de dados "*EN_03_terciary_data*", que contém os desafios em língua inglesa e algumas informações já foram tratadas, como a remoção de desafios sem as descrições, informações de entrada e saída ou ainda sem suas respectivas estatísticas[de Araújo 2019].

3. Preprocessamento, modelagem analítica e de classificação

3.1. Correlação entre nível de dificuldade e taxa de acertos

A primeira etapa de análise e preparação da base consistiu na tentativa de melhorar a correlação entre a taxa de acertos (dada pela quantidade de submissões corretas divididas pela quantidade total de submissões) e o nível de dificuldade, que varia de 1 a 10.

Uma vez com as informações carregadas em uma estrutura de dados [2], foi verificada a correlação existente as variáveis mencionadas (taxa de acerto e nível de dificuldade), que foi, aproximadamente, 0,80.

Embora já houvesse uma grande correlação existente entre as variáveis analisadas, foram observados alguns *outliers*, principalmente em problemas de nível 1 e 10. Portanto, a fim de amenizar os efeitos destes problemas, foi realizada uma regressão linear simples com os desafios que apresentavam taxas de acerto dentro da amplitude interquartil, considerando a taxa de acertos como a variável independente e o nível de dificuldade como variável dependente, para que fosse aplicada aos problemas tidos como *outliers*.

Como a regressão retorna valores reais que podem estar fora do intervalo de níveis de dificuldade padrão, $[1 - 10]$, foi necessário arredondar os resultados da regressão para "zero" casas decimais, onde a regressão foi aplicada, além de "reclassificar" problemas de acordo com a Equação 1, onde p é a instância do problema analisado.

$$nivel(p) = max(min(10, nivel(p)), 0) \tag{1}$$

[1]Plataforma que contém desafios típicos de competições de programação (descritos na seção 2).

[2]Para este trabalho, foi utilizada uma estrutura chamada *DataFrame* [McKinney 2019], disponível na Pandas, uma biblioteca que provê estruturas e ferramentas para análise de dados [McKinney et al. 2010].

3.2. Vetorização

Para fins de comparação, foram utilizadas duas formas de vetorização de texto: utilizando matriz uma matriz TF-IDF, que considera a frequência dos termos em seus respectivos documentos e aplica um fator de penalização suavizada a cada um deles com base na frequência de aparição em múltiplos documentos, e utilizando um transformador de sentença, um modelo de aprendizagem de máquina pré treinado capaz de mapear sentenças para vetores densos.

Para realizar a primeira versão da transformação, foi utilizada uma instancia da classe "*TfidfVectorizer*", disponível na biblioteca *sklearn*[Buitinck et al. 2013]. Para tal instância, foram fornecidos os textos de descrição, entrada e saída dos problemas concatenados por um caractere de espaço. Outros parâmetros relevantes personalizados de inicialização foram:

- **tokenizer:** função *word_tokenize*, disponibilizada na biblioteca *NLTK* [NLTK Project 2019];
- ***stopwords*:** "english".

Após obter a matriz, foi necessário aplicar um método de redução de dimensionalidade, pois, a matriz gerada era demasiadamente esparsa, o que poderia afetar o desempenho de classificadores treinados com base nela. Para tal tarefa, foi utilizado utilizada a Decomposição em Valores Singulares (SVD), dado que ela tende a trabalhar com matrizes esparsas de forma mais eficiente, uma vez que, diferentemente do PCA, não há necessidade de centralizar e normalizar os dados antes de computar a decomposição dos valores singulares [Pedregosa et al. 2020, Manning et al. 2008]. Após a aplicação da técnica, a matriz obtida passou a contar com 1152 atributos, a mesma quantidade que a obtida na técnica apresentada a seguir.

A segunda versão utiliza um transformador de sentenças baseado em modelo de aprendizagem de máquina, especificamente, "Embeddings" de Sentenças utilizando Redes Siamesas BERT (SBERT). Este transformador foi projetado para tarefas de identificação de similaridade semântica entre sentenças [Reimers and Gurevych 2019], de forma que cada transformador é treinado com um conjunto de dados diferente e apresenta diferentes objetivos de uso.

O transformador utilizado neste trabalho foi o "*multi-qa-MiniLM-L6-cos-v1*", um modelo treinado com mais de 214 milhões de sentenças, sendo elas pares perguntas e respostas de diferentes fontes. Ele foi escolhido, pois, consegue lidar com sentenças de até 512 *tokens* e as transforma em vetores densos de 384 dimensões [Reimers and Gurevych 2019].

Como mencionado anteriormente, o transformador selecionado lida com textos de até 512 palavras (*tokens*) e, portanto, não seria possível concatenar os conteúdos da descrição, de entrada e de saída dos problemas, como foi feito durante a geração da matriz TF-IDF. A fim de contornar a situação, foram gerados três "*embeddings*": um para a descrição, um para as informações de entrada e outro para as informações de saída que foram, posteriormente concatenados, resultando em vetores de 1152 atributos para cada problema.

Outro tratamento necessário foi a remoção de palavras de até 3 caracteres (potenciais *stopwords*, ou seja, palavras que não agregam informações ao contexto) em sentenças

com mais de 512 *tokens*, a fim de reduzir o a quantidade de "palavras", sem perder a "essência" da sentença.

3.3. Criação dos modelos de classificação

Dado que as estruturas de dados obtidas possibilitam a classificação com base em similaridade semântica, foram criados e analisados três tipos de classificadores: KNN utilizando distância do cosseno (busca por similaridade semântica), KNN utilizando distância euclidiana e florestas aleatórias, além do classificador aleatório estratificado[3], que foi utilizado para fins de comparação. Para este trabalho, todas as implementações utilizadas estão disponíveis na biblioteca *sklearn*[Pedregosa et al. 2011, Buitinck et al. 2013].

Conforme apresentado na Figura 1, as classes não possuíam desbalanceamento tão impactante, onde cada uma delas apresentou, em média, 183 problemas, com desvio padrão de 7. Portanto, não foi necessária a aplicação de técnicas adicionais para balanceamento de classes.

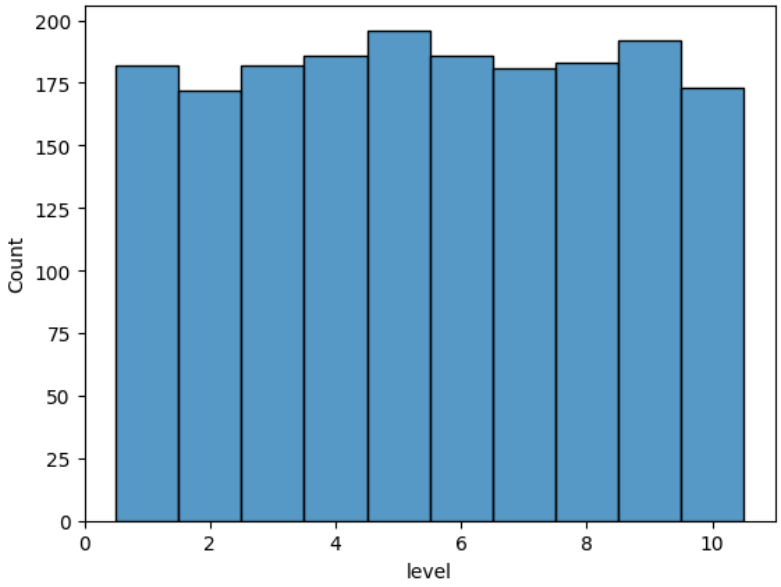


Figura 1. Histograma da contagem de problemas disponíveis na base de dados por nível de dificuldade.

Entretanto, dado a complexidade do conjunto de dados: muitos atributos e pouca representatividade de problemas por classe, foi necessário variar os hiperparâmetros dos modelos a fim de decidir os valores mais adequados para cada um deles.

Para a tarefa de seleção, foram utilizadas instâncias da classe "GridSearchCV"[4], outra utilidade implementada na biblioteca *sklearn*[Pedregosa et al. 2011, Buitinck et al. 2013],que é capaz de treinar o mesmo classificador com hiperparâmetros diferentes e buscar o melhor conjunto de argumentos para cada um deles com base em alguma métrica de avaliação de desempenho, que neste trabalho ficou definido que seriam

[3]"*DummyClassifier*" com parâmetro "*strategy*" definido como "stratified".

[4]Todos os modelos foram treinados em uma máquina com processador Intel i7-10750H com disponibilidade de 32GB de memória RAM e placa de vídeo RTX 2060.

avaliadas a acurácia, precisão, f1 e ROC AUC[5], com base em validações cruzadas de 10 dobras.

Para os modelos de floresta aleatória, foram testados os seguintes hiperparâmetros:

- **n_estimators (quantidade de árvores):** 20, 40, 60, 80, 100 e 120;
- **max_depth (profundidade máxima das árvores):** 5, 7, 9, 11, 13 e 15;
- **min_samples_leaf (quantidade mínima de amostras por folha):** 5, 10, 15 e 20;
- **min_impurity_decrease (quantidade mínima de ganho de informação para que haja ramificação nas árvores):** 0,001, 0,01 e 0,1.

Já para os modelos baseados em vizinhança (KNN), foram testados os seguintes hiperparâmetros:

- **n_neighbors (quantidade de vizinhos a serem considerados):** 11, 15, 21 e 31;
- **metric (métrica de distância para definir os vizinhos mais próximos):** "*euclidean*" (euclidiana) e "*cosine*" (cosseno);
- **weights (pesos):** "*uniform*" (votação uniforme) e "*distance*" (baseado em proximidade).

Por fim, foi aplicado uma nova "reclassificação" nos níveis de dificuldades dos problemas existentes na base de dados, que passaram a ser definidos pela Equação 2, onde p é a instância do problema analisado.

$$novo\_nivel(p) = ceil(nivel(p)/2) \tag{2}$$

Depois, todos os classificadores anteriores foram reavaliados com base nas novas classes atribuídas ao conjunto de problemas.

4. Resultados e análises

4.1. Tratamento de *outliers*

De forma geral, é possível afirmar que o tratamento de *outliers* foi um sucesso, dado que a quantidade deles foi consideravelmente reduzida e, dos poucos que sobraram, a maioria é justificável. Mas, o principal ponto, é que, mesmo passando por uma "reclassificação", as características estatísticas das taxas de acerto em relação ao nível de dificuldade (classes) não sofreram tanto impacto, conforme é possível observar na Figura 2.

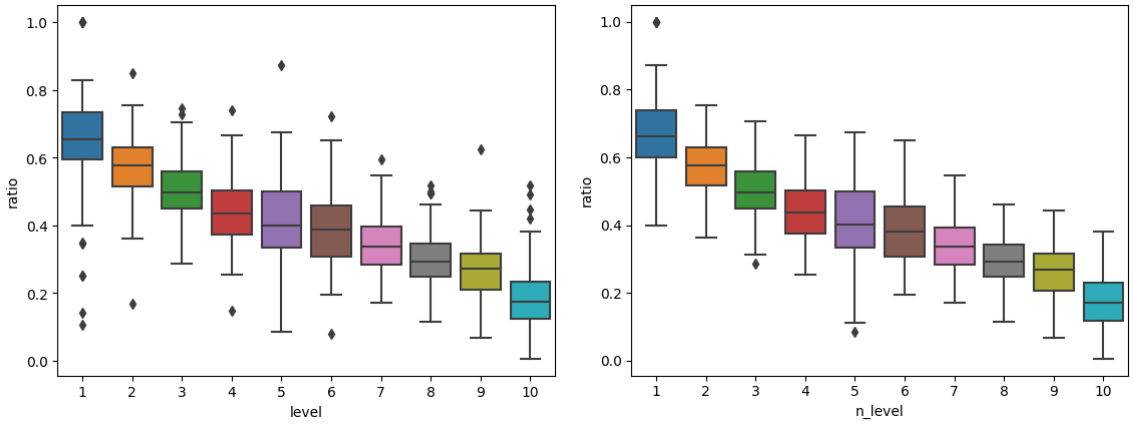
É possível notar, na Figura 2b que os *outliers* passaram a ficar mais próximos da amplitude interquartil, além de os problemas que apresentarem taxas de resoluções muito altas (próximas a 1), foram corretamente classificados como pertencentes ao nível 1.

Também é válido mencionar que, após a aplicação deste tratamento, a correlação entre a taxa de acertos e o nível de dificuldade aumentou de $\approx 0,80$ para $\approx 0,83$.

4.2. Modelos de classificação

Os resultados de pontuações da validação cruzada para cada uma das melhores versões dos modelos obtidos podem ser observados na Tabela 1.

[5]Todas as métricas consideradas foram utilizadas com suas respectivas variantes "balanceadas" ou "ponderadas", dado que trata-se de um problema de classificação multiclasse.



(a) Estatísticas iniciais. **(b) Estatísticas após tratamento.**

Figura 2. *Boxplots* das taxas de acertos para cada nível de dificuldade dos problemas antes e após a aplicação da regressão linear nos *outliers*.

É possível notar que o desempenho médio das melhores versões dos classificadores obtidos não ficou muito melhor que o classificador aleatório, ou seja, nenhum deles apresentou resultado que justifique seu uso para a classificação de nível de dificuldade de problemas com base no conteúdo textual dos enunciados.

Dentre as possíveis justificativas para o baixo desempenho dos modelos há:

- A quantidade de desafios disponíveis por classe: Conforme é mencionado na subseção 3.3, há apenas cerca de 183 problemas por classe. O que pode ser um problema para um modelo cuja matriz de atributos apresenta 1152 atributos (quase $\frac{2}{3}$ da quantidade de amostras);
- Falta de passos adicionais para tratamento das sentenças na construção da matriz TF-IDF: Algumas técnicas adicionais de PLN poderiam ter sido aplicadas (aplicação de *lematização* ou *stemização*, além de fornecer um conjunto de *stopwords* mais robusto), a fim de tornar a matriz gerada mais "robusta" a variação de termos similares e redundantes;
- Degradação de desempenho do modelo de transformação de sentenças SBERT: Conforme mencionado pelos autores do modelo, o desempenho dele começa a se degradar com textos muito grandes (motivo pelo qual as sentenças com mais de 512 palavras são truncadas), além de ter sido treinado com uma base de dados cujas sentenças não ultrapassaram 250 palavras, ao passo que a base trabalhada apresentou alguns problemas por excederem, inclusive, o limite máximo de 512 *tokens*.

Entretanto, também é possível observar que, embora as matrizes tenham como característica a possibilidade de trabalhar com similaridade semântica, a métrica utilizada pelo classificador KNN não impactou tanto no desempenho, dado que era esperado que a classificação baseada na distância do cosseno apresentasse resultados superiores à euclidiana.

Por fim, é válido citar que, embora as florestas aleatórias não tenham apresentado, de forma geral, resultados similares a modelos mais simples, como o KNN, seu desempenho melhorou consideravelmente ao reduzir a quantidade de classes para 5, com

Tabela 1. Avaliação de desempenho, com base nos resultados de validação cruzada, dos classificadores treinados.

Vetorização	Classes	Classificador	Acurácia	Precisão	F1	ROC AUC
TF-IDF	5	Aleatório	0,202619	0,201416	0,20102	0,499119
		KNN (cos)	0,243537	0,243057	0,238335	0,548972
		KNN (eucl)	0,245354	0,244505	0,237691	0,548973
		Fta. Aleatória	0,234482	0,239005	0,227365	0,536918
	10	Aleatório	0,101798	0,101294	0,100195	0,499469
		KNN (cos)	0,129085	0,121147	0,116302	0,534168
		KNN (eucl)	0,126589	0,120766	0,108535	0,545495
		Fta. Aleatória	0,120382	0,120888	0,117069	0,519224
Embedding	5	Aleatório	0,211696	0,211645	0,210216	0,502717
		KNN (cos)	0,252068	0,245244	0,239884	0,566641
		KNN (eucl)	0,252305	0,247358	0,241399	0,566321
		Fta. Aleatória	0,283362	0,275716	0,267257	0,604858
	10	Aleatório	0,094452	0,091462	0,091992	0,500522
		KNN (cos)	0,136186	0,134977	0,130048	0,559025
		KNN (eucl)	0,135630	0,134381	0,129561	0,560151
		Fta. Aleatória	0,141415	0,146079	0,127602	0,589584

a vetorização baseada em "*embeddings*", dado que a acurácia, precisão, f1 e ROC AUC apresentaram resultados melhores em 0,07, 0,06, 0,06 e 0,1 respectivamente, quando comparadas ao classificador aleatório.

5. Conclusão

Conforme as análises apresentadas na subseção 4.2, é possível afirmar que para o conjunto de dados analisado, os enunciados, provavelmente, não possuem informações suficientes que permitam classificar o nível de dificuldade de um problema com base apenas no conteúdo textual, dado que nenhum dos classificadores modelados foram capazes de superar significativamente o desempenho do classificador aleatório.

Todavia, é possível reaproveitar o método de tratamento de *outliers*, apresentado na subseção 3.1 e discutido na subseção 4.1, para casos similares, dado que ele apresentou bons resultados sem impactar profundamente as características estatísticas dos dados.

Por fim, para o caso analisado, também foi possível observar que a vetorização de sentenças baseada em "embeddings" de modelos SBERT não apresentou ganho de desempenho significativo quando comparada à clássica vetorização TF-IDF.

Referências

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122.

Chong, M., Specia, L., and Mitkov, R. (2010). Using natural language processing for automatic detection of plagiarism. In *Proceedings of the 4th International Plagiarism Conference (IPC-2010)*.

de Araújo, M. M. (2019). Heurísticas computacionais para extração de conhecimento em problemas de maratona de programação. Website online.

de Oliveira, G. F. (2019). XXIV Maratona de Programação. Website online.

Gupta, D., Vani, K., and Singh, C. K. (2014). Using natural language processing techniques and fuzzy-semantic similarity for automatic external plagiarism detection. In *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 2694–2699. IEEE.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *The term vocabulary and postings lists*. Cambridge University Press.

McKinney, W. (2019). pandas.dataframe - pandas 0.25.3 documentation.

McKinney, W. et al. (2010). Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX.

Meurers, D., Ziai, R., Ott, N., and Bailey, S. M. (2011). Integrating parallel analysis modules to evaluate the meaning of answers to reading comprehension questions. *International Journal of Continuing Engineering Education and Life-Long Learning*, 21(4):355–369.

NLTK Project (2019). nltk.tokenize package.

Paul, M. and Jamal, S. (2015). An improved srl based plagiarism detection technique using sentence ranking. *Procedia Computer Science*, 46:223–230.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V.and Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P.and Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2020). Decomposing signals in components (matrix factorization problems).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.