第12回アンサンブル学習_時間外課題

データspamに交差検証法を用いて、以下の問いに答えよ。

In [5]:
```
import pandas as pd # データフレーム
from sklearn.model_selection import KFold # 交差検証法
from sklearn.naive_bayes import GaussianNB # ナイーブベイズ分類器
from sklearn import tree # 決定木
from sklearn.neighbors import KNeighborsClassifier # KNN
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier # Bagging
from sklearn.ensemble import RandomForestClassifier# RandomForest
from sklearn.ensemble import GradientBoostingClassifier # Boosting
from sklearn.metrics import accuracy_score # 正解率計算のためのライブラリー
import statistics # 平均値や標準偏差を計算するライブラリー

import matplotlib.pyplot as plt # 図を描画するライブラリー
```

データセットの読み込み

In [3]:
```
df = pd.read_csv("https://datahub.io/machine-learning/spambase/r/spambase.csv", header = 0)
```

問1. 関数NaiveBayes (), rpart(), knn()、 bagging(), randomForest(), boosting()の5分割検証の各回の正解率を求めよ。

In [6]:
```
# 5分割交差検証を行うクラスのインスタンス化
kf = KFold(n_splits = 5, shuffle = True, random_state = 1)

# 各分類器のインスタンス化
gnb = GaussianNB()
clf = tree.DecisionTreeClassifier()
knn = KNeighborsClassifier(n_neighbors = 2)
bagging = BaggingClassifier(base_estimator=SVC(), n_estimators=10, random_state=0)
rf = RandomForestClassifier(max_depth = 2, random_state = 0)
boosting = GradientBoostingClassifier(n_estimators = 100, learning_rate = 1.0, max_depth = 1, random_

# 各分類器の正解率を保存するリスト
nb_acc = []
rpart_acc = []
knn_acc = []
bagging_acc = []
rf_acc = []
boosting_acc = []

# 各分割目における個体のindexの割り振り
for train_index, test_index in kf.split(df):

    # 学習、テストデータの設定
    train_x = df.iloc[train_index, 0:57]
    train_y = df.iloc[train_index, 57]
    test_x = df.iloc[test_index, 0:57]
    test_y = df.iloc[test_index, 57]

    # ナイーブベイズ分類器
    y_pred_nb = gnb.fit(train_x, train_y).predict(test_x)
    nb_res = accuracy_score(test_y, y_pred_nb)
    nb_acc.append(nb_res)

    # 決定木
    y_pred_clf = clf.fit(train_x, train_y).predict(test_x)
    clf_res = accuracy_score(test_y, y_pred_clf)
    rpart_acc.append(clf_res)

    # KNN
    y_pred_knn = knn.fit(train_x, train_y).predict(test_x)
    knn_res = accuracy_score(test_y, y_pred_knn)
    knn_acc.append(knn_res)

    # Bagging
    y_pred_bagging = bagging.fit(train_x, train_y).predict(test_x)
    bagging_res = accuracy_score(test_y, y_pred_bagging)
```

```
    bagging_acc.append(bagging_res)

    # RF
    y_pred_rf = rf.fit(train_x, train_y).predict(test_x)
    rf_res = accuracy_score(test_y, y_pred_rf)
    rf_acc.append(rf_res)

    # Boosting
    y_pred_boosting = boosting.fit(train_x, train_y).predict(test_x)
    boosting_res = accuracy_score(test_y, y_pred_boosting)
    boosting_acc.append(boosting_res)


print("NB:")
print(statistics.mean(nb_acc))
print(statistics.stdev(nb_acc))

print("Decision Tree: ")
print(statistics.mean(rpart_acc))
print(statistics.stdev(rpart_acc))

print("knn: ")
print(statistics.mean(knn_acc))
print(statistics.stdev(knn_acc))

print("Bagging: ")
print(statistics.mean(bagging_acc))
print(statistics.stdev(bagging_acc))

print("RandomForest: ")
print(statistics.mean(rf_acc))
print(statistics.stdev(rf_acc))

print("Boosting: ")
print(statistics.mean(boosting_acc))
print(statistics.stdev(boosting_acc))
```

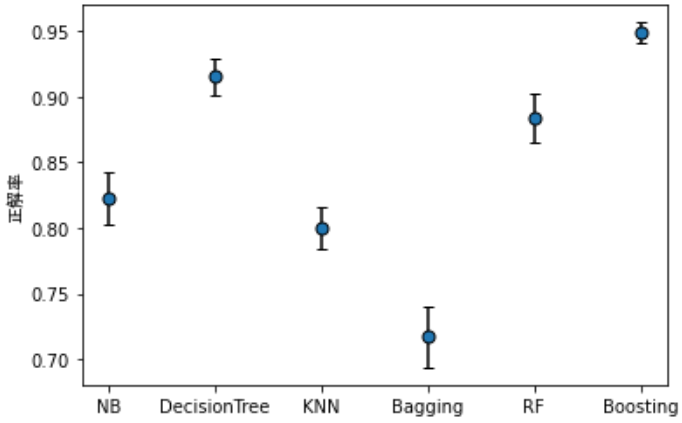
```
NB:
0.8228659774347354
0.019654273333528337
Decision Tree:
0.9152358022942926
0.01359788406210397
knn:
0.8000427229382052
0.0156877054708783
Bagging:
0.7170200160506066
0.023288817625602694
RandomForest:
0.8837178397771798
0.01851548337621335
Boosting:
0.948923665203229
0.007804116003694837
```

問2. 問1で求めた結果の平均プロットを作成し、考察せよ

In [7]:

```
# 平均プロットの作成のためのリストを作成
x = ["NB", "DecisionTree", "KNN", "Bagging", "RF", "Boosting"]
y = [statistics.mean(nb_acc), statistics.mean(rpart_acc), statistics.mean(knn_acc), statistics.mean(k
     statistics.mean(rf_acc), statistics.mean(boosting_acc)] # y軸に各分類器の正解率の平均値を保持
y_sd = [statistics.stdev(nb_acc), statistics.stdev(rpart_acc), statistics.stdev(knn_acc),
        statistics.stdev(bagging_acc), statistics.stdev(rf_acc), statistics.stdev(boosting_acc)] # 各分

fig, ax = plt.subplots()
ax.plot(x, y, marker='o', linewidth = 0) # 図の描画設定
ax.errorbar(x, y, yerr=y_sd, capsize=3, fmt='o', ecolor='k', ms=7, mfc='None', mec='k') # エラーバー
ax.set_ylabel('正解率', fontname = "MS Gothic")
plt.show()
```



ナイーブベイズや**Bagging**で求めた正解率が軒並み低く、**Boosting**が安定して高い正解率を出していることが分かった