Aprendizagem de Máquina (2020/Período Especial) - Impactos da Base de Aprendizagem

Diogo C. T. Batista[1]

[1]Universidade Federal do Paraná (UFPR)
Curitiba – Paraná – Brasil

diogo@diogocezar.com

1. Impactos da Base de Aprendizagem

Esta atividade laboratorial tem como objetivo a investigação exploratória dos impáctos da base de aprendizagem na performance de diferentes classificadores. Os classificadores a serem analisados neste laboratório são:

- KNN
- Naïve Bayes
- Linear Discriminant Analysis
- Logistic Regression
- Perceptron

Os dados estão dispostos em arquivo no formato *svmlight*. Este formato dispõe as informações de categorias e suas respectivas características. O código 1 mostra um pequeno trecho desta representação.

```
0 1:0.000000 2:0.000000 3:0.001412 4:0.000000 5:0.014124
  6:0.000000 ...
```

Código 1. Exemplo do Formato de Entrada

Neste exemplo, podemos notar que o 0 inicial mostra a qual categoria este dado pertence, na sequência, demonstra-se para cada característica o seu valor. A característica 1 possui o valor 0.000000.

Os dados utilizados neste experimento estão divididos em 2 partições. A primeira partição possui 20.000 registros, que foi definida para uma base de treinamento. Já a segunda, possui 58.646 informações, utilizadas para o teste.

1.1. Estratégia para Execução do Experimento

Para automatização da suite de experimentos, foi criado um sistema *orquestrador* no qual é possível definir quais vão ser os experimentos a serem executados. O Código 2 mostra os valores definidos em um arquivo no formato *JSON*.

```
[
  {
    "classifier": "knn",
    "chunk_start": 1000,
    "chunk_stop": 20000,
    "chunk_step": 1000
  },
```

```
8     {
9       "classifier": "naive\_bayes",
10      "chunk_start": 1000,
11      "chunk_stop": 20000,
12      "chunk_step": 1000
13    },
14    ...
15  ]
```

Código 2. JSON do Orquestrador

Neste JSON, *classifier* é o nome do classificador a ser utilizado. Estes, podem variar entre: knn, naive_bayes, lda, logistic_regression, e perceptron. Os outros atributos servem parar criar chunks que dividem os testes em função da disponibilidade da base de treinamento. Por exemplo, no primeiro bloco, *chunk_start* indica o tamanho do bloco inicial, *chunk_stop* indica a condição de parada, e por fim o *chunk_step* indica o incremento.

Para os testes, em todos os classificados variou-se a base de treinamento de 1000 até 20000, com o intervalo de 1000.

O script implementado ainda realiza a automação da coleta dos resultados, salvando em arquivo no formato *csv* todos os dados extraídos dos experimentos.

Os dados salvos são: *Classifier*, *Chunk*, *F1Score*, *Accuracy* e *Execution Time (s)*.

1.2. Parametrização dos Classificadores

Para este trabalho, foram implementados os parâmetros padrões para todos os classificadores anteriormente descritos. Assim sendo, nenhum ajuste específico foi realizado qualquer um dos classificadores testados.

1.3. Experimentos

Foram realizados 100 experimentos. 20 variação de 1000 20000 para cada um dos 5 classificadores. Os 20 melhores resultados estão demonstrados por ordem de *F1Score* seguidos de *Acurácia* e o *Tempo de Execução* na tabela 1

1.4. Análises dos Resultados

A análise dos resultados leva em considerações os quesitos:

1. Comparação do Desempenho dos Classificadores;
2. Classificador que tem o melhor desempenho com poucos dados;
3. Classificador que tem melhor desempenho com todos os dados;
4. Classiticador mais rápido;
5. Análise das Matrizes de Confusão;

1.4.1. Comparação do Desempenho dos Classificadores

A figura

Classifier	Chunk	F1Score	Accuracy	Execution Time (s)
knn	20.000	0,939	0,939	292,836
perceptron	11.000	0,938	0,938	5,501
knn	19.000	0,938	0,938	279,718
knn	18.000	0,938	0,937	268,223
knn	16.000	0,937	0,937	251,132
knn	17.000	0,937	0,937	257,359
knn	12.000	0,936	0,936	225,239
knn	13.000	0,936	0,936	237,264
knn	14.000	0,936	0,936	253,580
knn	15.000	0,936	0,935	270,468
perceptron	17.000	0,935	0,935	5,710
knn	10.000	0,935	0,935	198,619
knn	11.000	0,935	0,934	210,219
perceptron	18.000	0,934	0,934	5,717
knn	9.000	0,933	0,933	178,040
knn	8.000	0,930	0,929	157,567
lda	10.000	0,928	0,928	5,030
perceptron	13.000	0,928	0,928	5,337
lda	20.000	0,928	0,928	5,403
perceptron	20.000	0,926	0,927	5,677

Tabela 1. Melhores Resultados

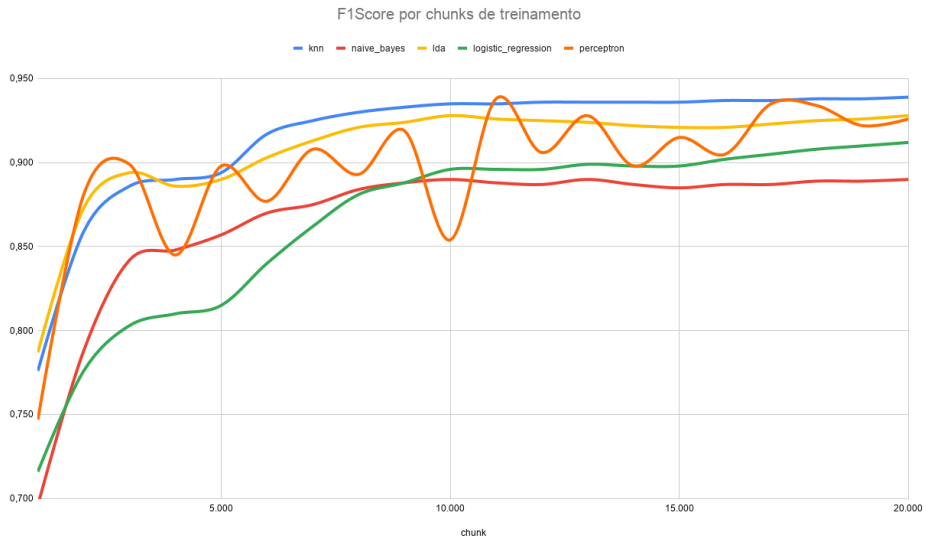

Figura 1. Comparação de Desempenho dos Classificadores

2. Experimentos

Assim como na adaptação do primeiro algoritmo, um sistema de entrada JSON possibilitou a variação de combinações para análise de diferentes resultados.

- O arquivo de representação fonte; as variações testadas foram: [features_20_10,

features_36_46, features_40_20, features_60_30, features_99_81]

- Ativar ou desativar a normalização; as variações testadas foram: [0,1]
- Alterar o tipo de distância; as variações testadas foram: [eucledian, manhattan]
- Alterar o K; as variações testadas foram: [1,3,4,5,6,7,8,9,10,11,12,13]. Foram considerados os K pares, apenas para efeito de comparações, sabendo-se se sua ineficiência por natureza.

O Código 3 mostra as opções de entrada para o algorítmo, podendo variar:

```
[
  {
    "data": "features_20_10",
    "normalized": 0,
    "distance": "euclidean",
    "k": 1
  },
  ...
]
```

Código 3. JSON para experimentos

3. Resultados

Foram realizadas *240* execuções para as variações listadas anteriormente. Os *20* melhores resultados (ordenados por Acurácia e F1Score) estão detalhados na Tabela 1. Já os 20 piores resultados (ordenados por Acurácia e F1Score) estão detalhados na Tabela 2.

Pode-se observar que os melhores resultados foram obtidos a partir das representações que consideraram as *maiores* alturas e larguras das imagens (99x81). Isso acontece pois, no momento em que se realiza a compressão das imagens, características são perdidas. O melhor resultado teve a *Acurácia* de $0,928$ e o *F1Score* de $0,929$ demorando 2.743 segundos para execução completa.

Já para nos piores resultados, nota-se que foram obtidos na variação de K: $K = 13$, $K = 12$, $K = 10$. Quanto ao K para os melhores resultados, percebe-se que foram obtidos com $K = 1$, seguido de $K = 3$ e $K = 5$. Isso mostra que a distribuição possui uma particularidade interessante, na qual ao se comparar apenas os vizinhos mais próximos se consegue um melhor resultado. Como já esperado, para os resultados nos quais K tinha um valor par, seus resultados foram menos eficientes.

Em outra análise, a estratégia de obter as médias das alturas e larguras também apresentaram resultados significativos (imagens de dimensão $36x46$), mas ficaram abaixo dos resultados da estratégia empírica que utiliza imagens $60x30$. O destaque é para o tempo de execução em *features_60_30*, que conseguiu-se uma Acurácia de $0,925$ e o F1Score de $0,926$ demorando apenas 203 segundos para execução completa.

As normalizações não parecem fazer muita diferença nos resultados, pois estes já se encontram bem distribuídos.

Em geral a alteração da estratégia de medição das distâncias (*euclidean* e *manhattan*) não influenciaram em uma melhoria da *Acurácia* ou do *F1Score*. O que se pode notar, foi que nos testes executados a estratégia *manhattan* impacta negativamente no tempo de execução.

4. Comparação das Matrizes de Confusão

Para cada execução, criou-se um arquivo de resultados que também armazena as matrizes de confusão.

Experiment	Normalized	Distance	K	Accuracy	F1Score	Execution Time (s)
features_20_10	1	manhattan	13	0,871	0,873	40
features_20_10	1	euclidean	13	0,871	0,873	30
features_20_10	0	manhattan	13	0,871	0,873	20
features_20_10	0	euclidean	13	0,871	0,873	10
features_20_10	1	manhattan	12	0,874	0,876	39
features_20_10	1	euclidean	12	0,874	0,876	29
features_20_10	0	manhattan	12	0,874	0,876	19
features_20_10	0	euclidean	12	0,874	0,876	9
features_99_81	1	manhattan	12	0,876	0,879	4.294
features_99_81	1	euclidean	12	0,876	0,879	3.894
features_99_81	0	manhattan	12	0,876	0,879	3.485
features_99_81	0	euclidean	12	0,876	0,879	3.090
features_20_10	1	manhattan	10	0,879	0,881	37
features_20_10	1	euclidean	10	0,879	0,881	27
features_20_10	0	manhattan	10	0,879	0,881	17
features_20_10	0	euclidean	10	0,879	0,881	7
features_99_81	1	manhattan	13	0,880	0,883	4.328
features_99_81	1	euclidean	13	0,880	0,883	3.928
features_99_81	0	manhattan	13	0,880	0,883	3.517
features_99_81	0	euclidean	13	0,880	0,883	3.126

Tabela 2. Piores Resultados

A Tabela 3 demonstra a matriz de confusão para o melhor resultado, enquanto que a Tabela 4 demonstra a matriz de confusão para o pior resultado.

É possível notar que apesar dos erros persistirem na classificação, como por exemplo em: [0,2],[1,2],[1,3],[1,4],[1,6],[1,7],[1,8],[1,9], eles diminuram consideravelmente. Como por exemplo, no pior caso, a classificação [1,4] foi classificada erroniamente 12 vezes, enquanto que no melhor caso, o erro ocorreu em apenas 8 vezes. Isso acontece, pois, aumentar o número de *pixels* analisados nas imagens, geram representações mais precisas, porém, como o método é o mesmo, pôde-se observar que os equívocos continuaram existindo nos mesmos lugares, mesmo que em menor número.

95	0	0	0	0	1	1	0	0	0
0	93	0	0	0	1	0	0	0	1
0	1	103	1	0	0	0	4	1	1
0	1	0	98	0	1	0	0	3	0
0	8	0	0	83	1	1	0	0	2
2	0	0	5	0	89	1	0	0	0
1	5	0	0	0	0	100	0	0	0
0	3	1	0	1	0	0	88	0	4
0	3	0	1	1	2	0	1	79	0
0	1	0	0	2	0	0	9	0	100

Tabela 3. Matriz de Confusão - Melhor Caso

95	1	0	0	0	1	0	0	0	0
0	95	0	0	0	0	0	0	0	0
3	9	89	1	0	0	3	6	0	0
1	1	1	98	0	0	0	2	0	0
0	12	0	0	78	0	1	0	0	4
0	1	0	7	0	88	1	0	0	0
1	6	0	0	0	0	99	0	0	0
0	11	0	0	3	0	0	79	0	4
0	5	0	3	1	6	0	5	65	2
0	3	0	0	7	0	0	17	0	85

Tabela 4. Matriz de Confusão - Pior Caso

5. Código Fonte

Os códigos preparados podem ser analisados através do repositório: https://github.com/diogocezar/machine-learning/tree/master/lab1/src