記事
Toshihiko Minamoto · 2021年12月9日 23m read

MLとIntegratedMLでCovid-19のICU入室予測を実行する（パート2）

キーワード: IRIS、IntegratedML、機械学習、Covid-19、Kaggle

前のパート1の続き... パート1では、Kaggleに掲載されているこのCovid-19データセットにおける従来型MLのアプローチを説明しました。

今回のパート2では、IRISのIntegratedMLを使用して、可能な限り単純な形態で同じデータとタスクを実行しましょう。IntegratedMLは、バックエンドAutoMLオプション用に洗練された優れたSQLインターフェースです。
同じ環境を使用します。

IntegratedMLアプローチとは

IRISにデータを読み込む方法

integredML-demo-templateには、IRISにデータを読み込む様々な方法が定義されています。 たとえば、このCSV形式のxlsファイルに固有のカスタムIRISクラスを定義し、それをIRISテーブルに読み込むことができます。
大量のデータをより適切に制御することができます。

ただし、この記事では、単純化された怠惰な方法
を使用します。データフレーム全体を私が作成したカスタムPython関数で読み込む方法です。 そうすることで、生のデータフレームや処理されたデータフレームのさまざまなステージをいつでもIRISに保存し、前のMLアプローチを使用して、類似性比較を行えます。

```
def to_sql_iris(cursor, dataFrame, tableName, schemaName='SQLUser', drop_table=False
):
        """
        Dynamically insert dataframe into an IRIS table via SQL by "excutemany"

        Inputs:
            cursor:      Python JDBC or PyODBC cursor from a valid and establised DB
connection
            dataFrame:   Pandas dataframe
            tablename:   IRIS SQL table to be created, inserted or apended
            schemaName:  IRIS schemaName, default to "SQLUser"
            drop_table:  If the table already exsits, drop it and re-
create it if True; othrewise keep it and appen
        Output:
            True is successful; False if there is any exception.
        """
        if drop_table:
            try:
                curs.execute("DROP TABLE %s.%s" %(schemaName, tableName))
            except Exception:
                pass

        try:
            dataFrame.columns = dataFrame.columns.str.replace("[() -]", "_")
            curs.execute(pd.io.sql.get_schema(dataFrame, tableName))
```

```
        except Exception:
            pass

        curs.fast_executemany = True
        cols = ", ".join([str(i) for i in dataFrame.columns.tolist()])
        wildc =''.join('?, ' * len(dataFrame.columns))
        wildc = '(' + wildc[:-2] + ')'
        sql = "INSERT INTO " + tableName + " ( " + cols.replace('-', '_') + " ) VALUE
S" + wildc
        #print(sql)
        curs.executemany(sql, list(dataFrame.itertuples(index=False, name=None)) )
        return True
```

#

Python JDBC接続のセットアップ

```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, roc_auc_score, roc_curve
import seaborn as sns
sns.set(style="whitegrid")
```

```
import jaydebeapi
url = "jdbc:IRIS://irisimlsvr:51773/USER"
driver = 'com.intersystems.jdbc.IRISDriver'
user = "SUPERUSER"
password = "SYS"
jarfile = "./intersystems-jdbc-3.1.0.jar"

conn = jaydebeapi.connect(driver, url, [user, password], jarfile)
curs = conn.cursor()
```

開始データポイントをセットアップする

類似性比較を行うために、前の記事の特徴量選択（「特徴量の選択 - 最終的な選択」セクション）の後のデータフレームから始めました。「DataS」はここで実際に開始するデータフレームです。

```
data = dataS
data = pd.get_dummies(data)
data.AGE_ABOVE65 = data.AGE_ABOVE65.astype(int)
data.ICU = data.ICU.astype(int)
data_new = data
data_new
```

	AGE_ABOVE65	GENDER	HTN	OTHER	CALCIUM_MEDIAN	CALCIUM_MIN	CALCIUM_MAX	CREATININE_MEDIAN	CREATININE_MEAN	CREATININE_MIN	...	HEART_RATE_	RESPIRATORY	TEMPERATURE	OXYGEN_SATU	ICU	WINDOW0-2	WINDOW2-4	WINDOW4-6	WINDOW6-12	WINDOWABOV

					AN			IAN	N			DIFFREL	RATE_DIFFREL	DIFFREL	RATION_DIFFREL						E12
	1	0.0	0.0	1.0	0.33035039	0.33035039	0.33035039	-0.891078	-0.891078	-0.891078	...	-1.000000	-1.000000	-1.000000	-1.000000		1				
1	1	0.0	0.0	1.0	0.33035039	0.33035039	0.33035039	-0.891078	-0.891078	-0.891078	...	-1.000000	-1.000000	-1.000000	-1.000000			1			
2	1	0.0	0.0	1.0	0.18367353	0.18367353	0.18367353	-0.868365	-0.868365	-0.868365	...	-0.817800	-0.719147	-0.771327	-0.886982				1		
3	1	0.0	0.0	1.0	0.33035039	0.33035039	0.33035039	-0.891078	-0.891078	-0.891078	...	-0.817800	-0.719147	-1.000000	-1.000000					1	
4	1	0.0	0.0	1.0	0.26531	0.26531	0.26531	-0.926398	-0.926398	-0.926398	...	-0.230462	0.096774	-0.242282	-0.814433	1					1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1920		1.0	0.0	1.0	0.33035039	0.33035039	0.33035039	-0.891078	-0.891078	-0.891078	...	-1.000000	-1.000000	-1.000000	-1.000000		1				
1921		1.0	0.0	1.0	0.24489848	0.24489848	0.24489848	-0.934890	-0.934890	-0.934890	...	-1.000000	-1.000000	-1.000000	-1.000000			1			
1922		1.0	0.0	1.0	0.33035039	0.33035039	0.33035039	-0.891078	-0.891078	-0.891078	...	-1.000000	-1.000000	-1.000000	-1.000000				1		
1923		1.0	0.0	1.0	0.33035039	0.33035039	0.33035039	-0.891078	-0.891078	-0.891078	...	-1.000000	-1.000000	-1.000000	-1.000000					1	
1924		1.0	0.0	1.0	0.30612	0.30612	0.30612	-0.944798	-0.944798	-0.944798	...	-0.763868	-0.612903	-0.513357	-0.835052						1

1925 rows × 62 columns

上記には、選択された58個の特徴量と、前の非数値列（WINDOW）から変換された4つの特徴量があることを示します。

IRISテーブルにデータを保存する

上記の**tosqliris**　関数を使用して、IRISテーブル「CovidPPP62」にデータを保存します。

```
iris_schema = 'SQLUser'
iris_table = 'CovidPPP62'


to_sql_iris(curs, data_new, iris_table, iris_schema, drop_table=True)

df2 = pd.read_sql("SELECT COUNT(*) from %s.%s" %(iris_schema, iris_table),conn)
display(df2)
```

	Aggregate1
	1925

次に、トレーニングビュー名、モデル名、およびトレーニングターゲット列（この場合は「ICU」）を定義します。

```
dataTable = iris_table
dataTableViewTrain = dataTable + 'Train1'
dataTablePredict = dataTable + 'Predict1'
dataColumn =  'ICU'
dataColumnPredict = 'ICUPredicted'
modelName = "ICUP621" #????? - ????????????????
```

すると、このデータをトレーニングビュー（1700行）とテストビュー（225行）に分割できます。
IntegratedMLではこれを行う必要はありませんが、前の記事との比較目的で行っています。

```
curs.execute("CREATE VIEW %s AS SELECT * FROM %s WHERE ID&lt;=1700" % (dataTableViewT
rain, dataTable))
```

```
df62 = pd.read_sql("SELECT * from %s" % dataTableViewTrain, conn)
display(df62)
print(dataTableViewTrain, modelName, dataColumn)
```

```
CovidPPP62Train1 ICUP621 ICU
```

IntegratedMLのデフォルトのAutoMLでモデルをトレーニングする

```
curs.execute("CREATE MODEL %s PREDICTING (%s)  FROM %s" % (modelName, dataColumn, dat
aTableViewTrain))
```

```
curs.execute("TRAIN MODEL %s FROM %s" % (modelName, dataTableViewTrain))

df3 = pd.read_sql("SELECT * FROM INFORMATION_SCHEMA.ML_TRAINED_MODELS", conn)
display(df3)
```

	MODELNAME	TRAINEDMODELNAME	PROVIDER	TRAINEDTIMESTAMP	MODELTYPE	MODELINFO
9	ICUP621	ICUP6212	AutoML	2020-07-22 19:28:16.174000	classification	ModelType:Random Forest, Package:sklearn, Prob...

したがって、IntegratedMLは「ModelType」を自動的に「Random Forrest」（ランダムフォレスト）として選択し、問題を「Classification」（分類）タスクとして扱っているという結果がわかります。 前の記事では、箱ひげ図を使った長々としたモデル比較と選択、およびグリッド検索による長々としたモデルパラメーターのチューニングなど、これとまったく同じことを達成しましたよね。

注意: 上記はIntergratedML構文による最低限のSQLです。 トレーニングアプローチやモデルの選択を指定していませんし、バックエンドMLプラットフォームも設定していません。 すべてはIMLの決定に委ねられており、IMLは

内部トレーニングストラテジーをある程度達成して、適切な最終結果を備えた合理的なモデルに落ち着いています。 わずかながら、私の期待を超えたと言ってよいでしょう。

では、予約しておいたテストセットに対し、現在トレーニングされているモデルの簡単な類似性テストランを実行してみましょう。

テストデータの結果を予測する

トレーニングには1700行を使用しました。
以下では、残りの225行を使用してテストデータのビューを作成し、これらのレコードにSELECT PREDICTを実行します。
その予測結果を「dataTablePredict」に保存して、データフレームとして「df62」に読み込みます。

```
dataTableViewTest = "SQLUSER.DTT621"
curs.execute("CREATE VIEW %s AS SELECT * FROM %s WHERE ID > 1700" % (dataTableViewTes
t, dataTable))


curs.execute("DROP TABLE %s" % dataTablePredict )
curs.execute("Create Table %s (%s VARCHAR(100), %s VARCHAR(100))" % (dataTablePredict
, dataColumnPredict, dataColumn))

curs.execute("INSERT INTO %s  SELECT PREDICT(%s) AS %s, %s FROM %s" % (dataTablePredi
ct, modelName, dataColumnPredict, dataColumn, dataTableViewTest))

df62 = pd.read_sql("SELECT * from %s ORDER BY ID" % dataTablePredict, conn)
display(df62)
```

その混同行列を手動で計算します。これを行う必要はありません。 これは比較のみを目的としています。

```
TP = df62[(df62['ICUPredicted'] == '1') & (df62['ICU']=='1')].count()['ICU']
TN = df62[(df62['ICUPredicted'] == '0') & (df62['ICU']=='0')].count()["ICU"]
FN = df62[(df62['ICU'] == '1') & (df62['ICUPredicted']=='0')].count()["ICU"]
FP = df62[(df62['ICUPredicted'] == '1') & (df62['ICU']=='0')].count()["ICU"]
print(TP, FN, '\n', FP, TN)
precision = (TP)/(TP+FP)
recall = (TP)/(TP+FN)
f1 = ((precision*recall)/(precision+recall))*2
accuracy = (TP+TN) / (TP+TN+FP+FN)
print("Precision: ", precision, " Recall: ", recall, " F1: ", f1, " Accuracy: ", accu
racy)

34 20
 8 163
Precision:  0.8095238095238095  Recall:  0.6296296296296297  F1:  0.7083333333333334
 Accuracy:  0.8755555555555555
```

または、IntegratedMLの組み込みの混同行列を取得する構文を使用することができます。

```
# ???????????
curs.execute("VALIDATE MODEL %s FROM %s" % (modelName, dataTableViewTest) )
df5 = pd.read_sql("SELECT * FROM INFORMATION_SCHEMA.ML_VALIDATION_METRICS", conn)
df6 = df5.pivot(index='VALIDATION_RUN_NAME', columns='METRIC_NAME', values='METRIC_VA
LUE')
```

```
display(df6)
```

METRICNAME	Accuracy	F-Measure	Precision	Recall
VALIDATIONRUN_NAME				
ICUP62121	0.88	0.71	0.81	0.63
...	...	...	...	...

パート1の「基本的なLRトレーニングを実行する」セクションにあった「元の結果」と比較すると、Recallは57%に対して63%、Accuracyは85%に対して88%という結果になっています。
したがって、IntegratedMLではより良い結果が得られています。

SMOTEを介して再調整されたトレーニングデータでIntegratedMLを再トレーニングする

上記のテストは、ICU入室と非入室の比率が1:3という不均衡なデータで行われました。 そこで、前の記事と同様に、SMOTEを適用してデータを均衡化し、その上で上記のIMLパイプラインを再実行することにしましょう。

「Xtrainres' and 'ytrainres」は、前のパート1の「基本的なLRトレーニングを実行する」セクションにあったSMOTE後のデータフレームです。

```
df_x_train = pd.DataFrame(X_train_res)
df_y_train = pd.DataFrame(y_train_res)
df_y_train.columns=['ICU']
```

```
df_smote = pd.concat([df_x_train, df_y_train], 1)
display(df_smote)

iris_schema = 'SQLUser'
iris_table = 'CovidSmote'
to_sql_iris(curs, df_smote, iris_table, iris_schema, drop_table=True) # save it into a new IRIS table of specified name
df2 = pd.read_sql("SELECT COUNT(*) from %s.%s" %(iris_schema, iris_table),conn)
display(df2)
```

	Aggregate1
	2490

SMOTEによって、ICU=1のレコードが増やされたため、データセットの行数は1700ではなく2490になりました。

```
dataTable = iris_table
dataTableViewTrain = dataTable + 'TrainSmote'
dataTablePredict = dataTable + 'PredictSmote'
dataColumn =  'ICU'
dataColumnPredict = 'ICUPredictedSmote'
modelName = "ICUSmote1" #????? - ???????????????? end
```

```
curs.execute("CREATE VIEW %s AS SELECT * FROM %s" % (dataTableViewTrain, dataTable))

df_smote = pd.read_sql("SELECT * from %s" % dataTableViewTrain, conn)
display(df_smote)
```

```
print(dataTableViewTrain, modelName, dataColumn)


CovidSmoteTrainSmote ICUSmote1 ICU

curs.execute("CREATE MODEL %s PREDICTING (%s)  FROM %s" % (modelName, dataColumn, dat
aTableViewTrain))


curs.execute("TRAIN MODEL %s FROM %s" % (modelName, dataTableViewTrain))

df3 = pd.read_sql("SELECT * FROM INFORMATION_SCHEMA.ML_TRAINED_MODELS", conn)
display(df3)
```

	MODELNAME	TRAINEDMODELNAME	PROVIDER	TRAINEDTIMESTAMP	MODELTYPE	MODELINFO
9	ICUP621	ICUP6212	AutoML	2020-07-22 19:28:16.174000	classification	ModelType:Random Forest, Package:sklearn, Prob...
12	ICUSmote1	ICUSmote12	AutoML	2020-07-22 20:49:13.980000	classification	ModelType:Random Forest, Package:sklearn, Prob...

次に、予約済みの225件のテストデータ行を再準備し、それに対してSMOTE再トレーニング済みモデルを実行します。

```
df_x_test = pd.DataFrame(X3_test)
df_y_test = pd.DataFrame(y3_test)
df_y_test.columns=['ICU']


df_test_smote = pd.concat([df_x_test, df_y_test], 1)
display(df_test_smote)

iris_schema = 'SQLUser'
iris_table = 'CovidTestSmote'

to_sql_iris(curs, df_test_smote, iris_table, iris_schema, drop_table=True)

dataTableViewTest = "SQLUSER.DTestSmote225"
curs.execute("CREATE VIEW %s AS SELECT * FROM %s" % (dataTableViewTest, iris_table))
curs.execute("Create Table %s (%s VARCHAR(100), %s VARCHAR(100))" % (dataTablePredict
, dataColumnPredict, dataColumn))
curs.execute("INSERT INTO %s  SELECT PREDICT(%s) AS %s, %s FROM %s" % (dataTablePredi
ct, modelName, dataColumnPredict, dataColumn, dataTableViewTest))

df62 = pd.read_sql("SELECT * from %s ORDER BY ID" % dataTablePredict, conn)
display(df62)

TP = df62[(df62['ICUPredictedSmote'] == '1') & (df62['ICU']=='1')].count()['ICU']
TN = df62[(df62['ICUPredictedSmote'] == '0') & (df62['ICU']=='0')].count()["ICU"]
FN = df62[(df62['ICU'] == '1') & (df62['ICUPredictedSmote']=='0')].count()["ICU"]
FP = df62[(df62['ICUPredictedSmote'] == '1') & (df62['ICU']=='0')].count()["ICU"]
print(TP, FN, '\n', FP, TN)
```

```
precision = (TP)/(TP+FP)
recall = (TP)/(TP+FN)
f1 = ((precision*recall)/(precision+recall))*2
accuracy = (TP+TN) / (TP+TN+FP+FN)
print("Precision: ", precision, " Recall: ", recall, " F1: ", f1, " Accuracy: ", accu
racy)


45 15
 9 156
Precision:  0.8333333333333334  Recall:  0.75  F1:  0.7894736842105262  Accuracy:  0.
8933333333333333



# SMOTE??????????????????????????
curs.execute("VALIDATE MODEL %s FROM %s" % (modelName, dataTableViewTest) )  #Covid19
aTest500, Covid19aTrain1000
df5 = pd.read_sql("SELECT * FROM INFORMATION_SCHEMA.ML_VALIDATION_METRICS", conn)
df6 = df5.pivot(index='VALIDATION_RUN_NAME', columns='METRIC_NAME', values='METRIC_VA
LUE')
display(df6)
```

METRICNAME	Accuracy	F-Measure	Precision	Recall
VALIDATIONRUN_NAME				
ICUP62121	0.88	0.71	0.81	0.63
ICUSmote122	0.89	0.79	0.83	0.75

前の63%に比べてはるかに優れた75%のRecallと、わずかに優れたAccuracyとF1スコアが得られました。

さらに**注目すべきこと**
は、この結果が、前の記事の「「グリッド検索によるパラメーターチューニング」をさらに行って、選択されたモデルを実行する」セクションに記録されたとおりに、「モデルの選択」と「グリッド検索によるパラメーターのチューニング」を集中的に行った「従来型MLアプローチ」と一致しているということです。
したがって、IMLの結果は全く悪くないということです。

IntegratedMLのH2Oプロバイダーに変更する

IMLのAutoMLプロバイダーを1行で変更してから、前のステップで行った通りにモデルを再トレーニングできます。

```
curs.execute("SET ML CONFIGURATION %H2O;  ")


modelName = 'ICUSmoteH2O'
print(dataTableViewTrain)
curs.execute("CREATE MODEL %s PREDICTING (%s)  FROM %s" % (modelName, dataColumn, dat
aTableViewTrain))
curs.execute("TRAIN MODEL %s FROM %s" % (modelName, dataTableViewTrain))

df3 = pd.read_sql("SELECT * FROM INFORMATION_SCHEMA.ML_TRAINED_MODELS", conn)
display(df3)
```


	MODELNAME	TRAINEDMODELNAME	PROVIDER	TRAINEDTIMESTAMP	MODELTYPE	MODELINFO
12	ICUSmote1	ICUSmote12	AutoML	2020-07-22 20:49:13.980000	classification	ModelType:Random Forest, Package:sklearn, Prob...
13	ICUPPP62	ICUPPP622	AutoML	2020-07-22 17:48:10.964000	classification	ModelType:Random Forest, Package:sklearn, Prob...
14	ICUSmoteH2O	ICUSmoteH2O2	H2O	2020-07-22 21:17:06.990000	classification	None

```
# ???????????
curs.execute("VALIDATE MODEL %s FROM %s" % (modelName, dataTableViewTest) )  #Covid19
aTest500, Covid19aTrain1000
df5 = pd.read_sql("SELECT * FROM INFORMATION_SCHEMA.ML_VALIDATION_METRICS", conn)
df6 = df5.pivot(index='VALIDATION_RUN_NAME', columns='METRIC_NAME', values='METRIC_VA
LUE')
display(df6)
```

METRICNAME	Accuracy	F-Measure	Precision	Recall
VALIDATIONRUN_NAME				
ICUP62121	0.88	0.71	0.81	0.63
ICUSmote122	0.89	0.79	0.83	0.75
ICUSmoteH2O21	0.90	0.79	0.86	0.73

H2O
AutoMLでは、F1は同じですが、Accuracyがわずかに高く、Recallがわずかに減少していることがわかります。
ただし、このCovid19 ICUタスクの主な目的は、可能であれば偽陰性を最小限に抑えることであるため、
プロバイダーをH2Oに変更しても、ターゲットパフォーマンスは向上しなかったようです。

もちろん、IntegratedMLのDataRobotプロバイダーもテストしたいのですが、残念ながらDataRobotのAPIキーは持っていないため、ここでストップとします。

まとめ:

1. **パフォーマンス**: この特定のCovid-19 ICUタスクでは、比較テストによって、IRIS IntegratedMLのパフォーマンスは従来型MLの類似性の結果に少なくとも同等か類似していることが示されています。 この特定のケースでは、IntegratedMLは内部トレーニングストラテジーを自動的に正しく選択することができ、適切なモデルに落ち着いて、期待される結果を出したように見えました。

2. **単純さ**: IntegratedMLのプロセスは、従来型MLのパイプラインよりもはるかに単純です。 上記に示されるとおり、モデルの選択やパラメーターのチューニングなどの通常のデータサイエンティスト作業を行わずに、同等のパフォーマンスを達成することができました。
比較の為でなければ、実際には特徴量の選択の不要です。 また、Integrated-demo-templateデモノートブックに示されているIntegratedMLの最低限の構文しか使用していません。 もちろん、従来型パイプラインで使用できる一般的なデータサイエンスツールのカスタマイズ性とファインチューニング機能は失われるという欠点はありますが、これは他のAutoMLプラットフォームにも多かれ少なかれ当てはまることです。

3.

データ前処理は依然として重要:
残念ながら特効薬はありません。または、特効薬には時間が必要でしょう。 このCovid-19 ICUタスクに限定して言えば、上記のテストでは、データが現在のIntegratedMLにとって依然として重要であることが示されています。生のデータ、欠落したデータを代入して選択された特徴量、および基本的なSMOTEオーバーサンプリングによる再調整データはすべて大幅に異なるパフォーマンスを見せました。
これは、IMLのデフォルトAutoMLとそのH2Oプロバイダーの両方に当てはまります。 DataRobotはわずかに優れたパフォーマンスを主張するかもしれませんが、IntegratedMLのSQLラッパーでさらにテストされると思います。 **要するに、データ正規化は、IntegratedMLでも依然として重要であるということです。**

4. **デプロイ可能性**: デプロイ可能性、API管理、モニタリング、および非関数サービス可能性などについてはまだ比較していません。次の記事で行えるでしょう。

今後の内容

1. **モデルのデプロイ**:
これまで、Covid-19のＸ線画像に対するデモAIと、バイタルサインおよび観測に対するCovid-19 ICU予測を実行しました。
これらをFlask/FastAPIおよびIRISサービススタックにデプロイし、REST/JSON APIを介してデモML/DL機能を開会できるでしょうか？
もちろん、次の記事でそのようなことを試すことはできます。 その後で、NLP APIなどを含むさらに多くのデモAI機能を徐々に追加していくことができます。

2. **FHIRラップAPIの相互運用性**:
この開発者コミュニティには、FHIRテンプレートやIRISネイティブAPIなどもあります。 デモAIサービスをFHIRアプリでSMARTに、または対応する標準に従ってFHIRラップAIサービスに変換することはできるでしょうか？ IRIS製品ラインには、AIデモスタックで利用できるAPIゲートウェイ、Kubernetesサポート付きのICM、SAMなどがあることも忘れないでください。

3. **HealthShare Clinical ViewerやTrakなどとのデモ統合は？** サードパーティAIベンダーのPACS Viewer（Covid-19 CT用）とHealthShare Clinical Viewerのデモ統合
は簡単に説明しました。したがって、おそらくいずれは、さまざまな専門分野での独自のAIデモサービスを最後まで説明することはできるでしょう。

#IntegratedML #Machine Learning (ML) #InterSystems IRIS

ソースURL:
https://jp.community.intersystems.com/post/ml%E3%81%A8integratedml%E3%81%A7covid-19%E3%81%AEicu%E5%85%A5%E5%AE%A4%E4%BA%88%E6%B8%AC%E3%82%92%E5%AE%9F%E8%A1%8C%E3%81%99%E3%82%8B%EF%BC%88%E3%83%91%E3%83%BC%E3%83%882%EF%BC%89