

Trabalho Prático 1

Henrique Furst Scheid - 2017014898

Janeiro 2022

Contents

1	Implementação	2
2	Análise experimental	6
3	Gráficos	8
3.1	australian.dat	8
3.2	banana.dat	10
3.3	hayes-roth.dat	12
3.4	heart.dat	15
3.5	monk-2.dat	17
3.6	phoneme.dat	19
3.7	ring.dat	21
3.8	tae.dat	23
3.9	titanic.dat	26
3.10	twonorm.dat	28

Implementação

<https://github.com/hfscheid/TP1-AlgoII>

O algoritmo

O algoritmo implementado desempenha a função de classificar pontos de acordo com sua proximidade espacial a pontos anteriormente avaliados. O mecanismo para tal avaliação consiste em um registro *a priori* de um conjunto de dados pertencente ao mesmo 'espaço' dos dados a serem avaliados. Desse modo, durante a classificação resgatam-se do registro um número arbitrário (oferecido pelo usuário do algoritmo) os dados espacialmente mais próximos do dado a se classificar. A classificação resultante corresponde à classificação presente na maioria de tais dados resgatados. Trata-se, pois, de uma implementação do algoritmo **k-nearest-neighbours**.

Denominam-se os dados oferecidos *a priori* de **conjunto de aprendizado** e os dados a serem classificados pelo algoritmo de **conjunto de teste**. O algoritmo está implementado em *python*.

Estruturas de dados

kdtree

O aspecto chave do algoritmo está no armazenamento e resgate de dados espaciais, com número arbitrário de dimensões. Foi utilizada uma árvore k-dimensional (*k-d tree*) com vértices estruturais e valores registrados exclusivamente em folhas. Dessa forma, o resgate de dados próximos a um ponto oferecido consistem em uma pesquisa (de complexidade logarítmica) a uma folha de valor mais próximo que atua como 'centro', seguido de 'retrocessos' a vértices em seu ramo para a adição de suas folhas vizinhas mais próximas (especialmente, os pontos dentro de um raio partindo de tal centro).

A implementação da *kdtree* consistiu na criação de duas classes principais: *KDtree* e *BaseNode*. Esta se ramifica em duas subclasses *Node* e *Leaf*, que representam os elementos da árvore.

KDtree possui métodos para:

- Construção de uma kdtree
- Pesquisa de pontos (retorna o ponto salvo mais próximo do ponto pesquisado)
- Resgate de todas as folhas que ramificam de um nó (retorna um conjunto de pontos)
- Exibição da kdtree, utilizado para testes

BaseNode possui métodos para:

- Construção de um vértice com valor interno (float) e referência para um vértice pai.
- Resgate do vértice pai
- Resgate de seu valor

Node possui métodos para:

- Construção de um vértice *BaseNode* com valor comparador e referência ao vértice paterno, adicionado de:
 - dois *BaseNodes* filhos (esquerdo para valores menores que o comparador, direito para valores maiores que o comparador)
 - dimensão do espaço à qual o comparador pertence
- Configuração e resgate do vértice filho esquerdo
- Configuração e resgate do vértice filho direito

Leaf possui métodos para:

- Construção de um vértice *BaseNode* com valor de classificação e referência ao vértice paterno, adicionado das coordenadas espaciais do ponto referente.

k-nearest-neighbours

Criou-se uma classe *xNN* para acomodar o conjunto de teste, assim como a *kdtree* populada com o conjunto de aprendizado. Além dos dados, a classe possui os métodos correspondentes ao algoritmo *k-nearest-neighbours* em si, que toma como parâmetro o número de vizinhos a serem utilizados.

A execução do algoritmo exige computação sucessiva de pontos vizinhos do ponto de teste *P*. Tais vizinhos são armazenados em uma **fila de prioridade** (implementada no módulo *heapq* da biblioteca padrão de *python*). A fila de prioridade permite a ordenação de novos vizinhos mediante menor distância com relação a *P* a cada inserção, de forma que pontos mais distantes ocupam as folhas. Tal fila é sempre “podada” para manter apenas os *K* vizinhos mais próximos de *P*.

Algoritmos

Construção de kdtree

A construção de uma árvore k-dimensional é feita através da construção recursiva de nós. Uma lista inicial com todos os pontos é separada “maiores” e “menores” através da comparação com o valor mediano de uma das dimensões. Um novo nó é criado cujos filhos esquerdos são o ramo “menores” e os direitos o ramo “maiores”. Tal processo é feito recursivamente para novas dimensões até que haja no máximo 2 pontos, instante no qual folhas são criadas para armazená-los e o algoritmo retorna.

```
constroiArvore(lista_pontos, dimensão, nó_pai):  
  Se 'lista_pontos' está vazio:  
    retorne  
  Se 'lista_pontos' possui 2 pontos:  
    crie um novo nó filho de 'nó_pai'  
    o valor do novo nó é a mediana dos pontos na dimensão atual  
    crie duas folhas para armazenar os pontos:  
      direita para o ponto maior que a mediana na dimensão atual  
      esquerda para o outro  
    insira as folhas como filhas do nó  
    retorne o novo nó  
  
  Se 'lista_pontos' possui 1 ponto:  
    retorne uma folha com o ponto  
  
  Senão:  
    crie um novo nó filho de 'nó_pai'  
    o valor do novo nó é a mediana dos pontos da dimensão atual  
    divida 'lista_pontos' em pontos maiores e menores que a mediana  
      na dimensão atual  
  
    filho direito do novo nó = constroiArvore(pontos maiores,  
                                              próxima dimensão,  
                                              nó atual)  
    filho esquerdo do novo nó = constroiArvore(pontos menores,  
                                              próxima dimensão,  
                                              nó atual)  
  
    retorne o novo nó
```

Consulta de ponto mais próximo

Para resgatar dados a partir de um ponto oferecido, basta comparar recursivamente o valor do input com o valor do comparador no nó em sua dimensão correspondente, a começar da raiz da árvore. Se o valor do input for maior, pesquise recursivamente no filho da direita. Caso contrário, pesquise recursivamente no filho da esquerda. Chegando-se em uma folha, retorne-a: este é o valor desejado.

Resgatar as folhas a partir de um nó

Dado um nó da *kdtree*, crie uma lista vazia. Adicione a essa lista as folhas dos nós do seu filho à esquerda, depois as folhas dos seus filhos da direita. Para encontrar tais filhos, execute este mesmo algoritmo, recursivamente.

```
folhas(nó):  
  Se 'nó' é vazio:  
    retorne lista vazia  
  Se 'nó' é uma folha:  
    retorne uma lista contendo 'nó'  
  Senão:  
    crie uma lista vazia  
    adicione à lista o valor de folhas(filho da esquerda de 'nó')  
    adicione à lista o valor de folhas(filho da direita de 'nó')  
    retorne a lista
```

K-nearest neighbours

O programa recebe uma base de dados e uma lista com os números de vizinhos a serem utilizados para a execução do algoritmo. Dessa forma, com apenas uma chamada o programa pode exibir resultados com diferentes graus de eficácia de acordo com o número de vizinhos oferecido. A base de dados é dividida em aprendizado (70% dos dados) e teste (30% dos dados). Um objeto *xNN* recebe tais conjuntos, criando uma *kdtree* a partir do primeiro e fazendo sucessivas classificações (uma para cada número de vizinhos oferecido) no segundo.

O programa realiza o seguinte procedimento para todos os elementos do conjunto de teste:

1. Dado um ponto P do conjunto de teste para classificação e um número K de vizinhos como parâmetro, o algoritmo pesquisa em sua *kdtree* (habitada pelo conjunto de aprendizado) o ponto X mais próximo de P .
2. Em seguida, o algoritmo retrocede para o pai X' de tal ponto. Se as folhas que ramificam de X totalizam um número de vizinhos menor que K , retroceda para o pai de X' , X'' e pegue as folhas que retrocedem de X'' . Repita o processo até encontrar K vizinhos.
3. Se o valor comparador do próximo pai a se retroceder estiver a uma distância maior do que a distância máxima entre os vizinhos atuais e o ponto P , interrompa laço. Senão, retroceda novamente, remova os vizinhos mais distantes para permanecer com K vizinhos sendo considerados.

Análise experimental

Foram escolhidos 10 conjuntos de dados, todos contendo apenas valores numéricos. As sucessivas avaliações de cada conjunto foram armazenadas em dois tipos de arquivo:

- Um *output* contendo cada ponto avaliado e sua classificação
- Um contendo o desempenho do algoritmo (acurácia, precisão e revocação) diante de todas as avaliações

O arquivo com os dados de desempenho foi utilizado para a geração de uma melhor visualização dos resultados. A seguir estão as observações acerca do desempenho do algoritmo para cada conjunto de dados. As classificações aos quais as visualizações se referem tratam-se das classes (*tags*) atribuídas pelo algoritmo, e não das originais.

australian.dat

Percebe-se que o aumento do número de vizinhos possui um efeito inversamente proporcional em dois momentos: entre a precisão e revocação de uma classe, e entre os comportamentos das classes. As curvas "Precisão da classificação 0" e "Revocação da classificação 1" são similares, tal como "Precisão da classificação 1" e "Revocação da classificação 0". Além disso, tais pares parecem espelhados. O pico de acurácia em 36 vizinhos coincide com o "ponto médio" de todas as curvas.

banana.dat

O comportamento se assemelha ao conjunto de dados anterior. Não há simetria evidente entre pares de gráficos todavia, visto que "Precisão da classificação -1" e "Revocação da classificação 1" se comportam como uma curva com concavidade para baixo, como o poderia fazer uma função quadrática. Isso se opõe ao comportamento de reta do outro par, resultando em um interessante comportamento logarítmico (observado) na curva de acurácia.

hayes-roth.dat

A classificação de valores "3" apresenta degeneração em todos os valores exceto 4. Isso se dá pois o algoritmo não classificou nenhum ponto como "3" a partir de 8 vizinhos, tornando o cálculo indefinido. A precisão da classificação "1" apresenta um comportamento (observado) oscilatório de amplitude crescente, enquanto a revocação da classificação "2" oscila com amplitude decrescente.

heart.dat

Todas as curvas apresentam concavidade para cima, em um comportamento ligeiramente similar. Como resultado, a acurácia se mostra mais alta nos pontos periféricos do gráfico observado.

monk-2.dat

A precisão da classificação "1", tal como a revocação da classificação "0", apresentam um comportamento bastante mais previsível que seu par complementar. Comum a todos, os melhores resultados ocorrem com poucos vizinhos, o que se reflete na acurácia.

phoneme.dat

Todas as métricas apresentam um decrescimento (observado) previsível similar a uma função inversa. Fugindo a este padrão estão as extremidades dos gráficos, que podem sugerir um comportamento oscilatório. A Acurácia reflete o comportamento previsível tal como a influência das exceções, resultando em valor ótimo com 8 vizinhos ao invés de 4.

ring.dat

Dois comportamentos distintos são observados. a Partir de 8 valores, a precisão das classificações com "0" é de 100%, tal como a revocação das classificações com "1". Paralelamente, a precisão das classificações com "1" e a revocação das classificações com "0" se comportam muito similarmente a uma função inversa. Como resultado, a acurácia máxima é tida com o menor número de vizinhos testado, 4.

tae.dat

Simetrias foram observadas tal como nos demais conjuntos de dados, apesar de não haverem comportamentos regulares. Observa-se uma tendência a maiores precisões em torno de 50 vizinhos, um valor mais alto comparado aos demais testes. Apenas a classificação "3" apresentou revocação igualmente alta nesse valor, enquanto os demais apresentam comportamento decrescente. A acurácia máxima foi atingida em torno de 40 vizinhos.

titanic.dat

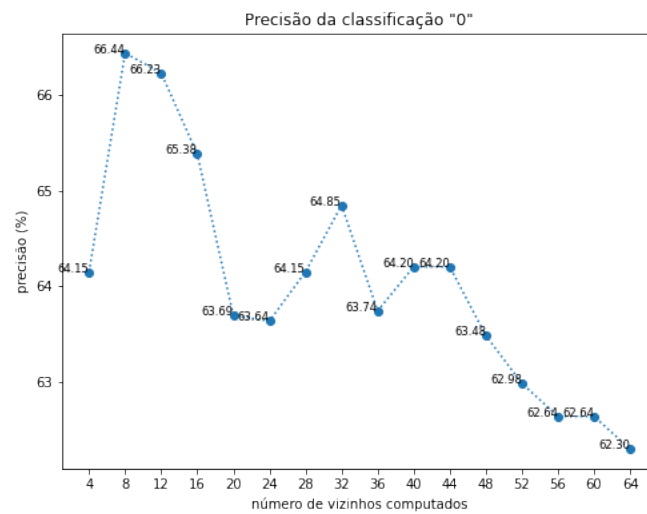
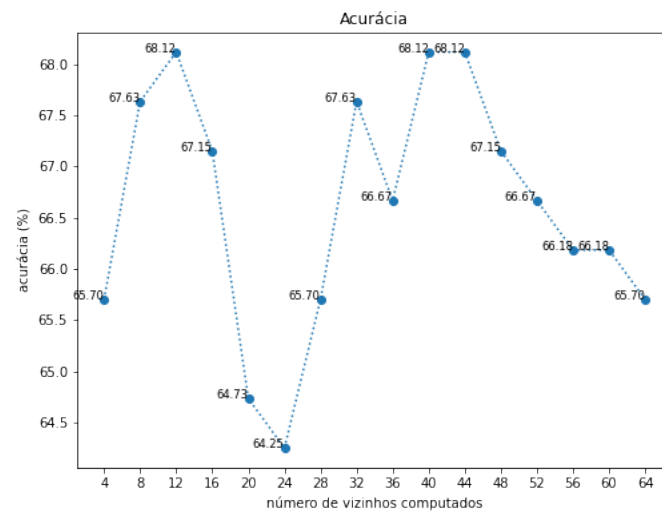
Todos os gráficos apresentam comportamento parecido com funções degrau: valores constantes seguidos de súbitas variações. Além disso, todos os valores máximos encontram-se com números pequenos de vizinhos. Acurácia máxima encontrada em torno de 12 vizinhos.

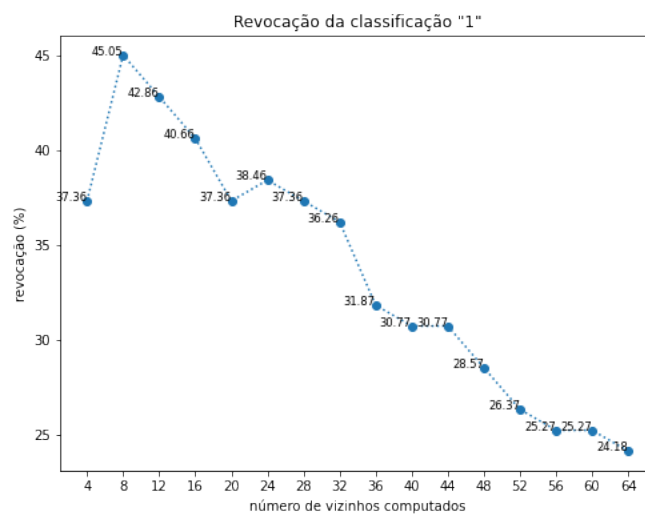
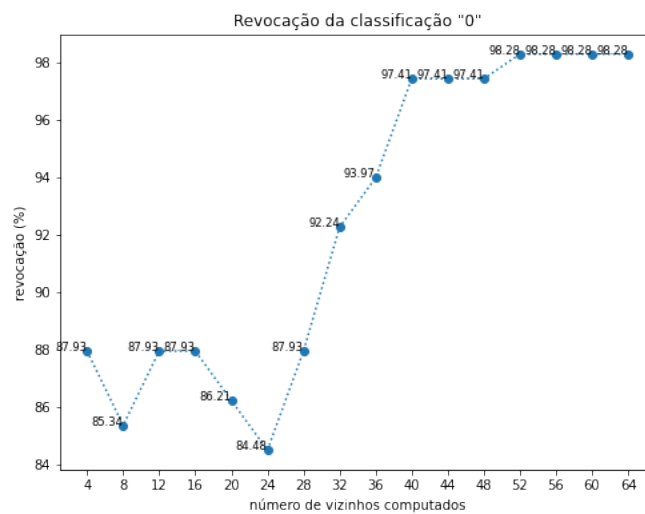
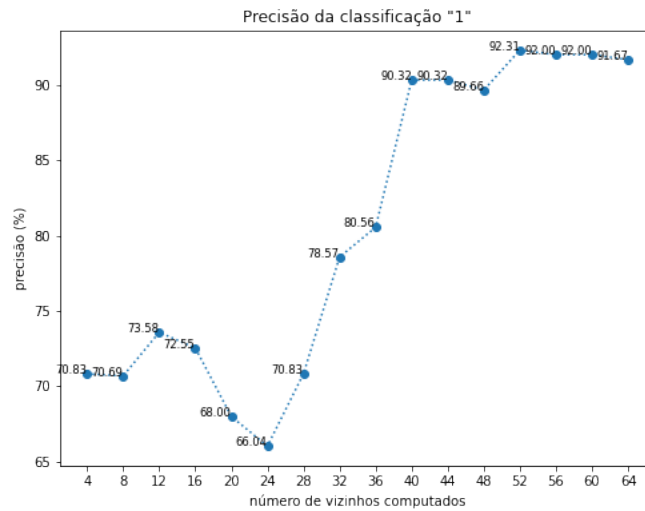
twonorm.dat

Todas as métricas apresentam súbito crescimento seguido de pequena oscilação em torno de uma constante próxima a 97%. Valores máximos encontrados com números médios de vizinhos (entre 20 e 40). Acurácia máxima obtida com 24 vizinhos.

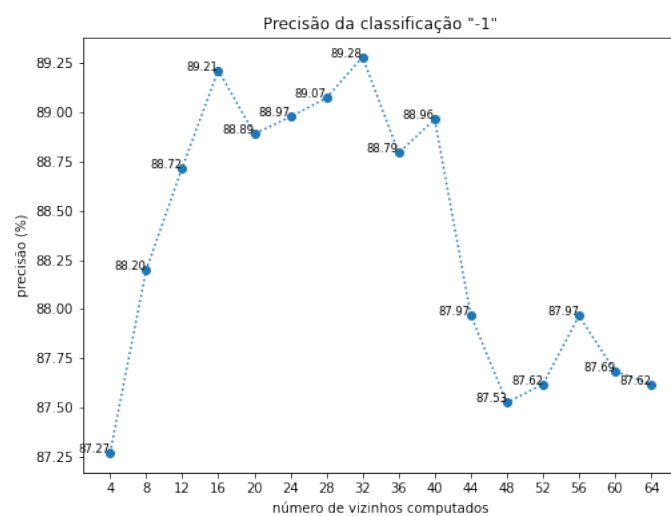
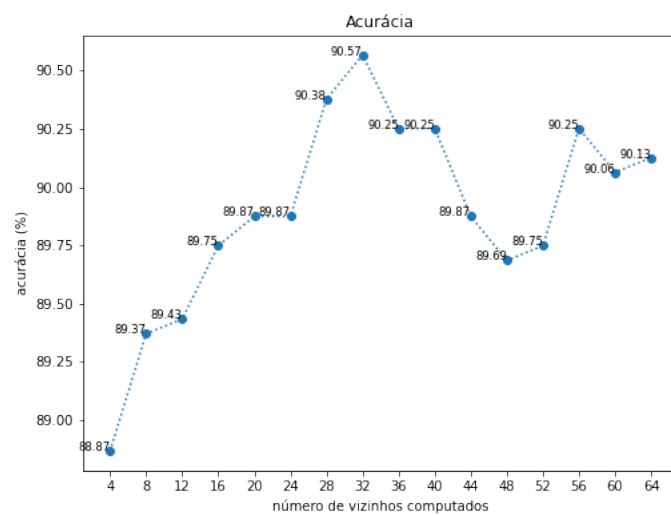
Gráficos

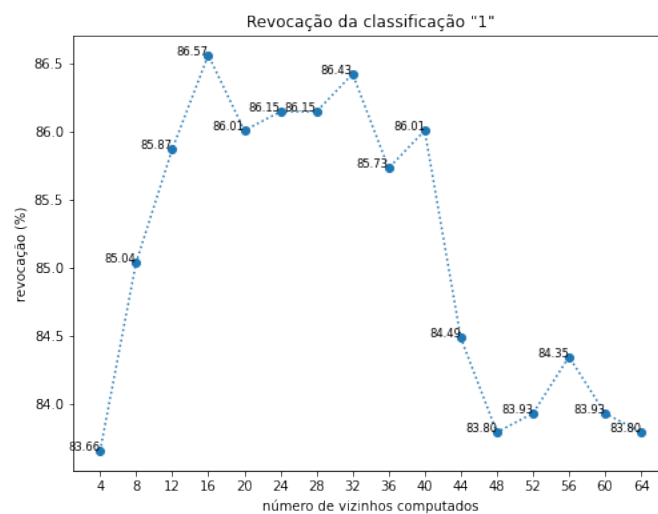
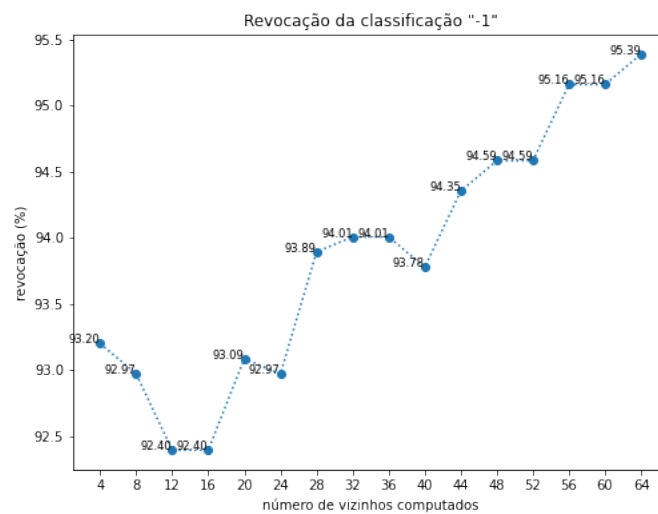
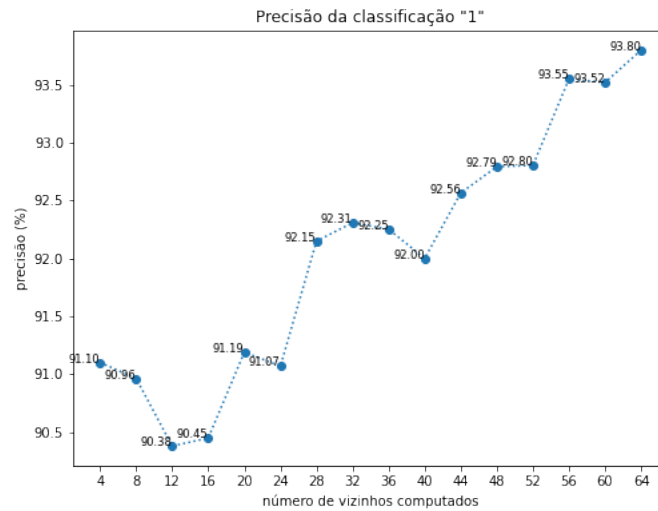
3.1 australian.dat



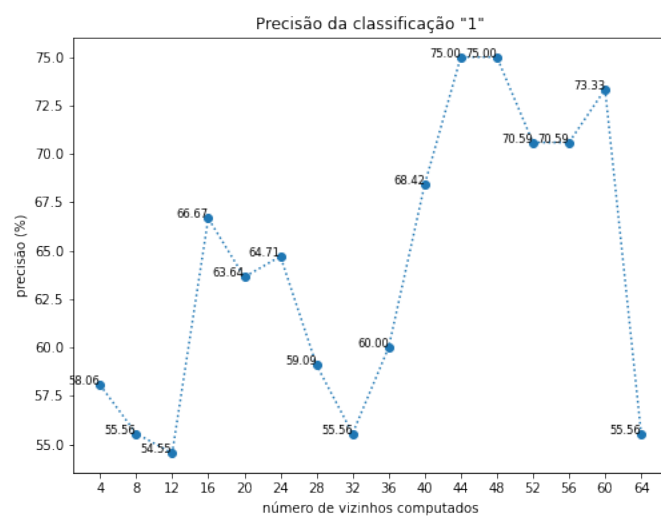
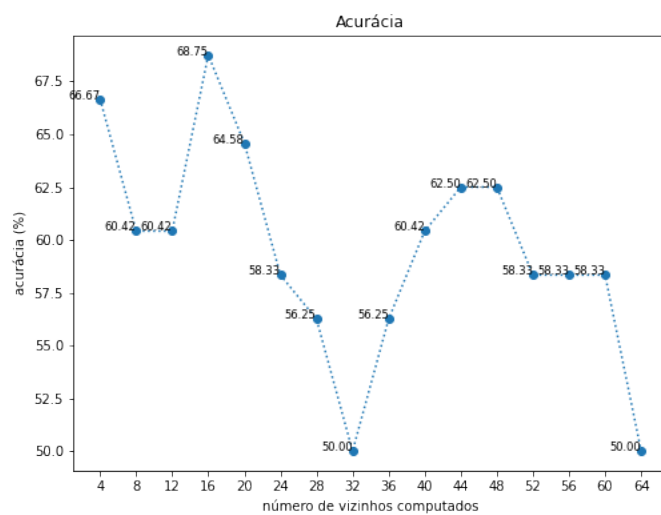


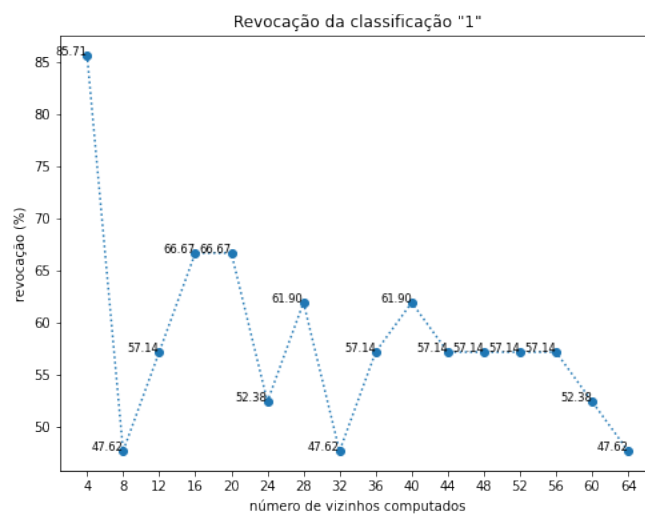
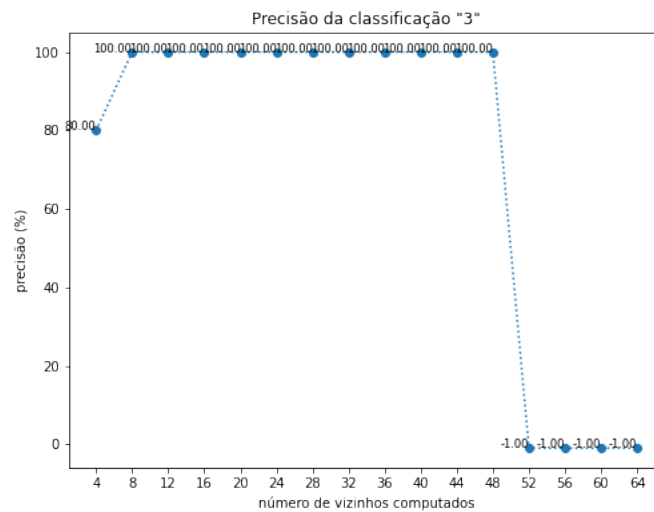
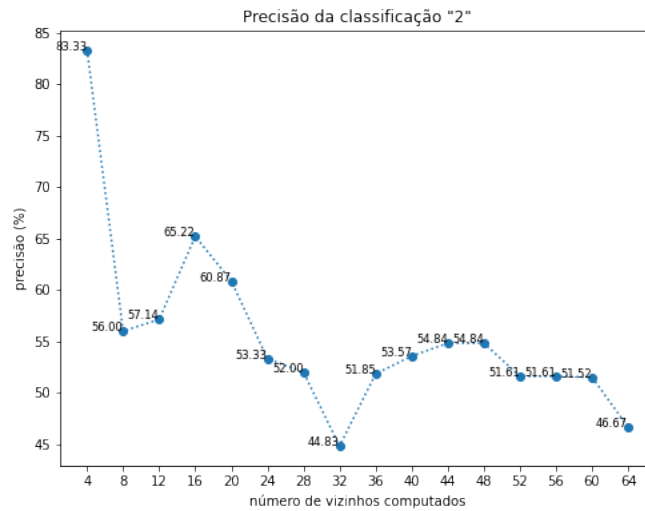
3.2 banana.dat

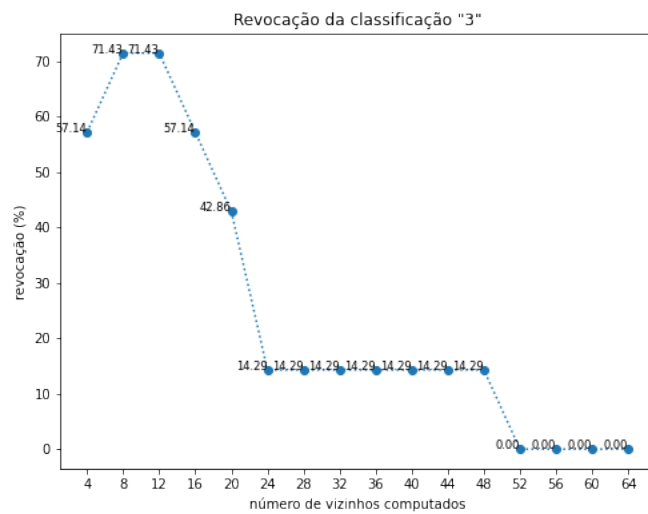
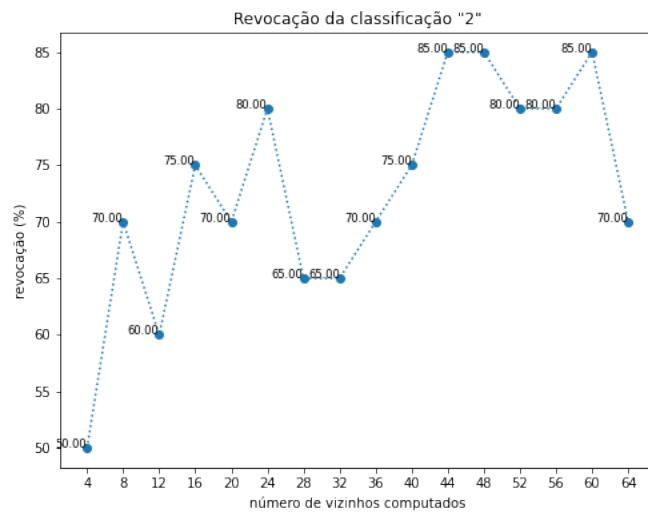




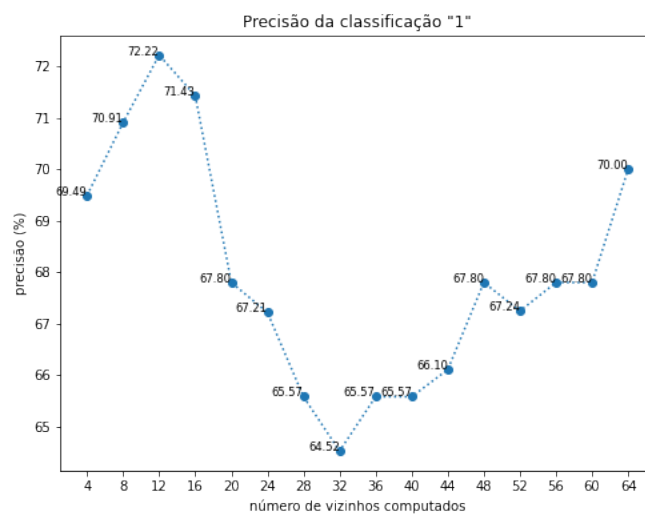
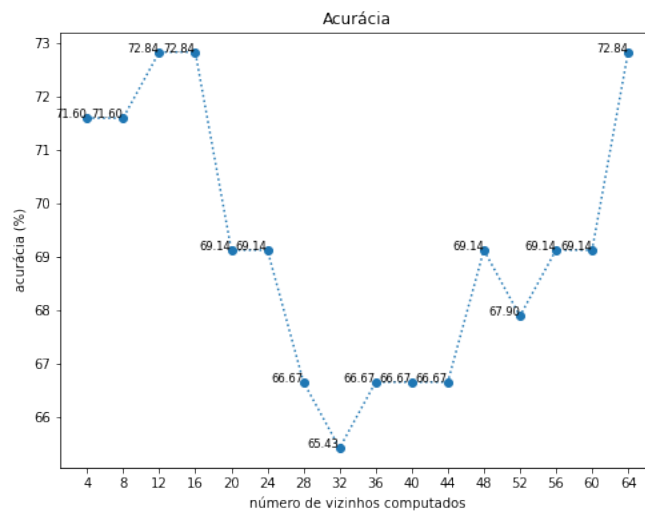
3.3 hayes-roth.dat

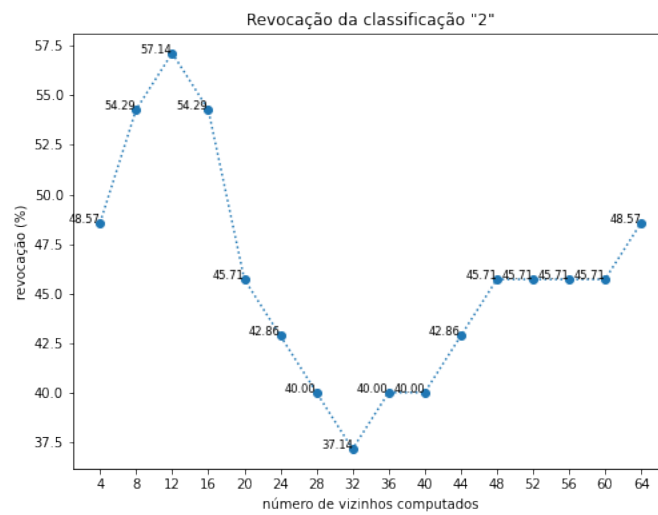
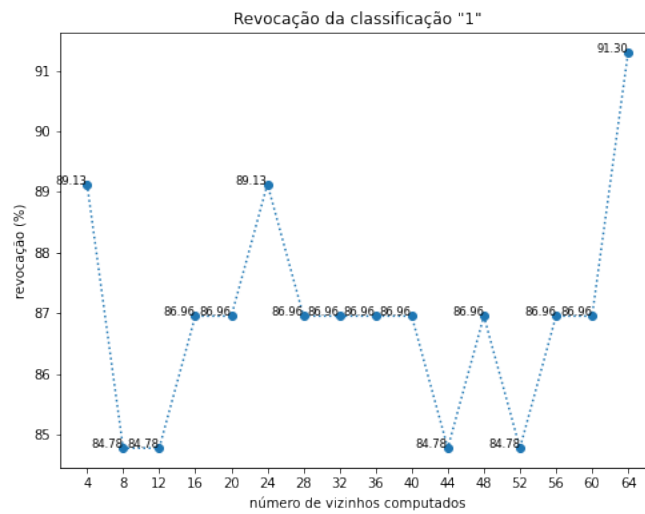
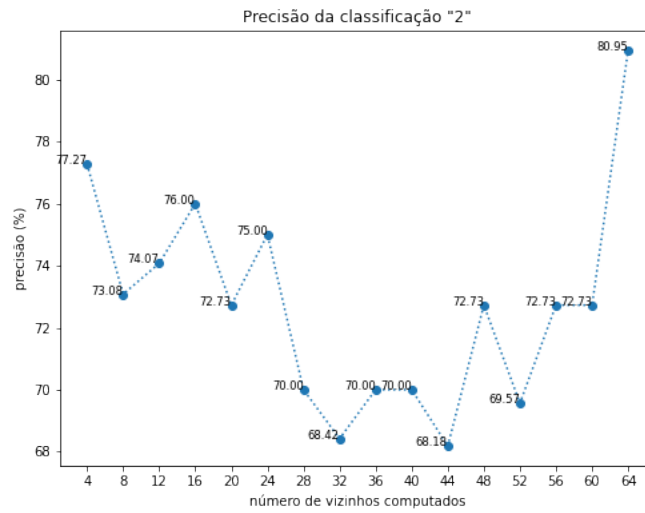




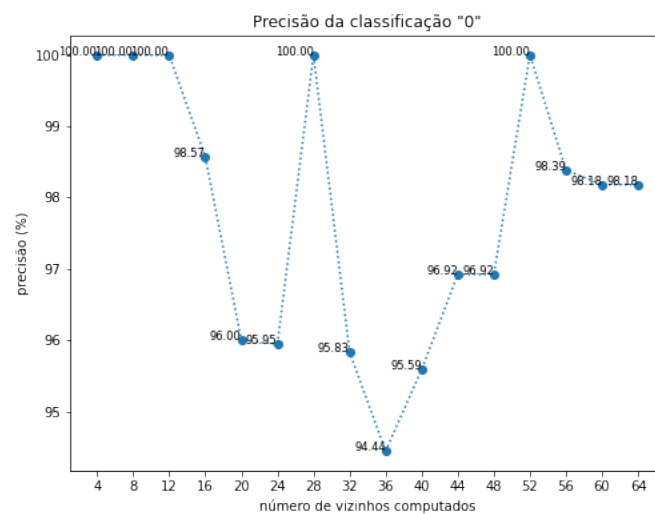
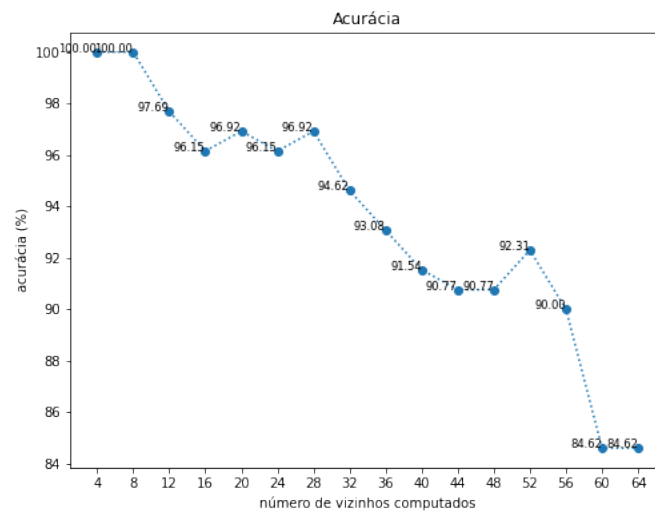


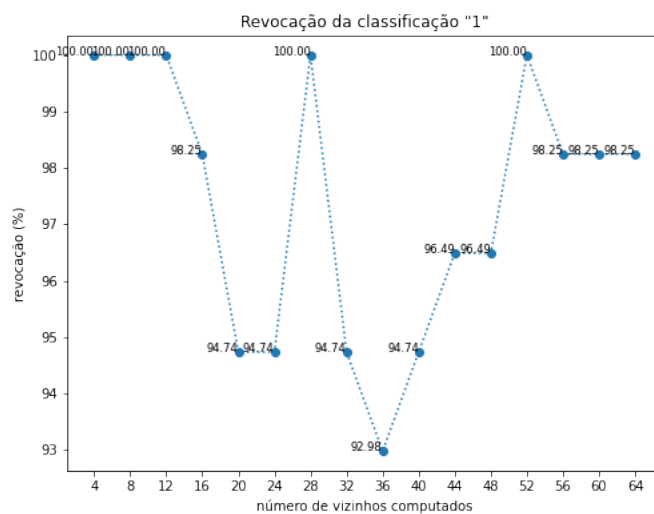
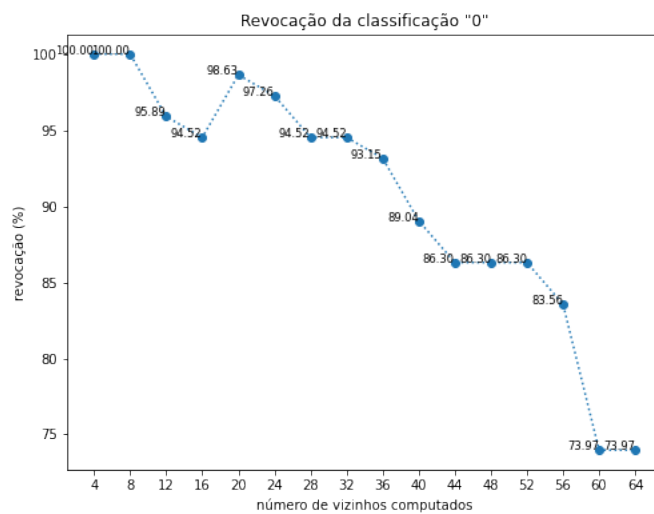
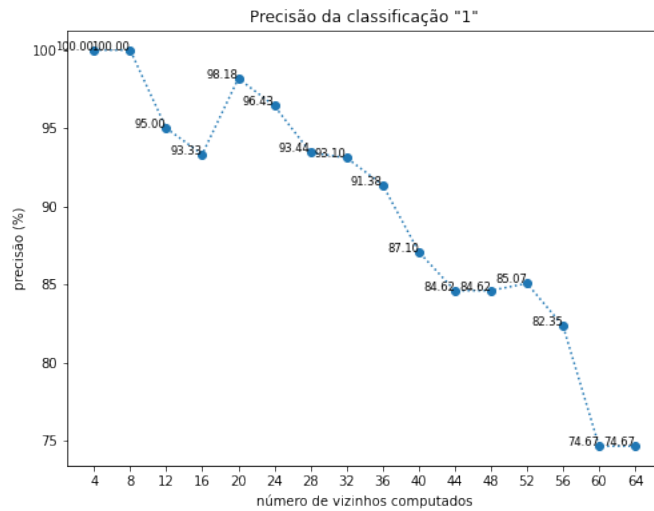
3.4 heart.dat



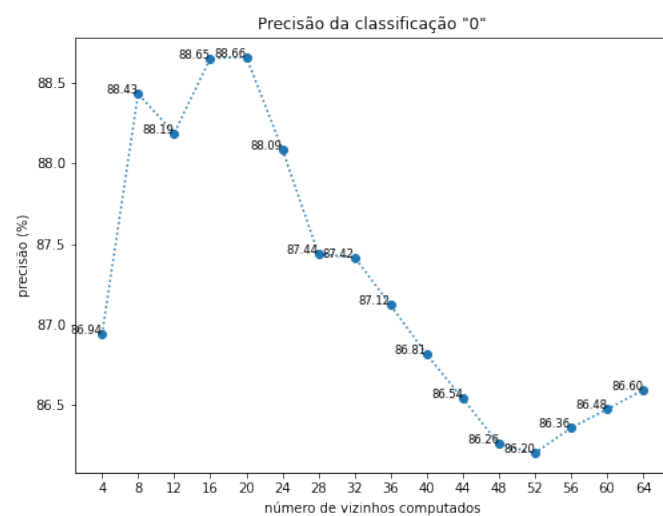
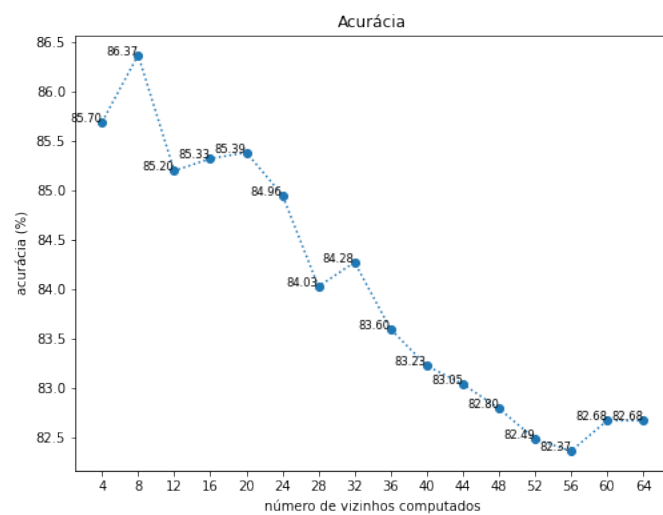


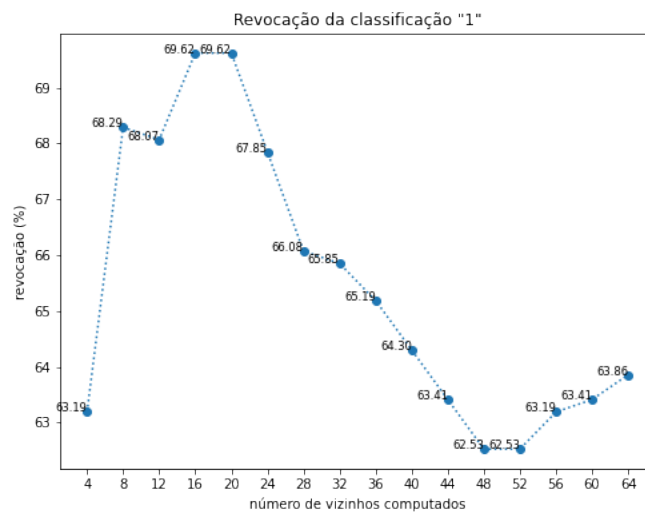
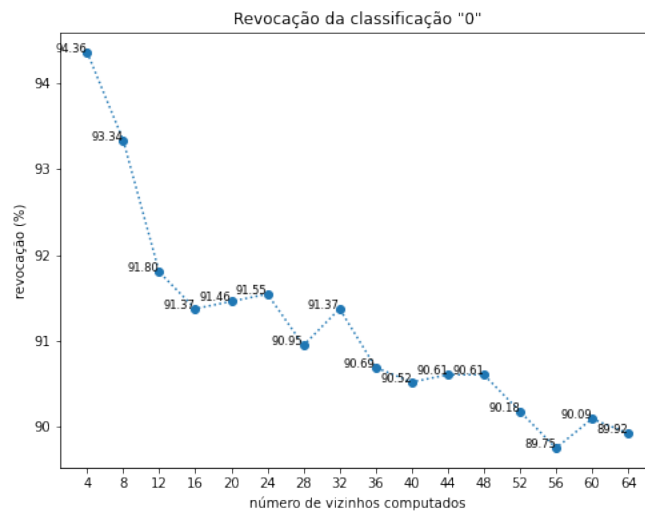
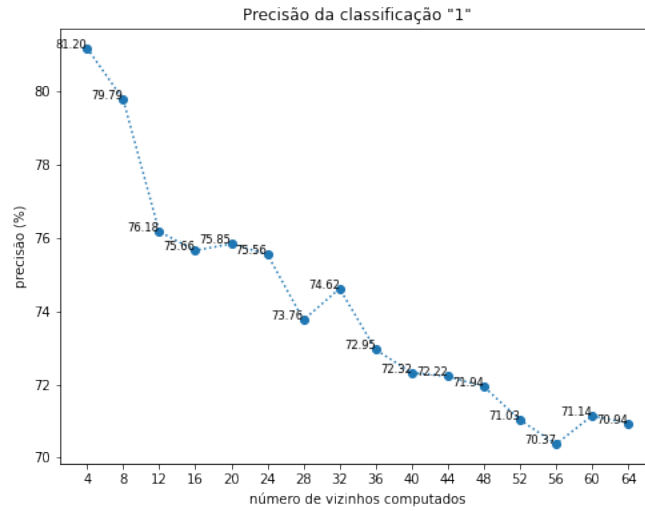
3.5 monk-2.dat



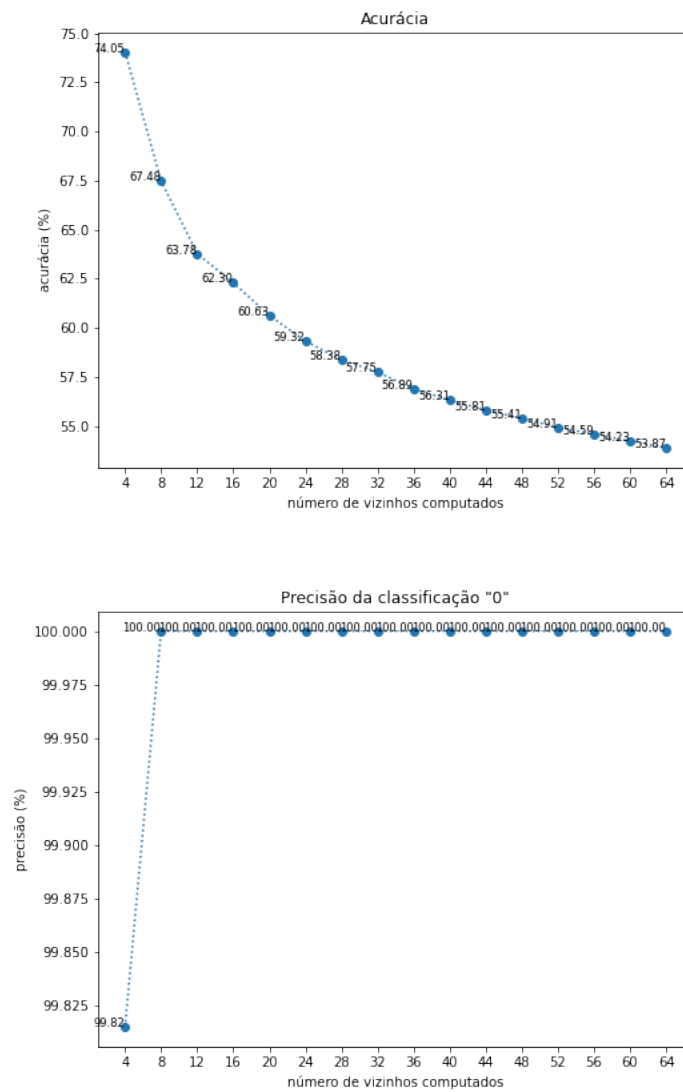


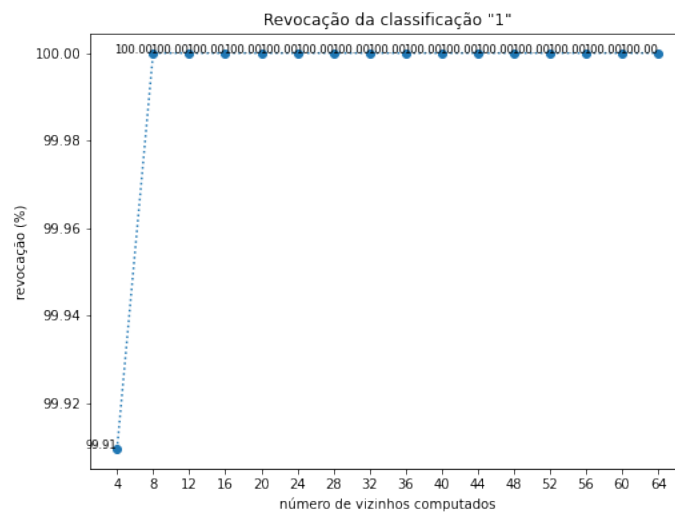
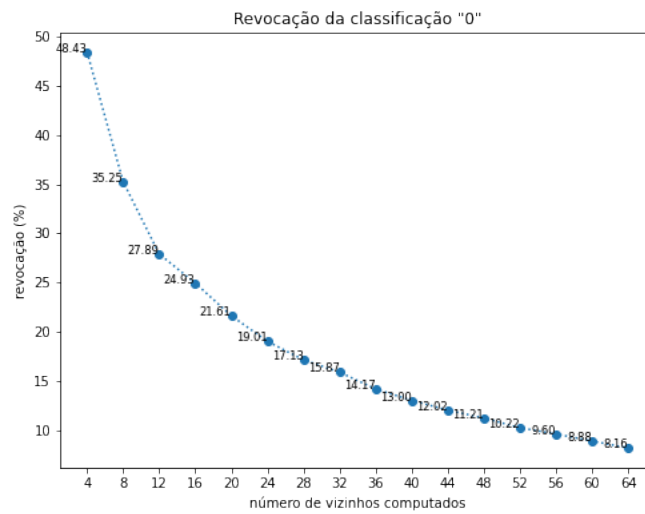
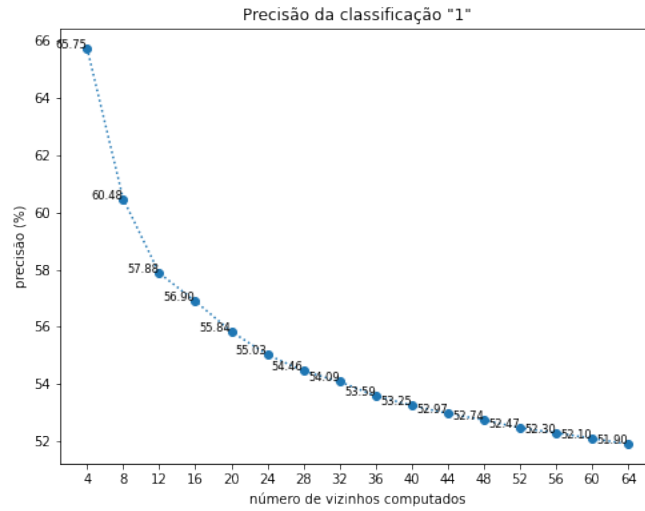
3.6 phoneme.dat



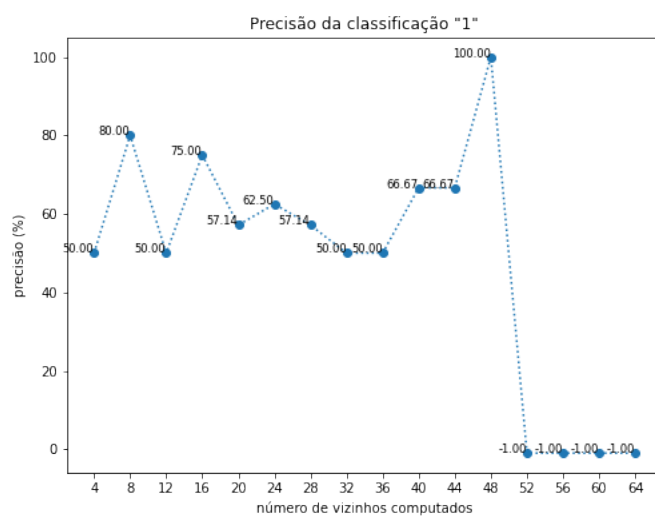
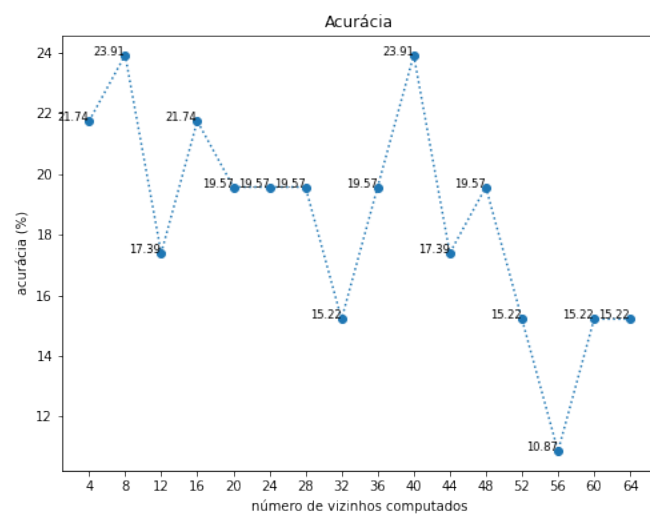


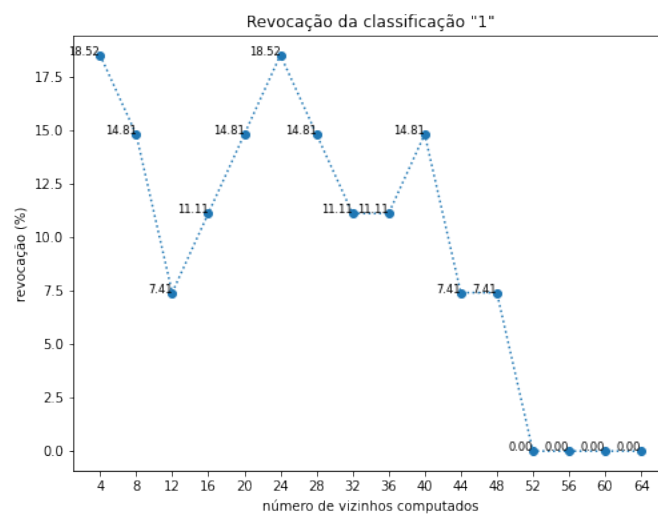
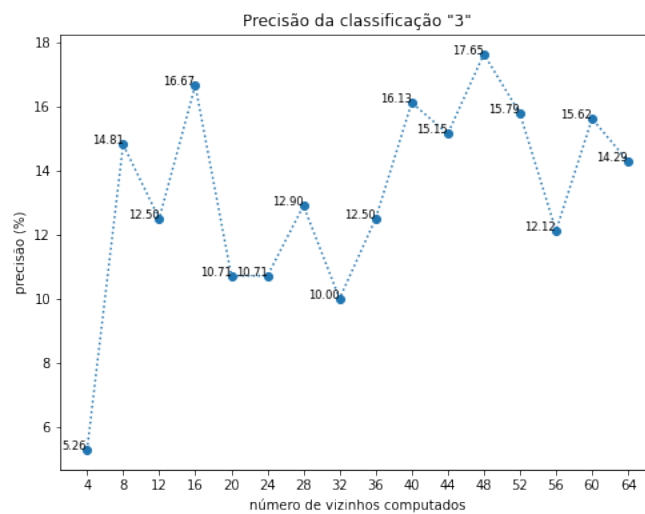
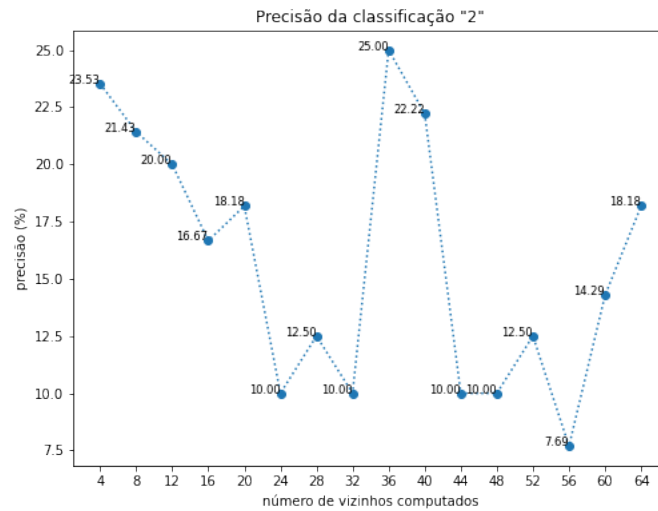
3.7 ring.dat

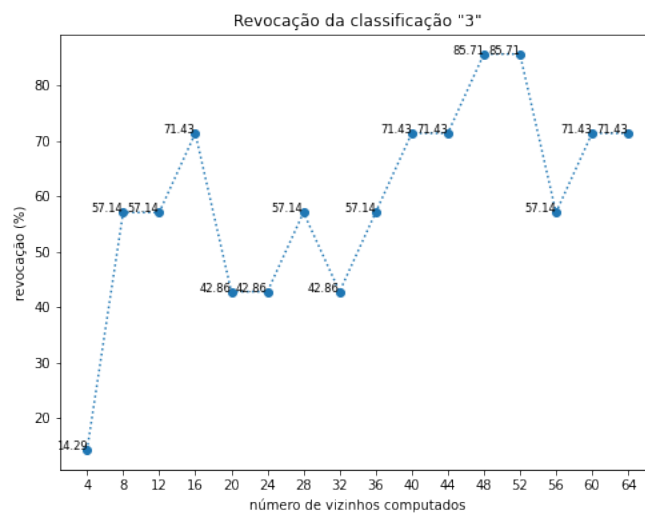
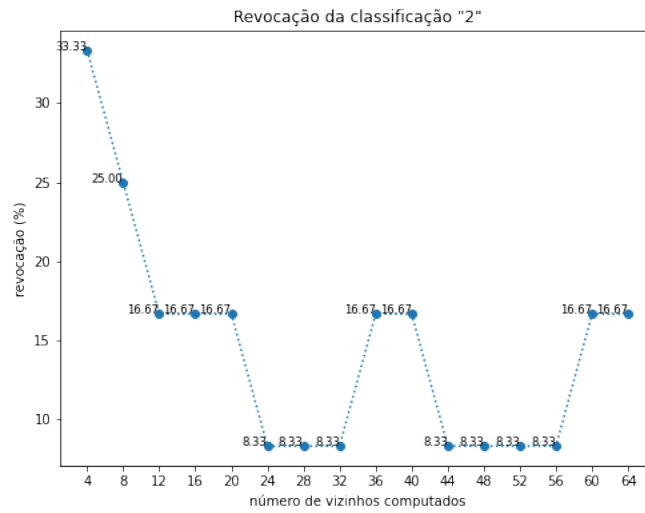




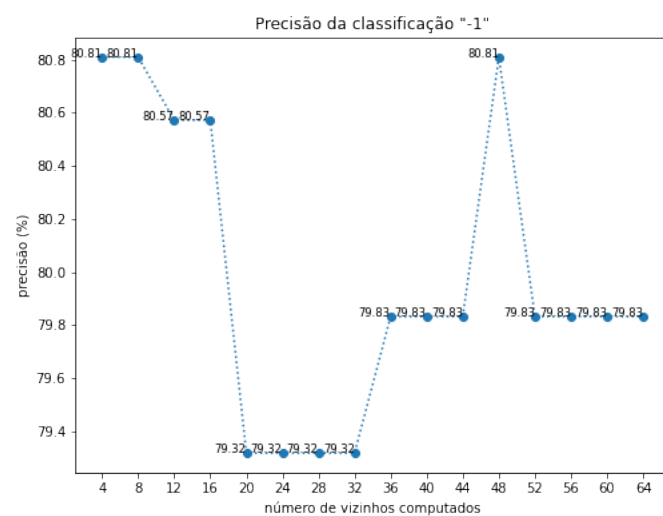
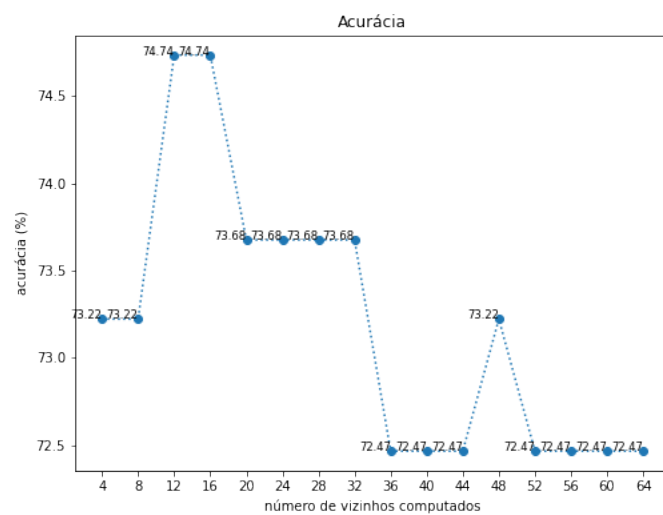
3.8 tae.dat

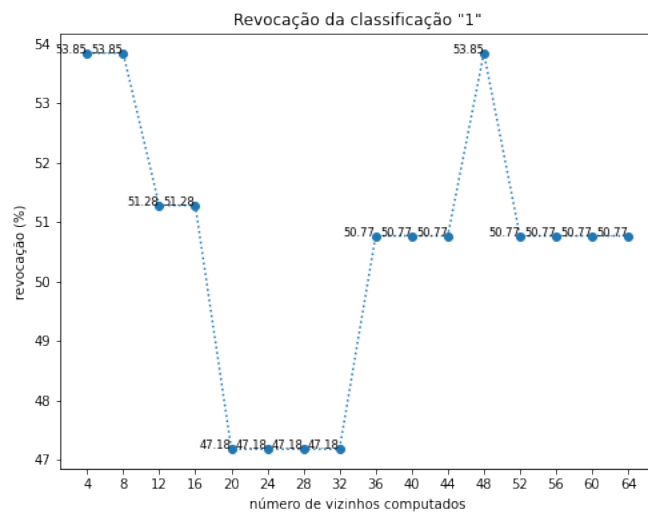
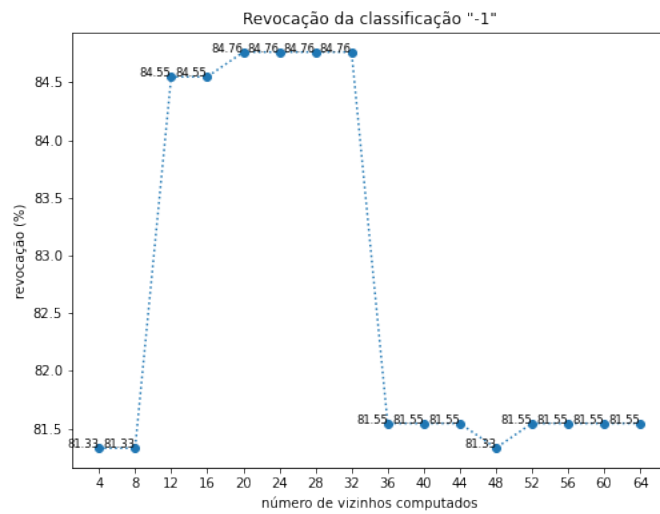
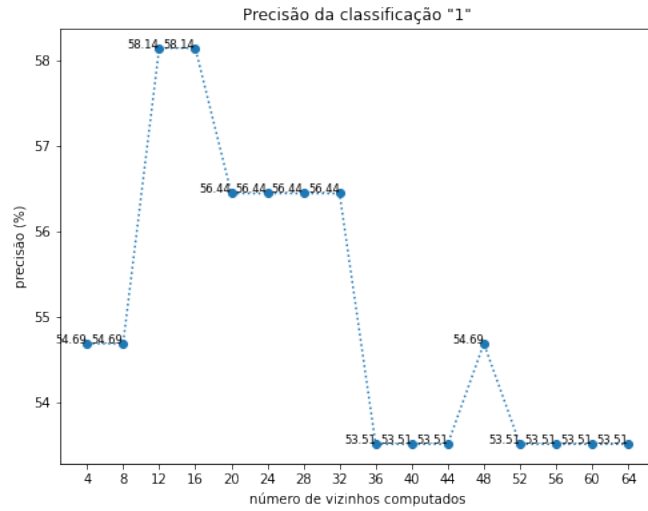






3.9 titanic.dat





3.10 twonorm.dat

