LANGLADE Alexandre
LOUAHADJ Aniss

TP1 - Machine Learning

Performances avec ou sans prétraitement

Résultats sans prétraitement :

```
Nombre fichiers pour apprentissage :  900
Nombre fichiers pour test :  100
Algorithme des centres mobiles
Bonnes réponses (environ) :  100.00 %

Perceptron multicouche
Bonnes réponses :  100.00 %

Separateur à Vaste Marge
Bonnes réponses :  100.00 %

Forêts d'arbres décisionnels
Bonnes réponses :  100.00 %
```

```
Nombre fichiers pour apprentissage :  100
Nombre fichiers pour test :  900
Algorithme des centres mobiles
Bonnes réponses (environ) :  84.00 %

Perceptron multicouche
Bonnes réponses :  97.22 %

Separateur à Vaste Marge
Bonnes réponses :  99.00 %

Forêts d'arbres décisionnels
Bonnes réponses :  95.67 %
```

Résultats avec ACP :

```
Nombre fichiers pour apprentissage :  900
Nombre fichiers pour test :  100
n_components pca =  7

Algorithme des centres mobiles
Bonnes réponses (environ) :  100.00 %

Perceptron multicouche
Bonnes réponses :  100.00 %

Separateur à Vaste Marge
Bonnes réponses :  100.00 %

Forêts d'arbres décisionnels
Bonnes réponses :  99.00 %
```

```
Nombre fichiers pour apprentissage :  100
Nombre fichiers pour test :  900
n_components pca =  7

Algorithme des centres mobiles
Bonnes réponses (environ) :  82.00 %

Perceptron multicouche
Bonnes réponses :  96.00 %

Separateur à Vaste Marge
Bonnes réponses :  99.11 %

Forêts d'arbres décisionnels
Bonnes réponses :  97.56 %
```

Ici nous avons appliqué un ACP avec le paramètre n_components valant 7, les vecteurs passent donc d'une dimension 13 à une dimension 7 et ainsi les calculs sont plus rapides.
Nous remarquons que pour une base d'apprentissage grande ou petite, la différence d'efficacité des algorithmes est négligeable par rapport à une absence de prétraitement. C'est donc une technique très intéressante mais il faut cependant rester vigilant puisque la perte de certaines informations pour réduire le temps de calcul peut dans certains cas pratiques amener à des résultats faux.

Choix du nombre de coefficients cepstraux retenus pour la description des individus

Nous avons laissé les paramètres par défaut pour l'exécution de la fonction mfcc qui renvoie donc 13 coefficients cepstraux. Ensuite nous avons regardé les ratios de variance de ces 13 coefficients et nous avons décidé d'en garder 7 et d'éliminer ceux avec un ratio de variance de l'ordre de 10^(-3) et plus petit. Nous avons donc gardé les 7 coefficients qui présentaient les variances les plus élevées.

```
Nombre fichiers pour apprentissage :  900
Nombre fichiers pour test :  100
n_components pca =  7

Algorithme des centres mobiles
Bonnes réponses (environ) :  100.00 %

Perceptron multicouche
Bonnes réponses :  100.00 %

Separateur à Vaste Marge
Bonnes réponses :  100.00 %

Forêts d'arbres décisionnels
Bonnes réponses :  99.00 %
```

```
Nombre fichiers pour apprentissage :  900
Nombre fichiers pour test :  100
n_components pca =  4

Algorithme des centres mobiles
Bonnes réponses (environ) :  98.00 %

Perceptron multicouche
Bonnes réponses :  98.00 %

Separateur à Vaste Marge
Bonnes réponses :  99.00 %

Forêts d'arbres décisionnels
Bonnes réponses :  98.00 %
```

On remarque cependant que les taux de réussites des algorithmes stagnent autour de 100% jusqu'à n_components = 5 puis baissent à 98% pour un n_components = 4 et enfin chutent dans les 80% pour un n_components = 3 avec notamment le perceptron multicouche qui n'arrive plus à attendre son état de convergence avec 200 itérations.

Performances des différentes méthodes de classification étudiées

Dans ce TP nous avons mis en pratique 4 méthodes d'apprentissages :

- Algorithme des centres mobiles
- Perceptron multicouche
- Séparateur à Vaste Marge
- Forêts d'arbres décisionnels

Pour implémenter ces algorithmes, nous avons utilisé la librairie sklearn.
Nous avons laissé les hyperparamètres des 4 agents comme sklearn les déclare par défaut et c'est dans ce cadre que nous allons comparer les résultats que l'on obtient.

Avant tout, voilà comment est défini le taux de réussite dans notre code:

```
reussite = metrics.confusion_matrix(liste_classe_test, y_predict)
taux = 0
for i in range(10):
    taux += reussite[i][i]
taux = taux*100/(10*pourcentage_test)
```

Cette façon de faire ne fonctionnent pas sur les centres mobiles puisque les labels de classe ne sont pas choisis, nous avons utilisé pour cette méthode les calculs suivants :

```
reussite = metrics.confusion_matrix(liste_classe_test, y_predict)
taux = 100
for i in range(10):
    aux = 0
    for j in range(10):
        if reussite[j][i]>aux:
            aux = reussite[j][i]
    if aux<pourcentage_test:
        taux += aux-pourcentage_test
```

Dans les 2 cas, pourcentage_test représente le pourcentage des fichiers de notre base de départ de 1000 fichiers qui sont conservés pour former la base de test.

```
Nombre fichiers pour apprentissage :  900
Nombre fichiers pour test :  100
n_components pca =  7

Algorithme des centres mobiles
Bonnes réponses (environ) :  100.00 %

Perceptron multicouche
Bonnes réponses :  100.00 %

Separateur à Vaste Marge
Bonnes réponses :  100.00 %

Forêts d'arbres décisionnels
Bonnes réponses :  99.00 %
```

```
Nombre fichiers pour apprentissage :  100
Nombre fichiers pour test :  900
n_components pca =  7

Algorithme des centres mobiles
Bonnes réponses (environ) :  82.00 %

Perceptron multicouche
Bonnes réponses :  98.11 %

Separateur à Vaste Marge
Bonnes réponses :  99.11 %

Forêts d'arbres décisionnels
Bonnes réponses :  98.00 %
```

Dans les 2 expériences nous avons gardé un prétraitement avec n_components qui vaut 7 comme expliqué précédemment mais la différence se fait sur le nombre de fichiers dans la base d'apprentissage. Dans un cas nous avons 900 fichiers et dans l'autre nous en avons 100. Nous remarquons que l'algorithme le plus impacté par ce changement est celui des centres mobiles, les autres restent cependant très décents.

Conclusion

Dans ce TP nous disposons de suffisamment de fichiers pour que l'algorithme des centres mobiles se montre aussi efficace que les 3 autres pour une base de test à 10%, les résultats de nos algorithmes sont très prometteurs pour un n_components à 7 puisqu'ils ne font quasiment aucune faute. Ainsi pour en choisir un seul à garder, on devrait comparer la vitesse d'exécution ou essayer de modifier les différents hyperparamètres des agents pour vérifier si on peut trouver une meilleure optimisation de l'utilisation des ressources et du temps de calcul.