Jeammy Côté - 536 963 858

IFT-4102 - TP3

Q2

— La précision (Precision)

Déterminer quel pourcentage des prédictions sont des vrais positifs par rapport aux faux positifs. TP / TP + FP

— Le rappel (Recall)

Déterminer quel pourcentage des prédictions sont des vrais positifs par rapport aux faux négatifs. TP / TP + FN

— Le F1-score

Est une mesure de précision de notre algorithme, il combine la précision et le rappel pour donner une idée équilibrée. 2 x precision x rappel / (precision + rappel)

— La matrice de confusion (Confusion matrix)

C'est une matrice nxn ou n est le nombre de classes de nos données. Par exemple, pour des données où il est seulement question de vrai ou de faux la matrice contiendrait les vrais positifs, les faux positifs, les vrais negatifs est les faux négatifs. Lorsque l'algorithme prédit une valeur vraie correctement, c'est un vrai positif. Lorsque l'algorithme prédit une valeur vraie incorrectement, c'est un faux positif. La matrice peut-être découpée comme suit [[true positive(TP), false positive(FP)], [false negative(FN), true negative(TN)]]. Il suffit de prendre les valeurs du tableau pour calculer la precision et le rappel.

Pour le calcul de la précision la formule est : TP / TP + FP

Pour le calcul du rappel la formule est : TP / TP + FN

Pour le calcul de f1-score la formule est : 2 x precision x rappel / (precision + rappel)

Pour le calcul de l'exactitude la formule est : total des TP / total des prédictions

Q2.1

2.1.1

La métrique de distance choisie est la distance d'Euclide puisque ces celle-ci qui est utilisé dans les diapos du cours et quel est plutôt simple à calculer.

2.1.2

function : predict

input :

x - an example to predict the class

data - data trained with the train function

k - nb of nearest neighbors

output :

bestClass - the predicted class for x

---

for entry in data

distance <- calculEuclideanDistance(x, entry)

distances <- addDistance(distance)

distances = distances.sort()

nearest_neighbors <- getNearestByK(k)

classCount <- getCountByClass(nearest_neighbors)

bestClass <- max(classCount)

return bestClass

Q2.2

2.2.1

function : train

input :

data - nxm array of data as n is the number of entries and m the number of features

classes - an array of length n as the label of classes in data

output :

summaries - nxm array as n is the number of classes and m the mean and standard deviation for each features

---

```
declare summaries as array
for class in classes
  for feature in data
    mean <- calculateMean(feature)
    standardDeviation <- calculateStandardDeviation(feature)
    summaries[class] <- (mean, standardDeviation)
  end for
end for
return summaries
```

##

function : predict

input : x - an example to predict the class

summaries - summary of data trained with the train function

output : bestClass - the predicted class for x

```
declare probabilityByClass as array
for class in classes
  for feature in x
    mean <- summaries[class][feature][0]
    standardDeviation <- summaries[class][feature][1]
    probability <- gaussianProbability(x, mean, standardDeviation)
    probabilityByClass[class] <- probabilityByClass[class] * probability
for probability in probabilityByClass
  return class with max probability
```

Q2.1.3 - 2.1.4 - 2.2.3 - 2.2.4

Résultats de l'algorithme naive Bayes, knn et comparaison avec scikit-learn pour chaques jeu de données:

```
################################################################################
##                          Evaluation on train dataset                       ##
################################################################################
```

----------Iris train----------

Train results for Naive Bayes with iris dataset:

confusion_matrix:

[[39. 0. 0.]

[0. 38. 3.]

[0. 3. 37.]]

accuracy : 0.95

precision : 0.9506097560975609

recall : 0.9506097560975609

f1 : 0.9506097560975609

Train results for sklearn Naive Bayes with iris dataset:

confusion_matrix:

[[39 0 0]

[0 38 3]

[0 3 37]]

accuracy : 0.95

precision : 0.9506097560975609

recall : 0.9506097560975609

f1 : 0.9506097560975609

Train results for knn with iris dataset:

confusion_matrix:

[[39. 0. 0.]

[0. 39. 2.]

[0. 1. 39.]]

accuracy : 0.975

precision : 0.9754065040650407

recall : 0.9754065040650407

f1 : 0.9754065040650407

Train results for sklearn knn with iris dataset:

confusion_matrix:

[[39 0 0]

[0 39 2]

[0 1 39]]

accuracy : 0.975

precision : 0.9754065040650407

recall : 0.9754065040650407

f1 : 0.9753086419753085

----------wine train----------

Train results for Naive Bayes with wine dataset:

confusion_matrix:

[[949. 358.]

[143. 710.]]

accuracy : 0.7680555555555556

precision : 0.7669208132691279

recall : 0.7792233361527925

f1 : 0.7730231297049723

Train results for sklearn Naive Bayes with wine dataset:

confusion_matrix:

[[993 314]

[151 702]]

accuracy : 0.7847222222222222

precision : 0.7794759374483784

recall : 0.7913664450864719

f1 : 0.7807426850376529

Train results for knn with wine dataset:

confusion_matrix:

[[1142. 165.]

[167. 686.]]

accuracy : 0.8462962962962963

precision : 0.8392660771177396

recall : 0.8389885466569675

f1 : 0.8391272889399546

Train results for sklearn knn with wine dataset:

confusion_matrix:

[[1142 165]

[167 686]]

accuracy : 0.8462962962962963

precision : 0.8392660771177396

recall : 0.8389885466569675

f1 : 0.8391265021320584

----------Abalone train----------

Train results for Naive Bayes with abalone dataset:

confusion_matrix:

[[86. 813. 320.]

[62. 771. 220.]

[37. 168. 865.]]

accuracy : 0.5152603231597845

precision : 0.5068639070016245

recall : 0.5370515259973194

f1 : 0.5215212384516231

Train results for knn with abalone dataset:

confusion_matrix:

[[847. 233. 139.]

[358. 624. 71.]

[169. 90. 811.]]

accuracy : 0.6828246558946739

precision : 0.6898968451103537

recall : 0.6817894490648574

f1 : 0.6858191875748081

Train results for sklearn knn with abalone dataset:

confusion_matrix:

[[847 233 139]

[358 624 71]

[169 90 811]]

accuracy : 0.6828246558946739

precision : 0.6898968451103537

recall : 0.6817894490648574

f1 : 0.6843342477008174

```
#############################################################################
##                                Evaluation on test dataset                 ##
#############################################################################
----------------------------------------Iris test----------------------------------------

  Test results for Naive Bayes with iris dataset:
  confusion_matrix:
  [[11.  0.  0.]
  [ 0.  9.  0.]
  [ 0.  0. 10.]]
  accuracy : 1.0
  precision : 1.0
  recall : 1.0
  f1 : 1.0


  Test results for sklearn Naive Bayes with iris dataset:
  confusion_matrix:
  [[11  0  0]
  [ 0  9  0]
  [ 0  0 10]]
  accuracy : 1.0
  precision : 1.0
  recall : 1.0
  f1 : 1.0


  Test results for knn with iris dataset:
  confusion_matrix:
```

[[11. 0. 0.]

[0. 8. 1.]

[0. 0. 10.]]

accuracy : 0.9666666666666667

precision : 0.9696969696969697

recall : 0.9629629629629629

f1 : 0.9663182346109176

Test results for sklearn knn with iris dataset:

confusion_matrix:

[[11 0 0]

[0 8 1]

[0 0 10]]

accuracy : 0.9666666666666667

precision : 0.9696969696969697

recall : 0.9629629629629629

f1 : 0.9645191409897292

----------wine test----------

Test results for Naive Bayes with wine dataset:

confusion_matrix:

[[264. 69.]

[28. 179.]]

accuracy : 0.8203703703703704

precision : 0.8129418912947415

recall : 0.8287635461548505

f1 : 0.8207764794808968

Test results for sklearn Naive Bayes with wine dataset:

confusion_matrix:

[[266 67]

[31 176]]

accuracy : 0.8185185185185185

precision : 0.809951365506921

recall : 0.8245201723462593

f1 : 0.8133333333333334

Test results for knn with wine dataset:

confusion_matrix:

[[275. 58.]

[57. 150.]]

accuracy : 0.7870370370370371

precision : 0.7747335495829472

recall : 0.7752317534926231

f1 : 0.7749825714691748

Test results for sklearn knn with wine dataset:

confusion_matrix:

[[275 58]

[57 150]]

accuracy : 0.7870370370370371

precision : 0.7747335495829472

recall : 0.7752317534926231

f1 : 0.7749796177189963

----------Abalone test----------

Test results for Naive Bayes with abalone dataset:

confusion_matrix:

[[22. 216. 71.]

[18. 181. 55.]

[5. 45. 222.]]

accuracy : 0.5089820359281437

precision : 0.5121073952716955

recall : 0.533324102262774

f1 : 0.5225004555334807

Test results for knn with abalone dataset:

confusion_matrix:

[[172. 94. 43.]

[136. 95. 23.]

[52. 27. 193.]]

accuracy : 0.5508982035928144

precision : 0.5542554459221126

recall : 0.5467362919226759

f1 : 0.5504701931271436

Test results for sklearn knn with abalone dataset:

confusion_matrix:

[[172 94 43]

[136 95 23]

[52 27 193]]

accuracy : 0.5508982035928144

precision : 0.5542554459221126

recall : 0.5467362919226759

f1 : 0.5484619794177524

Conclusion:

L'algorithme knn semble plus efficace que l'algorithme naive Bayes, mais prend beaucoup plus de temps lorsqu'il y a beaucoup de données. Cela est dû au fait que knn doit recalculer la distance de l'entrée à prédire avec toutes les autres données d'entraînement à chaque fois qu'il doit faire une nouvelle prediction. Par contre, les résultats de mon implémentation de l'algorithme naive Bayes, même s'ils sont très proches, sont parfois un peu moins performant que l'algorithme fourni avec la librairie scikit-learn.