Définitions des métriques

Un cas est positif lorsque l'instance est dans la classe qu'on teste.

Précision

Pour les problèmes de classification binaire, la précision est le pourcentage des cas positifs prédits qui sont bien classifiés (vrais positifs) par rapport à tous les cas positifs prédits. C'est donc une mesure pour savoir à quel point on se trompe lorsqu'on prédit un cas positif.

Pour les problèmes de classification ayant plus de deux étiquettes de classes, on peut utiliser l'approche de un-contre-tous. On calcule donc la précision de la classification binaire en utilisant une seule classe à la fois. La précision de la première classe serait donc lorsque cette classe a été prédit comme vrai et que c'était bien le cas par rapport à tous les cas positifs. \ Après avoir fait le calcul pour chaque classe, on peut faire la moyenne pondérée des précisions de chaque classe pour avoir la prédiction du modèle. La pondération se fait sur le nombre d'observation de chaque classe.

Rappel

Pour les problèmes de classification binaire, le rappel est le pourcentage des cas positif qui sont bien classifiés par rapport à tous les exemples qui sont réellement des cas positifs. C'est donc une mesure pour savoir à quel point on prédit bien les cas positifs.

Pour les problèmes de classification ayant plus de deux étiquettes de classes, on utilise l'approche de un-contre-tous de la même façon que pour la précision, mais en utilisant le calcul du rappel.

F1-score

Le F1-score est la moyenne harmonique de la précision et du rappel. Cela permet d'avoir un équilibre entre ces deux mesures. Le F1-score a donc le même objectif que l'exactitude, mais il est moins sensible aux cas où la classe à prédire est rarement présente dans les instances.

On utilise encore la même procédure pour les problèmes de classification ayant plus de deux étiquettes de classes.

Matrice de confusion

Pour les problèmes de classification binaire, la matrice de confusion contient deux lignes et deux colonnes. Elle sert à visualiser la performance d'un modèle.
Les lignes indiquent le nombre d'instance qui sont réellement dans la classe et le nombres d'instance qui ne sont pas dans la classe.
Les colonnes indiquent le nombre d'instance qui sont prédites dans la classe et le nombres d'instance qui ne sont pas prédits dans la classe.
On a donc le nombre de vrais positifs (VP) dans le coin supérieur gauche, le nombre de vrais négatifs (VN) dans le coin inférieur droit, le nombre de faux négatifs (FN) dans le coin supérieur droit et le nombre de faux positifs (FP) dans le coin inférieur gauche.
Une instance avec un vrai positif est lorsqu'on la prédit comme étant dans la classe alors

que cette instance est réellement dans cette classe.
Une instance avec un vrai négatif est lorsqu'on la prédit comme n'étant pas dans la classe alors que cette instance n'est pas réellement dans cette classe.
Une instance avec un faux positif est lorsqu'on la prédit comme étant dans la classe alors que cette instance n'est pas réellement dans cette classe.
Une instance avec un vrai négatif est lorsqu'on la prédit comme n'étant pas dans la classe alors que cette instance est réellement dans cette classe.

On peut calculer l'exactitude, la précision, le rappel et le F1-score à partir de la matrice de confusion.

Pour calculer l'exactitude, on additionne le nombre de vrai positif et de faux positif et on divise la somme par le nombre total d'instance. Le nombre total d'instance est la somme de tout les éléments de la matrice.

$$Exactitude = \frac{TP + TN}{TP + TN + FP + FN}$$

Pour calculer la précision, on divise le nombre de vrai positif par la somme des éléments de la première colonne de la matrice. Cette première colonne est la somme du nombre de vrai positif et de faux positif.

$$Précision = \frac{TP}{TP + FP}$$

Pour calculer le rappel, on divise le nombre de vrai positif par la somme des éléments de la première ligne de la matrice. Cette première ligne est la somme du nombre de vrai positif et de faux négatif.

$$Rappel = \frac{TP}{TP + FN}$$

Pour calculer le F1-score, on peut utiliser les résultats de la précision et du rappel, mais on peut aussi utiliser directement la matrice de confusion. Pour se faire, on utile la fonction suivant :

$$F_1 = \frac{TP}{TP + 0.5 * (FP + FN)}$$

Pour les problèmes de classification ayant plus de deux étiquettes de classes, on peut utiliser l'approche un-contre-tous. Dans ce cas, on a autant de matrice de confusion qu'on a de classes, car on fait une matrice par classe. On essaie donc de prédire si l'instance est dans la classe ou pas pour chaque classe. Les métriques sont calculées sur chaque matrices et on peut faire les moyennes pondérées pour avoir les performances du modèle.

K plus proches voisins

La métrique de distance utilisée est la distance euclidienne parce qu'on n'a pas un grand nombre d'attributs et ils sont tous numériques.

Pseudo-code

function train(train_data, train_labels)
    Save train_data and train_labels in sample list

function predict(x)
    Compute the Euclidean distance between x and all my points in my sample list
    Find the k nearest neighbors
    return the most popular label among the k nearest neighbors

Comparaison de notre implémentation avec scikit-learn

Nous obtenons quasiment les mêmes résultats pour les deux implémentations.

Classification Naïve Bayésienne

Nous avons décidé de faire la classifieur Naïf Bayésien Gaussien puisque la plupart des attributs sont numériques.

Pseudo-code

Fonction train

function train(train_data, train_labels)
    prior = [ ]
    For each unique class ∈ train_labels
        #Calcul des probabilités a priori
        Add to prior $\frac{number\ of\ instance\ in\ class}{total\ number\ of\ instance}$

    Initiate means, a matrix of number of class x number of features
    Initiate std, a matrix of number of class x number of features
    For each unique class ∈ train_labels
        cond = train_data for which the label is equal to class
        For each feature ∈ train_data
            means[class_position, feature_position] = mean of cond containing feature
            std[class_position, feature_position] = standard deviation of cond containing feature
            End For
        End For

Fonction predict

function predict(x)
    Initiate posterior_probs, a vector with a length of number of class
    For each unique class ∈ train_labels
        gaussian_probs = Gaussian density probabilities with x and the means and std associated with the class
        posterior_probs[class_position] = sum(log(gaussian_probs)) + log of the prior associated with the class
        End For
    Return the class with the maximum posterior_probs

Comparaison de notre implémentation avec scikit-learn

Pour les trois ensembles de données, on obtient exactement les mêmes résultats pour toutes les métriques avec notre implémentation qu'avec le classifieur de scikit-learn.

Discussion et comparaison entre les deux techniques d'apprentissage

	Iris		Wine		Abalone	
	KNN	CNB	KNN	CNB	KNN	CNB
Accuracy	91.30%	89.41%	84.94%	80.74%	89.31%	61.37%
Precision	90.00%	88.93%	88.52%	71.20%	73.28%	74.08%
Recall	85.71%	82.61%	86.75%	83.97%	77.73%	56.34%
F1-Score	85.03%	81.06%	87.63%	77.06%	75.32%	59.77%
Eval Time	78ms	32ms	31667ms	318ms	82249ms	648ms

On observe dans le tableau, les résultats des deux techniques d'apprentissage sur les données tests des trois jeux de données Iris, Wine et Abalone.

Pour le jeu de donnée Iris, le KNN et le CNB on des performances plutôt similaire. Cependant, le CNB obtient des résultats légèrement inférieurs pour toutes les métriques.

Pour le jeu de donnée Wine, la différence de performance entre les deux techniques est plus grande. Le KNN performe beaucoup mieux, surtout en terme de précision où il a une précision plus grande de plus de 14% par rapport au classifieur naïf Bayésien.

Pour le jeu de donnée Abalone, l'écart de performance est encore plus grande que dans les autres jeux de données. L'exactitude, le rappel et le F1-score sont beaucoup plus élevé pour le KNN, cependant la précision est légèrement meilleure pour le CNB.

La performance plus médiocre du classifieur naïf dans abalone est probablement expliquée par le fait que les attributs ne sont pas conditionnellement indépendants selon la classe. En effet, il serait surprenant que le poids des ormeaux soit conditionnellement indépendant de leurs dimensions. Il en est de même pour plusieurs autres attributs. Le KNN n'a pas la forte hypothèse d'indépendance conditionnelle donc ça ne nuit pas à ses performances.

Dans les trois jeu données, le classifieur naif de Bayes est beaucoup plus rapide que les k plus proches voisins.

Conclusion

Pour le jeu de données iris, le classifieur naif de Bayes est préférable aux k plus proches voisins car le temps d'exécution de l'algorithme du CNB est largement inférieur à celui du KNN pour une différence minime dans les performances observées. Pour les jeux de données Wine et Abalone, il est préférable d'utiliser le KNN car cette technique performe beaucoup mieux que le classifieur naïf de Bayes. La différence de performance vaut probablement la différence dans le temps d'exécution.