Fundamentos de Sistemas Inteligentes
Projeto 1 - Classificação de dígitos manuscritos

Hugo Nascimento Fonseca
Universidade de Brasília - UnB
Engenharia de Computação
16/0008166
hugonfonseca@hotmail.com

José Luiz Gomes Nogueira
Universidade de Brasília - UnB
Engenharia de Computação
16/0032458
joseluizgnogueira@live.com

***Resumo*—Este artigo tem como objetivo apresentar o relatório do Projeto 1 da disciplina de Fundamentos de Sistemas Inteligentes, semestre 2019/1 - Prof. Díbio Borges, na Universidade de Brasília.**

I. Introdução

O projeto consiste em comparar dois algoritmos de classificação: o LDA, *Linear Discriminant Analysis*, e o K-nn, *K- nearest neighbors*. Para tal, aplicaremos ambos na classificação de dígitos manuscritos obtidos do "MNIST dataset", sendo 60 mil para o set de treinamento e 10 mil para o set de teste/validação.

II. Fundamentação teórica

A. Descrição dos algoritmos

1) LDA - Linear Discriminant Analysis: O algoritmo LDA *Linear Discriminant Analysis* é um modelo simples em termos de preparação e aplicação, a representação é direta. Basicamente consiste de propriedades estatísticas dos dados, que são calculados separadamente para cada classe, essas propriedades estatísticas são usadas para gerar um modelo, que será usado para fazer as predições, o modelo usa o teorema de Bayes para estimar as probabilidades.

- Características
  1) O LDA assume que os dados são amostrados aleatoriamente, sem viés, ou seja, seus dados respeitam uma distribuição Gaussiana.
  2) O LDA assume que cada uma das *features* tem a mesma variância. Logo, quase sempre é uma boa ideia fazer a normalização dos dados antes de aplicar o LDA.
  3) *Features* colineares diminuem a acurácia do modelo.

2) K-nn - K nearest neighbors: O algoritmo *K nearest neighbors* (K-nn) é um dos algoritmos de classificação mais simples e ao mesmo tempo um dos mais utilizados. Ele se baseia na similaridade de características (*features*), o quão distante uma *feature* está do conjunto de treinamento determina como um dado será classificado. Um dado é classificado com base no voto majoritário dos seus vizinhos, ele será colocado como a classe mais comum dos *k* vizinhos ao redor dele, dessa forma, é importante utilizar um valor ímpar de vizinhos para evitar possíveis empates.

- Características:
  1) Não-paramétrico: não faz suposições sobre os dados, isso torna ele bom para classificação de dados não lineares e/ou dados dos quais não se tem muita informação.
  2) Preguiçoso: não possui nenhum pré-modelo para classificação, isso implica que todas as instâncias de treinamento permanençam salvas em memória.
  3) Flexibilidade da classificação: quanto maior o valor do K, menos flexível.

III. Projeto

A. Software

Para executar essa tarefa, escrevemos um projeto em Python utilizando os seguintes componentes, com suas respectivas versões, abaixo:

- Python 3.6.7
- NumPy 1.16.2
- Matplotlib 3.0.3
- Pandas 0.24.2
- OpenCV 4.0.0.21
- sklearn 0.20.3

B. Features

Inicialmente, levantamos e implementamos, utilizando a OpenCV [2], algumas *features* para serem calculadas em cada imagem [1], ou seja, basicamente, estaremos medindo a similaridade entre dois formatos baseados nos resultados das features. Para que isso seja eficiente, a *feature* precisa satisfazer os critérios abaixo:

- Os descritores (*features*) devem ser tão completos quanto possível para poder representar o conteúdo da informação.
- O custo de computação dos descritores deve ser o menor possível, de forma a manter um bom tempo de execução.
- Os descritores devem poder ser representados e armazenados de forma compacta, o vetor de *features* não deve ser muito grande.

1) Centro de massa (Centroid)
   O centro de massa (centroid) de uma forma é a média

aritmética de todos os pontos pertencentes a forma, isto é:

$$c = \frac{1}{n}\sum_{i=1}^{n} P_i \tag{1}$$

2) Eixo de menor inércia (Axis of least inertia)
O eixo de menor inercia é usado para prover informação sobre a orientação relativa do objeto com relação ao plano da imagem, ele pode ser descrito como a linha que requer a menor quantidade de energia para rotacionar o objeto.

3) Excentricidade (Eccentricity)
A excentricidade é uma medida de proporção da imagem, é a razão entre o comprimento do maior eixo da forma pelo menor eixo. Neste projeto calculamos utilizando a técnica do menor retângulo delimitador.

4) Circularidade (Circularity)
A circularidade é uma representação de quanto um formato é similar a um circulo, uma das formas que pode ser usada para calcular é a seguinte: é a razão entre a área da forma e a do circulo que possui o mesmo perímetro.

$$Circularity = \frac{A_S}{A_C}, \tag{2}$$

5) Retangularidade (Rectangularity)
A retangularidade é uma métrica que representa o quanto retangular é a forma, ou seja, o quanto ela preenche o menor retângulo delimitador.

$$Rectangularity = \frac{A_S}{A_R} \tag{3}$$

6) Convexidade (Convexity)
A convexidade é definida como a razão entre o perímetro do casco convexo da forma e o perímetro da forma. O casco convexo de uma forma é dado pela menor região convexa que inclui a forma totalmente dentro dela.

$$Convexity = \frac{O_{ConvexHull}}{O} \tag{4}$$

7) Solidez (Solidity)
A solidez é uma métrica para descrever a extensão na qual uma forma é côncava ou convexa, é definida pela razão entre a área da forma e a área do casco convexo.

$$Solidity = \frac{A_S}{A_{CH}} \tag{5}$$

IV. Avaliação/Resultados

A. Máquina utilizada nos testes

- Processador: i3 - 3217u - 1.8GHz
- Memória: 8GB - DDR3 - 1066MHz
- Sistema Operacional: Ubuntu 18.04 LTS

1) LDA - Linear Discriminant Analysis: Para o LDA, foi realizado a classificação utilizando tanto combinações de *features* como entrada, quanto os próprios pixels da imagem.

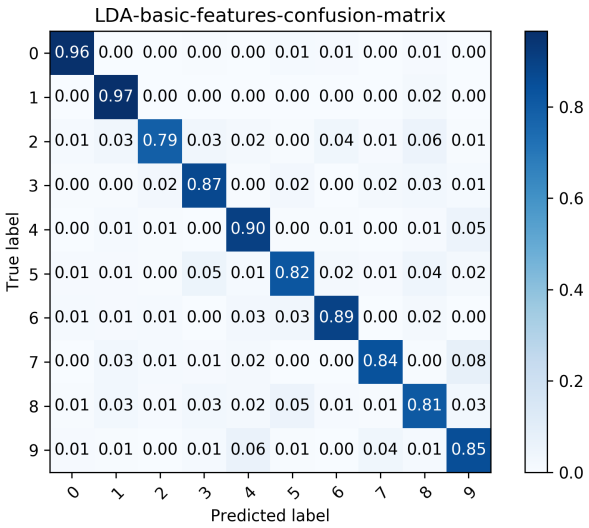

Figura 1: Matriz de confusão para LDA, tendo como entrada todos os pixels

Utilizando os próprios pixels da imagem, obtivemos a matriz de confusão da figura 1, onde podemos observar uma alta acurácia (aproximadamente 0,873), entretanto o vetor que descreve cada item é grande demais (784 inteiros de 0 a 255) levando à um alto tempo de treinamento, porém ainda assim mantém baixo o tempo de predição.
Além do tamanho deste vetor que descreve cada item ser muito grande, uma rápida análise nos indica que há muitas colinearidades que poderiam ser descartadas, diminuindo assim o tamanho da entrada e consequentemente o tempo. Uma forma de se fazer isso poderia ser, um vetor somente com as coordenadas dos pixels preenchidos (após binarização), realizando um truncamento no tamanho tendo como parâmetro o maior tamanho da maior amostra. Outra forma é extrair informações da imagem, que poderiam descrever unicamente uma classe, informações essas que chamamos de features, as features escolhidas e que serão aqui analisadas estão detalhadas na seção acima.

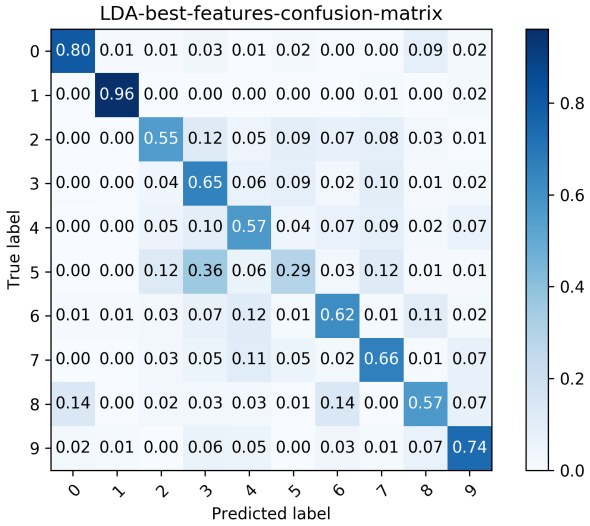

Figura 2: Matriz de confusão para LDA, tendo como entrada todas as features

Utilizando as melhores features, obtivemos a matriz de confusão da figura 2, que obteve uma acurácia abaixo do esperado.
Para descobrir quais das features possuem maior relevância e quais conjuntos de features apresentavam a maior acurácia, foi executado o treino e obtido a acurácia no teste utilizando todas as combinações possíveis features (o arquivo detalhando a acurácia de cada combinação esta disponível nos arquivos do projeto > output > features combination > combinacoes de features). Por meio desta analise constatamos que para o LDA, quanto mais features melhor, uma vez que a própria linearidade da classificação irá ignorar as features colineares do vetor, não sofrendo interferência por permutação das features nem por suas colinearidades.
Contudo, mesmo utilizando todas as features, obtivemos uma baixa acurácia. Levantamos que poderia ser dois os problemas:

1- As features escolhidas não descrevem as classes unicamente muito bem.
2- Uma, ou varias features, dentro da mesma classe, possui grande flutuação.

Olhando os valores de features para algumas imagens, observamos que ela varia muito para a mesma classe, mas descrevem razoável bem a classe quando o numero é legível, logo, para um aumente da acurácia, poderia ser levantado outras features que cubram melhor os dígitos mais deformados.

2) K-nn - K nearest neighbors: Para o K-nn, foi realizado a classificação utilizando tanto combinações de features como entradas, quanto os próprios pixels da imagem.
Para ambos foram realizados o cálculo da curva de erro em função da quantidade de vizinhos, variando os vizinhos entre 1-60.

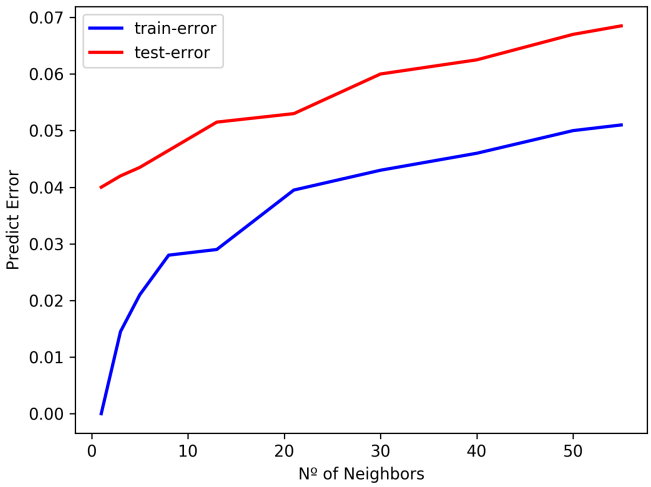

Figura 3: Curva de erro para KNN em função do número de vizinhos - usando todos os pixels como entrada

Como é possível visualizar na figura 3, quanto maior o número de vizinhos maior o erro da predição, isto tendo como entradas os próprios pixels das imagens.
A curva não segue a distribuição esperada de possuir baixo erro em treino e alto erro em teste para valores muito baixos e altos de k, e valores próximos de erro em treino e teste para valores medianos de k. Acreditamos que isso se deve a grande quantidade de informações que descrevem a imagens (784 pixels) e a grande quantidade de colinearidade entre eles. Por meio do gráfico, podemos perceber que os melhor valor de k para obter uma alta acurácia são os menores valores, logo escolhemos para esta amostra o valor de k = 3.

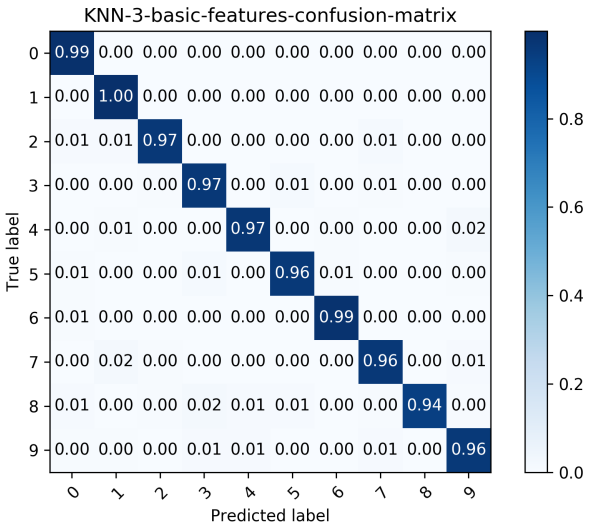

Figura 4: Matriz de confusão para KNN, tendo como entrada todas as features

Como podemos ver em sua matriz de confusão (figura 4) o knn apresenta alta acurácia para todos as classes.
Entretanto, assim como o LDA, o knn apresentou, devido a grande quantidade de informação, grande lentidão tanto no treinamento, e apresentou ainda mais lentidão nos testes.

Assim como LDA utilizamos uma features para descrever a imagem, reduzindo a quantidade de dados de entrada.

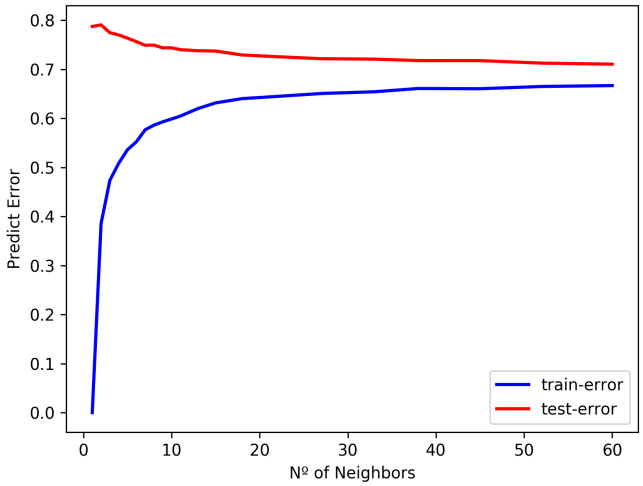

Figura 5: Curva de erro para KNN em função do número de vizinhos - usando todos os features como entrada

Na figura 5 podemos ver a relação de erros na predição em função da quantidade de vizinhos.
Diferentemente do que ocorreu na figura 3, a curva aqui se aproxima mais do esperado onde os valores de vizinhos extremos apresentam aumentam o erro de predição nos teste (aqui não vemos o erro se elevar com k extremo pois não á um k extremo para facilitar a visualização da parte importante da curva).
Através da figura podemos perceber que, para baixos valores de k temos um especialização muito grande no treino e aumento no erro nos testes, já para valores próximos a 60 possuímos um valor de erro mais próximo para ambos e o mais baixo para os testes.
Logo, escolhemos o valor de k=3 e k=60 para analisarmos mais a funda, uma vez que k=3 é um caso de especialização e 60 é um valor proximo do melhor caso.

Novamente, assim como no LDA, surgiu a duvida, será que a permutação e combinação das features interferem no resultado do knn?
A resposta curta é sim. A permutação de fato não faz muita diferença nos resultados, porém a combinação faz, e muita. Por exemplo, somente a features 3 pode fornecer um resultado melhor que utilizando todas as features juntas. Acreditamos que isso se deva ao fato do knn ser sensível a colinearidade e a distribuição dos dados.
Para a analise do knn com os valores de k=3 e k=60, otimizamos os resultados utilizando as features que elevam a acurácia pada cada valor de k (o arquivo detalhando a acurácia de cada combinação esta disponível nos arquivos do projeto > output > features combination > combinacoes de features).

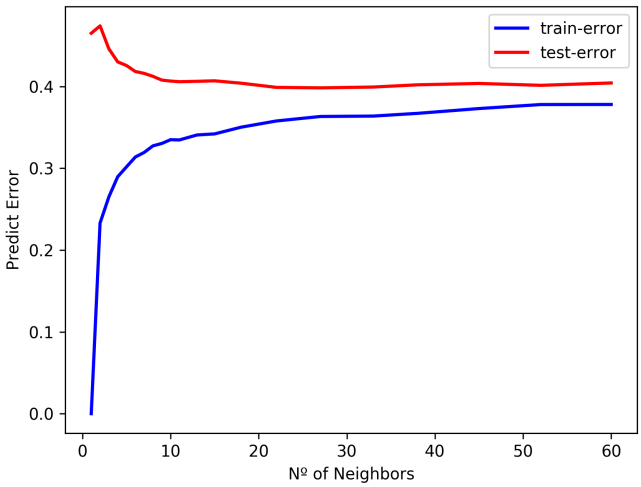

Figura 6: Curva de erro para KNN em função do número de vizinhos - usando features para otimização de k=3 como entrada

Na figura 6 podemos visualizar a curva de erro em função do número de vizinhos novamente, entretanto, agora não mais utilizando todas as features, e sim as features (3,4,5,6,7 descritas na seção acima) que melhoram o caso de k=3.
Podemos observar uma leve quebra na ponta da curva, onde para k=3 a uma queda no valor do erro de predição em teste.

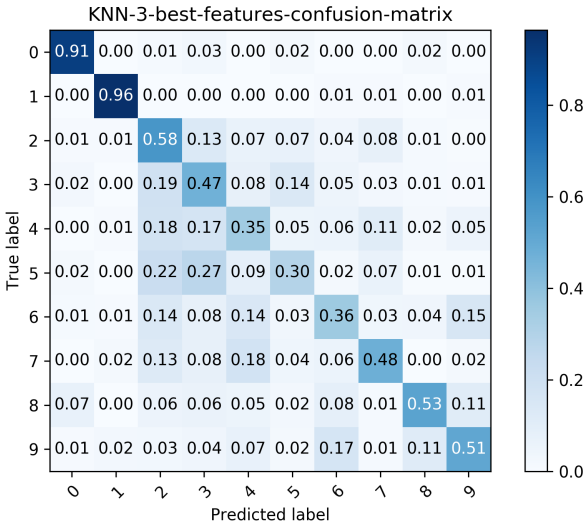

Figura 7: Matriz de confusão para KNN - usando features para otimização de k=3 como entrada

A matriz de confusão para este caso encontra-se na figura 7, onde podemos observar uma leve dispersão, alguns falsos positivos e falsos negativos, e ambos razoavelmente centralizados, nos números 2,3,4 e 5. Acreditamos que isso se deva ao mesmo fator que possa ter levado os valores de acurácia do LDA estarem abaixo do esperado.

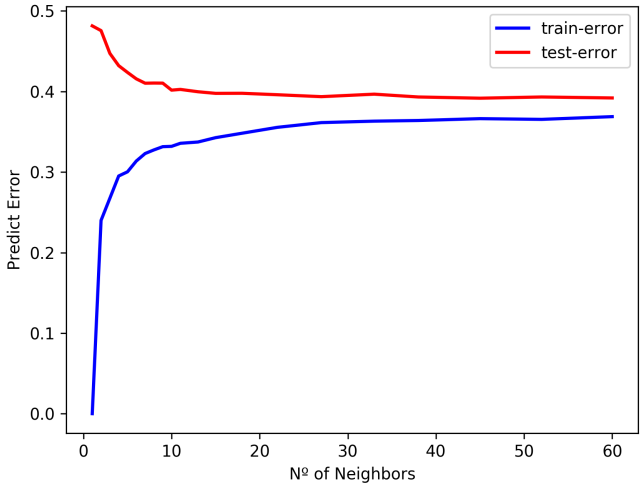

Figura 8: Curva de erro para KNN em função do número de vizinhos - usando features para otimização de k=60 como entrada

Na figura 8 podemos visualizar a curva de erro em função do número de vizinhos novamente, entretanto, agora não mais utilizando todas as features (3,4,5,6,7 descritas na seção acima), e sim as features (3,5,6,7 descritas na seção acima) que melhoram o caso de k=60.
Podemos observar uma leve queda na região assintótica da curva, juntamente com uma suave queda na região onde k possui baixo valor, quando comparado com a curva de erro utilizando todas as features. Isso indica que as features utilizadas para k=60, melhoram não só o caso de 60, mas de forma leve todos os valores de k.

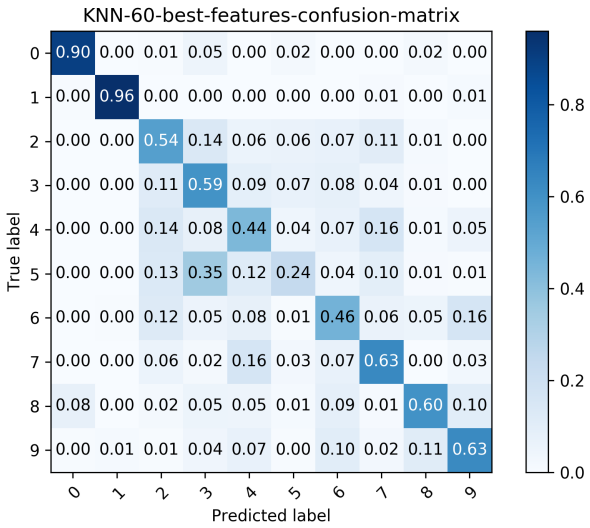

Figura 9: Matriz de confusão para KNN - usando features para otimização de k=60 como entrada

A matriz de confusão para este caso encontra-se na figura 9, onde podemos observar uma leve dispersão, alguns falsos positivos e falsos negativos, e ambos razoavelmente centralizados também, nos números 2,3,4 e 5. Novamente acreditamos que isso se deva ao mesmo fator que possa ter levado os valores de acurácia do LDA estarem abaixo do esperado.

Tabela I: Tabela de Resultados

Método	**Features**	**K**	**T. Treino (60mil) (s)**	**T. Teste (10mil) (s)**	**Acurácia**
KNN	Pixels	3	61.8808	1254.9243	0.9705
KNN	3,4,5,6,7	3	4.8184	1.6305	0.5539
KNN	3,5,6,7	60	4.543	2.13812	0.608
LDA	Pixels	-	30.5976	0.07756	0.873
LDA	1,2,3,4,5,6,7	-	10.5212	1.6891	0.651

3) Comparação: Como podemos observar na tabela, assim como explicado nas duas secções acima, para o método KNN, qualquer feature é relevante, onde features colineares podem alterar muito a acurácia, assim como o valor escolhido de vizinhos. Entretanto para o LDA quanto mais features melhor, uma vez que o fato de ser uma separação linear, toda e qualquer colinearidade irá desaparecer.

No caso onde utilizamos os próprios pixel da imagem como conjunto de entrada, obtivemos uma acurácia bastante alta, principalmente para o KNN que conseguiu se especializar mais sem aumentar muito o erro nos conjuntos de teste, porém como o conjunto de entrada possui tamanho de 784 e não há nenhum método de redução, como em redes neurais, os tempos de treino aumentaram significativamente, já os tempos de predição ocorreu algo bastante interessante, para o método KNN as predições obtiveram um valor muito alto, tornando inviável sua utilização nestas condições, já o LDA justamente por ser linear conseguiu se especializar bem, e manter os testes com baixo nivel de erro, ao mesmo tempo que obteve uma aceleração no tempo de resposta, fenômeno bastante parecido com o ocorrido em redes neurais, onde o treino possui um elevado tempo, mas devido a simplicidade da regreção obtida as predições possuem resposta bastante rápidas.

V. Conclusão

Diante dos dados apresentados, e as analises realizadas, podemos concluir que o knn possui alta acurácia para dados com muita informação, entretanto perde em desempenho nesse caso, o tornando mais indicados para dados com poucas informações.
Podemos concluir também que dentre as features escolhidas, as que apresentam melhor custo benefício são as features 3,5,6 e 7, uma vez que utilizando somente elas obtivemos uma acurácia de 0,60 no knn e 0,54 no lda.
Acreditamos que seria possível o aumento significativo da acurácia em ambos os metodos, utilizando outras features que possuam menor flutuação entre as mesmas classes e que consigam representar de forma mais distinta os valores 2,3,4 e 5 que são os valores que possuem a maior quantidade falsos positivos e falsos negativos.

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.

[2] OpenCV: Contour Features, disponível em: https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html, acessado em: 7 de abril de 2019.

[3] Sklearn: Linear Discriminant Analysis Module, disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html, acessado em: 7 de abril de 2019.

[4] Sklearn: Regression based on k-nearest neighbors Module, disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html, acessado em: 7 de abril de 2019.

[5] Sklearn: Classifier implementing the k-nearest neighbors Module, disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html, acessado em: 7 de abril de 2019.

[6] Sklearn: Confusion matrix Module, disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html, acessado em: 7 de abril de 2019.

[7] Sklearn: Plot Confusion matrix Function, disponível em: https://scikit-learn.org/0.17/auto_examples/model_selection/plot_confusion_matrix.html#example-model-selection-plot-confusion-matrix-py, acessado em: 7 de abril de 2019.

[8] Sklearn: Accuracy Module, disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html, acessado em: 7 de abril de 2019.

[9] J. Brownlee, Your First Machine Learning Project in Python Step-By-Step, disponível em: https://machinelearningmastery.com/machine-learning-in-python-step-by-step/, acessado em: 7 de abril de 2019.