

Detectando questões duplicadas: Quora Questions Pairs

Airine Carmo e Christian Cardoso

Mestrado em Engenharia de Sistemas e Computação – Universidade Federal do Rio de Janeiro

(UFRJ) – Campus Cidade Universitária

21941-592 – Rio de Janeiro – RJ – Brasil

carmoa@cos.ufrj.com e christiancardozo@cos.ufrj.br

1. Introdução

Um problema bastante comum em mineração de dados é a identificação de itens duplicados em uma base. Neste trabalho, utilizaremos a base de dados de perguntas “Quora Questions Pairs” [1] para classificar perguntas duplicadas. O conjunto de dados usado está presente no site do Kaggle [2] como uma competição online do Quora [3], um site de perguntas e respostas. No Kaggle, foram fornecidos dois conjuntos de dados: treinamento e teste. O conjunto de treinamento contém uma série de pares de perguntas etiquetadas como similares ou não similares. O conjunto de teste contém uma série de pares de perguntas que devem ser classificadas e depois enviadas ao site da competição. Para os devidos fins deste trabalho, apenas a base de treinamento foi utilizada. Para a implementação deste trabalho foi utilizada a linguagem Python [4], versão 3, além da biblioteca scikit-learn [5] e do framework NLTK [6]. Todo o código desenvolvido para este relatório, assim como o próprio, se encontram no GitHub [7]. A seção 2 é dedicada à uma análise dos dados. A seção 3 descreve os procedimentos de pré-processamento e extração de features. A seção 4 enumera os classificadores utilizados e seus resultados. Por fim, na seção 5, as conclusões e considerações finais.

2. Análise dos dados

Como mencionado na seção 1, apenas a base de treinamento fornecida pelo Kaggle foi utilizada neste trabalho. Nesta seção, faremos uma breve análise dos dados. O arquivo de dados está disponível em formato CSV, e contém um cabeçalho na primeira linha seguido de 404.290 registros. O CSV possui seis colunas de dados:

1. identificador do registro
2. identificador da primeira pergunta
3. identificador da segunda pergunta
4. texto da primeira pergunta
5. texto da segunda pergunta
6. classe ('0' para não duplicada e '1' para duplicada)

Uma análise na quantidade de registros de cada classe nos permite concluir que 255.027 registros são da classe '0' e os outros 149.263 são da classe '1'. No gráfico da figura 2.1 temos a proporção de cada classe. Em relação às questões, a base possui 537.933 perguntas com ids diferentes, onde 111.780 delas aparecem mais de uma vez. O histograma da figura 2.2 contém a contagem das vezes que perguntas se repetiram em registros diferentes. À princípio, temos 863.550 palavras distintas. Esta quantidade

consequentemente será reduzida durante o pré-processamento devido à remoção de stopwords e ao processo de stemming.



Figura 2.1 - Proporção de cada classe no conjunto de dados

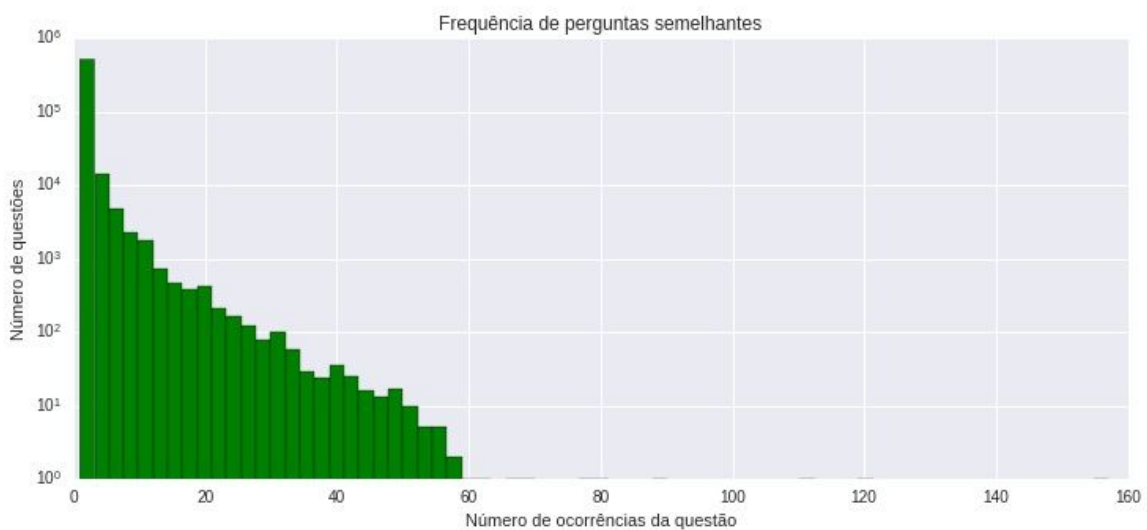


Figura 2.2 - Frequência de perguntas que aparecem em registros diferentes

3. Pré-processamento dos dados

Antes de começarmos os experimentos de classificação, é necessário pré-processar o conjunto de dados. Como sabemos, os dados brutos são dados por pares de Strings. Esses dados precisam ser tratados para se tornarem atributos a serem utilizados no momento do treinamento dos classificadores. O pré-processamento é feito em seis passos, detalhados nas subseções a seguir.

3.1. Transformação e Tokenização

O primeiro passo do pré-processamento consiste em transformar todos os caracteres das perguntas para caixa baixa. Em seguida, é realizado um processo de tokenização por palavras para os pares de perguntas. A tokenização leva em consideração sequências de caracteres alfanuméricos como um token. Caracteres de pontuação são removidos neste momento. Ao final deste processo, cada registro irá possuir um par de perguntas devidamente tokenizadas com caracteres em caixa baixa.

3.2. Remoção de Stop Words

Algumas palavras são extremamente frequentes no uso da língua. Porém, a maioria destas palavras frequentes não contribuem com nenhum significado semântico às sentenças. Estas palavras são classificadas como “Stop Words”. Como sua importância semântica é mínima, a remoção delas garante que as mesmas não acabem atrapalhando o processo de mineração de dados. Com isso, é preciso definir uma lista de Stop Words. Há diversas versões desta lista dependendo do idioma e dos objetivos. As perguntas no Quora são majoritariamente em inglês, e o NLTK possui uma lista pré-definida de Stop Words para a língua inglesa. Esta lista foi utilizada para remover as Stop Words dos pares de perguntas do conjunto de dados.

3.3 Stemming

O processo de stemming consiste em identificar a raiz de uma palavra e remover as suas derivações. O objetivo é reduzir palavras flexionadas à sua raiz. Com isso, palavras como “connection” e “connected” são cortadas e transformadas para “connect”. Novamente, o NLTK foi usado para realizar o stemming, utilizando o Porter Stemmer [8]. Após o processo de stemming, precedido da remoção de Stop Words, o tamanho do vocabulário presente nas perguntas passou a ser de 65.483.

3.4. Construção da matriz de presença e n-gramas

A construção da matriz de presença permite transformar toda a informação de texto em números. E, para captar o máximo possível de informação contida nos textos, utilizamos n-gramas, com n de 1 até 3. Sendo assim, foram geradas todas as combinações presentes no texto de 1-gramas, 2-gramas e 3-gramas. Além disso, para o problema em questão, foi preciso um esquema mais elaborado para tentar representar bem as informações contidas nos registros. Construir apenas uma matriz de presença levando em consideração somente um texto concatenado das duas questões perderia informações de quais tokens pertencem à qual pergunta e como isso afeta na decisão de similaridade entre as perguntas. Pensando em manter o máximo de informação neste tratamento de dados, foram construídas três matrizes de presença. A primeira matriz se refere à presença de n-gramas na primeira questão do par. A segunda, se refere à presença de n-gramas na segunda questão do par. E, a terceira, é feita utilizando a presença de n-gramas que aparecem nas duas perguntas. Após a construção dessas matrizes, elas são concatenadas horizontalmente, formando assim uma grande matriz com um total de 3.315.693 colunas, além de muito esparsa.

3.5. Redução de dimensionalidade (SVD)

A matriz construída no passo 3.4 é computacionalmente impossível de ser usada no processo de aprendizagem e classificação. Para prosseguirmos, foi necessário realizar uma redução de dimensionalidade. O algoritmo utilizado para tal tarefa foi o SVD. Executando o SVD para a matriz gigante, podemos extrair as variáveis latentes mais importantes para o conjunto de dados. O gráfico da figura 3.1 mostra os valores de importância dos primeiros 50 autovalores da matriz diagonal Sigma gerada pelo SVD.

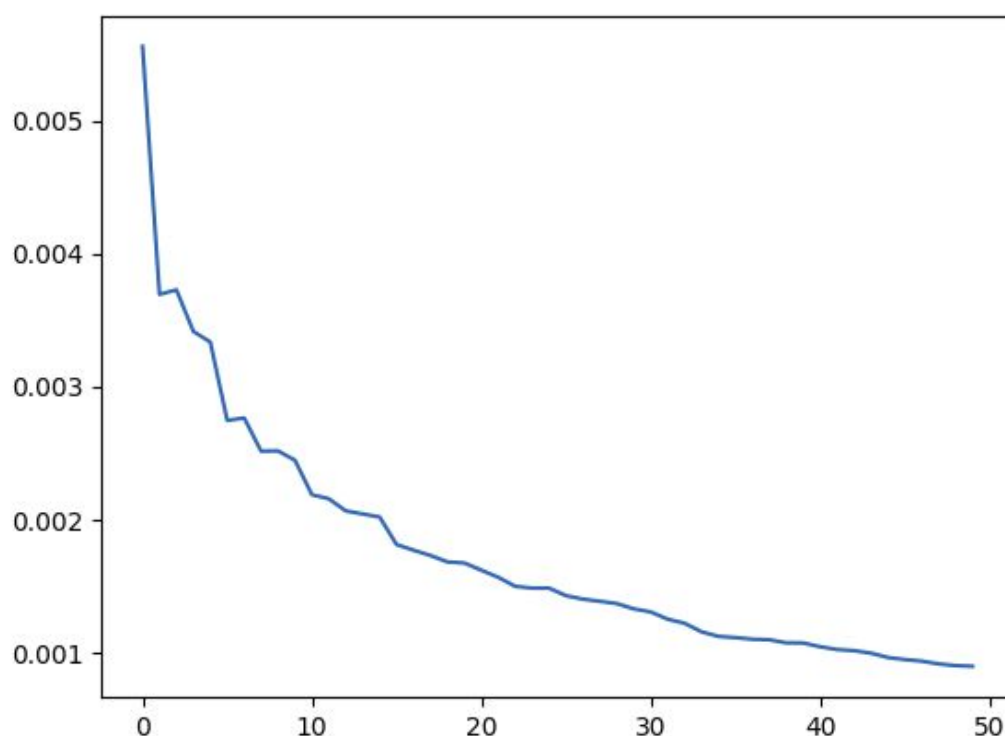


Figura 3.1 - Valores da matriz diagonal Sigma gerada pelo SVD

Com o auxílio da curva empírica da figura 3.1, foi possível definir um ponto de corte K para reduzir a dimensionalidade da matriz. Buscando um resultado computacionalmente possível e que houvesse o mínimo de perda de informações, foi escolhido K=10 como ponto de corte. Sendo assim, o SVD nos dará como saída uma nova matriz com apenas 10 colunas. Essa matriz será combinada no passo 3.6 para formar, enfim, a matriz final de features.

3.6 Combinando features

O pré-processamento, enfim, é finalizado no passo onde criamos seis features adicionais. Essas features são concatenadas horizontalmente à matriz de saída do SVD gerada no passo 3.4. Cada exemplo de treinamento receberá, além das 10 features geradas no SVD, as seguintes:

1. A diferença de quantidade de tokens entre as questões dividido pela quantidade de tokens na questão 1: $diff1 = \frac{abs(len(q1) - len(q2))}{len(q1)}$

2. A diferença de quantidade de tokens entre as questões dividido pela quantidade de tokens na questão 2: $diff2 = \frac{abs(len(q1) - len(q2))}{len(q2)}$
3. Distância de Jaccard [REF_num] entre os conjuntos de tokens das questões
4. Distância de Leveinshtein [REF_num] entre os conjuntos de tokens das questões
5. Quantidade de tokens que aparecem nas duas perguntas dividido pela quantidade de tokens na questão 1
6. Quantidade de tokens que aparecem nas duas perguntas dividido pela quantidade de tokens na questão 2

Com essas seis features adicionais, nossa matriz final de features possui exatamente 16 colunas.

4. Experimentos

Para os experimentos, três modelos de aprendizado de máquina foram utilizados: Naive Bayes do Scikit-learn, Redes Neurais do scikit-learn e o XGBoost [9]. A seguir, apresentamos os resultados obtidos. Nos três experimentos, o conjunto de exemplos foi primeiramente dividido em 10% para exemplos de validação (exemplos que nunca aparecerão no processo de aprendizagem) e 90% para treinamento. Nos três algoritmos de aprendizado, foi utilizado o método de K-Fold Cross Validation, com K=3, a fim de melhorar as métricas de avaliação. Duas métricas de avaliação foram utilizadas: acurácia e log loss.

4.1. Naive Bayes

O Naive Bayes foi utilizado como um “base line”, ou seja, um ponto de partida para que os resultados em outros modelos sejam satisfatoriamente melhores. No Scikit-learn, existem três modelos de Naive Bayes: Gaussiano, Multinomial e Bernoulli. O resultado obtido pelos três modelos pode ser conferido nas seguintes tabelas:

Naive Bayes Gaussiano		
	Acurácia	Log loss
Conjunto de treinamento	68.2416% (+/- 0.0422%)	1.3230 (+/- 0.0097)
Conjunto de validação	68.1317%	1.3590

Naive Bayes Multinomial		
	Acurácia	Log loss
Conjunto de treinamento	66.6351% (+/- 0.0734%)	0.5994 (+/- 0.0001)
Conjunto de validação	62.1954%	0.6483

Naive Bayes Bernoulli		
-----------------------	--	--

	Acurácia	Log loss
Conjunto de treinamento	62.0856% (+/- 0.0993%)	0.6463 (+/- 0.0014)
Conjunto de validação	66.7095%	0.5987

Podemos notar que o Naive Bayes Gaussiano foi o modelo que melhor se adaptou e generalizou os dados para classificação, tendo a maior acurácia. Porém, a pontuação em log loss foi bastante elevada comparada aos outros dois modelos.

4.2. Rede Neural

Definir os parâmetros ótimos em uma rede neural é uma tarefa trabalhosa e árdua. Algumas estratégias como executar um algoritmo genético para encontrar os melhores parâmetros podem durar dias, semanas e até meses. Neste trabalho, buscando a viabilidade de resultados, apenas algumas variações de parâmetros foram exploradas. Todas as combinações dos seguintes parâmetros foram testadas:

1. Quantidade de camadas: 1 e 2
2. Quantidade de neurônios por camada: 10, 30 e 50
3. Função de ativação: relu, tangente hiperbólica e logística
4. Ajuste de pesos: adam (stochastic gradient-based optimizer), lbfgs (métodos de quasi-Newton) e SGD (stochastic gradient descent)

Com as definições acima, temos 54 redes com parâmetros diferentes. A rede com maior acurácia também obteve o menor valor de log loss. A seguinte tabela mostra a configuração da rede com melhor desempenho:

Quantidade de camadas	2
Quantidade de neurônios por camada	50
Função de ativação	relu
Ajuste de pesos	adam

A tabela a seguir contém o resultado de aprendizado desta rede:

Rede neural		
	Acurácia	Log loss
Conjunto de treinamento	75.6323% (+/- 0.0554%)	0.4633 (+/- 0.0012)
Conjunto de validação	75.6511%	0.4629

4.3. XGBoost

O XGBoost é uma biblioteca de gradient boosting otimizada e muito eficiente. Assim como na rede neural, o XGBoost possui diversos parâmetros para serem otimizados. Um guia online [10] nos permite ter um ponto de partida para variação dos parâmetros. O `learning_rate` inicial foi de 0.15 e permaneceu com o mesmo valor final. O `n_estimators` começou como 170 e seu valor ajustado foi para 200. O parâmetro `max_depth` começou valendo 6 e seu valor ajustado foi para 12. Valores menores para `n_estimators` e `max_depth` pioraram o desempenho do modelo. Valores maiores para esses dois parâmetros aumentavam a acurácia e o diminuía o log loss apenas nos dados de treinamento, ou seja, causava overfitting dos dados.

Os parâmetros finais podem ser conferidos na seguinte tabela:

learning_rate	0.15
n_estimators	200
max_depth	12

O melhor resultado obtido pelo XGBoost, referente à configuração acima, pode ser conferido na tabela a seguir:

XGBoost		
	Acurácia	Log loss
Conjunto de treinamento	78.8276% (+/- 0.05%)	0.422606 (+/- 0.0008)
Conjunto de validação	78.9012%	0.4196

5. Resultados e Considerações finais

O XGBoost foi o classificador com melhor desempenho dado o conjunto de features e os parâmetros utilizados. A rede neural, por sua vez, obteve um bom resultado e muito próximo ao XGBoost, mas, ainda assim precisaria de uma exploração de parâmetros maior para verificar se poderia ser o modelo de aprendizado mais adequado ao problema. O naive bayes por sua vez cumpriu o papel proposto de base line para os outros classificadores. O gráfico da figura 5.1 compara o valor de acurácia obtidos pelos classificadores nos conjuntos de treinamento e de validação. O gráfico da figura 5.2 faz a mesma comparação com o valor de log loss.

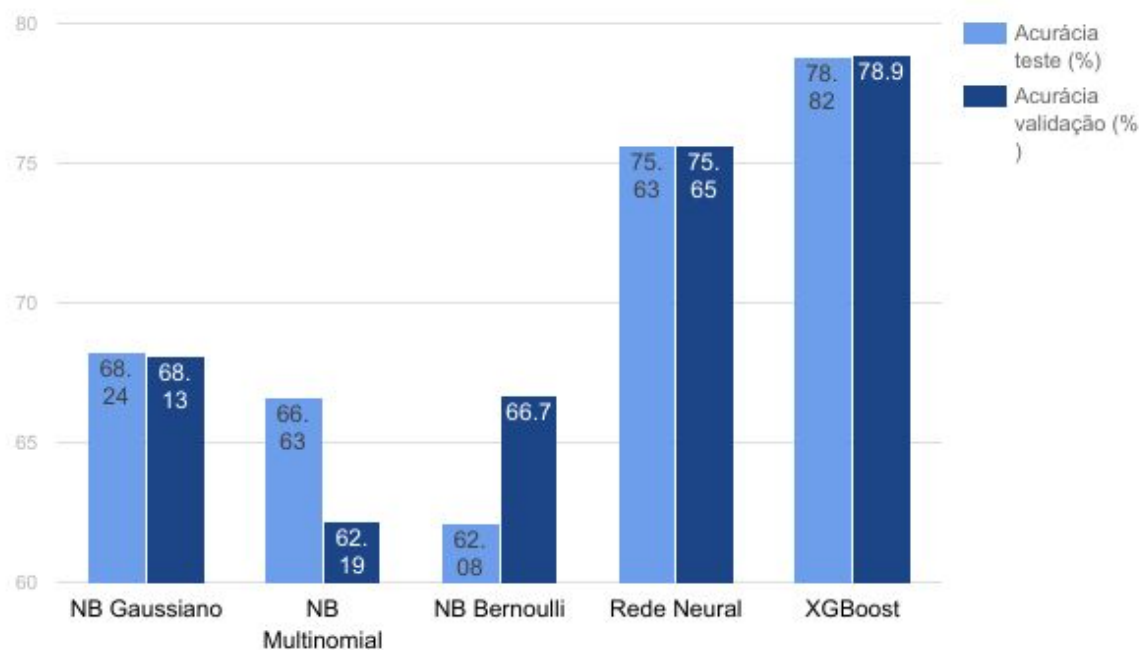


Figura 5.1 - Gráfico comparativo entre as acurácias dos classificadores

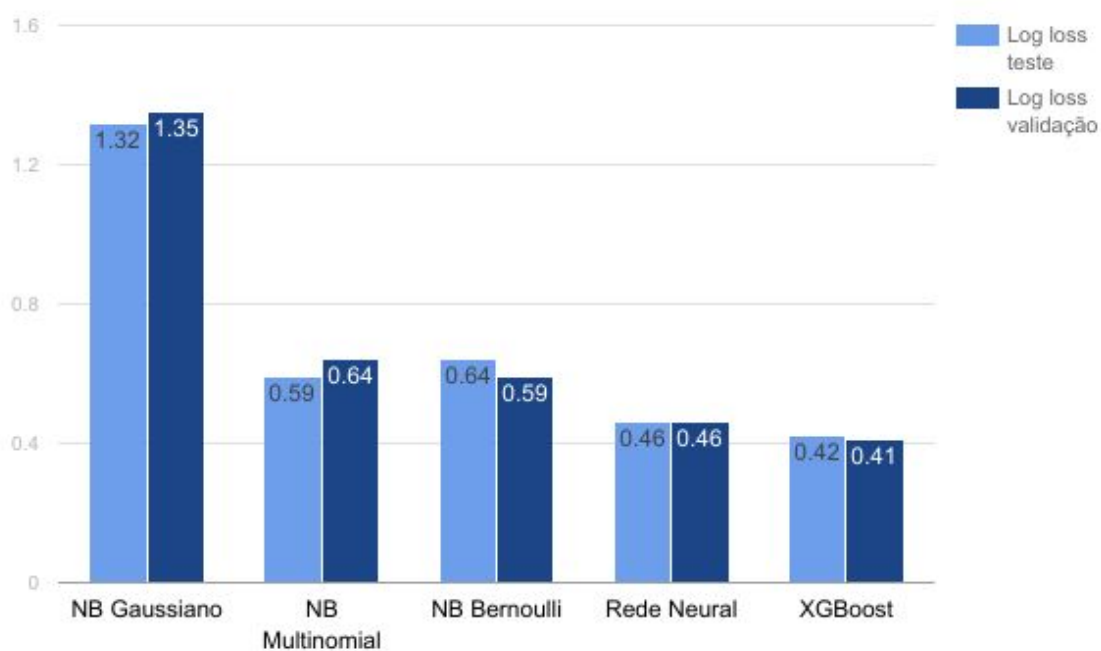


Figura 5.2 - Gráfico comparativo entre os valores de log loss dos classificadores

O presente trabalho foi realizado com êxito dentro de seus objetivos. De acordo com o escopo do problema, foi proposta uma solução para criação do modelo de features que

seria usado como o conjunto de entrada para os classificadores. Diversas técnicas de mineração de dados foram utilizadas e aplicadas juntamente ao aprendizado de máquina em busca da melhor solução para o problema de identificação de perguntas duplicadas.

6. Referências

- [1] Quora Questions Pairs, 2017. Disponível em: <https://www.kaggle.com/c/quora-question-pairs>. Acesso em: 15 de mai. 2017.
- [2] Kaggle, 2017. Disponível em: <https://www.kaggle.com/>. Acesso em: 15 de mai. 2017.
- [3] Quora, 2017. Disponível em: <https://www.quora.com/>. Acesso em: 15 de mai. 2017.
- [4] Python Software Foundation, 2017. Disponível em: <https://www.python.org/>. Acesso em: 15 de mai. 2017.
- [5] scikit-learn: Machine Learning in Python, 2017. Disponível em: <http://scikit-learn.org/stable/>. Acesso em: 15 de mai. 2017.
- [6] Natural Language Toolkit, 2017. Disponível em: <http://www.nltk.org/>. Acesso em: 15 de mai. 2017.
- [7] GitHub, 2017. Disponível em: <https://github.com/chriiscardozo/QuoraQuestionPair/>. Acesso em: 15 de mai. 2017.
- [8] Stemmers, 2017. Disponível em: <http://www.nltk.org/howto/stem.html>. Acesso em: 15 de mai. 2017.
- [9] XGBoost Documents, 2017. Disponível em: <https://xgboost.readthedocs.io/en/latest/>. Acesso em: 15 de mai. 2017.
- [10] Complete Guide to Parameter Tuning in XGBoost (with codes in Python), 2017. Disponível em: <https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/>, 2017. Acesso em: 15 de mai. 2017.

