Machine Learning Package

Portfolio de algoritmos de Machine Learning

Sumário

- O objectivo de clustering consiste em criar grupos de observações que contêm dados semelhantes.
- O objectivo da decomposição consistem em reduzir as observações nas suas partes constituintes.
- No nosso portefólio, métodos de clustering e decomposição podem seguir a estrutura de um ***Transformer***.
- Os métodos de clustering têm um método adicional chamado *predict*. Este método indica qual o cluster de cada amostra.

Datasets

- Os datasets estão disponíveis em:
  - https://www.dropbox.com/sh/oas4yru2r9n61hk/AADpRunbqES44W49gx9deRN5a?dl=0

statistics sub-package

- No sub-package *statistics* adiciona o módulo chamado *euclidean_distance.py.*
- *def euclidean_distance*
  - assinatura/argumentos:
    - x – uma amostra
    - y – várias amostras
  - ouput esperado:
    - array com distância entre X e as várias amostras de Y
  - algoritmo:
    - Calcula a distância euclidiana entre X e Y usando a seguinte formula:
    - distance_y1n = np.sqrt((x1 - y11)^2 + (x2 - y12)^2 + ... + (xn - y1n)^2)
    - distance_y2n = np. sqrt((x1 - y21)^2 + (x2 - y22)^2 + ... + (xn - y2n)^2)
    - …

Objeto *KMeans*

- Na pasta *clustering*, adiciona o modulo *kmeans.py* que deve conter o objeto *KMeans.*
- O algoritmo *k-means* agrupa amostras em grupos chamados *centroids.* O algoritmo tenta reduzir a distância entre as amostras e o *centroid*.
- *class KMeans:*
  - Parâmetros:
    - k – número de clusters/centroids
    - max_iter – número máximo de iterações
    - distance – função que calcula a distância
  - Parâmetros estimados:
    - centroids – média das amostras em cada centroid
    - labels – vetor com a label de cada centroid
  - Métodos:
    - fit – infere os centroids minimizando a distância entre as amostras e o centroid
    - transform – calcula as distâncias entre as amostras e os centroids
    - predict – infere qual dos centroids está mais perto da amostra

Objeto *KMeans*

- *KMeans.fit*:
  1. Inicializa k centroids usando o *np.random.permutation*.
     Resultado: k centroids
  2. Calcula a distância entre uma amostra e os vários centroids. Escolhe o centroid com distância mais curta. Aplica o método a todas as amostras do dataset.
     Resultado: labels
  3. Agrupa as amostras pelo seus centroid. Calcula a média de cada centroid.
     Resultado: k centroids
  4. Repete o passo 2 e 3 até não existirem diferenças nas labels
- *KMeans.transform*:
  1. Calcula a distância entre cada amostra e os vários centroids
  2. Retorna as distâncias
- *KMeans.predict*:
  1. Calcula a distância entre uma amostra e os vários centroids. Escolhe o centroid com distância mais curta. Aplica o método a todas as amostras do dataset.
     Resultado: labels

Avaliação

- Exercício 3: Implementar o *PCA*
  - 3.1) Adiciona o objeto *PCA* ao sub-package *decomposition*. Deves criar um módulo chamado *pca.py* para implementar este objeto
  - 3.2) Considera a estrutura do objeto *PCA* apresentada no diapositivo seguinte.
  - 3.3) Podes testar o objeto *PCA* num jupyter notebook usando o dataset iris.csv (classificação)

Objeto PCA

- Técnica de álgebra linear para reduzir as dimensões do dataset. O PCA a implementar usa a técnica de álgebra linear SVD (Singular Value Decomposition)

- *class PCA:*
  - Parâmetros:
    - n_components – número de componentes
  - Parâmetros estimados:
    - mean – média das amostras
    - components – os componentes principais aka matriz unitária dos eigenvectors
    - explained_variance – a variância explicada aka matriz diagonal dos eigenvalues
  - Métodos:
    - fit – estima a média, os componentes e a variância explicada
    - transform – calcula o dataset reduzido usando os componentes principais.

Vê o slide seguinte para mais detalhes sobre a técnica SVD

PCA c/ SVD

- Os passos seguintes permitem implementar o método *fit do PCA* usando SVD:
  1. Começa por centrar os dados:
     - Infere a média das amostras
     - Subtrai a média ao dataset (X – mean)
  2. Calcula o SVD:
     - SVD de X pode ser calculado pela seguinte formula $X = U*S*V^T$
     - A função *numpy.linalg.svd(X, full_matrices=False)* dá-nos o U, S, V^T
  3. Infere os componentes principais:
     - Os componentes principais (*components*) correspondem aos primeiros *n_components* de V^T
  4. Infere a variância explicada:
     - A variância explicada pode ser calculada pela seguinte formula $EV = S^2/(n-1)$ – n corresponde ao número de amostras e S é dado pelo SVD
     - A variância explicada (*explained_variance*) corresponde aos primeiros *n_componentes* de EV.

PCA c/ SVD

- Os passos seguintes permitem implementar o método *transform do PCA* usando SVD:
  1. Começa por centrar os dados:
     - Subtrai a média ao dataset (X – mean).
     - Usa a média inferida no método *fit*
  2. Calcula o X reduzido:
     - A redução de X pode ser calculado pela seguinte formula Xreduced = X*V
     - A função *numpy.dot(X, V) – multiplicação de matrizes* - dá-nos a redução de X às componentes principais
     - NOTA: V corresponde à matriz transporta de V^T