Inteligência Artificial
Reconhecimento Facial

Mauri S. Moretti Junior[1], Diego S. Cintra[1]

[1]Faculdade de Computação – Universidade Federal de Mato Grosso do Sul (UFMS)
Caixa Postal 549 – 79.070-900 – Campo Grande – MS – Brasil

{mauri_junior18,diego_2337}@hotmail.com

Abstract. *It was requested to us the implementation of a system capable of recognizing the facial expression of an individual and classify it in either "happy" or "sad", with the program written in Python. For such, it is necessary the understanding of the artificial intelligence algorithm's behavior which automatically generate knowledge representation: Decision trees, KNN and neural networks. It is also necessary, for completion of the task, to utilize the "scikit-learn" (for the aforementioned algorithms) and "OpenCV" (for webcam image capture) libraries. These requests aim to broaden the knowledge about the supervised algorithms behavior, their precision and the best scenario for their use.*

Resumo. *Foi-nos requisitado a implementação de um sistema capaz de reconhecer a expressão facial de um indivíduo e classificá-la em "feliz" ou "triste", com o programa escrito em Python. Para isso, é necessário a compreensão do funcionamento dos algoritmos de inteligência artificial que geram automaticamente a representação do conhecimento: Árvores de decisão, KNN e redes neurais. Também é necessário, para completar a tarefa, utilizar das bibliotecas "scikit-learn" (para os algoritmos previamente mencionados) e "OpenCV" (para captura de imagens via webcam). Essas requisições visam ampliar os conhecimentos sobre o funcionamento dos algoritmos supervisionados, a precisão desses e o melhor cenário para sua utilização.*

Representação dos dados de entrada

Para a primeira parte do trabalho, foi-nos solicitado um conjunto de dados com, no mínimo, 300 fotos para o conjunto de treinamento. Essas deveriam ser de tamanho 64x64 pixels, a fim de, ao serem transpostas para o programa, se transformassem em 4096 colunas, mais uma representando a classe correspondente. As imagens foram representadas como sendo do tipo "ndarray", que é bastante utilizado pela biblioteca "OpenCV" para processamento de imagem. Primeiramente, fizemos um programa em Python que tirava uma foto convencional (somente a face), porém em uma resolução maior. Depois, outro programa abria as fotos tiradas (em formato "iplimage", que representa a imagem em somente um canal) e diminuía sua resolução para 64x64 pixels, utilizando as funções da descrição do trabalho. Depois, a foto redimensionada (em formato "iplimage") era transformada no formato "ndarray"; porém, esse vetor é multidimensional, portanto era necessário convertê-lo para um vetor com 4096 elementos (64 vezes 64). Utilizando a função "np.ravel(imagem)", isso foi feito. Por último, era necessário salvar a imagem em um arquivo ".csv", e para isso, utilizamos o seguinte trecho de código:

```
ffile = file(“facesall.csv”, “a”)

np.savetxt(ffile, np.atleast_2d(imagem), fmt = '%d', delimiter = ';', newline = “;1 \n”)

ffile.close()
```

Vamos explicar algumas das funções utilizadas nesse trecho. A função “file()” é a padrão para abertura de arquivos durante a execução do programa. A função “np.savetxt” recebe como parâmetros o arquivo a ser escrito a imagem (no caso, ffile), a imagem propriamente dita, o formato (fmt = '%d'), que no caso foi especificado para inteiros, um delimitador (dado por ponto e vírgula) e o que fazer ao final da linha (aqui, adicionamos uma nova coluna representando a classe – tanto 0 quanto 1, dependendo da imagem – e inserimos o '\n' como quebra de linha). É importante notar que, para o parâmetro da imagem, utilizamos a função “np.atleast_2d(imagem)”, que escreve todo o vetor em uma única linha (ao invés de uma única coluna, como ele faz por padrão). Por último, “ffile.close()” termina a escrita no arquivo. Caso haja alguma dúvida em relação a execução desses processos, foram incluídos os arquivos “photos.py” e “resize_save.py” a esse relatório, que especificam os passos explicados previamente.

Durante a execução do programa “MLearning.py”, módulo que faz parte do programa “Recognition.py”, seu construtor abre todas as imagens salvas no arquivo previamente mencionado através do seguinte trecho de código:

```
ffile = file(“facesall.csv”, “r”)

dataset = np.loadtxt(ffile, dtype = int, delimiter = ';', ndmin = 2)
```

A função “np.loadtxt” funciona de maneira similar àquela que salva as imagens em um arquivo, possuindo parâmetros quase idênticos. O parâmetro “ndmin” especifica qual será o tamanho da dimensão da variável a receber os arquivos, e o “dtype” especifica o tipo da variável que recebe o método. Depois disso, para poder utilizar esses dados nos algoritmos fornecidos pela biblioteca “scikit-learn”, deve-se especificar o conjunto de treinamento e a classe correspondente a cada um dos itens, o que foi feito através das seguinte instruções:

```
x = dataset[:,0:4096]

y = dataset[:,4096]
```

Dessa maneira, separamos todos os itens da última coluna como sendo as classes específicas de cada um dos itens do conjunto de dados, que estão armazenados na variável “x”. Finalmente, através do método “fit(x, y)”, carregamos todos os dados necessários para a execução dos algoritmos.

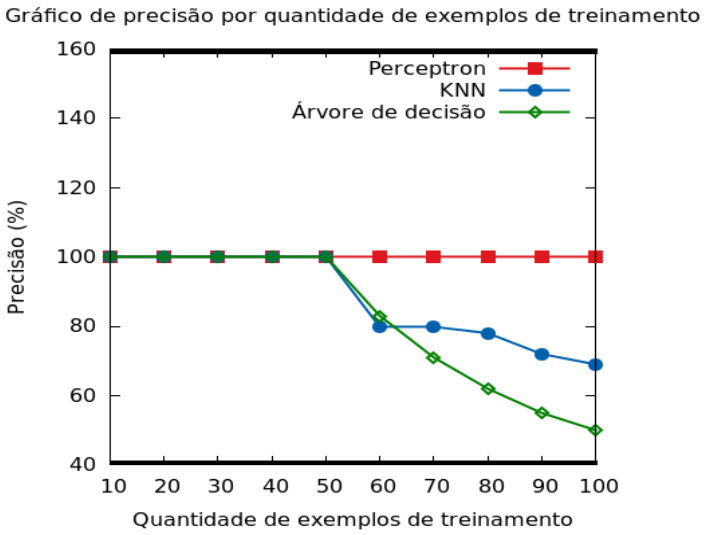
É importante também especificar o funcionamento dos métodos da biblioteca “scikit-learn”, que implementam os algoritmos de aprendizado de máquina. Para o KNN (*K Nearest Neighbors*), dentre os parâmetros utilizados, temos o *n_neighbors*, que especifica a quantidade de k vizinhos a ser utilizada, *weights*, que define se o algoritmo executado será o KNN sem pesos ou com pesos (definido por “distance”), *radius*, que define o tamanho da indexação (caso ela seja utilizada), *algorithm*, que define qual será a estrutura de dados utilizada para indexar os atributos e *p*, que define qual a métrica de distância a ser utilizada (ou euclidiana, quando p for igual a 2, ou de *manhattan*, quando p for igual a 1).

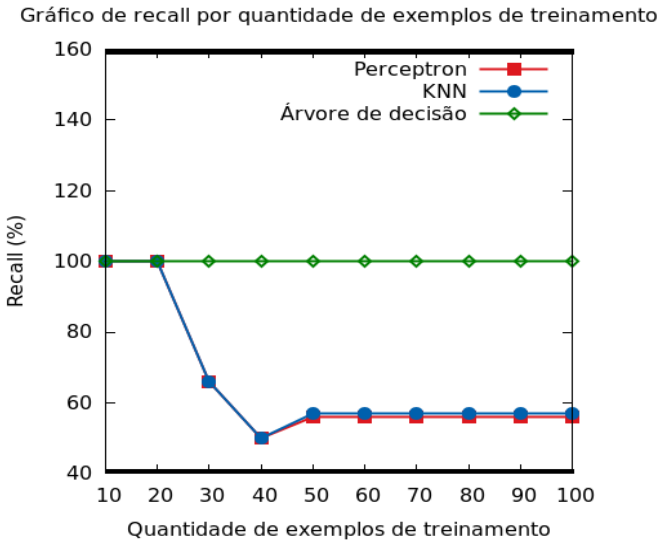
Para o Perceptron, utilizamos o método “SGDClassifier”, que possui o parâmetro *loss*: ao defini-lo como “perceptron”, esse passa a ser equivalente ao método “Perceptron”. Para o restante dos parâmetros, temos *alpha*, que define a taxa de aprendizado na regressão, *penalty*, que define a norma de regularização e *shuffle*, que permuta os atributos do conjunto de treinamento antes de aplicar a regressão.

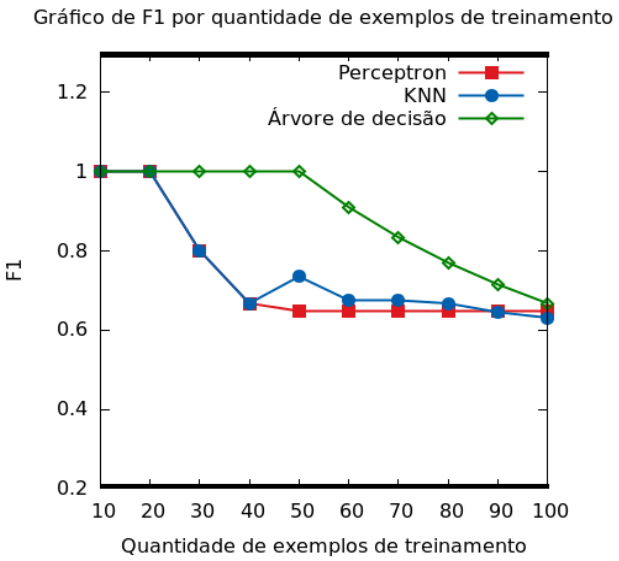
Por último, na árvore de decisão, utiliza-se o método “DecisionTreeClassifier”, consistindo dos parâmetros *criterion*, definindo qual será a medida divisória da árvore (utilizando “entropy”, usa-se o ganho de informação), *max_depth*, que define a profundidade máxima de uma árvore (fazendo a poda para evitar overfitting) e *min_samples_split*, definindo o mínimo de exemplos para poder dividir um nó.

Gráficos de precisão, *recall* e F1

Como requisitado na descrição do trabalho, segue abaixo os gráficos de precisão, recall e F1 por quantidade de exemplos dos três algoritmos, respectivamente:







É importante lembrar que, assim como também foi requisitado, foram selecionados duzentos exemplos do conjunto de treinamento para serem o treino, enquanto os outros 100 serviram de casos de teste.

Como podemos ver, o algoritmo que apresentou resultados mais estáveis durante a classificação dos casos de teste foi o Perceptron, apresentando uma precisão de 100% ao longo do aumento da quantidade de exemplos (a precisão mostra a porcentagem de instâncias classificadas corretamente como positivas dentre todas as que foram classificadas como positivas) e uma certa estabilidade no gráfico de média harmônica entre precisão e sensitividade (F1); seu recall (que é a porcentagem de instâncias classificadas corretamente como positivas dentre todas as que realmente são positivas), apesar de se estabilizar após uma certa quantidade de exemplos, apresentou valores indesejados durante a classificação com poucos exemplos de treinamento.

Conclusão

Após agregar conhecimentos a respeito do funcionamento dos algoritmos de aprendizado de máquina, a implementação e operação desse trabalho nos possibilitou compreender mais ainda como o aprendizado de máquina é dado. Fomos capazes de entender e manusear as bibliotecas necessárias para o funcionamento do programa e também a complexidade que existe em classificar casos de teste, visto que em uma boa parte das situações, a classificação é feita incorretamente, variando muito de acordo com a luz ambiente e o local onde a face está centrada. Apesar dos arquivos não executarem de maneira ideal, o importante foi o aprendizado adquirido ao longo da implementação do programa.