Programa de Engenharia de Sistemas e Computação - PESC/Coppe/UFRJ
Aluno: Christian da Silva Cabral Cardozo **DRE:** 117029985
Data Mining - Professor Geraldo Zimbrão

Relatório de Classificação de Dígitos do MNIST Dataset

Introdução

Neste relatório estão presentes os dados e procedimentos realizados para a execução do segundo trabalho da disciplina Data Mining. O objetivo deste trabalho é realizar a classificação de dígitos escritos à mão do MNIST Dataset [1]. Foram utilizados dois algoritmos para classificação. O primeiro escolhido como “base line” foi o Naive Bayes [2]. O segundo algoritmo foi o kNN (K-nearest neighbors) [3], onde foi realizada uma análise exaustiva de parâmetros para obter o melhor resultado dentre os testados.

Dataset

O MNIST é uma base de dados de dígitos escritos à mão. Seu dataset está organizado em duas partes: treinamento e teste. O conjunto de treinamento possui 60 mil exemplos, e o conjunto de teste possui 10 mil exemplos. Estes conjuntos foram construídos de modo que todas as classes tivessem igual relevância durante o processo de aprendizado. Na seguinte tabela temos a quantidade das classes em cada conjunto:

Dígito (Classe)	Quantidade no conjunto de treinamento	Quantidade no conjunto de teste
0	5923	980
1	6742	1135
2	5958	1032
3	6131	1010
4	5842	982
5	5421	892
6	5918	958
7	6265	1028
8	5851	974
9	5949	1009

Este tipo de situação elimina a necessidade expressa da realização de um procedimento de k-Fold durante o aprendizado dos algoritmos. Como já exemplificado na tabela, temos dez classes neste dataset, que são exatamente os caracteres numéricos: 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9.

Desenvolvimento

Neste trabalho, foi utilizada a linguagem Python para o desenvolvimento. Algumas bibliotecas para Python foram utilizadas e estão listadas a seguir:

1. Python-mnist [4]: biblioteca de terceiros desenvolvido para realizar o parser dos arquivos de dados do MNIST. Este programa lê como entrada os arquivos binários da base de dados e retorna como saída as informações de cada exemplo em formato de vetor, onde cada posição do vetor representa o valor em escala cinza (0 à 255) do pixel.
2. Scikit-learn [5]: biblioteca de aprendizado de máquina que contém diversos algoritmos implementados. Nela, foram utilizadas as implementações dos algoritmos Naive Bayes e kNN.
3. NumPy [6]: biblioteca comumente utilizada em Python que contém rotinas numéricas. É dependência da biblioteca scikit-learn.
4. SciPy [7]: biblioteca comumente utilizada em Python que contém diversas rotinas utilizadas na matemática, engenharia e ciência geral. É dependência da biblioteca scikit-learn.

O código deste trabalho pode ser encontrado no GitHub [8] do autor de seu autor, assim como o presente relatório.

Naive Bayes

O Naive Bayes foi escolhido como “base line” para este trabalho. O valor de acurácia obtido com ele fornece um ponto de partida e uma expectativa mínima de acertos para os outros algoritmos. No scikit-learn, estão disponíveis três versões do Naive Bayes: Gaussiano, Multinomial e de Bernoulli. As três versões foram exploradas em busca de suas acurácias, além da variação de parâmetros na versão Multinomial e de Bernoulli. A versão Gaussiana não obteve um bom resultado, porém, as outras duas obtiveram um erro menor que 20% em suas execuções, revelando um melhor ponto de partida. Pela variação dos parâmetros na versão Multinomial e de Bernoulli, é possível perceber que a variação do parâmetro alpha tende a não melhorar o erro de forma expressiva.

Naive Bayes Gaussiano

Parâmetros	Erro (%)
Sem parâmetros para ajustar no scikit-learn	44,42

Naive Bayes Multinomial

Parâmetros	Erro (%)
alpha=1e-10	16,31
alpha=1e-06	16,3
alpha=0.0001	16,31
alpha=0.1	16,33
alpha=0.2	16,33
alpha=0.5	16,34
alpha=0.8	16,35
alpha=1.0	16,35

Naive Bayes Bernoulli

Parâmetros	Erro (%)
alpha=1e-10	15,81
alpha=1e-06	15,82
alpha=0.0001	15,82
alpha=0.1	15,85
alpha=0.2	15,85
alpha=0.5	15,86
alpha=0.8	15,86
alpha=1.0	15,87

kNN (k-nearest neighbors)

O algoritmo escolhido para alcançar valores melhores na classificação foi o kNN. O kNN faz parte de um grupo de algoritmos de aprendizado chamado “Lazy Learning” [9], onde a generalização dos dados de treinamento é feita apenas quando uma questão é feita ao modelo. Para classificar uma entrada desconhecida, o kNN utiliza os k vizinhos mais próximos da entrada para tomar a decisão.

Os experimentos realizados usando o scikit-learn levou em consideração uma análise de parâmetros exaustiva. Os seguintes parâmetros foram variados durante os vários experimentos:

1. Peso (weight): “uniform” para tratar todas as distâncias com mesmo peso e “distance” para que os vizinhos mais próximos tenha um peso maior na decisão.
2. n: chamado de “n” no scikit-learn, nada mais é do que o k na formulação citada do parágrafo anterior. Leva em consideração os n (ou k) vizinhos mais próximos para a classificação. Neste trabalho, o n foi variado entre os valores: 2, 3, 5, 10, 20, 50 e 100.
3. Métrica: métrica de distância para ser utilizada. Pode assumir os valores: “euclidean”, “manhattan” ou “minkowski”.
4. p: valor da potência caso o parâmetro de métrica seja igual à “minkowski”. O parâmetro variou entre os valores 1, 2 e 3.

Dado os valores acima, todas as combinações foram testadas em busca do menor erro. Na tabela a seguir, temos os valores dos quatro melhores resultados no experimento:

#	Weight	n (k)	p	metric	Erro (%)
1	distance	5	3	minkowski	2,65
2	distance	3	3	minkowski	2,72
3	uniform	5	3	minkowski	2,81
4	uniform	3	3	minkowski	2,82

Considerações Finais

Os resultados obtidos no presente trabalho foram satisfatórios. Primeiro, o “base line” do naive bayes deu um ponto de partida e meta de melhoria em relação ao valor do erro. A escolha do kNN se mostrou eficiente devido aos resultados obtidos com o erro pequeno. Apesar de ser um método lento para realizar a classificação, pois é um modelo de “Lazy Learning”, é possível paralelizar o processo com o scikit-learn e acelerar o término do algoritmo.

Além do objetivo inicial de classificação ter sido alcançado, os resultados obtidos são competitivamente comparáveis aos disponíveis no próprio site do MNIST. Utilizar pesos de acordo com as distâncias se mostrou mais eficaz para este problema. A obtenção de um erro igual à 2,65% revela que a escolha de k=5 foi a melhor abordagem obtida dentre os testados. Este resultado foi ligeiramente melhor que o listado no site do MNIST na categoriza do kNN sem pré-processamento. Nenhum pré-processamento foi realizado neste desenvolvimento, mas alguma abordagem anterior ao algoritmo de classificação poderia ser adotada para melhorar o resultado final, alcançando valores menores de erro como é listado no sitedo MNIST.

Referências

[1] MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. Disponível em: http://yann.lecun.com/exdb/mnist/. Acessado em 24/04/2017.
[2] Naive Bayes classifier. https://en.wikipedia.org/wiki/Naive_Bayes_classifier. Acessado em 24/04/2017.
[3] k-nearest neighbors. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm. Acessado em 24/04/2017.
[4] python-mnist. Disponível em https://pypi.python.org/pypi/python-mnist/. Acessado em 24/04/2017
[5] scikit-learn: machine learning in Python. Disponível em http://scikit-learn.org/stable/. Acessado em 24/04/2017.
[6] NumPy. Disponível em http://www.numpy.org/. Acessado em 24/04/2017
[7] SciPy.org. Disponível em https://www.scipy.org/. Acessado em 24/04/2017
[8] mnistclassification. Disponível em: https://github.com/chriiscardozo/mnistclassification. Acessado em 27/04/2017.