IFT-7025

TP2

April 3, 2021

Marc HENRIOT
Adrien TURCHINI

Université Laval - Faculté des sciences et de génie

UNIVERSITÉ
LAVAL

1 Définitions

1.1 Précision

La précision ou valeur prédictive positive en statistique représente, dans un cas binaire, la proportion de documents correctement classés positifs qui le sont réellement sur tous les documents classés positifs. Cette mesure nous permet donc de voir, parmi les données classées positives, la proportion de données correctement classés soit celles qui sont réellement positives. Dans un cas non binaire, la précision pour une classe précise revient au nombre de données classées correctement dans cette classe sur le nombre de données classées dans la classe i soit ceux bien et mal classés.

1.2 Rappel

Le rappel, sensibilité ou true positive rate dans un cas binaire est le ratio des données correctement classés positives sur toutes les données réellement positives. Dans un cas non binaire cela revient au ratio des données correctement classés dans une classe i sur toutes les données appartenant à la classe i. Nous avons donc une mesure de la quantité de données classées pertinentes parmi toutes les données pertinentes. Cette mesure nous permet de tirer des conclusions lorsqu'elle est interprétée avec la spécificité.

1.3 F1-score

Le score F1 est la moyenne harmonique de la précision et du rappel (voir définitions au-dessus). Dans le cas ou l'on souhaite comparer plusieurs modèles donc certains ayant une meilleure précision alors que d'autres ont un meilleur rappel, le score F1 nous permet de prendre en compte ces deux mesures au sein d'une seule et de comparer les modèles.

1.4 Matrice de confusion

La matrice de confusion est une matrice carré représentant la classification des données. Dans un cadre binaire, sur les lignes on trouve les valeurs réelles des classes (positif et négatif) et sur les colonnes les valeurs attribuées (positif et négatif). La matrice de confusion nous permet donc très rapidement de voir le nombre de données correctement classées pour chaque classe et le nombre de données mal classées pour chaque classe également. Nous retrouvons donc pour une matrice de confusion dans un cas binaire le nombres de vrais positifs (classe positive classés positifs), de faux positifs (classe négative classés positifs), de vrais négatifs (classe négative classés négatifs) et de faux négatifs (classe positive classés négatifs).

La matrice de confusion nous permet de trouver les trois mesures précédentes. En effet nous avons :

$$précision = \frac{vrais\ positifs}{vrais\ positifs\ +\ faux\ positifs}$$

$$rappel = \frac{vrais\ positifs}{vrais\ positifs\ +\ faux\ négatifs}$$

$$F1\ score = 2 * \frac{précision\ *\ rappel}{précision\ +\ rappel} = \frac{2 *\ vrais\ positifs}{2 *\ vrais\ positifs\ +\ faux\ positifs\ +\ faux\ négatifs}$$

2 K plus proches voisins (K-nearest neighbors)

2.1 Métrique

Comme toutes les données sont des valeurs continues, la métrique choisit est la distance euclidienne. La ou une distance de Manhattan aurait été plus préférable si les donnée avait été des valeurs discrètes.

2.2 Pseudo-code

Algorithm 1 euclidean_distance

Require: $x1, x2$
Ensure: $len(x1) = len(x2)$
  $distance \leftarrow (((x1 - x2) * *2).sum()) * *0.5$

Algorithm 2 train

Require: $train, train\_labels$
Ensure: $len(train) = len(train\_labels)$
  $X\_train \leftarrow train$
  $y\_train \leftarrow train\_labels$
  $n\_class \leftarrow Nombre\ de\ classes\ dans\ train\_labels$

Algorithm 3 getNeighbors

Require: X_test_row
  $distances \leftarrow liste$
  for `X_train_row, y_train_row in zip(X_train, y_train)` **do**
    $distances.append([X\_train\_row, euclidean\_distance(X\_test\_row, X\_train\_row), y\_train\_row])$
  end for
  $distances.sort()$
  $K\_neighbors \leftarrow liste$
  for `i in range(K)` **do**
    $K\_neighbors.append(distances[i])$
  end for
  $retourne\ K\_neighbors$

Algorithm 4 predict

Require: X
  $y\_pred \leftarrow liste$
  for `X_test_row in X` **do**
    $K\_neighbors \leftarrow getNeighbors(X\_test\_row)$
    $classes \leftarrow liste\ des\ classes\ dans\ K\_neighbors$
    $prediction \leftarrow max(count(classes))$
    $y\_pred.append(prediction)$
  end for
  retourne y_pred

2.3 Choix de K

Pour la valeur de K, nous utilisons une approche par validation croisée. Cela permet de déterminer la meilleure valeur pour K. Pour définir K_{min} et K_{max} nous appliquons la règle du pouce, $K_{min} = 1$ donc et $K_{max} = min(0.1 * len(y), 50)$. Ce maximum de 50 est arbitraire, elle sauve du temps de calcul déjà très long pour une validation croisée. D'autant plus que KNN est long en complexité temporelle $O(n + m)$ (n taille train, m taille de test) cela se remarque sur les jeux de données abalones et Vinho Verde. En revanche les 10% de la taille des datas et la pour limiter le sur apprentissage sur des data-sets se plus petites taille comme iris.

Le processus de validation croisée est celui décrit dans l'énoncé du TP et peut être trouvé das la fonction *cv_knn*() dans *entrainer_tester.py*

Pour différencier les valeurs de K nous nous basons sur la précision moyenne du modèle sur n-splits le meilleur score moyen est retenu ainsi que la valeur de K correspondante.

Après compilation du programme nous avons :

$$K_{iris} = 8$$

$$K_{wine} = 37$$

$$K_{abalone} = 9$$

2.4 Comparaison avec sklearn

2.4.1 Resultat du programme

On voit que le modèle performe très bien sur Iris, il n'y a que 2 classes mal classé dans le test et une sur le train. Tout les scores sont du meme ordre de grandeurs, le modèle se comporte très biens sur Iris.

Le modèle a plus de mal sur le dataset wine malgré $K = 37$, on voit qu'il y a beaucoup de faux positif dans le train comme dans le test, cela se retrouve dans la précision qui est plus basse que le rappelle. Le modèle n'est sûrement pas adapté a ce problème de classification.

Sur Abalone, l'accuracy est relativement bonne par rapport à l'ensemble, mais la précision n'est vraiment pas bonne comparé à Iris par exemple. Le modèle performe bien de manière générale, mais n'est pas très spécifique.

```
---------- IRIS TEST ----------
mean_accuracy
 0.9555555555555556
mean_precision
 0.9521531100478469
mean_recall
 0.9521531100478469
mean_F1-score
 0.9521531100478469
Confusion_matrix
 [[15  0  0]
 [ 0 10  1]
 [ 0  1 18]]
---------- WINE TEST ----------
mean_accuracy
 0.7358024691358025
mean_precision
 0.7257798855111288
mean_recall
 0.7314183055252947
mean_F1-score
 0.7276669285153181
Confusion_matrix
 [[368  93]
 [121 228]]
---------- ABALONE TEST ----------
mean_accuracy
 0.8387869114126097
mean_precision
 0.5664706576521291
mean_recall
 0.7116518272650261
mean_F1-score
 0.6108075362792343
Confusion_matrix
 [[ 64  59   0]
 [ 23 959  24]
 [  0  96  28]]
```

```
---------- IRIS TRAIN ----------
mean_accuracy
 0.9904761904761905
mean_precision
 0.989247311827957
mean_recall
 0.9916666666666667
mean_F1-score
 0.9903161098429827
Confusion_matrix
 [[35  0  0]
 [ 0 39  0]
 [ 0  1 30]]
---------- WINE TRAIN ----------
mean_accuracy
 0.7555555555555555
mean_precision
 0.7197689524484384
mean_recall
 0.7450951175349132
mean_F1-score
 0.72713125
Confusion_matrix
 [[1019  160]
 [ 302  409]]
---------- ABALONE TRAIN ----------
mean_accuracy
 0.8460485802257954
mean_precision
 0.6473660177013979
mean_recall
 0.8040446767439214
mean_F1-score
 0.6966743989872196
Confusion_matrix
 [[ 205  120    0]
 [  54 2139   40]
 [   0  236  129]]
```

2.4.2 Resultat de sklearn

Pour la comparaison avec sklearn nous entraînons 3 kNN différents (un pour chaque jeu de données) avec les valeurs de K trouvé lors de la validation croisée. La comparaison est faite sur l'ensemble de tests.

```
sklearn matrice de confusion iris :
 [[15  0  0]
 [ 0 10  1]
 [ 0  1 18]]
sklearn précision iris = 0.9555555555555556
sklearn matrice de confusion iris :
 [[368  93]
 [121 228]]
sklearn précision iris = 0.7358024691358025
sklearn matrice de confusion iris :
 [[ 64  59   0]
 [ 23 959  24]
 [  0  96  28]]
sklearn précision iris = 0.8387869114126097
```

Cette partie va être très rapide, nous obtenons les même résultats avec sklearn à la classe près. Ce qui valide notre algorithme de classification.

3 Classification Naïve Bayésienne

3.1 Pseudo-code de l'algorithme

Algorithm 5 Classifieur Bayésien Naïf

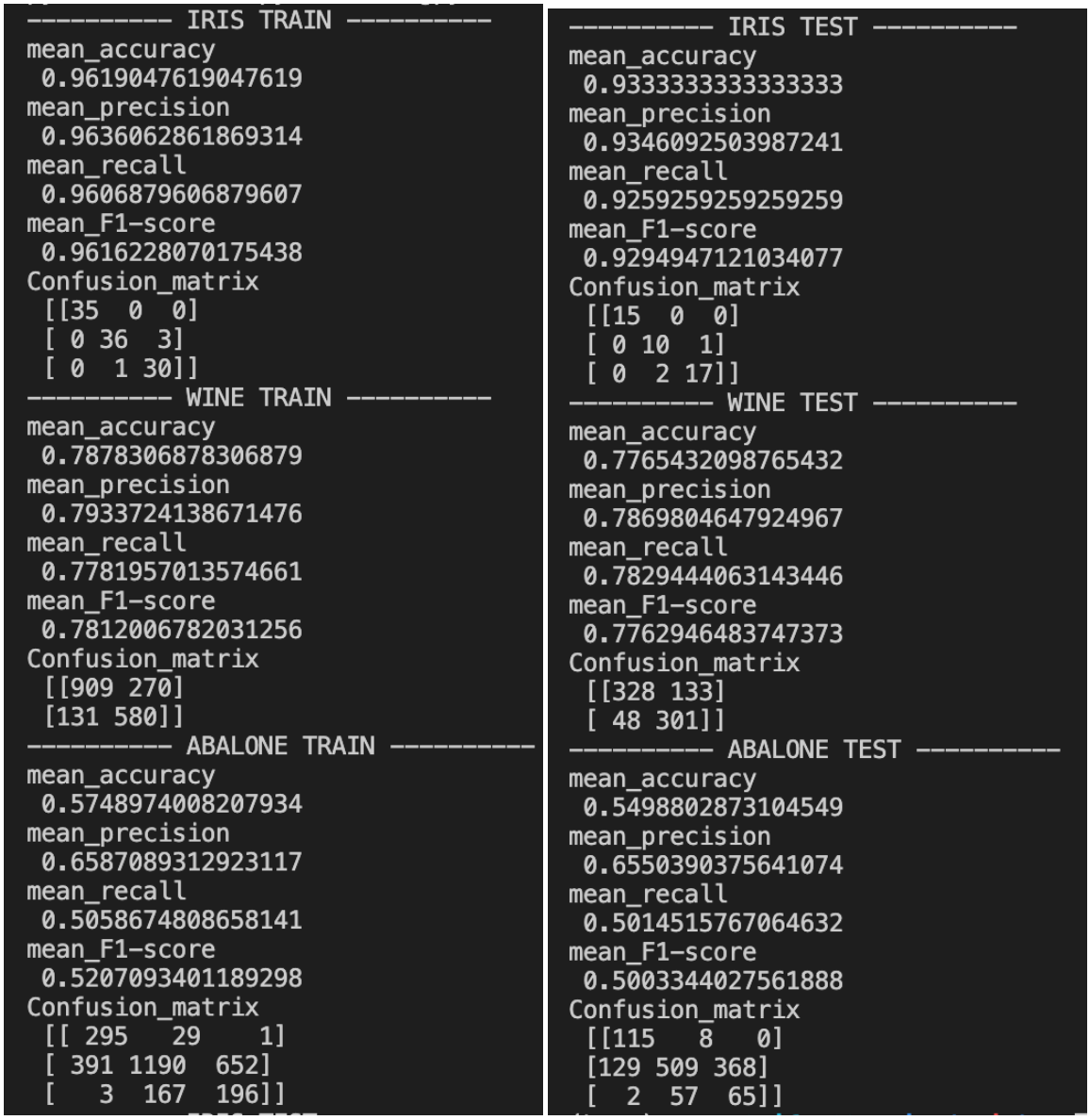
Require: *X_train*, *y_train*, *X_test*
Lire X_train et y_train.
Pour chaque classe calculer sa fréquence dans le dataset entier.
Dans X_train, pour chaque variables prédictives de chaque classe, calculer sa moyenne et sa variance.
y_test_pred $\leftarrow$ *tableau*
for *Chaque donnée du jeu de test* **do**
  for *Chaque classe* **do**
    for *Chaque attribut* **do**
      Utiliser la fonction gaussienne pour calculer les probabilités que l'attribut appartienne à la classe
    end for
    Calculer la vraisemblance que la donnée appartienne à la classe par la formule de Bayes
  end for
  Trouver la classe qui maximise la vraisemblance
  Ajouter à y_pred la classe attribuée
end for
return *y_pred*

3.2 Évaluation des résultats

On remarque que notre classifieur performe très bien sur Iris avec 0.96 de score F1 sur le jeu d'entraînement et 0.92 de score F1 sur le jeu de test. Lorsqu'on passe sur le dataset Wine nos performances descendent avec tout de même 0.78 de score F1 pour le jeu d'entraînement et 0.77 de score F1 pour le jeu de test. Cependant lorsqu'on passe au dataset abalone, notre score F1 pour le jeu d'entraînement et de test est respectivement de 0.52 et 0.50.

La performance de notre classifieur est donc intimement reliée aux types données utilisée. Pour iris avec 3 classes il est très performant mais pour abalone qui se rapporte plus à un problème de regression linéaire, il n'est pas bon.

On remarque que le classifieur nous fourni des mesures très similaires pour le rappel, l'accuracy, la précision et la score F1. Notre modèle performe donc bien de manière générale sur des données adaptées à ce type de problème.



3.3 Comparaison avec Scikit-Learn

Nous avons également programmé un classifieur Bayesien Naif Gaussien en utilisant la bibliothèque Scikit Learn. Nous observons à peu de chose près les même résultats que notre classifieur fait à la main. La plus grande différence que nous trouvons se trouve dans le jeu de donnée de test d'abalone ou nous avons 0.56 de score F1 en utilisant le modèle sklearn et 0.50 en utilisant notre modèle maison. On retrouve par exemple la même accuracy pour le jeu de test d'iris et le jeu de test abalone avec les deux modèles.

On peut donc conclure que notre classifieur est donc performant car presque aussi bon que celui de scikit learn.