Stacking

Combinaison de modèles prédictifs

Ricco Rakotomalala

Université Lumière Lyon 2

Principe de la combinaison de modèles

- La combinaison d'un pool de modèles peut s'avérer plus performant que chacun d'entre eux pris individuellement.

- Ce sera d'autant plus vrai que les modèles sont individuellement performants, tout en étant très différents les uns des autres c.-à-d. se complètent, ne classent pas de la même manière (ou se trompent sur des observations distinctes).

- On peut appliquer le même algorithme sur des versions modifiées des données d'apprentissage (ex. bagging [tirage avec remise], boosting [pondération des individus]).

- On peut appliquer sur les mêmes données d'apprentissage des familles d'algorithmes différentes et/ou le même algorithme mais avec des paramétrages permettant d'avoir un comportement hétérogène.

Données et modèles utilisés pour les illustrations

Données « sonar », reconnaître le type d'un objet (roche ou mine) à partir des mesures d'un appareil. Apprentissage : 108 obs. ; Test : 100 obs.

Trois types de modèles prédictifs : Arbre de décision, Analyse discriminante linéaire, SVM Noyau RBF. Avec chacun, un taux de reconnaissance de 71%, 70% et 59%.

```
#charger les données
import pandas
sonar = pandas.read_excel("sonar.xlsx",header=0)
#données : descripteurs + cible
D = sonar.values
X = D[:,1:]
y = D[:,0]
#subdivision app-test
from sklearn import model_selection
XTrain,XTest,yTrain,yTest =
model_selection.train_test_split(X,y,train_size=108,random_state=1)
#arbre de décision
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(random_state=1)
dt.fit(XTrain,yTrain)
from sklearn import metrics
metrics.accuracy_score(yTest,dt.predict(XTest)) #71%
#analyse discriminante
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis()
lda.fit(XTrain,yTrain)
metrics.accuracy_score(yTest,lda.predict(XTest)) #70%
#SVM - noyau RBF
from sklearn.svm import SVC
svm = SVC(kernel="rbf", gamma="auto",random_state=1, probability=True)
svm.fit(XTrain,yTrain)
metrics.accuracy_score(yTest,svm.predict(XTest)) #59%
```

Plan

1. Combinaison par vote simple
2. Combinaison par vote pondéré
3. Stacking
4. Conclusion
5. Références

Faire voter les modèles lors de la phase de prédiction

COMBINAISON PAR VOTE SIMPLE

Principe du vote à la majorité simple (VotingClassifier sous Scikit-Learn)

- Sur les mêmes données d'apprentissage, construire indépendamment un ensemble de modèles prédictifs (ex. arbre de décision, analyse discriminante, k-plus proches voisins). Il faut que les modèles aient des caractéristiques très différentes.
- En prédiction, utiliser un vote simple des prédictions pour attribuer les classes aux individus supplémentaires.

```
#librairie ensemble de modèles
from sklearn.ensemble import VotingClassifier

#modèles individuels - instances
dt = DecisionTreeClassifier(random_state=1)
lda = LinearDiscriminantAnalysis()
svm = SVC(kernel="rbf", gamma="auto",random_state=1,probability=True)

#construction du meta-modèle par vote "hard"
groupe = VotingClassifier(estimators=[('arbre',dt),
('linearDA',lda),('SVMRbf',svm)], voting="hard")

#apprentissage
groupe.fit(XTrain,yTrain)

#prediction + évaluation en test
metrics.accuracy_score(yTest,groupe.predict(XTest))
```

Taux de reconnaissance 78%, meilleur que le meilleur des modèles (qui était l'arbre avec 71%)

Variante : s'appuyer sur les probabilités d'appartenance aux classes

- Même démarche sauf qu'au moment de la prédiction, calculer les probabilités d'affectation et faire leur somme pour décider de la classe prédite.
- N'est vraiment intéressante que si les estimations des probabilités sont bien calibrées c.-à-d. sont réparties de la même manière entre [0, 1], comparables d'un modèle à l'autre (si certains produisent des probas autour de 0.5 systématiquement, d'autres des probas très tranchées proches de 0 ou 1, l'agrégation est faussée).

```
#construction du meta-modèle par vote "soft"
groupeBis =
VotingClassifier(estimators=[('arbre',dt),('linearDA',
lda),('SVMRbf',svm)], voting="soft")

#apprentissage
groupeBis.fit(XTrain,yTrain)

#prediction + évaluation en test
metrics.accuracy_score(yTest,groupeBis.predict(XTest))
```

Taux de reconnaissance 73%, l'amélioration est moindre que pour "hard".

Accorder une importance différenciée aux classifieurs

COMBINAISON PAR VOTE PONDÉRÉ

Donner des poids différents aux modèles prédictifs

- Accorder des poids différents aux modèles.
- Les poids peuvent être indexés sur leurs performances individuelles par exemple. Mais en réalité, on ne sait pas vraiment, rappelons que le gain provient aussi de la diversité des modèles.

```
#modèles
dt = DecisionTreeClassifier(random_state=1)
lda = LinearDiscriminantAnalysis()
svm = SVC(kernel="rbf",
gamma="auto",random_state=1,probability=True)

#agrégateur avec vote pondéré
groupeTer =
VotingClassifier(estimators=[('arbre',dt),('linearDA',
lda),('SVMRbf',svm)], voting="hard",weights=[5,5,1])

#apprentissage
groupeTer.fit(XTrain,yTrain)

#prediction + évaluation en test
metrics.accuracy_score(yTest,groupeTer.predict(XTest))
```

Taux de reconnaissance 78%, autant que le vote simple finalement, mais étais-ce la bonne combinaison de poids à utiliser ?

Une manière d'apprendre les pondérations à partir des données

STACKING

Principe du stacking

- Toujours construire différents modèles de manière indépendante sur les données.
- S'appuyer sur ces modèles pour élaborer un jeu de données intermédiaire où leurs prédictions constituent les variables explicatives, les étiquettes observées constituent toujours la variable cible.
- On construit alors un classifieur à partir de ces données. Nous avons un modèle de modèles, un métamodèle.
- Si nous construisons le métamodèle à partir d'une régression logistique, les coefficients correspondent aux pondérations des modèles intermédiaires, apprises à partir des données.
- En prédiction, nous calculons la prédiction de chaque modèle et nous appliquons la pondération (si rég. logistique) du méta-modèle.
- On peut affiner l'idée en utilisant les probabilités d'appartenance plutôt que les prédictions brutes, avec toujours la réserve concernant leur calibration.

Stacking en pratique

- On ne peut pas utiliser les prédictions sur l'échantillon d'apprentissage pour construire les nouvelles explicatives, ce serait favoriser les modèles qui font du surapprentissage (prédictions toujours correctes en resubstitution)
- On pourrait réserver un troisième échantillon dédié à la construction du métamodèle (une sorte d'échantillon de validation). Mais l'approche est gourmande en données.
- En pratique, on utilise surtout la validation croisée pour constituer l'échantillon intermédiaire.

Algorithme (M modèles)

Découper les données d'apprentissage (n obs.) en K blocs de tailles (à peu près) identiques ($n_k = n/K$)

Pour k = 1 à K
  Construire les M modèles sur les (K-1) blocs
  Prédire sur le $k^{ème}$ bloc [nous avons une matrice de taille (n_k x M)]
Fin Pour

Assembler ces sous-matrices (rbind), nous avons une matrice de taille (n x M), la nouvelle matrice des M variables explicatives.

La variable cible est constituée des étiquettes des données d'apprentissage (vecteur de taille n)

Le métamodèle est élaboré à partir de ce nouvel ensemble de données.

« Stacked ensemble » sous H2O

```
#subdivision app-test
from sklearn import model_selection
DTrain,DTest = model_selection.train_test_split(sonar,train_size=108,random_state=1)

#librairie H2O
import h2o

#démarrage du serveur
h2o.init(nthreads=-1)

#conversion des données au format H2O
HTrain = h2o.H2OFrame(DTrain)
HTest = h2o.H2OFrame(DTest)

#régression logistique binaire (family = ''binomial'')
from h2o.estimators import H2OGeneralizedLinearEstimator
reg = H2OGeneralizedLinearEstimator(seed=100,family="binomial",nfolds=5,keep_cross_validation_predictions=True)
reg.train(x=HTrain.columns[1:],y="classe",training_frame=HTrain)
reg.model_performance(HTest) #AUC = 0.859

#naive bayes
from h2o.estimators import H2ONaiveBayesEstimator
nb = H2ONaiveBayesEstimator(seed=100,nfolds=5,keep_cross_validation_predictions=True)
nb.train(x=HTrain.columns[1:],y="classe",training_frame=HTrain)
nb.model_performance(HTest) #AUC = 0.832

#gradient boosting
from h2o.estimators import H2OGradientBoostingEstimator
gb = H2OGradientBoostingEstimator(seed=100,distribution="bernoulli",nfolds=5,keep_cross_validation_predictions=True)
gb.train(x=HTrain.columns[1:],y="classe",training_frame=HTrain)
gb.model_performance(HTest) #AUC = 0.907

#super-learner, régression logistique
from h2o.estimators import H2OStackedEnsembleEstimator
meta = H2OStackedEnsembleEstimator(seed=100,base_models=[reg,nb,gb],metalearner_algorithm="AUTO")

#apprentissage – construction du métamodèle
meta.train(x=HTrain.columns[1:],y="classe",training_frame=HTrain)

#AUC du meta modèle
meta.model_performance(test_data=HTest) #AUC = 0.921

#fin
h2o.shutdown()
```

Validation croisée en 5 blocs, et conservation des prédictions en validation croisée pour pouvoir les exploiter dans la construction du méta-modèle. Il faut que les blocs aient été constitués de la même manière (strictement identiques).

Passage en paramètre des modèles déjà entraînés en validation croisée.

Métamodèle construit à l'aide de la régression logistique, avec pour contrainte des coefficients non-négatifs (GLM with non negative weights).

Le modèle composite est meilleur que les modèles pris individuellement. C'est l'intérêt du stacking.

CONCLUSION

Stacking - Conclusion

- Faire coopérer des modèles permet de les rendre collectivement plus performants.
- Cela est d'autant plus vrai qu'ils sont individuellement performants tout en étant suffisamment hétérogènes pour être complémentaires.
- La stratégie d'agrégation peut être le vote simple ou pondéré.
- Elle peut être également apprise à partir des données en construisant un modèle intermédiaire (métamodèle) permettant d'attribuer les poids « optimaux » aux différents modèles constituant le groupe. En ce sens, la régression logistique est privilégiée.
- On peut utiliser tout type d'algorithme pour constituer le métamodèle (ex. un arbre de décision, un random forest, etc.). Mais on est moins en phase avec l'idée de pondération optimale dans ce cas.

RÉFÉRENCES

Références

- Wolpert D., « Stacked Generalization », Neural Networks, 5:241-259, 1992.
- Van der Laan M., Polley E., Hubbard A.E., « Super Learner », Statistical Applications in Genetics and Molecular Biology, 6(1), 2007.
- H2O Documentation, « Stacked Ensembles ».
- Scikit-Learn Documentation, « Ensemble Methods – Voting Classifier ».