SY19 TP2

Introduction

Dans ce TP, nous disposons de trois jeux de données. Le premier représente des expressions de visage, le second des lettres de l'alphabet et le troisième des sons. Le but est alors de trouver des classifieurs efficaces pour ces trois jeux de données en utilisant des méthodes d'apprentissage supervisé.

Character

Analyse

Nous commençons dans un premier temps par analyser notre jeu de données. Grâce à la commande 'summary', nous pouvons voir que chaque élément est décrit par 16 variables quantitatives et que nos individus vont logiquement être divisés en 26 classes représentant l'alphabet. On note également que les variables semblent issues d'une loi centrée autour de 0. Enfin les individus sont repartis assez équitablement dans les differentes classes.

```
table(character_data$Y)
```

```
##
##   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q   R
## 399 345 367 397 401 418 390 347 372 366 382 381 397 367 396 420 412 360
##   S   T   U   V   W   X   Y   Z
## 366 367 404 392 383 414 395 362
```

Approche

Pour trouver le meilleur classifieur pour ces données, nous allons appliquer plusieurs méthodes étudiées en cours. Nous pourrons alors comparer l'efficacité de ces méthodes en comparant l'erreur de chaque classifieur fournie par validation croisée. Nous allons détailler certaines méthodes ci-dessous et l'ensemble des méthodes appliquées avec leurs résultats seront présentées ultérieurement dans un tableau récapitulatif.

Application

Dans un premier temps, nous allons appliquer simplement plusieurs méthodes conjointement avec la validation croisée pour obtenir des résultats comparrables et significatifs. Nous commençons donc par diviser nos données en deux parties, un ensemble train comportant les deux tiers des données et un ensemble de test comportant le reste.

Une fois la séparation faite, nous mettons en place une validation croisée. Nous divisons alors nos données en dix parties de même taille. Ainsi nous pouvons entrainer notre modèle sur une partie de nos données et ensuite prédire les données de test. On fait alors une moyenne des résultats que nous avons obtenu pour obtenir une erreur stable et significative qu'on utilisera pour comparer nos differentes méthodes et ensuite choisir celle qui nous donne la plus petite erreur.

Après avoir testé nos differents modèles, nous avons également mis en place une méthode de réduction de la dimension (ACP). Les données étant déjà centrées autour de 0 et avec des valeurs assez proches, nous n'avons ni besoin de normaliser ces données ni de les redimensionner avant d'appliquer l'ACP.

Pour ce jeu de données, c'est le modèle du random foret qui a donné le meilleur résultat.

```
n_folds <- 10
folds_i <- sample(rep(1:n_folds, length.out = n))
CV<-rep(0,10)
for (k in 1:n_folds) {
  test_i <- which(folds_i == k)
  train_xy <- character[-test_i, ]
  test_xy <- character[test_i, ]
  rf <- randomForest(Y ~ ., data = train_xy)
  pred_rf<-predict(rf, newdata = test_xy, type = "response")
  prop.table(table(test_xy$Y,pred_rf))
  cm= as.matrix(table(test_xy$Y,pred_rf))
  CV[k]<- sum(diag(cm)) / sum(cm)
}
CVerror= sum(CV)/length(CV)
```

En effet, pour ce jeu de données les méthodes de résolution linéaire sont moins efficaces. C'est pour cette raison que le LDA a des performances moindres comparé au random forest qui est moins dépendant des variables. De même on peut remarquer que le QDA nous donne de meilleures performances que le LDA. Ceci est probablement expliqué par le fait que les variables n'ont pas les mêmes matrices de variance-covariance. Dans ce cas, les méthodes linéaires comme le LDA sont beaucoup moins efficaces pour differencier les differentes classes. Dans ce cas de figure, les méthodes quadratiques comme le QDA ainsi que d'autres méthodes moins dépendantes de cette caractéristique comme le random forest nous donne de meilleurs résultats.

Résultats

Voici les résultats de précision obtenus pour les differentes méthodes testées:

- RandomForest: 0.9343
- SVM: 0.913
- Naive-Bayes: 0.69
- LDA: 0.70
- QDA: 0.88
- RDA: 0.87
- SVM + PCA: 0.71
- Naive-Bayes + PCA: 0.64
- LDA + PCA: 0.65
- QDA + PCA: 0.68

Comme prévu, les méthodes simples et linéaires sont celles qui nous donnent les classifieurs les moins précis. Naive-Bayes est ici peu performant ce qui pourrait s'expliquer par une trop grande corrélation entre plusieurs prédicteurs.
Nous avons alors ensuite ajouté une partie de traitement des données avec l'ACP. En effet, notamment pour améliorer les résultats de modèles comme Naive-Bayes nous avons appliqué l'ACP pour réduire les dimensions de notre jeu de données. Cependant, quel que soit le modèle auquel on a appliqué l'ACP, le résultat de notre classifieur devenait moins bon. Ceci s'explique par le fonctionnement de ce traitement. En effet, l'ACP prend en compte seulement les coordonnées des points de nos données. L'ACP peut donc supprimer des informations qui sont pourtant importantes pour classifier nos données. En fonction de notre jeu de données ce processus de construction peut mener à de mauvaises composantes n'expliquant pas bien nos classes et donc résultant en des classifieurs moins performants. C'est pour cette raison que cette phase de traitement n'a pas été maintenue pour notre classifieur final.

Conclusion

Après le test de nos différents modèles et même de l'ajout d'une phase de traitement des données, les résultats obtenus par validation croisée nous permet de choisir le meilleur classifieur pour ce jeu de données. Ainsi c'est le randomForest qui nous donne le meilleur résultat avec une erreur de seulement 6.6% ce qui est assez satisfaisant.

Paroles

Analyse

Nous commençons dans un premier temps par analyser notre jeu de données. Grâce à la commande 'summary', nous pouvons voir que chaque élément est décrit par 16 variables quantitatives et que nos individus vont logiquement être divisés en 26 classes représentant l'alphabet. On note également que les variables semblent issues d'une loi centrée autour de 0. Enfin les individus sont repartis assez équitablement dans les differentes classes.