Rio de Janeiro, 11 de Junho de 2014

Relatório do Terceiro Trabalho

Aprendizado de Máquina

Carlos Mattoso e Leonardo Kaplan

Processo de Extração de Características

A extração de características é feita pela função **extract_features** implementada no arquivo **feature_extractor.py** e passa pelas seguintes etapas:

- Primeiramente, a imagem é convertida para o espaço de cores **HLS**, sendo a luminosidade reduzida em **75%**. Isto, para a grande maioria dos casos, possibilita uma boa segmentação da mão com relação ao fundo (utilizando-se a função de ***threshold*** com ***binarization*** e ***otsu***).
- A partir daí, aplicamos 3 etapas de **dilations** seguidas de 2 de **erosions.**
- Extraindo-se então os contornos, **p**ode-se obter o maior contorno. Este, para grande parte dos casos, acaba por segmentar bem a mão. Com isso fazem-se as extrações dos seguintes atributos:

    a. **contour_area_ratio** - fração da área do contorno com relação a àrea do convex hull
    b. **rectangle_area_ratio** - fração da área do retângulo com relação a área do convex hull
    c. **aspect_ratio** - aspect ratio calculada com base no retângulo mínimo englobante
    d. **convex_hull_perimeter** - perímetro do convex hull
    e. **ellipse_angle** - ângulo da elipse fitada
    f. ***ellipse_center_x*** - coordenada x do centro da elipse fitada
    g. ***ellipse_center_y*** - coordenada y do centro da elipse fitada
    h. ***ellipse_major_axis*** - eixo maior da elipse fitada
    i. ***ellipse_minor_axis*** - eixo menor da elipse fitada

j. ***fixpt_depth*** - "fixed-point approximation (with 8 fractional bits) of the distance between the farthest contour point and the hull."
k. **centroid_x** - coordenada x do centro do hull
l. **centroid_y** - coordenada y do centro do hull
m. **hu_moment_1**
n. **hu_moment_2**
o. **hu_moment_3**
p. **hu_moment_4**
q. **hu_moment_5**
r. **hu_moment_6**
s. **hu_moment_7**

Em um primeiro momento, como exibido na apresentação, utilizou-se um conjunto menor de ***features***. Ao ampliar-se tal conjunto (as características acima em itálico ***[f-j]***) não houve expressiva melhoria no resultado dos algoritmos. Optamos, então, por testar mais a fundo os algoritmos apenas com o **conjunto básico**.

O arquivo **traverse_examples.py** passa por todas as pastas dos exemplos e, utilizando a função **extract_features,** coleta em cada pasta um conjunto de características concatenadas extraídas de uma sequência de **{10,17,35}** imagens o mais espaçadas possível. Cada exemplo de treinamento é resultado da concatenação descrita previamente.

Resultados

Abaixo disponibilizamos uma série de tabelas com os resultados de execução de cada algoritmo no **Weka** (cv 10 folds). Todos os resultados obtidos podem ser encontrados na pasta **logs**. Os melhores resultados para cada algoritmo foram **destacados**.

Atributos Básicos (logs/basic_features/35_images) - 35 Imagens

900 instâncias - 491 atributos

KNN (lazyIBK)

Arquivo	K= | distance func.	Taxa de Reconhecimento (%)	Tempo de Execução (discretizado)
knn1.log	**1** | *euclidean*	69.7778	Muito rápido
knn2.log	**2** | *euclidean*	65.4444	Muito rápido
knn3.log	**3** | *euclidean*	64.8889	Muito rápido
knn4.log	**4** | *euclidean*	67	Muito rápido
knn1_manhattan.log	**1** |manhattan	**73.1111**	Muito rápido
knn2_manhattan.log	**2** | manhattan	71.4444	Muito rápido

SVM (libSVM)

Arquivo	Kernel Usado	Taxa de Reconhecimento (%)	Tempo de Execução (discretizado)
svm1.log	Radial	13.1111	Rápido
svm2.log	Polinomial (grau 3)	**55.5556**	Lento
svm3.log	Sigmoid	11.1111	Médio

Árvore de Decisão (J48)

Arquivo	Confidence Factor	Taxa de Reconhecimento (%)	Tempo de Execução (discretizado)
decision_tree1.log	0.25	53.2222	Médio
decision_tree2.log	0.4	53.3333	Médio
decision_tree3.log	0.1	**53.6667**	Médio

Rede Neural (MultilayerPerceptron)

Arquivo	Taxa de Reconhecimento (%)	Tempo de Execução (discretizado)
nn1.log	**52.1111**	Muito lento

RANK : **KNN > SVM > Árvore de Decisão > Rede Neural**

Atributos Básicos (logs/basic_features/17_images) - 17 Imagens

900 instâncias - 239 atributos

KNN (lazyIBK)

Arquivo	K= | distance func.	Taxa de Reconhecimento (%)	Tempo de Execução (discretizado)
knn1.log	**1** | *euclidean*	72.1111	Muito rápido
knn2.log	**2** | *euclidean*	70.4444	Muito rápido
knn3.log	**3** | *euclidean*	71.1111	Muito rápido
knn4.log	**4** | *euclidean*	70.7778	Muito rápido
knn1_manhattan.log	**1** |manhattan	**77.1111**	Muito rápido
knn2_manhattan.log	**2** | manhattan	75.6667	Muito rápido

SVM (libSVM)

Arquivo	Kernel Usado	Taxa de Reconhecimento (%)	Tempo de Execução (discretizado)
svm1.log	Radial	13.1111	Rápido
svm2.log	Polinomial (grau 3)	**52.7778**	Lento
svm3.log	Sigmoid	11.1111	Médio
svm2_normalized.log	Polinomial (grau 3)	31	Médio

A normalização piorou os resultados obtidos, afetando principalmente a classificação de exemplos das classes de 5 até 8.

Árvore de Decisão (J48)

Arquivo	Confidence Factor	Taxa de Reconhecimento (%)	Tempo de Execução (discretizado)
decision_tree1.log	0.25	**62.6667**	Médio
decision_tree2.log	0.4	**62.6667**	Médio
decision_tree3.log	0.1	**62.6667**	Médio

Rede Neural (MultilayerPerceptron)

Arquivo	Taxa de Reconhecimento (%)	Tempo de Execução (discretizado)
nn1.log	**75.8889**	Muito lento

RANK : **KNN** > **Rede Neural > Árvore de Decisão > SVM**

Atributos Básicos (logs/basic_features/10_images) - 10 Imagens

900 instâncias - 141 atributos

KNN (lazyIBK)

Arquivo	K= | distance func.	Taxa de Reconhecimento (%)	Tempo de Execução (discretizado)
knn1.log	1 | *euclidean*	74.8889	Muito rápido
knn2.log	2 | *euclidean*	71.3333	Muito rápido
knn3.log	3 | *euclidean*	71.5556	Muito rápido
knn4.log	4 | *euclidean*	70.8889	Muito rápido
knn1_manhattan.log	1 |manhattan	**79.4444**	Muito rápido
knn2_manhattan.log	2 | manhattan	78.4444	Muito rápido

KNN (implementado por nós - segmentação 70/30)

Arquivo	K= | distance func.	Taxa de Reconhecimento [max|mean] (%)	Tempo de Execução (discretizado)
knn1_implemented_euclidean.log	1 | euclidean	**74.81481 | 72.07407**	Muito lento
knn1_implemented_manhattan.log	1| manhattan	**81.85185 | 76.7777**	Muito lento

Os resultados aqui encontrados ficaram próximos do que obtivemos no **Weka** para mesma entrada, como observado na tabela anterior a esta.

SVM (libSVM)

Arquivo	Kernel Usado	Taxa de Reconhecimento (%)	Tempo de Execução (discretizado)
svm1.log	Radial	14	Rápido
svm2.log	Polinomial (grau 3)	**56.5556**	Lento
svm3.log	Sigmoid	11.1111	Médio
svm2_normalized.log	Polinomial (grau 3)	39.8889	Médio

A normalização piorou os resultados obtidos, novamente com uma perda maior na classificação de exemplos das classes de 5 até 8.

Árvore de Decisão (J48)

Arquivo	Confidence Factor	Taxa de Reconhecimento (%)	Tempo de Execução (discretizado)
decision_tree1.log	0.25	**61.7778**	Médio
decision_tree2.log	0.4	**61.7778**	Médio
decision_tree3.log	0.1	61.4444	Médio

Rede Neural (MultilayerPerceptron)

Arquivo	Taxa de Reconhecimento (%)	Tempo de Execução (discretizado)
nn1.log	76	Muito lento

RANK : **KNN** > **Rede Neural > Árvore de Decisão > SVM**

Atributos Extras (logs/extra_features) - 35 Imagens

900 instâncias - 666 atributos

KNN (lazyIBK)

Arquivo	K= | distance func.	Taxa de Reconhecimento (%)	Tempo de Execução (discretizado)
knn_extra1.log	1 | euclidean	69.2222	Muito rápido
knn_extra2.log	1 | manhattan	72.5556	Muito rápido

KNN (implementado por nós - segmentação 70/30)

Arquivo	K= | distance func.	Taxa de Reconhecimento [max] (%)	Tempo de Execução (discretizado)
knn1_implemented_euclidean.log	1 | euclidean	69.62963	Muito lento
knn1_implemented_manhattan.log	1 | manhattan	73.7037	Muito lento

SVM (libSVM)

Arquivo	Kernel Usado	Taxa de Reconhecimento (%)	Tempo de Execução (discretizado)
svm_extra1.log	Polinomial (grau 3)	53.2222	Rápido

Árvore de Decisão (J48)

Arquivo	Confidence Factor (C)	Taxa de Reconhecimento (%)	Tempo de Execução (discretizado)
decision_tree_extra1.log	0.25	55.7778	Rápido

RANK: KNN > SVM > Árvore de Decisão

Em razão da inexpressiva mudança nas taxas de reconhecimento gerais, não rodamos a rede neural, que ficaria absurdamente lenta para um conjunto de características tão grande.

Comentários

Observa-se que o emprego de imagens mais espaçadas para a composição dos exemplos de treinamento facilitou a diferenciação de diferentes classes para praticamente todos os algoritmos. Destaca-se em especial a expressiva melhoria da **rede neural**, que passou de **~50%** quando baseado em **35** imagens para **76%** no caso de **10** imagens. Além disso, procuramos fazer uma normalização da entrada ao alimentar a **SVM**, mas isto prejudicou o poder preditivo do algoritmo.

Chegamos a rodar ainda **KNN** *(k=1, manhattan distance)* para um conjunto de sequências de 5 imagens, sendo disponibilizado também este *log*. Contudo, obteve-se um resultado de **78%,** inferior ao obtido para o conjunto de sequências de 10 imagens, que foi **79.4%**. Assim, encerramos a análise neste conjunto.

Uma análise das matrizes de confusão disponibilizadas nos arquivos de *log* revela que a classe com maiores problemas de classificação foi a **5ª**. Aplicando-se o algoritmo de extração de características adotado observa-se que ele tem um desempenho fraco nas imagens de exemplo. Isto se dá em razão da distribuição de **luminosidade** nas mesmas, a propriedade principal em que se baseia nosso algoritmo para detecção de mãos. Neste caso, ele fica "confuso" e demarca uma região maior do que deveria, embora funcione muito bem para a maioria dos demais exemplos.

Uma modificação do processo de extração que aplicasse, além da redução de luminosidade um filtro de pele, o tornaria mais robusto para casos como da classe 5 e talvez melhorasse seu funcionamento para as outras. Experimentamos algumas formas de fazer isso durante o desenvolvimento, mas não atingimos um resultado muito satisfatório.

Enfim, como todos as análises o **KNN** demonstrou maior habilidade preditiva, desenvolvemos sua versão simples em **R**. O algoritmo está altamente comentado e é de clara compreensão, embora não seja muito rápido. Contudo, como pode-se ver nos resultados do algoritmo apresentados na análise (para os casos de 10-basic e 35-extras), este chega em valores próximos daqueles encontrados pelo **Weka.**