Análise de técnicas de aprendizado de máquina com o conjunto de dados MNIST

André Filipe Caldas Laranjeira
Curso de Engenharia da computação
Universidade de Brasília
Matrícula: 16/0023777
Email: andrecaldaslaranjeira@gmail.com

Victor André Gris Costa
Curso de Engenharia da computação
Universidade de Brasília
Matrícula: 16/0019311
Email: victorandr98@gmail.com

***Resumo*—Existem várias escolhas relevantes a serem feitas em um projeto de aprendizado de máquinas para classificação de entradas. Fatores como características analisadas (ou *features*), algoritmos de aprendizado e seus parâmetros e métricas de sucesso devem ser bem escolhidos para possibilitar a geração de um sistema classificador satisfatório. Esse artigo busca detalhar as conclusões obtidas por meio da comparação de um conjunto de algoritmos de aprendizado utilizadas na construção de um sistema classificador de dígitos manuscritos alimentado com exemplos do conjunto de dados MNIST.**

***Index Terms*—algoritmos, aprendizado, sistema, classificador, MNIST**

I. Problema a ser resolvido

O problema a ser resolvido é a criação de sistemas inteligentes para classificar corretamente os dígitos manuscritos da base de dados MNIST. Com base nos resultados obtidos por nosso programa, devemos analisar como as características e algoritmos de aprendizagem escolhidos influenciaram de forma positiva ou negativa a performance de nossos sistemas inteligentes.

II. Especificações técnicas do programa

- Linguagem de programação utilizada: Python 3.7.3
- Pacotes utilizados:
  - numpy (1.15.4)
  - opencv-python (4.0.0.21)
  - python-mnist (0.6)
  - scikit-learn (0.20.3)
- Tempo médio de execução do programa: 5 minutos
- Repositório do programa: https://github.com/AndreLaranjeira/Machine_learning-MNIST

III. Extração das características

A. Dados originais

A base de dados utilizada é a base de dados MNIST para dígitos manuscritos. Após baixarmos os arquivos com os dados do MNIST, utilizamos o pacote *python-mnist* para carregar os dados dos arquivos da base de dados em listas do pacote *numpy*. Originalmente, o pacote *python-mnist* extraiu cada dígito manuscrito na forma de uma lista de 784 posições, onde cada posição era um valor de escala de cinzas (0 a 255) de um pixel da imagem 28x28. Já os labels dos dígitos manuscritos foram extraídos na forma de um inteiro representando o número representado pelo dígito manuscrito.

B. Criação de contorno

O primeiro passo para começarmos a extrair características significativas dos nossos dados de entrada foi gerar um contorno a partir dos pixels de escala de cinzas utilizados. Para isso, utilizamos funções do pacote *opencv-python*, as quais nos permitiram extrair os contornos dos dígitos manuscritos e nos forneceram valores como a área e o perímetro do contorno.

Nesse momento, nos deparamos com a escolha de um parâmetro de funcionamento de sistema, que foi o **limite de escala de cinza**. O limite de escala de cinza determina o valor de escala de cinza a partir do qual os pontos seriam ou não incluídos no contorno gerado. Inicialmente, pensamos em utilizar o valor de 127, que seria o valor médio possível da escala de cinza de um ponto. Infelizmente, isso gerou contornos com uma quantidade muito baixa de pontos, de forma que alguns contornos possuíam área igual a 0 conforme o retorno das funções do pacote *opencv-python*. Após realizarmos um ajuste experimental, decidimos utilizar **o limite de escala de cinzas como sendo 70**. Isso nos forneceu contornos úteis e aumentou nossa acurácia para os algoritmos utilizados em aproximadamente 0,5%.

C. Escolha das características

Inicialmente, planejávamos utilizar apenas 3 características de contorno em nossos sistemas de aprendizagem (Número de Euler, Retangularidade e Circularidade), mas obtivemos resultados com performance abaixo do esperado (aproximadamente 50% de acurácia em todos os métodos). Suspeitamos que essa baixa acurácia inicial se deva à grande variação da retangularidade e circularidade no conjunto de dados estudado, até mesmo em dígitos da mesma classe, causando certa sobreposição nas classes de dados estudadas. Ou seja, podemos dizer que essas características não se correlacionam muito bem com as classes de dígitos manuscritos estudados, ao contrário, por exemplo, do número de Euler. Isso fez com que dígitos que tivessem um número de Euler diferente fossem muito constantemente confundidos.

Com o intuito de solucionar o problema anteriormente mencionado, nós adicionamos elongação, solidez, convexidade

e ângulo do eixo de menor inércia como características dos dados. Com cada característica adicionada, obtivemos um ganho de aproximadamente $3,5\%$ de acurácia, indicando que, embora cada característica estivesse nos fornecendo novas informações acerca de cada dígito manuscrito, essas informações ainda apresentavam uma baixa correlação com as classes de dígito estudadas e certa dependência entre si, não sendo suficientes para distinguir de forma clara o dígito manuscrito.

Com as características em mãos, experimentamos as várias combinações lineares possíveis entre as características disponíveis, buscando constatar se a adição ou retirada de alguma característica melhoraria a performance do sistema. Constatamos que a retirada de qualquer característica nos fornecia uma perda de acurácia, sendo que a mais notável perda ocorreu quando retiramos o número de Euler das características estudadas. Nós também notamos que não haviam perdas significantes de acurácia nos algoritmos utilizados ao retirarmos a característica de circularidade, motivo pelo qual retiramos essa característica da análise realizada. Acreditamos que esse fato decorre do fato de já estarmos utilizando a característica de retangularidade, a qual possui uma grande dependência do valor da circularidade de um dígito manuscrito.

1) Número de Euler: O número de Euler é a subtração entre a quantidade de partes contínuas e os seus buracos. Nós acreditamos que esse número poderia auxiliar a diferenciar números com formatos parecidos e quantidade de buracos diferentes como o 0 e 8. Como havia perdas em acurácia grandes ao retirá-lo, o mantivemos e chegamos a conclusão de que ele era uma das características mais relevantes extraídas dos dados, devido a sua alta correlação (em comparação com as outras características utilizadas) com as classes de dígitos manuscritos.

2) Retangularidade: A retangularidade é a razão entre a área do menor retângulo que envolve o contorno e a área da próprio contorno. Acreditamos que essa característica auxiliaria a diferenciar números cheios de curvas e com espaços vazios, como o 2, de números mais retangulares e preenchidos, como o 1. Como havia perdas em acurácia médias ao retirá-lo, o mantivemos e chegamos a conclusão de que ele possuía certa correlação com as classes de dígitos estudadas, embora não no nível esperado. Em retrospectiva, esse nível de correlação deveria ser esperado, dado a grande variação de formas que um mesmo dígito pode ser escrito.

3) Elongação: A elongação de uma forma pode ser obtida pela equação 1, onde Elo é o valor de elongação (que pertence ao intervalo [0; 1]), W é a largura do menor retângulo no qual o contorno pode ser inscrito e L é a altura do menor retângulo no qual o contorno pode ser inscrito. [1] Note que essa fórmula assume que $W < L$, de forma que os valores de W e L foram trocados sempre que $W > L$ e que o retângulo mencionado anteriormente pode possuir qualquer angulação.

$$Elo = 1 - \frac{W}{L} \tag{1}$$

Portanto, números que possuam dimensões de largura e altura próximas terão uma elongação mais próxima de 0, enquanto números que possuem uma grande desproporção entre largura e altura terão sua elongação mais próxima de 1. Acreditamos que essa característica tenha se correlacionado bem com as classes de dígitos a serem classificados, em especial com o dígito '1', característico por sua grande elongação.

4) Convexidade: A convexidade denota quão convexo um contorno é. Ela é expressa por meio da *razão entre o perímetro da borda convexa de um contorno e o perímetro de um contorno*. [1] A construção da borda convexa dos contornos foi feita por meio do pacote *opencv-python*. Acreditamos que essa característica tenha se correlacionado relativamente bem apenas com algumas das classes de dígitos estudadas, como '0' e '8', que possuem grande convexidade.

5) Solidez: A solidez é outra medida utilizada para descrever quão convexo um contorno é. Ao contrário da Convexidade, que compara os perímetros de um contorno e de sua borda convexa, a Solidez compara as áreas de um contorno e de sua borda convexa. Dessa forma, podemos escrever a Solidez como sendo *a razão entre a área de um contorno e a área de sua borda convexa*. [1] Acreditamos que essa característica se correlacione bem com certas classes de dígitos, mas que ela possua um certo grau de dependência em relação à Convexidade, tornando os ganhos de acurácia obtidos com o seu uso diminutos, dado que já estávamos utilizando a Convexidade em nossas características.

6) Ângulo do eixo de menor inércia (ALI): O Ângulo do eixo de menor inércia (ALI) pode ser descrito como sendo o ângulo θ que a reta que melhor ajusta os pontos de um contorno, em termos de distâncias quadráticas, possui em relação ao eixo das abscissas (eixo x). [1] Essa característica foi uma das mais custosas, em relação ao tempo de execução do programa, de ser implementada, pois seu cálculo requer o percorrimento de todos os pixels da imagem. Acreditamos que, devido a alta variabilidade na inclinação de dígitos manuscritos, essa característica não tenha se correlacionado tão bem com as classes de dígitos estudadas.

IV. Análise de resultados

A. Algoritmo LDA

1) Análise teórica: O método da *Linear Discriminant Analysis* procura encontrar uma ou mais linhas que separem as classes estudadas. Para fazer isso ele assume que as classes seguem distribuição normal e com variâncias iguais. Note que esse algoritmo, por seguir uma separação linear, não é muito flexível. Isso é uma vantagem caso o que foi assumido pelo método for cumprido, porém, caso não seja cumprido, ele não irá possuir um bom nível de acurácia.

2) Resultados observados: Conseguimos alcançar acurácia de 61,32% com o método do LDA e notamos que a maioria de nossas características possuem variâncias diferentes, exceto o número de Euler. Isso fez com que o número de Euler fizesse a maior diferença. Isso também não permitiu que nosso algoritmo funcionasse muito bem. A menor precisão foi com o número 5 de 30% e a maior foi com o número 1 de 98%.

3) Matriz de confusão obtida:

-	0	1	2	3	4	5	6	7	8	9
0	843	2	14	46	18	15	2	0	22	18
1	0	1092	1	3	0	1	0	22	6	10
2	5	2	423	175	112	52	85	144	26	8
3	5	1	44	708	32	153	28	28	9	2
4	9	1	110	52	464	122	31	136	10	47
5	2	1	83	342	155	199	27	68	9	6
6	25	5	123	31	59	19	369	28	100	199
7	0	4	36	27	118	48	15	737	1	42
8	49	1	21	60	7	5	137	1	628	65
9	48	3	17	31	42	56	68	7	68	669

B. Algoritmo KNN com K = 3

1) Análise teórica: O método dos *K nearest neighbors* é um método baseado em observar quais são as K classes mais próximas por distância (geralmente euclideana) das características. Esse algoritmo é mais flexível que o LDA por poder separar as observações com mais graus de liberdade que uma análise linear, especialmente no caso em que K = 1. Este é o caso mais flexível de nosso trabalho, com K = 3. Como ele não assume coisa alguma acerca de suas entradas, ele pode funcionar bem com as características que escolhemos.

2) Resultados observados: A acurácia foi menor que a do algoritmo de LDA com 60,14%. Provavelmente, o algoritmo foi flexível demais, gerando uma função que se ajustou de forma excessiva aos pontos fornecidos em nossos exemplos de treinamento (*overfitting*). A classe com maior precisão foi a do número 1 com 94% e a de pior precisão foi o número 5 com 33%.

3) Matriz de confusão obtida:

-	0	1	2	3	4	5	6	7	8	9
0	887	0	6	38	8	11	3	1	18	8
1	2	1108	8	2	0	5	2	5	2	1
2	8	19	536	116	124	50	84	78	12	5
3	32	1	168	556	48	161	20	11	8	5
4	5	6	197	105	417	67	36	103	7	39
5	28	10	147	293	136	203	18	36	11	10
6	10	6	188	41	66	29	419	18	48	133
7	0	22	130	43	149	41	27	611	1	4
8	26	1	73	38	24	11	79	2	665	55
9	16	4	32	49	59	33	146	12	46	612

C. Algoritmo KNN com K = 7

1) Análise teórica: Este é o algoritmo de K vizinhos mais próximos consultando os primeiros 7 vizinhos. Este algoritmo é menos flexível que o KNN com K = 3, mas ainda é mais flexível que o LDA. Acreditamos que o fato do algoritmo não assumir coisa alguma acerca de suas entradas e do algoritmo ser menos flexível que o algoritmo KNN com K = 3 contribuíram para o aumento na acurácia observado.

2) Resultados observados: A taxa de acurácia observada foi de 63,25%. Esta acurácia é maior que a do LDA e KNN com K = 3. A classe com maior precisão observada foi o número 1 com 98% e a menor foi o número 5 com 33%, novamente.

3) Matriz de confusão obtida:

-	0	1	2	3	4	5	6	7	8	9
0	883	2	2	45	7	13	3	0	17	8
1	1	1108	4	0	2	5	3	9	2	1
2	4	1	514	117	127	44	110	91	17	7
3	10	0	108	628	47	157	21	19	16	4
4	0	2	138	80	495	80	30	107	11	39
5	10	4	92	355	137	199	28	44	15	8
6	6	3	153	28	53	19	477	29	57	133
7	0	11	87	36	174	36	16	660	2	6
8	21	1	41	19	16	10	98	2	716	50
9	9	2	21	39	41	38	142	12	60	645

D. Algoritmo KNN com K = 11

1) Análise teórica: Este é o algoritmo de K vizinhos próximos consultando os primeiros 11 vizinhos. Este algoritmo é menos flexível que o KNN com K = 3 e K =7, mas é mais flexível que o LDA. Acreditamos que o fato do algoritmo não assumir coisa alguma acerca de suas entradas e do algoritmo ser menos flexível que o algoritmo KNN com K = 7 contribuíram para o aumento na acurácia observado.

2) Resultados observados: A acurácia foi de 64,48%, o melhor resultado entre todos os algoritmos estudados nesse trabalho. O número com maior precisão foi 1 com 98% e a menor precisão foi para o número 5 com 34%.

3) Matriz de confusão obtida:

-	0	1	2	3	4	5	6	7	8	9
0	872	2	3	51	7	18	3	0	16	8
1	2	1105	4	0	0	5	4	9	3	3
2	2	0	521	130	112	41	107	95	18	6
3	4	0	106	649	45	150	18	15	19	4
4	0	1	123	65	515	81	38	110	13	36
5	6	4	83	352	148	203	31	44	14	7
6	4	3	142	24	56	20	490	26	57	136
7	0	10	71	28	167	35	19	690	4	4
8	20	1	39	16	13	5	100	2	726	52
9	8	2	19	40	45	35	107	11	65	677

V. Conclusão

No quesito de performance, todos os algoritmos atingiram o objetivo de forma mediana (acurácia entre 59% e 65%), e nenhum deles atingiu uma taxa de acurácia que seria capaz de se aproximar de (e muito menos substituir) um ser humano. Notavelmente, todos eles tiveram resultados parecidos, com pouca melhora de acurácia entre os algoritmos estudados. Isso indica que o problema não foi o grau de liberdade utilizado para modelar os dados, mas sim o uso de características que, em sua maioria, não se correlacionavam bem com as classes estudadas. Talvez a escolha de características que fossem menos dependentes entre si e que se correlacionem melhor com as classes estudadas gerassem melhores resultados para um ou mais algoritmos analisados. Também é possível que a utilização dessas características em outros tipos de algoritmos, por exemplo um algoritmo com um grau de liberdade médio gerasse melhoras na taxa de acurácia observada.

O algoritmo que obteve a melhor taxa de acurácia foi o KNN com K = 11. Acreditamos que a flexibilidade desse algoritmo (nem tão rígido quanto o LDA, mas nem tão solto

como os KNNs com K = 3 e K = 7) geraram uma aproximação melhor da função representativa dos dados de treinamento.

Em uma análise geral dos algoritmos, o dígito ’1’ foi o que obteve maior precisão. Isso está conforme os resultados esperados, pois reflete a distinção desse dígito em termos de características observadas como elongação e retangularidade. Já o dígito com a pior precisão foi o ’5’, refletindo, talvez, a enorme semelhança desse dígito com os dígitos ’3’ e ’4’ em relação à medidas de contorno.

REFERÊNCIAS

[1] Mingqiang Yang, Kidiyo Kpalma, Joseph Ronsin. A Survey of Shape Feature Extraction Techniques. Peng-Yeng Yin. Pattern Recognition, IN-TECH, pp.43-90, 2008. <hal-00446037>