In [ ]:

【ハイパーパラメタ・チューニング】
問8
グリッドサーチは、パラメータとして提示した値すべての組合せをしらみつぶしに探索する方法でした。このやり方はたいへん優れたやり方だとは思いますが、如何せん、組合せが増えると、探索に結構な時間を要する場合があります。これを解消する方法として、「ランダムサーチ」という方法があります。
ランダムサーチは、しらみつぶしではなく、指定した回数だけランダムな組合せで探索を行います。コード的には、グリッドサーチとまったく同じで、GridSearchCVの代わりに、RandomizedSearchCVを使い、パラメータ'n_iter'でランダムサーチの回数を指定します.
上記タイタニックでランダムサーチを試してみてください。

In [5]:

```
# 問4と同じ探索範囲設定でランダムフォレストでやってみます
import pandas as pd

df = pd.read_csv('train.csv')
df = df.drop(['PassengerId','Age','SibSp','Parch','Ticket',
              'Cabin','Name'], axis=1)
df['Embarked'] = df['Embarked'].fillna('S')
df['Embarked'] = df['Embarked'].map({'S': 0, 'C': 1, 'Q': 2})
df['Sex'] = df['Sex'].apply(lambda x: 0 if x=='male' else 1)
df.head()
```

Out[5]:

	Survived	Pclass	Sex	Fare	Embarked
0	0	3	0	7.2500	0
1	1	1	1	71.2833	1
2	1	3	1	7.9250	0
3	1	1	1	53.1000	0
4	0	3	0	8.0500	0

In [6]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import accuracy_score

x = df.drop('Survived', axis=1)
y = df['Survived']
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size=0.3,
                                                    random_state=0)


params = {
    'n_estimators': [10, 50, 100, 500, 1000],
    'max_depth': [1, 2, 10, 100, 1000],
    'max_features': [1, 2, 2.2, 3, 3.5, 4]
}

rs = RandomizedSearchCV(RandomForestClassifier(),
                        param_distributions=params, cv=5,
                        n_jobs=-1, n_iter=50)
#しらみつぶしは 5×5×6=150 なので、その 1/3 でやってみます
rs.fit(x_train, y_train)
```

Out[6]:
```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(), n_ite
r=50,
                   n_jobs=-1,
                   param_distributions={'max_depth': [1, 2, 10, 10
0, 1000],
                                        'max_features': [1, 2, 2.2
, 3, 3.5, 4],
                                        'n_estimators': [10, 50, 1
00, 500,
                                                         1000]})
```

In [7]:
```
print('result : {}' .format(rs.cv_results_))
print()
print('best score : {:.5f}' .format(rs.best_score_))
print()
print('best parameters : {}' .format(rs.best_params_))
```

```
                    100, 2, 1, 1, 1000, 100, 100, 1, 1000, 10, 10
, 100, 1,
                    1000, 1, 10, 2, 10, 10, 1, 2, 1000, 100, 2],
              mask=[False, False, False, False, False, False, Fal
se, False,
                    False, False, False, False, False, False, Fal
se, False,
                    False, False, False, False, False, False, Fal
se, False,
                    False, False, False, False, False, False, Fal
se, False,
                    False, False, False, False, False, False, Fal
se, False,
                    False, False, False, False, False, False, Fal
se, False,
                    False, False],
       fill_value='?',
            dtype=object), 'params': [{'n_estimators': 100, 'max
_features': 2.2, 'max_depth': 1000}, {'n_estimators': 10, 'max_f
eatures': 3.5, 'max_depth': 100}, {'n_estimators': 50, 'max_feat
```

In [8]:
```
rf = RandomForestClassifier(max_depth=1000, max_features=4,
                            n_estimators=500)
rf.fit(x_train, y_train)

pred = rf.predict(x_test)
acc = accuracy_score(pred, y_test)

print('accuracy score : {:.5f}' .format(acc))
```

```
accuracy score : 0.81343
```

In [ ]:
```
# だいたい同精度が出ています。必ずしもうまくいくとは限りませんが、パラメータの組合
# ちょっと試しにやってみよう的な場合には有効な方法だと思います。
# もう一つ、有名な方法に「ベイズ最適化」という方法があり、なんでもよさげなパラメー
# の方向性を探索し、徐々に絞っていくやり方だとか…。私自身、あまり理解できていない
# 機に紹介させてもらうこととします。
```