Article
Zhong Li · Aug 23, 2020 15m read

Run some Covid-19 ICU predictions via ML vs. IntegratedML (Part II)

Keywords: IRIS, IntegratedML, Machine Learning, Covid-19, Kaggle

Continued from the previous Part I ... In part I, we walked through traditional ML approaches on this Covid-19 dataset on Kaggle.

In this Part II, let's run the same data & task, in its simplest possible form, through IRIS integratedML which is a nice & sleek SQL interface for backend AutoML options. It uses the same environment.

IntegratedML Approach?

How to load data into IRIS

integredML-demo-template defined various ways to load data into IRIS. For example, I can define a custom IRIS class specific to this xls file in CSV format then load it into an IRIS table. It allows better control for large data volumes.

However, in this post I go for a simplified and lazy method, by just loading the whole dataframe into an IRIS table via a customised Python function I scratched up. Doing such allows us save various stages of the raw or processed dataframes into IRIS anytime, for like-alike comparisons with previous ML approach.

```
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
```

Setup Python JDBC connection

```
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()
```

Set up starting data point

For like-alike comparisons, I started from the data frame after feature selections in the previous post (in is section "Feature Selection - Final Selection"), where "DataS" is the exact dataframe that we would start here.

```
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	CALCIUMMEDIAN	CALCIUMMIN	CALCIUMMAX	CREATININ_MEDIAN	CREATININ_MEAN	CREATININ_MIN	...	HEART_RATEDIFF_REL	RESPIRATORY_RATEDIFF_REL	TEMPERATURE_DIFFREL	OXYGENSATURATION_DIFFREL	ICU	WINDOW0-2	WINDOW2-4	WINDOW4-6	WINDOW6-12	WINDOWABOVE12
0	1	0.0	0.0	1.0	0.330359	0.330359	0.330359	-0.891078	-0.891078	-0.891078	...	-1.000000	-1.000000	-1.000000	-1.000000	0	1	0	0	0	0
1	1	0.0	0.0	1.0	0.330359	0.330359	0.330359	-0.891078	-0.891078	-0.891078	...	-1.000000	-1.000000	-1.000000	-1.000000	0	0	1	0	0	0
2	1	0.0	0.0	1.0	0.183673	0.183673	0.183673	-0.868365	-0.868365	-0.868365	...	-0.817800	-0.719147	-0.771327	-0.886982	0	0	0	1	0	0
3	1	0.0	0.0	1.0	0.330359	0.330359	0.330359	-0.891078	-0.891078	-0.891078	...	-0.817800	-0.719147	-1.000000	-1.000000	0	0	0	0	1	0
4	1	0.0	0.0	1.0	0.326531	0.326531	0.326531	-0.926398	-0.926398	-0.926398	...	-0.230462	0.096774	-0.242282	-0.814433	1	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1920	0	1.0	0.0	1.0	0.330359	0.330359	0.330359	-0.891078	-0.891078	-0.891078	...	-1.000000	-1.000000	-1.000000	-1.000000	0	1	0	0	0	0
1921	0	1.0	0.0	1.0	0.244898	0.244898	0.244898	-0.934890	-0.934890	-0.934890	...	-1.000000	-1.000000	-1.000000	-1.000000	0	0	1	0	0	0
1922	0	1.0	0.0	1.0	0.330359	0.330359	0.330359	-0.891078	-0.891078	-0.891078	...	-1.000000	-1.000000	-1.000000	-1.000000	0	0	0	1	0	0
1923	0	1.0	0.0	1.0	0.330359	0.330359	0.330359	-0.891078	-0.891078	-0.891078	...	-1.000000	-1.000000	-1.000000	-1.000000	0	0	0	0	1	0
1924	0	1.0	0.0	1.0	0.306122	0.306122	0.306122	-0.944798	-0.944798	-0.944798	...	-0.763868	-0.612903	-0.551337	-0.835052	0	0	0	0	0	1

1925 rows × 62 columns

The above indicated we had 58 selected features plus those another 4 features that was converted from the previous non-numeric column ('WINDOW').

Save data into IRIS table

We use the above **tosqliris** function to save the data into IRIS table "CovidPPP62":

```
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
0	1925

Then define the training view name, model name, and training target column which is 'ICU' in this case.

```
dataTable = iris_table
dataTableViewTrain = dataTable + 'Train1'
dataTablePredict = dataTable + 'Predict1'
dataColumn =  'ICU'
dataColumnPredict = 'ICUPredicted'
modelName = "ICUP621" #chose a name - must be unique in server end
```

Then we can split the data into a Training View (1700 rows) and a Test View (225 rows). We don't have to do this in Integrated ML; it's just for comparisons purpose with previous post.

```
curs.execute("CREATE VIEW %s AS SELECT * FROM %s WHERE ID<=1700" % (dataTableViewTrai
n, dataTable))
```

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

CovidPPP62Train1 ICUP621 ICU

Train model by IntegratedML's default AutoML

```
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...

So, we can see the result shows IntegratedML automatically chose "ModelType" as "Random Forest", and treat the problem as a "Classification" task. That's exactly we achieved out of the lengthy model comparisons and selections by box plot, as well as lengthy model parameter tuning by grid search etc within the previous post, right?

Note: the above SQL is the bare minimum per IntegratedML syntax. I didn't specify any training approach or model

selection, nor did set any backend ML platform. Everything was left to IML's decision, and it somewhat manages to achieve its internal training strategy, then settled down to a reasonable model with the correct end results. I'd say it's slightly beyond my expectations.

Let's do a quick like-alike test run of the currently trained model on our reserved test set.

Predict result on test data

We used 1700 rows for training. Below we create a view of test data with the left 225 rows, and run SELECT PREDICT on these records. We will save the predicted result into 'dataTablePredict', and load it into 'df62' as data frame.

```
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)
```

Manually calculate its confusion matrix - we don't have to do this. It's just for comparison purpose:

```
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
```

Or we can use the IntegratedML syntax to get its built-in confusion matrix:

```
# validate the test data
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
VALIDATIONRUNNAME				
ICUP62121	0.88	0.71	0.81	0.63
...	...	...	...	...

Comparing with the "Orignal Result" in section "Run Basic LR Training" in Part I, the above result has a Recall 63% vs. 57%, and Accuracy 88% vs 85%. So it is a better result with IntegratedML.

Re-train IntegratedML on re-balanced training data via SMOTE

The above test was done on imbalanced data, in which ICU Admitted vs. Non-admitted has a ratio of 1:3. So as in previous post, let's simply SMOTE it to make the data balanced, then re-run the IML pipeline above.

'Xtrainres' and 'ytrainres' are dataframes after SMOTE from the previous Part I in its section "Run Basic LR Training"

```
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
0	2490

Now the dataset has 2490 rows instead of 1700, since SMOTE enriched more records with ICU = 1.

```
dataTable = iris_table
dataTableViewTrain = dataTable + 'TrainSmote'
dataTablePredict = dataTable + 'PredictSmote'
dataColumn =  'ICU'
dataColumnPredict = 'ICUPredictedSmote'
modelName = "ICUSmote1" #chose a name - must be unique in server 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...

Then we re-prepare the reserved set of 225 test data rows, ad run the SMOTE re-trained model on them:

```
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') &amp;amp; (df62['ICU']=='1')].count()['ICU']

TN = df62[(df62['ICUPredictedSmote'] == '0') &amp;amp; (df62['ICU']=='0')].count()["ICU"]
FN = df62[(df62['ICU'] == '1') &amp;amp; (df62['ICUPredictedSmote']=='0')].count()["ICU"]
FP = df62[(df62['ICUPredictedSmote'] == '1') &amp;amp; (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
```

```
# validate the test data via SMOTE re-trained model
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
VALIDATIONRUNNAME				
ICUP62121	0.88	0.71	0.81	0.63
ICUSmote122	0.89	0.79	0.83	0.75

The result indicated a significantly better Recall of 75% over previous 63%, and a slightly better Accuracy and F1 score.

More **noticeably**, this result is in line with our "tradition ML approach" in the previous post, after intensive "model selection" and "parameter tuning by grid searches" as recorded in section "Run Selected Model by further "Parameter Tuning via Grid Search". So the IML result is not bad at all.

Change to IntegratedML's H2O provider

We can change the IML's AutoML provider by a single line, then re-train the model as done in the previous step:

```
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

```
# validate the test data
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
VALIDATIONRUNNAME				
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

The result seems to show that H2O AutoML has a slightly better Accuracy, the same F1, but a slightly less Recall. However, our core objective in this Covid19 ICU task is to minimise the False Negatives if we could. So it seems that the provider change to H2O didn't drive up our target performance yet.

Certainly, I would like to test IntegratedML's DataRobot Provider as well, but unfortunately I don't have an API key from DataRobot yet, so I will park it here.

Recap:

1. **Performance**: For this specific Covid-19 ICU task, our test comparisons indicate that the IRIS IntegratedML's performance is at least on par with or similar to traditional ML results like-alike. In this specific case IntegratedML was able to automatically choose the internal training strategy correctly, and appeared to settle down the right model, and delivered the expected result.

2. **Simplicity**: IntegratedML has a much more simplified process than traditional ML pipelines. As shown above, I don't need to bother anything with Model Selection and Parameter Tuning etc etc routine data scientist jobs anymore, while achieving equivalent performance. I don't actually need Feature Selection either, if not for comparability purpose. Also, I only used IntegratedML bare minimum syntax as shown in the demo notebook of Integrated-demo-template. Sure, the downside is we would sacrifice the customisation and fine-tuning capabilities of common data science tools via its traditional pipelines, however, this is also more or less true to other AutoML platforms as well.

3. **Data pre-processing still matters**: There is no silver bullet unfortunately; or say, silver bullet would need time. Specific to this Covid19 ICU task, the above tests show that data still matters a lot to current IntegratedML: raw data, selected features with imputed missing data, and re-balanced data with basic SMOTE over-sampling, they all resulted in significantly different performance. This is true to both IML's default AutoML and its H2O provider. I imagine DataRobot might claim a slightly better performance but to be further tested with IntegratedML's SQL wrapper. **In short, data normalisation still matters in IntegratedML.**

4. **Deployability**: I didn't compare its deployability, API management, Monitoring, and non-functional Serviceability etc etc yet - we can do it in next post.

Next

1. **Model deployments**: So far we did some demo AI on Covid-19 X-Rays, and Covid-19 ICU predictions on vital signs and observations. Can we deploy them into Flask/FastAPI and IRIS service stacks, and expose their demo ML/DL capabilities via REST/JSON APIs? Sure we can try such in next post. After then we could add on more demo AI capabilities over the time, including NLP APIs etc.

2. **FHIR wrapped API Interoperability**: We also have FHIR template, as well as IRIS Native API etc in this developer community. Could we turn our demo AI service into SMART on FHIR apps, or FHIR wrapped AI services per corresponding standards - could we try such? And please remember in IRIS product line we also have API Gateway, ICM with Kubernetes support, and SAM etc that we could leverage too with our AI demo stacks.

3. **Demo Integration with HealthShare Clinical Viewer and/or Trak etc**? I briefly showed a demo integration of a 3rd party AI vendor's PACS Viewer (for Covid-19 CTs) with HealthShare Clinical Viewer, hence maybe we could finish that hiking with our own AI demo services too, in various specialty domains over the time.

#IntegratedML #Machine Learning (ML) #InterSystems IRIS

Source URL:https://community.intersystems.com/post/run-some-covid-19-icu-predictions-ml-vs-integratedml-part-ii