Relatório de Aprendizagem de Máquina
2015.1

Centro de Informática - UFPE
Aluno: Victor Carriço Santos - vcs2
Professor: George Darminton

Introdução

As questões da lista foram desenvolvidas com a linguagem Python versão 3.4.2, juntamente com as bibliotecas NumPy e MatPlotLib. Foi escolhida essa tecnologia pela flexibilidade da linguagem, simplicidade da sintaxe, por possuir diversas bibliotecas open-source e por ser bastante usada no meio científico, sendo muito fácil conseguir informações a respeito da mesma.

Questão 1

kNN sem peso

Primeiramente, foi feita uma classe KNN com os seguintes atributos:

- raw_data: contém o dataset do jeito que ele está na base UCI
- class_position: o número da coluna correspondente a classe do dado
- classification: um dicionário com todos os tipos de classe, sendo a chave igual ao nome da classe e o item uma representação em número real dela, foi feito isso para manter uma matriz apenas com números, para facilitar a manipulação
- all_data: todos os dados, já sem a coluna ID
- data: os dados de treino
- data_test: os dados de teste

Alguns métodos da classe são apenas para carregar os dados numa matriz e tratá-los, como retirar a coluna ID, identificar as classes, etc. São eles: _get_class_position, _get_raw_data, _find_classes, _load_data, get_train_data, get_test_data. Dessa forma o algoritmo ficou mais genérico, podendo ser reusado para quaisquer dados, necessitando apenas informar a coluna que contém o ID e a coluna que contém a classe. Nesta questão a base de dados foi dividida em treino e teste, aleatoriamente a cada vez que o programa é executado.

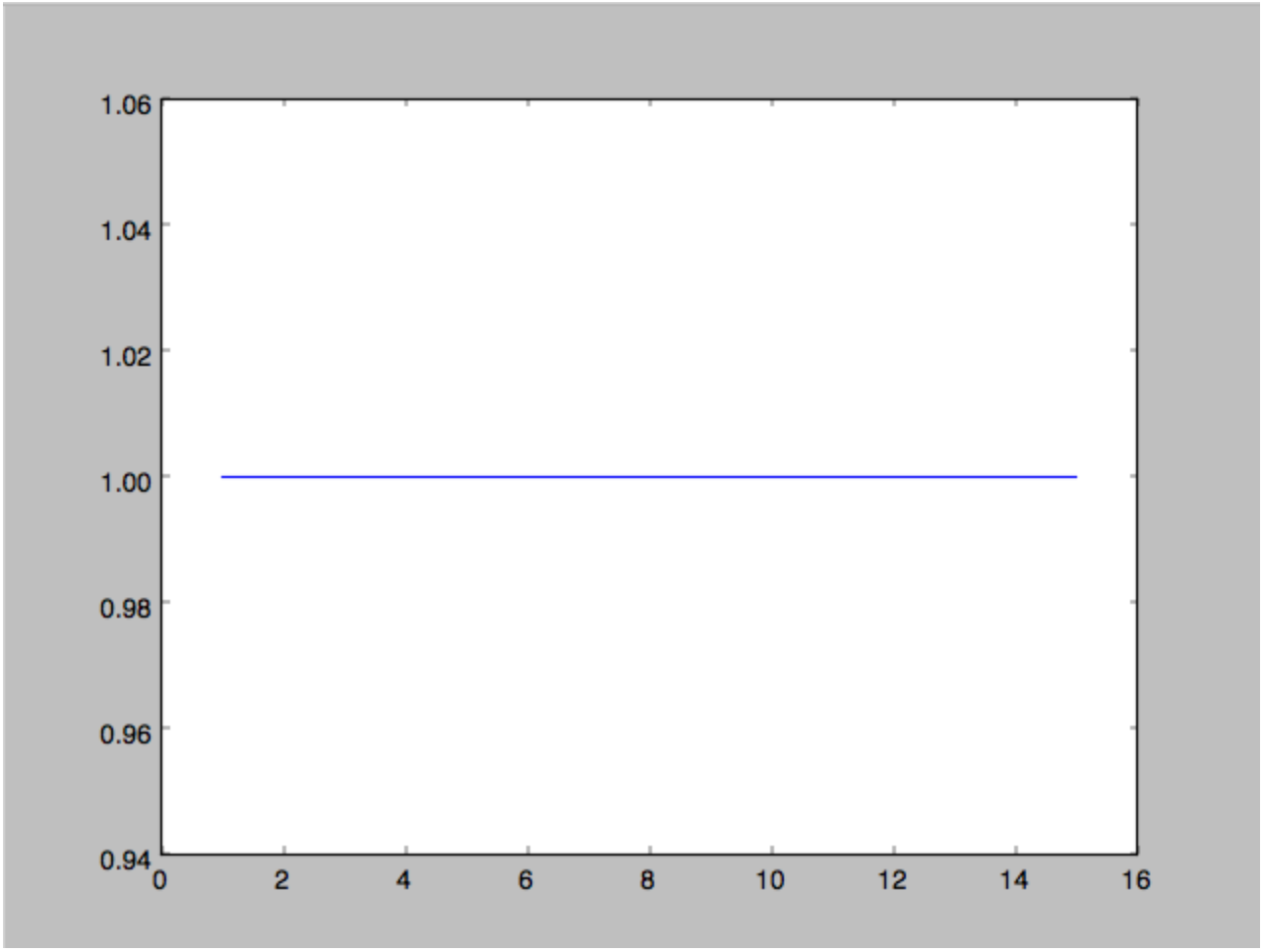
O método _get_distance faz o cálculo da distância euclidiana entre dois pontos e foi usada a função *linalg.norm* da biblioteca NumPy.

O método get_k_nearest retornam os k pontos dos dados de treino a um outro ponto e o método get_k_nearest_test aplica o get_k_nearest para todos os elementos dos dados de teste.

O método solve aplica a fórmula do KNN dos dados obtidos com os métodos acima. Basicamente, para cada ponto dos dados de teste é visto qual a classe predominante dentre os k elementos mais próximos dele.

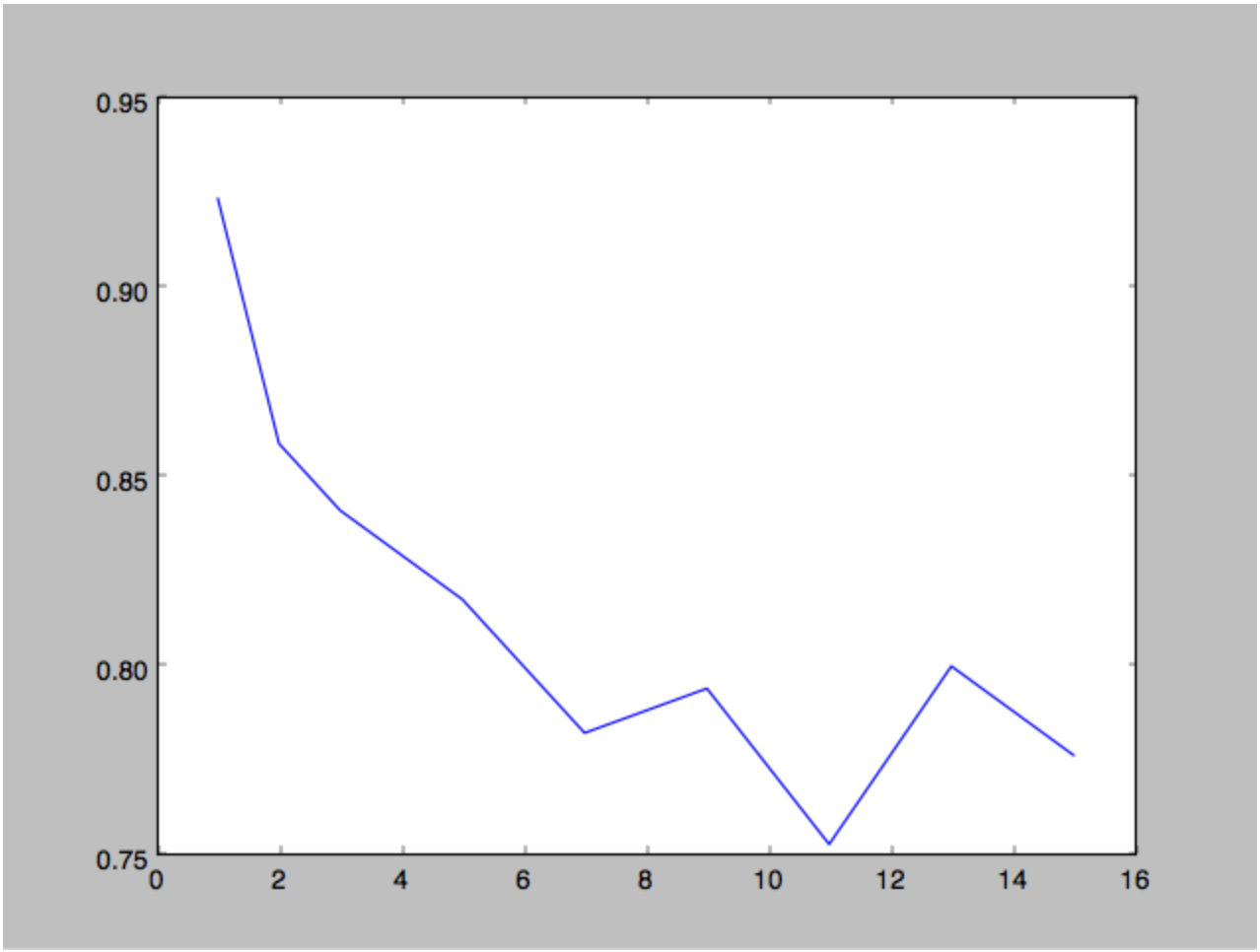
Foram usadas as bases de teste da UCI ‘Iris’ e ‘Brest cancer Wisconsin - Diagnostic’.

Segue abaixo o resultado obtido para a base ‘Iris’:



Como se pode ver, o algoritmo está com 100% de acerto para todos os K’s analisados. A princípio achei estranho, mas depois de analisar bastante o código não consegui achar nenhum erro. Para garantir, foi testado com uma quantidade menor de dados de treino e os dados variaram. Então associei o resultado inesperado ao fato dos dados serem simples, também aos dados de testes não serem muito numerosos. Além da função que usei que pode fazer algo mais além da distância euclidiana.

Abaixo temos o resultado para a base Breast cancer Wisconsin - Diagnostic':

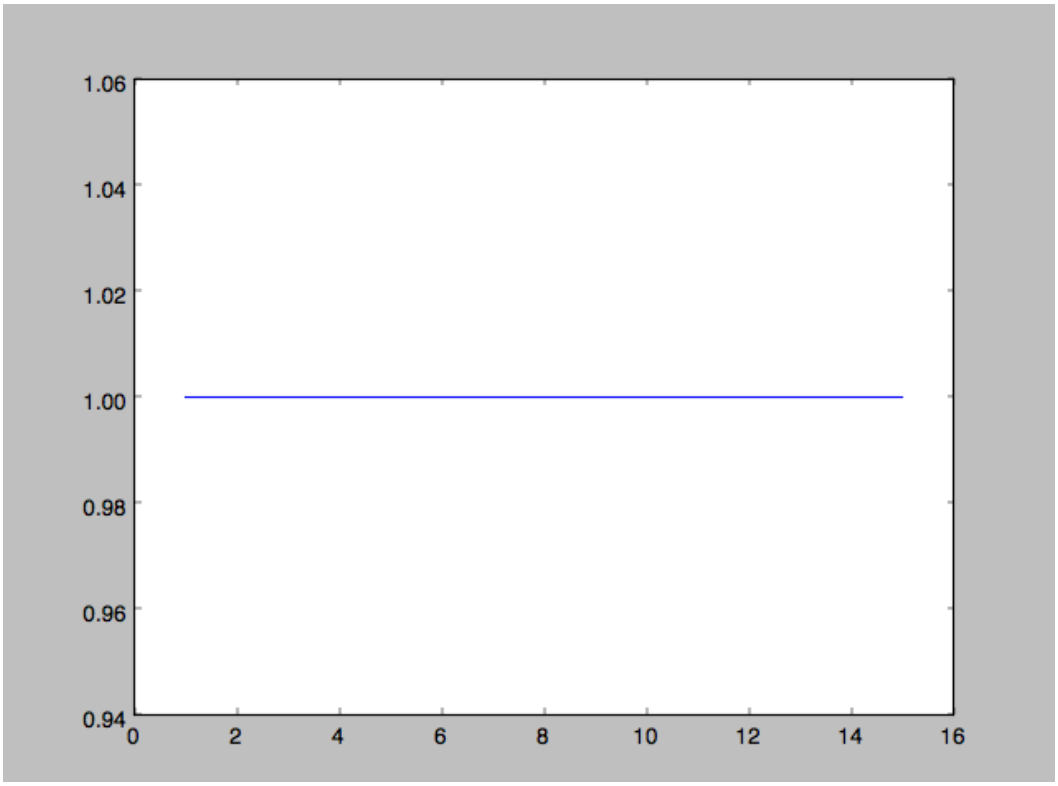


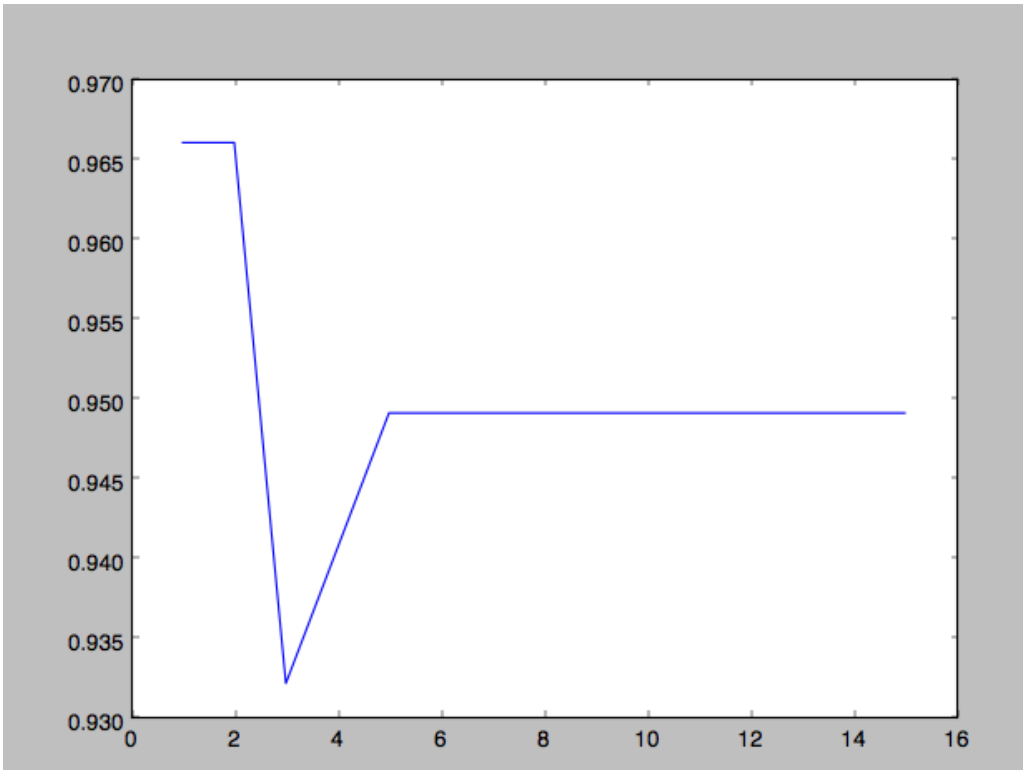
O algoritmos tem maior taxa de acerto para k=1, provavelmente pela proximidade dos pontos. Fazendo com que a medida que o k aumenta, um número maior de pontos que não pertençam aquela classe sejam pegos pelo algoritmo.

KNN com peso

Para essa parte, foi criada uma classe KNNWeighted que herda da classe KNN. O único método novo é o argmax e o delta. Esses usam a expressão literal do kNN com peso, para o cálculo dos k mais próximos. Foram usadas as mesmas bases passadas.

Segue abaixo os resultados obtidos ‘Iris’ e ‘Breast cancer Wisconsin’, respectivamente:





:

Com isto, inferimos que para a base de dados ‘Breast cancer Wisconsin’ o algoritmo com peso tem uma taxa de acerto de até 5% a mais que o algoritmo sem peso.

Questão 2

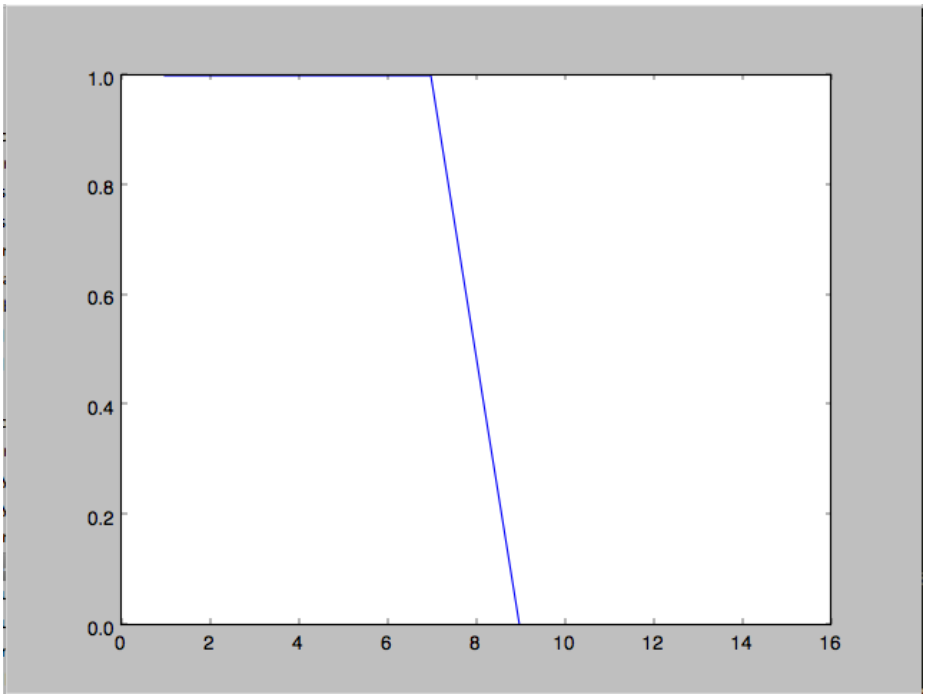
Foram usadas as bases de dados ‘hayes-roth’ e ‘shuttle-landing-control’.

Nesta questão o maior esforço foi para o cálculo da distância VDM, pois o algoritmo do kNN já estava modularizado suficiente para apenas substituir a função que calcula a distância.

Primeiramente, carreguei os dados de forma análoga ao da primeira questão. Em seguida temos o método ‘vdm_i’ que calcula o VDMi, ou seja, o argumento do cáculo da distância entre dois pontos correspondente a coluna ‘i’. Na linha 68, foi feito um fator de correção, o qual não existe no algoritmo real. Porém, eventualmente apareciam algumas divisão por zero e foi necessário acrescentá-lo. Dentro desse métodos, ainda usei mais dois métodos separados o ‘_get_N_ic’ e o ‘_get_N_i’ que retornam, respectivamente, o número de elementos dos dados de treino que tem o atributo do vetor de teste pertencem a uma determinada classe e o número de elementos dos dados de treino que tem determinado atributo. Após isso, calcula-se o argumento do cálculo do VDM para todos os atributos do vetor e retorna-se a distância VDM.

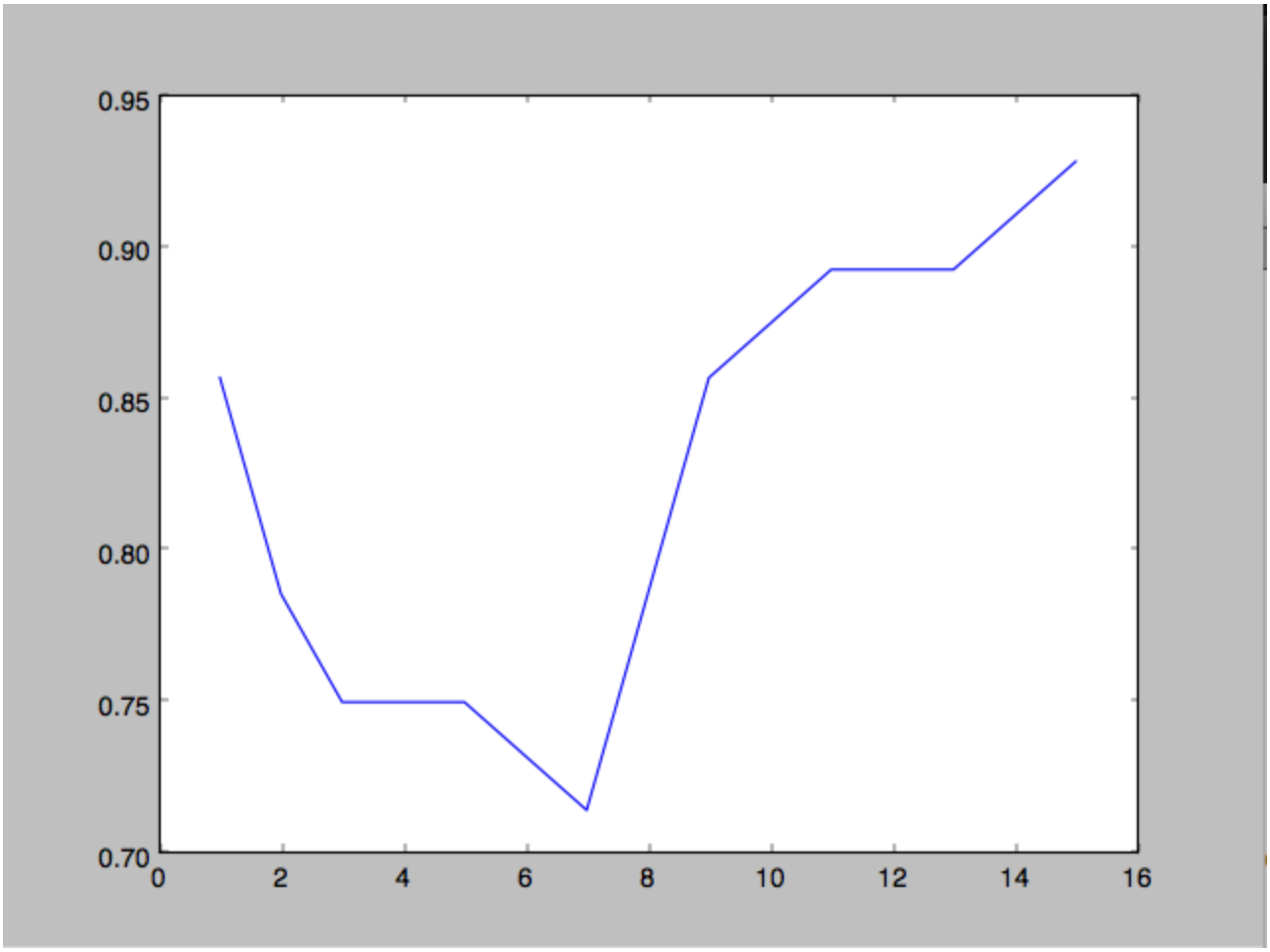
Seguem abaixo os resultados:

Resultados para a base yellow-small':



A taxa de acerto apresenta esse declínio porque a quantidade de dados é pequena, ou seja, aumenta o k até um determinado momento onde o k é grande suficiente para pegar pegar quase todos elementos de teste.

Segue abaixo o resultado para os dados 'hayes-roth':


0.95
0.90
0.85
0.80
0.75
0.70
0
2
4
6
8
10
12
14
16