Nom et Prénom : BATTACHE Nassim, DJOUAD Iskander.

Master 1 OIVM - Optique Image Vision MultiMedia

Apprentissage supervisé

Abstract :

Ce rapport résume les méthodes statiques pour l'apprentissage supervisé .

Introduction

Bagging

Forets aléatoires

Boosting

Conclusion

Introduction :

0.0.1 Forets aléatoire

Les forets aléatoire ou connue sous le nom de random forest est un algorithme affété, dans la machine learning. Il permet de prédire d'une façon fiable avec son système de forêt d'arbres décisionnels. Les forets aléatoires sont composés de plusieurs arbres de décision, qui fonctionnent indépendamment sur un problème. ou une estimation est donner par chacun de ces derniers, et à partir de ces estimations assembler on obtiendra une estimation globale.

0.0.2 Bagging

Les méthodes de type bagging construisent plusieurs instances d'un estimateur, calculées sur des échantillons aléatoires tirés de la base d'apprentissage (et éventuellement une partie des attributs, également sélectionnés de façon aléatoire), et ensuite combine les prédictions individuelles en réalisant leur moyenne pour réduire la variance de l'estimateur.

0.0.3 Boosting

Le but du boosting est d'évaluer une séquence de classifieurs faibles (weak learners) sur plusieurs versions légèrement modifiées des données d'apprentissage. Les décisions obtenues sont alors combinées par une somme pondérée pour obtenir le modèle final. Avec scikit-learn, c'est la classe AdaBoostClassifier qui implémente cet

algorithme.

Bagging:

Construire la variance de la valeur accuracy sur 100 tirages pour la séparation apprentissage/test. Que pouvons-nous conclure ?

```
from sklearn.model_selection import train_test_split
# 90% des données pour le test, 10% pour l'apprentissage
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.90)
clf = tree.DecisionTreeClassifier()
clf.fit(X_train, y_train)
Z = clf.predict(X_test)
accuracy = clf.score(X_test,y_test)
print(" 90% des données pour le test, 10% pour l'apprentissage :"+str(accuracy))


from sklearn.model_selection import train_test_split
# 90% des données pour le test, 10% pour l'apprentissage
accuracy100 = []
for i in range(100) :
 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.90)
 clf = tree.DecisionTreeClassifier()
 clf.fit(X_train, y_train)
 Z = clf.predict(X_test)
 accuracy100.append(clf.score(X_test,y_test) )

print(" 90% des données pour le test, 10% pour l'apprentissage :"+str(accuracy100))


import numpy as np
var = np.std(accuracy100)
print("la moyennne est de : "+str(np.mean(accuracy100)))

print ("la variance de la valeur accuracy sur 100 tirages "+str(var))
```

Voici le résultat du programme :

```
In [72]: runfile('/home/etudiant/Bureau/untitled1.py', wdir='/home/etudiant/Bureau')
1.0
 90% des données pour le test, 10% pour l'apprentissage :0.6396786155747837
 90% des données pour le test, 10% pour l'apprentissage :[0.673053152039555, 0.6644004944375772,
0.7163164400494437, 0.69221260815822, 0.6872682323856613, 0.6718170580964153, 0.7268232385661311,
0.7212608158220025, 0.6699629171817059, 0.7200247218788628, 0.6687268232385661, 0.6891223733003708,
0.6625463535228677, 0.7138442521631644, 0.734857849196539, 0.7187886279357231, 0.7033374536464772,
0.695920889987639, 0.6594561186650185, 0.6860321384425216, 0.7119901112484549, 0.6458590852904821,
0.6674907292954264, 0.6934487021013597, 0.6514215080346106, 0.657601977750309, 0.6965389369592089,
0.6971569839307787, 0.6773794808405439, 0.6903584672435105, 0.688504326328801, 0.6724351050679852,
0.7336217552533992, 0.7342398022249691, 0.7194066749072929, 0.6990111248454882, 0.7194066749072929,
0.6044499381953028, 0.7187886279357231, 0.6841779975278122, 0.7410383189122374, 0.6681087762669963,
0.7342398022249691, 0.6681087762669963, 0.6792336217552534, 0.6606922126081582, 0.6569839307787392,
0.6786155747836835, 0.7101359703337453, 0.7051915945611866, 0.6674907292954264, 0.7045735475896168,
0.65389369592089, 0.6878862793572311, 0.6236093943139679, 0.6971569839307787, 0.681087762669963,
0.7033374536464772, 0.6779975278121138, 0.6866501854140915, 0.6427688504326329, 0.7311495673671199,
0.6835599505562423, 0.6291718170580964, 0.688504326328801, 0.6786155747836835, 0.6582200247218789,
0.6792336217552534, 0.6761433868974042, 0.703955500618047, 0.6755253399258344, 0.6755253399258344,
0.6971569839307787, 0.7428924598269468, 0.711372064276885, 0.6501854140914709, 0.673053152039555,
0.6983930778739185, 0.6223733003708282, 0.7051915945611866, 0.673053152039555, 0.6619283065512979,
0.7187886279357231, 0.6705809641532756, 0.6440049443757726, 0.6779975278121138, 0.7243510506798516,
0.6718170580964153, 0.6940667490729295, 0.6971569839307787, 0.6711990111248455, 0.7243510506798516,
0.6415327564894932, 0.7126081582200248, 0.6606922126081582, 0.6940667490729295, 0.6749072929542645,
0.6755253399258344, 0.6928306551297899, 0.6650185414091471]
la variance de la valeur accuracy sur 100 tirages 0.02777459631318312
```

Si on exécute le code plusieurs fois on obtient des valeurs autour de mean=0.68 et sigma=0.02 .Si on considère une distribution normale, les valeurs de accuracy vont se retrouver avec une probabilité de 71.44 % qui est ±3 autour de la moyenne. Les arbres de décision produisent des classifieurs avec un taux d'erreur qui n'est pas très stable, selon l'échantillon d'apprentissage.

Calculer la variance de la valeur accuracy sur 100 tirages pour la séparation apprentissage/test. Comparer avec la variance du classifieur de base. Que pouvons-nous conclure ?

```
clf_bag = BaggingClassifier(tree.DecisionTreeClassifier(),
max_samples=0.5, max_features=0.5, n_estimators=200)

clf_bag.fit(X_train, y_train)
Z = clf.predict(X_test)
accuracy_bag=clf.score(X_test,y_test)
print("score Bagging  :  "+str(accuracy_bag))

N = 100
accuracy_bag = []

for i in range(N):
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.9)
            clf_bag = BaggingClassifier(tree.DecisionTreeClassifier(), max_samples=0.5, max_features=0.5, n_estimators=(i+1)*50)
            clf_bag.fit(X_train, y_train)
            Z = clf_bag.predict(X_test)
            accuracy_bag.append(clf_bag.score(X_test,y_test))
```

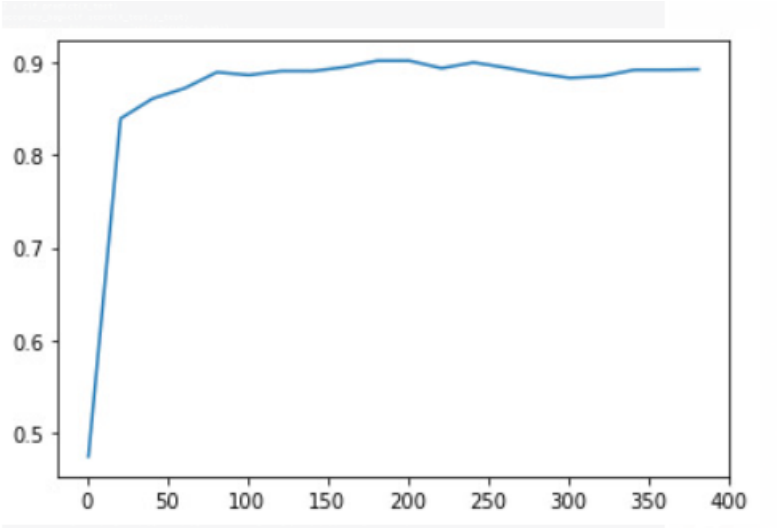
Voici le résultat suivant :

```
la moyennne est de : 0.6882323856613103
la variance de la valeur accuracy sur 100 tirages 0.025252089315711698
score Bagging  :  0.7144622991347342
```

Construire le graphique accuracy vs n_estimators. Que constatez-vous ?

```
import matplotlib.pyplot as plt
plt.plot([50*(i+1) for i in range(N)], accuracy_bag)
```

On obtient la courbe suivante :



Le taux d'erreur diminue avec n_estimators, mais à partir de la valeur 97 il se stabilise, donc rien ne change à partir de cette valeur de n_estimators. Dans le code suivant nous avons pris test_size = 0.9 pour que l'effet soit plus visible.

Varier les paramètres max_samples et max_features. Pour quelles valeurs on obtient le meilleur résultat ? On pourra notamment utiliser GridSearchCV pour réaliser une recherche systématique.

```
for i in range(N):
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.9)
            clf_bag = BaggingClassifier(tree.DecisionTreeClassifier(), max_samples=0.5, max_features=0.5, n_estimators=(i+1)*50)
            clf_bag.fit(X_train, y_train)
            Z = clf_bag.predict(X_test)
            accuracy_bag.append(clf_bag.score(X_test,y_test))

import matplotlib.pyplot as plt
plt.plot([50*(i+1) for i in range(N)], accuracy_bag)
plt.show()

X, y = digits.data, digits.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.9)

from sklearn.model_selection import GridSearchCV
pgrid = {"max_samples": [0.1,0.2, 0.4, 0.6, 0.16,0.32],
                "max_features": [0.1,0.2, 0.8, 0.16, 0.32]}
grid_search = GridSearchCV(BaggingClassifier(tree.DecisionTreeClassifier()), param_grid=pgrid, cv=5)
grid_search.fit(X_train, y_train)
best_score = grid_search.best_estimator_.score(X_test, y_test)
max_sample = grid_search.best_estimator_.max_samples
maxFeatures = grid_search.best_estimator_.max_features
print("best score : " + str(best_score))
print("max_samples: " + str(max_sample))
print("max_features: " + str(maxFeatures))
```

Le résultat est le suivant :

Pour chercher des bonnes valeurs pour les paramètres on utilise la validation croisée avec GridSearchCV, comme expliqué dans le TP sur l'évaluation et la sélection des modèles décisionnels <tpEvaluationSelectionModeles>_.

```
best score : 0.8368355995055624
max_samples: 0.6
max_features: 0.8
```

On a un score de 83% obtenu pour max_samples = 0.6 et max_features = 0.8.

Forets aléatoires:

Comment la valeur de la variable accuracy se compare avec le cas bagging qui utilise le même nombre d'arbres (200 dans notre cas) ? Construire la variance de la valeur accuracy sur 100 tirages pour la séparation apprentissage/test. Que pouvons-nous conclure en comparant avec la séction précedente (bagging) ?

```
############################################################
######### random forest ####################################
############################################################

digits = load_digits()
X, y = digits.data, digits.target
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.90)
from sklearn.ensemble import RandomForestClassifier
clf_random_forest = RandomForestClassifier(n_estimators=200)
clf_random_forest.fit(X_train, y_train)
y_pred_ramdom_forest = clf_random_forest.predict(X_test)
accuracy_ramdom_forest= clf_random_forest.score(X_test,y_test)
print("score ramdom forest  :  "+str(accuracy_random_forest))


clf = tree.DecisionTreeClassifier()
clf.fit(X_train, y_train)
pred = clf.predict(X_test)
score_abr = clf.score(X_test,y_test)
print("score Arbre de desition : " + str(score_abr))

clf = BaggingClassifier(tree.DecisionTreeClassifier(), max_samples=0.5, max_features=0.5, n_estimators=200)
clf.fit(X_train, y_train)
pod_bag= clf.predict(X_test)
score_bag = clf.score(X_test,y_test)
print("score Bagging foret de 200 arbres : " + str(score_bag))

clf = RandomForestClassifier(n_estimators=200)
clf.fit(X_train, y_train)
prod_rdf = clf.predict(X_test)
score_rdf = clf.score(X_test,y_test)
print("random forest pour une foret de  200 arbres: " + str(score_rdf))
```

Le résultat est le suivant :

```
score ramdom forest  :  0.9085290482076638
score Arbre de desition : 0.6149567367119901
score Bagging foret de 200 arbres : 0.8683559950556242
random forest pour une foret de  200 arbres: 0.9215080346106304
```

les forêts aléatoires produisent un classifieur un peu meilleur sur cette base de données.

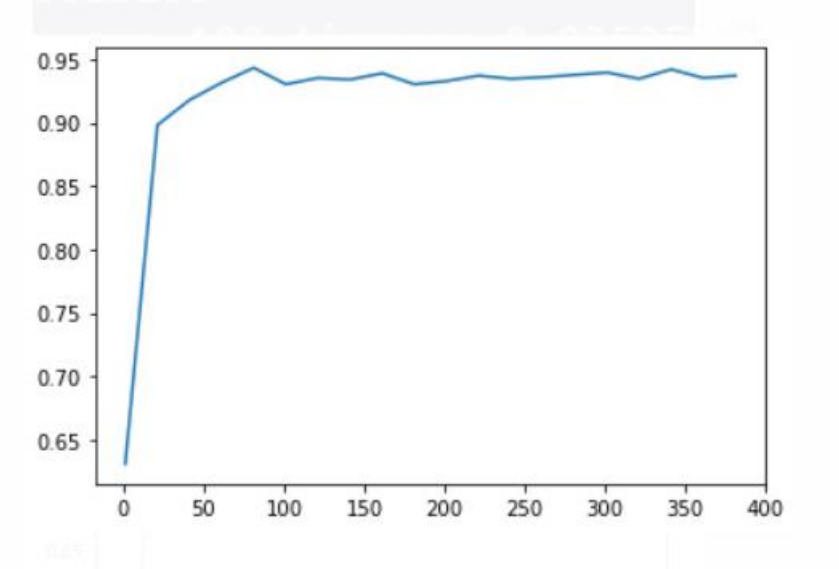
Construire le graphique accuracy vs n_estimators. Que constatez-vous ? A partir de quelle valeur on n'améliore plus ?

```
N = 400
accuracy_rf = []
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.9)
for i in range(1,N,20):

            clf_bag = RandomForestClassifier(n_estimators=i)
            clf_bag.fit(X_train, y_train)
            Z = clf_bag.predict(X_test)
            accuracy_rf.append(clf_bag.score(X_test,y_test))

import matplotlib.pyplot as plt
plt.plot([i for i in range(1,N,20)], accuracy_rf)
plt.show()
```

On obtient la figure suivante :



A partir de n_estimators = 50 le résultat commence à stabiliser autour de la moyenne donc on ne gagne rien en augmentant la valeur.

Regardez dans la documentation les ExtraTreesClassifier et refaites la classification avec ce type de classifieur. Comparez avec RandomForestClassifier.

```
from sklearn.ensemble import ExtraTreesClassifier

import numpy as np
digits = load_digits()
X, y = digits.data, digits.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.90, random_state=0)

clf = RandomForestClassifier(n_estimators=200)
clf.fit(X_train, y_train)
accuracy_rf=clf.score(X_test,y_test)
print("score random foret: " + str(accuracy_rf))

clf = ExtraTreesClassifier(n_estimators=200)
clf.fit(X_train, y_train)
accuracy_xt = clf.score(X_test,y_test)
print("score ExtraTrees : " + str(accuracy_xt))
```

On trouve le résultat suivant :

```
La moyennne des score de bagging est de : 0.870778739184178
La variance de la valeur accuracy sur 100 tirages 0.024652665783103576
La moyennne des score de romdom forest est de : 0.9121384425216317
La variance de la valeur accuracy sur 100 tirages 0.013189973825711921
```

Par rapport aux forêts aléatoires, ils abandonnent l'utilisation des échantillons de bootstrap et pour chaque feature candidat choisissent un seuil de coupure aléatoire. Souvent leur taux d'erreur est supérieur aux forêts aléatoires, le temps de calcul plus faible et les arbres générés sont plus grands.

Boosting:

Le paramètre max_depth contrôle la profondeur de l'arbre. Essayez plusieurs valeurs pour voir l'impact de l'utilisation d'un classifieur faible vs plus fort (max_depth élevé ou éliminer le paramètre). Testez aussi l'effet du paramètre learning_rate et le nombre de classifieurs.

```
from sklearn.ensemble import AdaBoostClassifier
digits = load_digits()
X, y = digits.data, digits.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.90)
# AdaBoost basé sur 200 arbres de décision
max_depth_ = [1,2,4,8,10]
for j in max_depth_ :
 clf =AdaBoostClassifier(base_estimator=tree.DecisionTreeClassifier(max_depth=j),
 n_estimators=200, learning_rate=2)
 clf.fit(X_train, y_train)
 accuracy = clf.score(X_test, y_test)
 print("le score du boosting avec max depth égale a : " +str(j)+" est de : "+str( accuracy))
```

On obtient le résultat suivant :

```
le score du boosting avec max depth égale a : 1 est de : 0.7867737948084055
le score du boosting avec max depth égale a : 2 est de : 0.8590852904820766
le score du boosting avec max depth égale a : 4 est de : 0.8974042027194067
le score du boosting avec max depth égale a : 8 est de : 0.9134734239802225
le score du boosting avec max depth égale a : 10 est de : 0.688504326328801
```

max_depth faible génère des arbres qui généraliser mal. max_dept élevé génère des arbres qui cause du sur-apprentissage d'où une mauvaise généralisation .

Conclusion :

- Les arbres de décision, les forêts aléatoires et le boosting de gradient figurent parmi les modèles statistiques et de machine learning les plus utilisés par les data scientists. Pourtant, les trois méthodes sont similaires et se chevauchent en grande partie.
- L'erreur de variance fait référence à l'ampleur de la variation d'un résultat en fonction des modifications apportées au jeu de données consacré à l'apprentissage. Les arbres décisionnels ont une variance élevée, ce qui signifie que des changements minimes dans les données d'apprentissage peuvent entraîner des changements importants dans le résultat final
- Les random forests réduisent la variance observée dans les arbres de décision :

- En utilisant différents échantillons pour l'entraînement,
- En spécifiant des sous-ensembles de caractéristiques aléatoires,
- En construisant et combinant de petits arbres (peu profonds).
- La difirence entre le boosting et le random forests est dans la façon dont les arbres sont construits , les algorithmes de random forest construisent chaque arbre indépendamment tandis que le boosting construit un arbre à la fois. Ce de dernier fonctionne de manière progressive, en introduisant un algorithme apprenant faible pour améliorer les lacunes des apprenants faibles existants.
- les forêts aléatoires combinent les résultats à la fin du processus (en calculant la moyenne ou en appliquant les « règles de la majorité »), tandis qu'un modèle de boosting combine les résultats en cours de route.
- Tout comme le boosting, les méthodes bagging sont des méthodes d'agrégation.
- out comme le boosting, le bagging permet de rendre performante des règles dites "faibles". Bag-gons par exemple la règle du 1 plus proche voisin (on rappelle que cette règle n'est pas universellement consistante). On désigne par $\hat{m}(., \theta k)$ la règle du 1 plus proche voisin construite sur l'échantillon bootstrap $\theta k(Dn)$, le bootstrap ayant été réalisé selon la technique.