Trabalho prático 3

Técnicas de aprendizado não supervisionado

Julianny Favinha Donda
156059
julianny.favinha@gmail.com

José Henrique Ferreira Pinto
155976
joseh.fp@gmail.com

Objetivo

O objetivo deste trabalho é explorar o conceito de técnicas não supervisionadas para aprendizado de máquina. Nos métodos estudados até agora, a entrada era composta de observações e *target*, o que permitia regular o resultado obtido através de funções de custo como *mean squared error* e *cross entropy*. Os métodos não supervisionados, diferentemente, são aplicados para encontrar padrões, e agrupar os dados de acordo com esses padrões, porém a interpretação desse agrupamento é responsabilidade do programador.

Atividade

Vamos avaliar uma base de dados de *tweets*[1] relacionados com o tema de saúde, no formato *bag of words* disponibilizado pela professora. Os dados não foram separados em treinamento e teste e, com esses dados, pretendemos testar os métodos K-means e DBSCAN de clusterização para agrupar os dados e verificar se eles mostram algum sentido para cada grupo formado.

K-means

O método K-means recebe como entrada um número de grupos. Então, uma variação escolhe pontos aleatórios são escolhidos como centro desses grupos. A partir daí, o algoritmo arranja todas as observações nos grupos, de acordo com a distância dos pontos aos centróides. Depois disso, os centróides são atualizados para ficar na posição central do novo grupo. Repetindo esses passos, encontramos os *clusters*.

Experimento

Utilizamos a implementação da biblioteca Sci-Kit Learn[3]. Para encontrar um número de *clusters* que fosse satisfatório, executamos o algoritmo para os valores de 10 a 300 *clusters*, ao passo de 10 em 10. Para cada valor, encontramos a métrica *inertia*, um dos resultados da execução do algoritmo da biblioteca, que indica a soma das distâncias quadradas das amostras para o centro do *cluster* mais próximo.

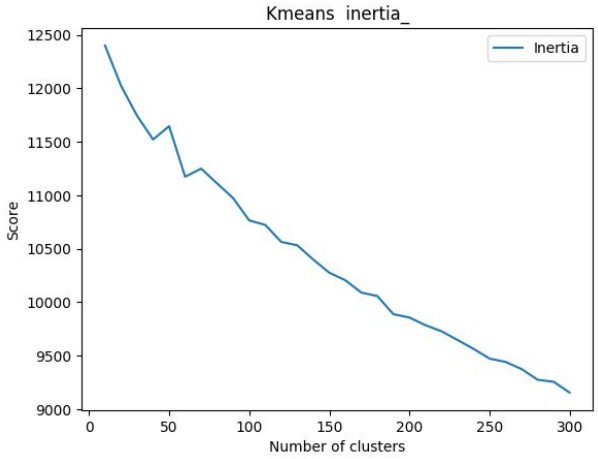

Figura 1 - Gráfico da métrica inertia pelo número de *clusters* para o método de K-means, com as features originais do *bag of words*.

Além disso, também calculamos o coeficiente de silhueta, que não variou muito, ficando entre 0,02 e 0,07.

Ao analisar os *clusters* gerados, estranhamos a distribuição das frases. Alguns *clusters* continham mais de 1000 sentenças outros com 10 ou menos. Entretanto, ficou evidente que o algoritmo foi aplicado de forma correta pois foram formados *clusters* onde a maioria possuía o mesmo texto de *tweets*. Identificamos por exemplo algumas mensagens que podem ser consideradas *spam*.

```
Cluster 7
6321: The real secret to a healthy complexion? Eating more of these:
6434: The real secret to a healthy complexion? Eating more of these:
7536: The real secret to a healthy complexion? Eating more of these:
7681: The real secret to a healthy complexion? Eating more of these:
8354: The real secret to a healthy complexion? Eating more of these:
8615: The real secret to a healthy complexion? Eating more of these:
8928: The real secret to a healthy complexion? Eating more of these:
```

Figura 2 - Captura de tela de trecho do arquivo de saída (DictionaryOfClustersKmeans.txt) do programa Kmeans.py para um determinado *cluster* gerado.

Outros *clusters* analisados tentaram agrupar *tweets* que possuíam as palavras "AUDIO:", "VIDEO:", "RT @". Entretanto, outros *clusters* pareciam fazer pouco sentido. Para verificar se o resultado melhorava ou piorava,

aplicamos a técnica de redução de dimensionalidade PCA[6], com 0,95 de variância total.

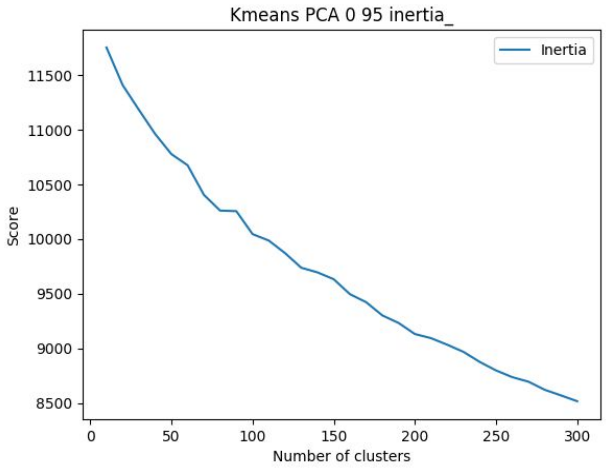

Figura 3 - Gráfico da inertia pelo número de *clusters* para o método de K-means, com as features de *bag of words* selecionadas pelo método de redução de dimensionalidade PCA, variância 95%.

Notamos que o resultado não obteve uma mudança significativa com 0,95 de variância, nem para os *clusters* gerados. Dessa forma, mais um teste foi feito, com 0,65 de variância total.

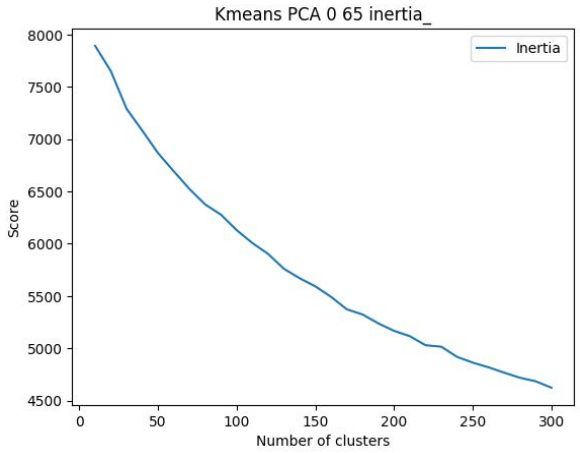

Figura 4 - Gráfico da inertia pelo número de *clusters* para o método de K-means, com as features de *bag of words* selecionadas pelo método de redução de dimensionalidade PCA, variância 65%.

Nota-se que também não houve diferenças significativas. Como os *clusters* ainda aparentavam estar mal distribuídos, começamos a discutir a morfologia dos dados, e decidimos testar outra forma de agrupamento.

DBSCAN

Diferentemente do K-means, o método DBSCAN define *clusters* a partir da densidade de observações[2], considerando parte do *cluster*, todos os pontos que estão próximos (a uma determinada vizinhança) de outros pontos do *cluster*.

Experimento

Utilizamos a implementação da biblioteca Sci-Kit Learn[4]. Fizemos uma tabela de possíveis *epsilon (distância entre pontos do mesmo cluster)* e mínimo de observações para cada *cluster* e buscamos o que apresentou um resultado melhor. O número mínimo de observações nos permite desconsiderar *outliers*, pois se existem *clusters* pequenos (abaixo de um determinado limite) e separados dos *clusters* principais, podemos considerá-los como ruído.

A primeira implementação do programa executava o algoritmo de DBSCAN da biblioteca e calculava o coeficiente de silhueta. Notou-se que o coeficiente estava negativo, ou seja, a clusterização estava extremamente insatisfatória, mesmo mudando os parâmetros, o que nos levou a questionamentos sobre os dados. Logo percebemos que, no cálculo desse coeficiente, estávamos considerando as sentenças que o algoritmo de clusterização estava rotulando como *outlier*. Portanto, retiramos do cálculo essas sentenças, o que fez o coeficiente aumentar significativamente e a partir daí pudemos fazer uma análise mais assertiva.

Tabela 1 - Dados coletados ao executar o algoritmo DBSCAN. Como entrada, temos os parâmetros eps e min_samples, sendo o restante a saída do programa.

eps	min_samples	Clusters	Observations without outlier	Silhouette Score
0.050	2	950	2809	1.00
0.050	3	286	1481	1.00
0.050	4	145	1058	1.00
0.050	5	88	830	1.00
0.100	2	950	2809	1.00
0.100	3	286	1481	1.00
0.100	4	145	1058	1.00
0.100	5	88	830	1.00
0.719	2	1402	5110	0.74
0.719	3	601	3508	0.71
0.719	4	353	2684	0.70
0.719	5	215	2049	0.70
0.769	2	1364	5937	0.65
0.769	3	614	4437	0.57
0.769	4	418	3676	0.56
0.769	5	297	3056	0.59
0.788	2	1272	6253	0.58
0.788	3	538	4785	0.52

0.788	4	371	4072	0.45
0.788	5	273	3461	0.46
1.000	2	21	12926	0.19
1.000	3	7	12898	0.18
1.000	4	5	12890	0.18
1.000	5	2	12873	0.18

Para poder estudar o possível formato dos dados, como dito, fizemos um *grid search*. Começamos com os valores 0,1 e 1,0 e fizemos uma busca binária para encontrar o melhor valor, verificando se o coeficiente de silhueta aumentava.

Notamos que para *epsilon* 0,05, os valores de *min_samples* não mudaram o coeficiente de silhueta, que resultou em 1,0, o resultado ideal para essa métrica. Entretanto, vemos que cerca de 90% dos dados foram considerados *outliers* para obter esse resultado.

Para epsilon 1,0, vemos que o algoritmo quase não conseguiu clusterizar os dados. Isso indica que os dados estavam a uma proximidade menor de *epsilon*, logo a maioria ficou no mesmo *cluster*, sendo que poucos foram considerados *outliers*. Dessa forma, o coeficiente de silhueta diminuiu consideravelmente. Nota-se que há uma relação entre epsilon e número de observações que não são consideradas *outliers*: quanto maior o epsilon, a maioria das sentenças são agrupadas no mesmo *cluster*, ou seja, é necessário encontrar um valor de *epsilon* e um valor de número mínimo de sentenças para que o algoritmo consiga considerar o máximo valor de sentenças (poucas sentenças sendo consideradas *outliers*) e possua um número relativamente bom para o coeficiente de silhueta.

Para verificar se o resultado melhorava ou piorava, aplicamos a técnica de redução de dimensionalidade PCA[6], com 0,95 de variância total. Notamos que o resultado foi pouco diferente, e para epsilon 1,0 o algoritmo não conseguiu clusterizar os dados, tanto que o coeficiente de silhueta não pôde ser calculado, pois é necessário um mínimo de 2 *clusters* para essa métrica. Também foi feito o teste para um PCA com 0,65 de variância total

Tabela 2 - Dados coletados ao executar o algoritmo DBSCAN depois de executar PCA no conjunto de dados. Como entrada, temos os parâmetros eps e min_samples, sendo o restante a saída do programa.

eps	min_samples	Clusters	Observations without outlier	Silhouette score
0.050	2	951	2813	0.99
0.050	3	286	1483	0.99
0.050	4	146	1063	0.99
0.050	5	89	835	0.99
0.100	2	961	2842	0.99
0.100	3	291	1502	0.99
0.100	4	147	1070	0.99
0.100	5	89	838	0.99
0.719	2	162	12012	0.30
0.719	3	41	11770	0.31
0.719	4	19	11661	0.33
0.719	5	14	11599	0.33
0.769	2	77	12693	0.31
0.769	3	13	12565	0.29
0.769	4	6	12528	0.32
0.769	5	6	12517	0.32
0.788	2	49	12893	0.30
0.788	3	8	12811	0.29
0.788	4	3	12787	0.31
0.788	5	3	12782	0.31
1.000	2	1	13229	---
1.000	3	1	13229	---
1.000	4	1	13229	---
1.000	5	1	13229	---

Nota-se que a partir de um epsilon igual ou maior a 0,719, a maioria das sentenças entra para algum cluster.

Análise de *clusters*

K-means

Como a redução de dimensionalidade não melhorou os resultados, então o melhor resultado que tivemos foi utilizando os dados originais.

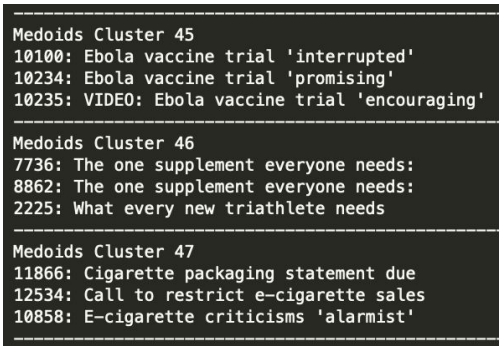

Figura 5 - Captura de tela de trecho do arquivo de saída (MedoidsOfClustersKmeans.txt) do programa Kmeans.py para um determinados *clusters* gerados.

Metade dos dados estão em um mesmo *cluster*, não foi possível encontrar relação entre eles. Acreditamos que esse resultado pode ter sido referente à falta de limpeza dos

dados, pois 95% das frases possuem palavras comuns como “to”, “you”, “the”. Entretanto, nota-se que a maioria dos medóides dos *clusters* fazem sentido, ou seja, de certa forma, o algoritmo conseguiu clusterizar certos padrões, pela frequência de palavras.

DBSCAN

Notamos que, ao executar a redução de dimensionalidade PCA, o algoritmo conseguiu incluir mais sentenças nos clusters. O resultado que encontramos foi melhor para variância 65%. Acreditamos que o resultado mais satisfatório foi para epsilon igual a 0,719 e num_samples igual a 2, que considerou a maioria massiva dos dados (cerca de 90% dos dados), como exibido na Tabela 2.

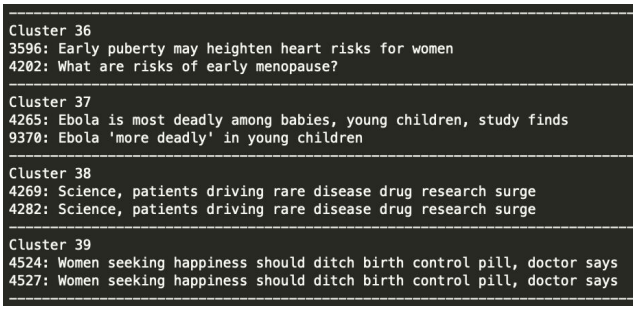

Figura 6 - Captura de tela de trecho do arquivo de saída do programa DBSCAN.py para um determinados *clusters* gerados.

Como efeito colateral, nota-se que muitos clusters acabaram ficando com um número mínimo de sentenças, que, se min_samples fosse maior que 2, poderiam ser potencialmente considerados como *outliers*. Nota-se que, assim como no algoritmo de K-means, o algoritmo clusterizou sentenças que possuíam as palavras "AUDIO:", "VIDEO:", "RT @". O algoritmo DBSCAN não usa o conceito de medóides, e sim da densidade (vizinhança) entre as observações.

Conclusões e trabalho futuro

Ao realizar o trabalho, percebemos que a noção de "qualidade de um cluster" varia muito de acordo com os dados, da mesma forma que a qualidade das métricas. Elas devem ser escolhidas para o que queremos avaliar e como a base de dados está organizada. Usamos o coeficiente de silhueta para esse trabalho por ser um coeficiente que avalia as distâncias intra e inter *cluster*, nos dando uma noção da morfologia dos *clusters.*

Existem outras métricas, como a Davies-Bouldin, que avalia a similaridade intra *cluster* e diferenças inter *cluster.*

Outro aspecto importante a se considerar é se as métricas são normalizadas ou não. Por exemplo, o coeficiente de silhueta retorna sempre um valor no intervalo $[-1, 1]$, sendo -1 uma péssima clusterização e 1 o ideal. Por outro lado, o coeficiente Davies-Bouldin não é normalizado, o que torna difícil comparar dois valores de *datasets* diferentes.

Uma outra forma de medir qualidade seria utilizar a mesma métrica que o algoritmo de clusterização utiliza, se não a qualidade estará enviesada.

Como trabalho futuro, podemos aplicar outros métodos de clusterização, para verificar se fornecem resultados melhores. E, além disso, aplicar outras métricas para validar a qualidade dos clusters.

Notamos que a clusterização tenta agrupar frases que possuem as mesmas palavras. Gostaríamos de limpar a base de dados retirando palavras que não tem muito significado, como artigos, conjunções e pronomes.

Seria bom testar um número maior de clusters com o algoritmo de K-means e testar um número menor de redução de dimensionalidade para o algoritmo DBSCAN, a fim entender melhor o balanço e relação entre *outliers* e qualidade dos *clusters*.

Finalmente, para fins de comparação, poderíamos realizar os mesmos experimentos para o arquivo word2vec disponibilizado pela professora.

Referências

[1] **Health News in Twitter dataset.** Disponível em <https://archive.ics.uci.edu/ml/datasets/Health+News+in+Twitter#> Acesso em 19 out 2018.

[2] **Visualizing DBSCAN Clustering.** Disponível em <https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/> Acesso em 2 nov 2018.

[3] **Kmeans of Sci-kit Learn.** Disponível em <http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html> Acesso em 29 out 2018.

[4] **DBSCAN of Sci-kit Learn.** Disponível em <http://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html#sklearn.cluster.DBSCAN> Acesso em 2 nov 2018.

[5] **Silhouette score of Sci-kit Learn.** Disponível em <http://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html#sklearn.metrics.silhouette_score> Acesso em 29 out 2018.

[6] **Principal component analysis (PCA).** Disponível em http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html> Acesso em 29 out 2018.