Article
Zhong Li · Aug 22, 2020 24m read

Open Exchange

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

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

Purpose

Recently I noticed a Kaggle dataset for the prediction of whether a Covid-19 patient will be admitted to ICU. It is a spreadsheet of 1925 encounter records of 231 columns of vital signs and observations, with the last column of "ICU" being 1 for Yes or 0 for No. The task is to predict whether a patient will be admitted to ICU based on known data.

This dataset seems to be a good example of what's called "traditional ML" task. The data seem to have the right quantity and relatively right quality. It might have a better chance of being applied directly on the IntegratedML demo kit, so what could be the simplest approach for a quick test based on normal ML pipelines vs. possible IntegratedML approach?

Scope

We will briefly run through some normal ML steps such as :

- Data EDA
- Feature selection
- Model selection
- Model parameter tuning by grid search

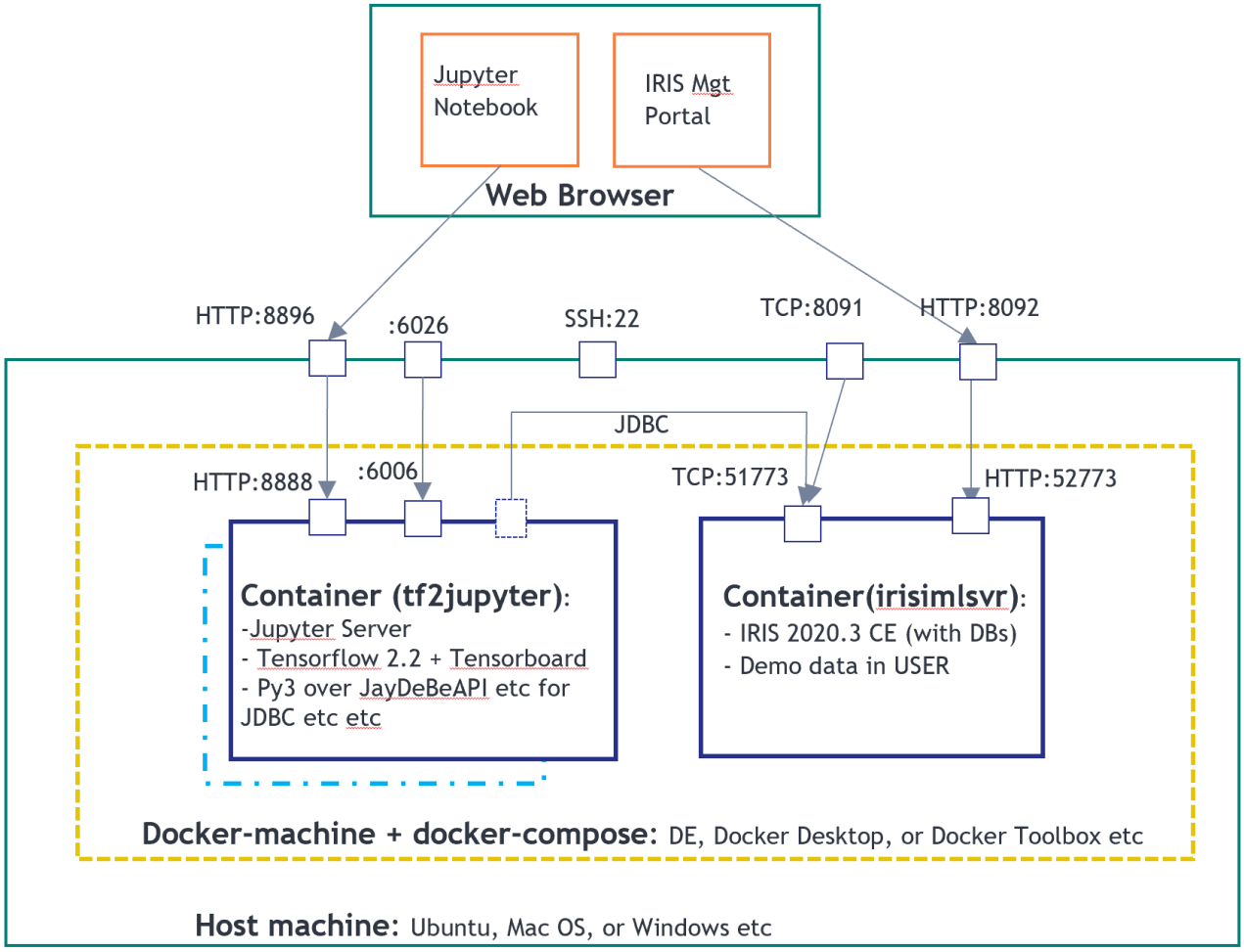
Vs.

- Integrated ML approaches via SQL.

It's run on an AWS Ubuntu 16.04 server with Docker-compose etc.

Environment

We will re-use the integredML-demo-template's Docker environment:



The following notebook file is running on "tf2jupyter", and IRIS with IntegratedML on "irismlsrv". Docker-compose runs on an AWS Ubuntu 16.04.

Data & Tasks

The dataset contains 1925 records collected from 385 patients, each with 5 records of encounters exactly. Out of its 231 columns there is one "ICU" being our training and predicting target, and other 230 columns could all be used as inputs somehow. ICU has a binary value of 1 or 0. Apart from 2 columns which seem to be categorical strings (presented as "object" in dataframe), all others are numeric.

```
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 os
for dirname, _, filenames in os.walk('./input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
./input/datasets_605991_1272346_Kaggle_Sirio_Libanes_ICU_Prediction.xlsx
```

```
df = pd.read_excel("./input/datasets_605991_1272346_Kaggle_Sirio_Libanes_ICU_Predicti
on.xlsx")
df
```

	PATIENT_VISIT_IDENTIFIER	AGE_ABOVE65	AGE_PERCENTIL	GENDER	DISEASE GROUPING 1	DISEASE GROUPING 2	DISEASE GROUPING 3	DISEASE GROUPING 4	DISEASE GROUPING 5	DISEASE GROUPING 6	...	TEMPERATURE_DIFF	OXYGEN_SATURATION_DIFF	BLOOD PRESSURE_DIASTOLIC_DIFF_REL	BLOOD PRESSURE_SISTOLICDIFF_REL	HEART_RATE_DIFF_REL	RESPIRATORY_RATE_DIFF_REL	TEMPERATURE_DIFF_REL	OXYGEN_SATURATION_DIFF_REL	WINDOW	ICU
0	0	1	60th	0	0.0	0.0	0.0	0.0	1.0	1.0	...	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	0-2	0
1	0	1	60th	0	0.0	0.0	0.0	0.0	1.0	1.0	...	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	2-4	0
2	0	1	60th	0	0.0	0.0	0.0	0.0	1.0	1.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4-6	0
3	0	1	60th	0	0.0	0.0	0.0	0.0	1.0	1.0	...	-1.000000	-1.000000	NaN	NaN	NaN	NaN	-1.000000	-1.000000	6-12	0
4	0	1	60th	0	0.0	0.0	0.0	0.0	1.0	1.0	...	-0.238095	-0.818182	-0.389967	0.407558	-0.230462	0.096774	-0.242282	-0.814433	ABOVE12	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1920	384	0	50th	1	0.0	0.0	0.0	0.0	0.0	0.0	...	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	0-2	0
1921	384	0	50th	1	0.0	0.0	0.0	0.0	0.0	0.0	...	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	2-4	0
1922	384	0	50th	1	0.0	0.0	0.0	0.0	0.0	0.0	...	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	4-6	0
1923	384	0	50th	1	0.0	0.0	0.0	0.0	0.0	0.0	...	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	6-12	0
1924	384	0	50th	1	0.0	0.0	1.0	0.0	0.0	0.0	...	-0.547619	-0.838384	-0.701863	-0.585967	-0.763868	-0.612903	-0.551337	-0.835052	ABOVE12	0

1925 rows × 231 columns

```
df.dtypes
```

```
PATIENT_VISIT_IDENTIFIER        int64
AGE_ABOVE65                     int64
AGE_PERCENTIL                  object
GENDER                          int64
DISEASE GROUPING 1            float64
                               ...
RESPIRATORY_RATE_DIFF_REL     float64
TEMPERATURE_DIFF_REL          float64
OXYGEN_SATURATION_DIFF_REL    float64
WINDOW                         object
ICU                             int64
Length: 231, dtype: object
```

There are certainly a few options to frame this problem and its approaches. On top of our heads the first apparent option would be, this can be a basic "binary classification" problem. We can treat all 1925 records as "stateless" individual record, regardless of whether they are from the same patient or not. Sure, it could be a "regression" problem too if we treat ICU and other values as all numeric.

There are certainly other possible approaches too. For example, we can have a further perspective that the dataset has 385 distinct sets of short "time series", each per a patient. We could dissolve the whole set into 385 separate sets for Train/Val/Test, and could we try Deep Learning models such as CNN or LSTM to capture the "symptom development phase or pattern" hidden in each set per individual patients? We might. By doing that we could also apply some data augmentation to enrich the test data by various means. That could be another topic beyond this post.

In this post we will just test a quick run of so called "traditional ML" approach vs. IntegratedML (an AutoML) approach.

"Traditional" ML Approach?

This is a relatively normalised dataset comparing with most real-world cases, apart from some missing values, so we might be be able to skip the feature engineering part, and use the columns as features directly. So let's go straight into the feature selection.

Impute the missing data

First to make sure all missing values are filled with simple imputation:

```
df_cat = df.select_dtypes(include=['object'])
df_numeric = df.select_dtypes(exclude=['object'])
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
idf = pd.DataFrame(imp.fit_transform(df_numeric))
idf.columns = df_numeric.columns
idf.index = df_numeric.index
idf.isnull().sum()
```

Feature Selections

We can certainly use the normal correlation function built in the data frame, to calculate each column values' correlation against ICU's.

featuring engineering - correlation

```
idf.drop(["PATIENT_VISIT_IDENTIFIER"],1)
idf = pd.concat([idf,df_cat ], axis=1)
cor = idf.corr()
cor_target = abs(cor["ICU"])
relevant_features = cor_target[cor_target>0.1]   # correlation above 0.1
print(cor.shape, cor_target.shape, relevant_features.shape)
#relevant_features.index
#relevant_features.index.shape
```

It will list 88 features that has a correlation of >0.1 with the target value of ICU. These columns could be directly used as our model input

I also ran a few other "feature selection methods" that are normally used in traditional ML tasks:

Feature Selection - Chi squared

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.preprocessing import MinMaxScaler
X_norm = MinMaxScaler().fit_transform(X)
chi_selector = SelectKBest(chi2, k=88)
chi_selector.fit(X_norm, y)
chi_support = chi_selector.get_support()
chi_feature = X.loc[:,chi_support].columns.tolist()
print(str(len(chi_feature)), 'selected features', chi_feature)
```

```
88 selected features ['AGE_ABOVE65', 'GENDER', 'DISEASE GROUPING 1', ... ... 'P02_VEN
OUS_MIN', 'P02_VENOUS_MAX', ... ... RATURE_MAX', 'BLOODPRESSURE_DIASTOLIC_DIFF', ...
... 'TEMPERATURE_DIFF_REL', 'OXYGEN_SATURATION_DIFF_REL']
```

Feature Selection - **Pearson correlation**

```
def cor_selector(X, y,num_feats):
    cor_list = []
    feature_name = X.columns.tolist()
    # calculate the correlation with y for each feature
    for i in X.columns.tolist():
        cor = np.corrcoef(X[i], y)[0, 1]
        cor_list.append(cor)
    # replace NaN with 0
    cor_list = [0 if np.isnan(i) else i for i in cor_list]
    # feature name
    cor_feature = X.iloc[:,np.argsort(np.abs(cor_list))[-num_feats:]].columns.tolist(
)
    # feature selection? 0 for not select, 1 for select
    cor_support = [True if i in cor_feature else False for i in feature_name]
```

```
    return cor_support, cor_feature



cor_support, cor_feature = cor_selector(X, y, 88)
print(str(len(cor_feature)), 'selected features:  ', cor_feature)
```

```
88 selected features:   ['TEMPERATURE_MEAN', 'BLOODPRESSURE_DIASTOLIC_MAX', ... ... '
RESPIRATORY_RATE_DIFF', 'RESPIRATORY_RATE_MAX']
```

Feature Selection - Recursive Feature Elimination (RFE)

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
rfe_selector = RFE(estimator=LogisticRegression(), n_features_to_select=88, step=100,
 verbose=5)
rfe_selector.fit(X_norm, y)
rfe_support = rfe_selector.get_support()
rfe_feature = X.loc[:,rfe_support].columns.tolist()
print(str(len(rfe_feature)), 'selected features: ', rfe_feature)
```

```
Fitting estimator with 127 features.
88 selected features:  ['AGE_ABOVE65', 'GENDER', ... ... 'RESPIRATORY_RATE_DIFF_REL',
 'TEMPERATURE_DIFF_REL']
```

Feature Selection - **Lasso**

```
ffrom sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler
X_norm = MinMaxScaler().fit_transform(X)
embeded_lr_selector = SelectFromModel(LogisticRegression(penalty="l2"), max_features=
88)
embeded_lr_selector.fit(X_norm, y)
embeded_lr_support = embeded_lr_selector.get_support()
embeded_lr_feature = X.loc[:,embeded_lr_support].columns.tolist()
print(str(len(embeded_lr_feature)), 'selected features', embeded_lr_feature)
```

```
65 selected features ['AGE_ABOVE65', 'GENDER', ... ... 'RESPIRATORY_RATE_DIFF_REL', '
TEMPERATURE_DIFF_REL']
```

Feature Selection - **RF Tree-based**: SelectFromModel

```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier
embeded_rf_selector = SelectFromModel(RandomForestClassifier(n_estimators=100), max_f
eatures=227)
embeded_rf_selector.fit(X, y)
```

```
embeded_rf_support = embeded_rf_selector.get_support()
embeded_rf_feature = X.loc[:,embeded_rf_support].columns.tolist()
print(str(len(embeded_rf_feature)), 'selected features', embeded_rf_feature)
```

```
48 selected features ['AGE_ABOVE65', 'GENDER', ... ... 'TEMPERATURE_DIFF_REL', 'OXYGE
N_SATURATION_DIFF_REL']
```

Feature Selection - LightGBM or XGBoost

```
from sklearn.feature_selection import SelectFromModel
from lightgbm import LGBMClassifier
```

```
lgbc=LGBMClassifier(n_estimators=500, learning_rate=0.05, num_leaves=32, colsample_by
tree=0.2,
            reg_alpha=3, reg_lambda=1, min_split_gain=0.01, min_child_weight=40)
```

```
embeded_lgb_selector = SelectFromModel(lgbc, max_features=128)
embeded_lgb_selector.fit(X, y)
```

```
embeded_lgb_support = embeded_lgb_selector.get_support()
embeded_lgb_feature = X.loc[:,embeded_lgb_support].columns.tolist()
print(str(len(embeded_lgb_feature)), 'selected features:  ', embeded_lgb_feature)
embeded_lgb_feature.index
```

```
56 selected features:   ['AGE_ABOVE65', 'GENDER', 'HTN', ... ... 'TEMPERATURE_DIFF_RE
L', 'OXYGEN_SATURATION_DIFF_REL']
```

Feature Selection - Ensemble them all

```
feature_name = X.columns.tolist()
# put all selection together
feature_selection_df = pd.DataFrame({'Feature':feature_name, 'Pearson':cor_support, '
Chi-2':chi_support, 'RFE':rfe_support, 'Logistics':embeded_lr_support, 'Random Forest
':embeded_rf_support, 'LightGBM':embeded_lgb_support})
# count the selected times for each feature
feature_selection_df['Total'] = np.sum(feature_selection_df, axis=1)
# display the top 100
num_feats = 227
feature_selection_df = feature_selection_df.sort_values(['Total','Feature'] , ascendi
ng=False)
feature_selection_df.index = range(1, len(feature_selection_df)+1)
feature_selection_df.head(num_feats)
```

```
df_selected_columns = feature_selection_df.loc[(feature_selection_df['Total'] &gt; 3)
```

```
]
df_selected_columns
```

We can list those features that were selected by at least 4 methods:

Out[23]:

	Feature	Pearson	Chi-2	RFE	Logistics	Random Forest	LightGBM	Total
1	TEMPERATURE_MAX	True	True	True	True	True	True	6
2	TEMPERATURE_DIFF_REL	True	True	True	True	True	True	6
3	RESPIRATORY_RATE_MEDIAN	True	True	True	True	True	True	6
4	RESPIRATORY_RATE_MEAN	True	True	True	True	True	True	6
5	RESPIRATORY_RATE_MAX	True	True	True	True	True	True	6
6	RESPIRATORY_RATE_DIFF_REL	True	True	True	True	True	True	6
7	RESPIRATORY_RATE_DIFF	True	True	True	True	True	True	6
8	OXYGEN_SATURATION_MIN	True	True	True	True	True	True	6
9	LACTATE_MIN	True	True	True	True	True	True	6
10	LACTATE_MEDIAN	True	True	True	True	True	True	6
11	LACTATE_MEAN	True	True	True	True	True	True	6
12	HTN	True	True	True	True	True	True	6
13	HEART_RATE_MIN	True	True	True	True	True	True	6
14	HEART_RATE_DIFF	True	True	True	True	True	True	6
15	BLOODPRESSURE_SISTOLIC_MIN	True	True	True	True	True	True	6

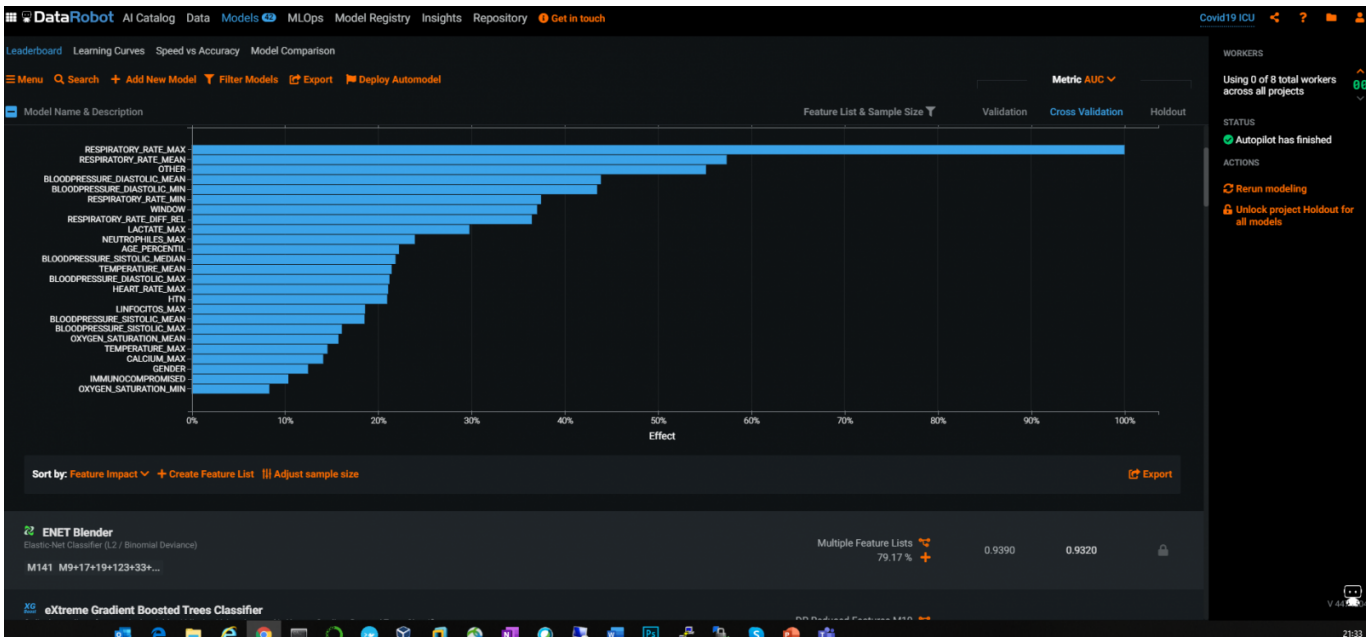
... ...

51	BE_VENOUS_MIN	True	True	True	True	False	False	4
52	BE_VENOUS_MEDIAN	True	True	True	True	False	False	4
53	BE_VENOUS_MEAN	True	True	True	True	False	False	4
54	BE_VENOUS_MAX	True	True	True	True	False	False	4
55	ALBUMIN_MIN	True	True	True	True	False	False	4
56	ALBUMIN_MEDIAN	True	True	True	True	False	False	4
57	ALBUMIN_MEAN	True	True	True	True	False	False	4
58	ALBUMIN_MAX	True	True	True	True	False	False	4

We can certainly choose these 58 features. In the meantime, experiences told us Feature Selection is not necessarily always a democratic vote; more often it could be specific to the domain problem, to the specific data, and sometimes to the specific ML model or approach that we are going to adopt later.

Feature selection - **3rd party Tools**

There are widely used industry tools and AutoML tools, for example DataRobot can give a good & automatic selection of Features :



From the above DataRobot graph, not surprisingly, we can see various RespiratoryRate and BloodPressure values to be the most relevant features to ICU admission.

Feature Selection - **Final Selection**
And in this case, I did some quick experiments and noticed that the LightGBM selection of features actually resulted in a tiny bit of better result, so we will use this selection method only.

```
df_selected_columns = embeded_lgb_feature  # better than ensembled selection
```

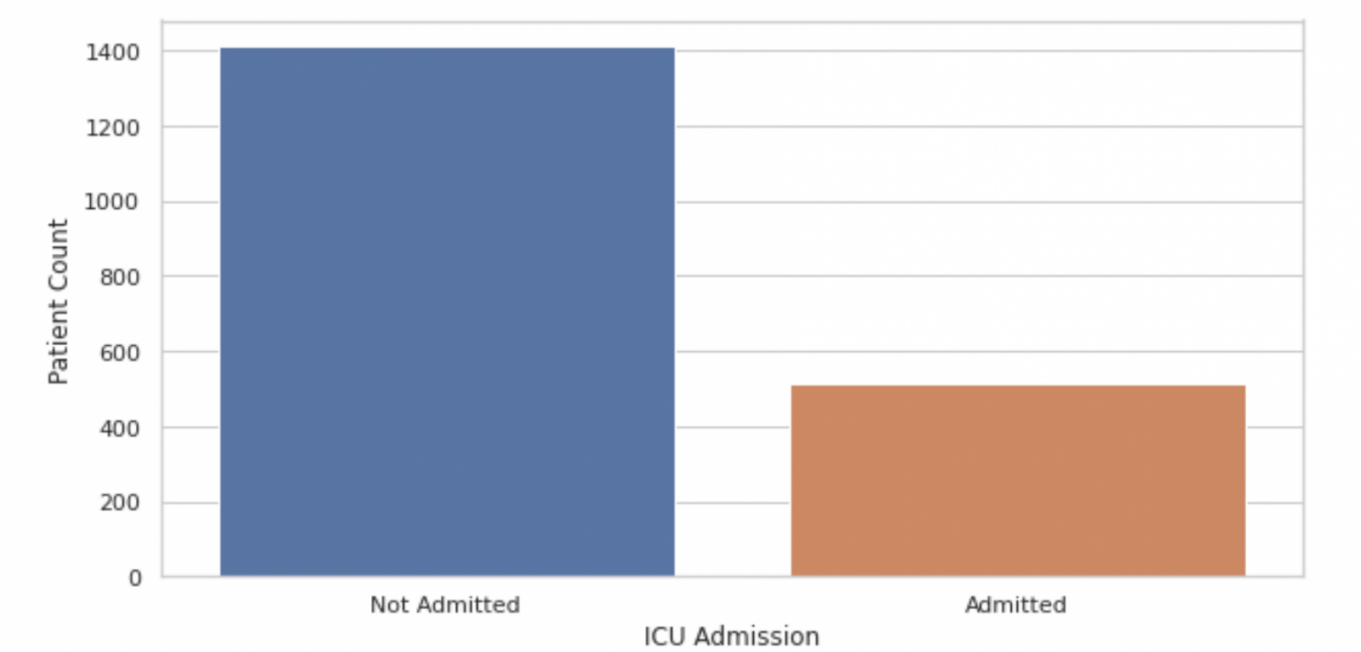
```
dataS = pd.concat([idf[df_selected_columns],idf['ICU'], df_cat['WINDOW']],1)
dataS.ICU.value_counts()
print(dataS.shape)
```

```
(1925, 58)
```

We can see 58 features are being selected; not too few, not too many; seems the right amount for this specific single target binary classification issues.

Data Imbalance

```
plt.figure(figsize=(10,5))
count = sns.countplot(x = "ICU",data=data)
count.set_xticklabels(["Not Admitted","Admitted"])
plt.xlabel("ICU Admission")
plt.ylabel("Patient Count")
plt.show()
```



This indicated that the data is imbalanced, only 26% records is ICU admitted. This will impact the result so we can think of normal data balancing approaches such as SMOTE etc.

We can try all sort of other EDAs here to understand various data distributions accordingly.

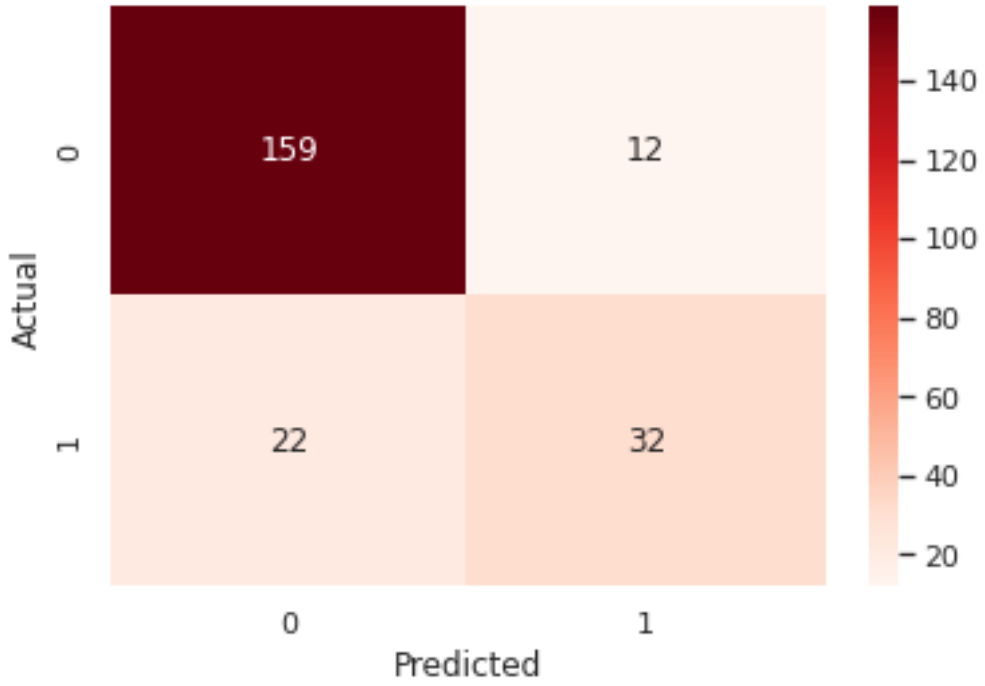
Run basic LR training

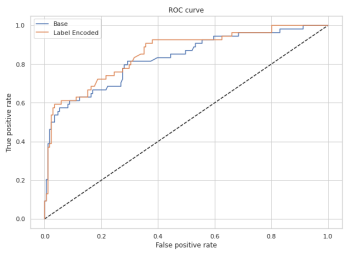
There are some nice quick training notebooks on the Kaggle site we can run quickly based on our own selection of feature columns. Let's just start with a quick run of LR classifier for the training pipeline:

```
data2 = pd.concat([idf[df_selected_columns],idf['ICU'], df_cat['WINDOW']],1)
data2.AGE_ABOVE65 = data2.AGE_ABOVE65.astype(int)
data2.ICU = data2.ICU.astype(int)
X2 = data2.drop("ICU",1)
y2 = data2.ICU

from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
X2.WINDOW = label_encoder.fit_transform(np.array(X2["WINDOW"].astype(str)).reshape((-
1,)))

confusion_matrix2 = pd.crosstab(y2_test, y2_hat, rownames=['Actual'], colnames=['Pred
icted'])
sns.heatmap(confusion_matrix2, annot=True, fmt = 'g', cmap = 'Reds') print("ORIGINAL"
)
print(classification_report(y_test, y_hat))
print("AUC = ",roc_auc_score(y_test, y_hat),'\n\n')
print("LABEL ENCODING")
print(classification_report(y2_test, y2_hat))
print("AUC = ",roc_auc_score(y2_test, y2_hat))
y2hat_probs = LR.predict_proba(X2_test)
y2hat_probs = y2hat_probs[:, 1] fpr2, tpr2, _ = roc_curve(y2_test, y2hat_probs) plt.f
igure(figsize=(10,7))
plt.plot([0, 1], [0, 1], 'k--')
```

```
plt.plot(fpr, tpr, label="Base")
plt.plot(fpr2,tpr2,label="Label Encoded")
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc="best")
plt.show()
```

```
ORIGINAL
              precision    recall  f1-score   support
           0       0.88      0.94      0.91       171
           1       0.76      0.57      0.65        54
    accuracy                           0.85       225
   macro avg       0.82      0.76      0.78       225
weighted avg       0.85      0.85      0.85       225
AUC =  0.7577972709551657
LABEL ENCODING
              precision    recall  f1-score   support
           0       0.88      0.93      0.90       171
           1       0.73      0.59      0.65        54
    accuracy                           0.85       225
   macro avg       0.80      0.76      0.78       225
weighted avg       0.84      0.85      0.84       225
AUC =  0.7612085769980507
```





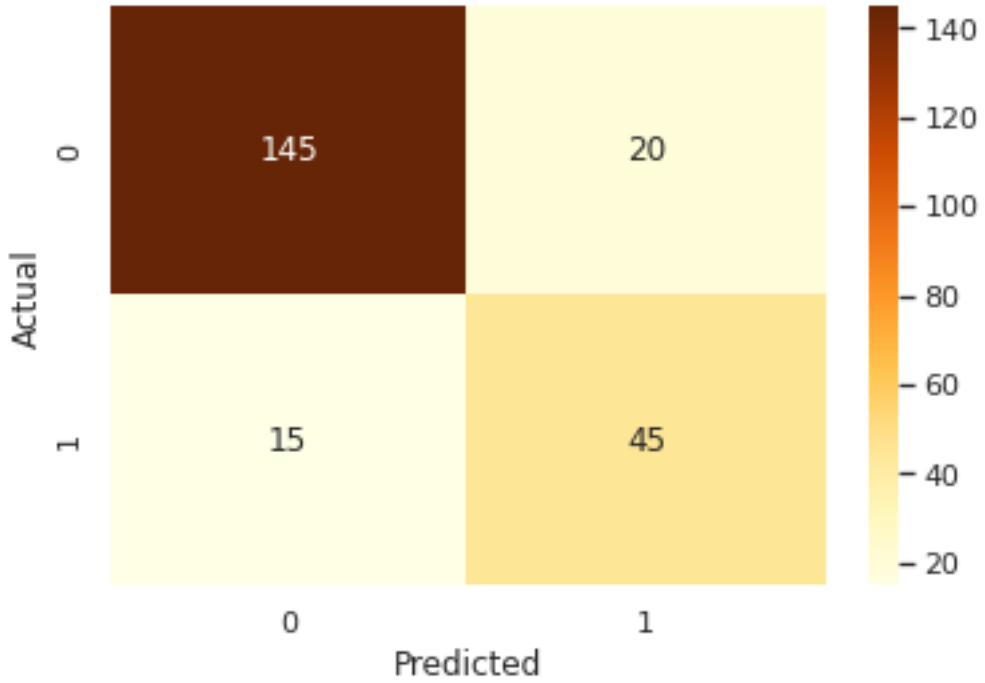
It looks like it achieves an AUC 76%, with accuracy 85%, but Recall for ICU Admitted is only 59% - there seems to be too many False Negative. That's certainly not ideal - we don't want to miss the real ICU risks for a patient record. So all the following tasks will focus on **the objective** on how to drive up the Recall rate, by driving down FN, with a somewhat balanced overall accuracy hopefully.

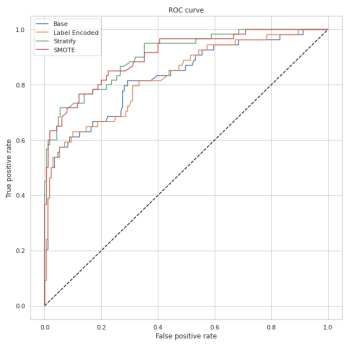
In previous sections we mentioned imbalanced data, so the firsts instinct would usually be Stratify the test set, and SMOTE it to make a more balanced dataset.

```
#stratify the test data, to make sure Train and Test data have the same ratio of 1:0
X3_train,X3_test,y3_train,y3_test = train_test_split(X2,y2,test_size=225/1925,random_
state=42, stratify = y2, shuffle = True)
# train and predict
LR.fit(X3_train,y3_train)
y3_hat = LR.predict(X3_test)

#SMOTE the data to make ICU 1:0 a balanced distribution
from imblearn.over_sampling import SMOTE sm = SMOTE(random_state = 42)
X_train_res, y_train_res = sm.fit_sample(X3_train,y3_train.ravel())
LR.fit(X_train_res, y_train_res)
y_res_hat = LR.predict(X3_test)

#draw confusion matrix etc again
confusion_matrix3 = pd.crosstab(y3_test, y_res_hat, rownames=['Actual'], colnames=['P
redicted'])
sns.heatmap(confusion_matrix3, annot=True, fmt = 'g', cmap="YlOrBr")
print("LABEL ENCODING + STRATIFY")
print(classification_report(y3_test, y3_hat))
print("AUC = ",roc_auc_score(y3_test, y3_hat),'\n\n')
print("SMOTE")
print(classification_report(y3_test, y_res_hat))
print("AUC = ",roc_auc_score(y3_test, y_res_hat))
y_res_hat_probs = LR.predict_proba(X3_test)
y_res_hat_probs = y_res_hat_probs[:, 1]
fpr_res, tpr_res, _ = roc_curve(y3_test, y_res_hat_probs) plt.figure(figsize=(10,10))

#And plot the ROC curve as before.
```

```
LABEL ENCODING + STRATIFY
              precision    recall  f1-score   support
           0       0.87      0.99      0.92       165
           1       0.95      0.58      0.72        60
    accuracy                           0.88       225
   macro avg       0.91      0.79      0.82       225
weighted avg       0.89      0.88      0.87       225
AUC =  0.7856060606060606
SMOTE
```

```
              precision    recall  f1-score   support
           0       0.91      0.88      0.89       165
           1       0.69      0.75      0.72        60
    accuracy                           0.84       225
   macro avg       0.80      0.81      0.81       225
weighted avg       0.85      0.84      0.85       225
AUC =  0.8143939393939393
```





So STRATIFY and SMOT processing of the data does seem to improve the Recall from 0.59 to 0.75, with an overall accuracy of 0.84.

Now the data processing is largely done as usual for tradition ML, we want to know what could be the best model(s) in this case; can they do any better, and can we then try a relative comprehensive comparison?

Run Training Comparisons of Various Models:

Let's carry on evaluating some commonly used ML algorithms, and generate a result dashboard of comparison by box plots:

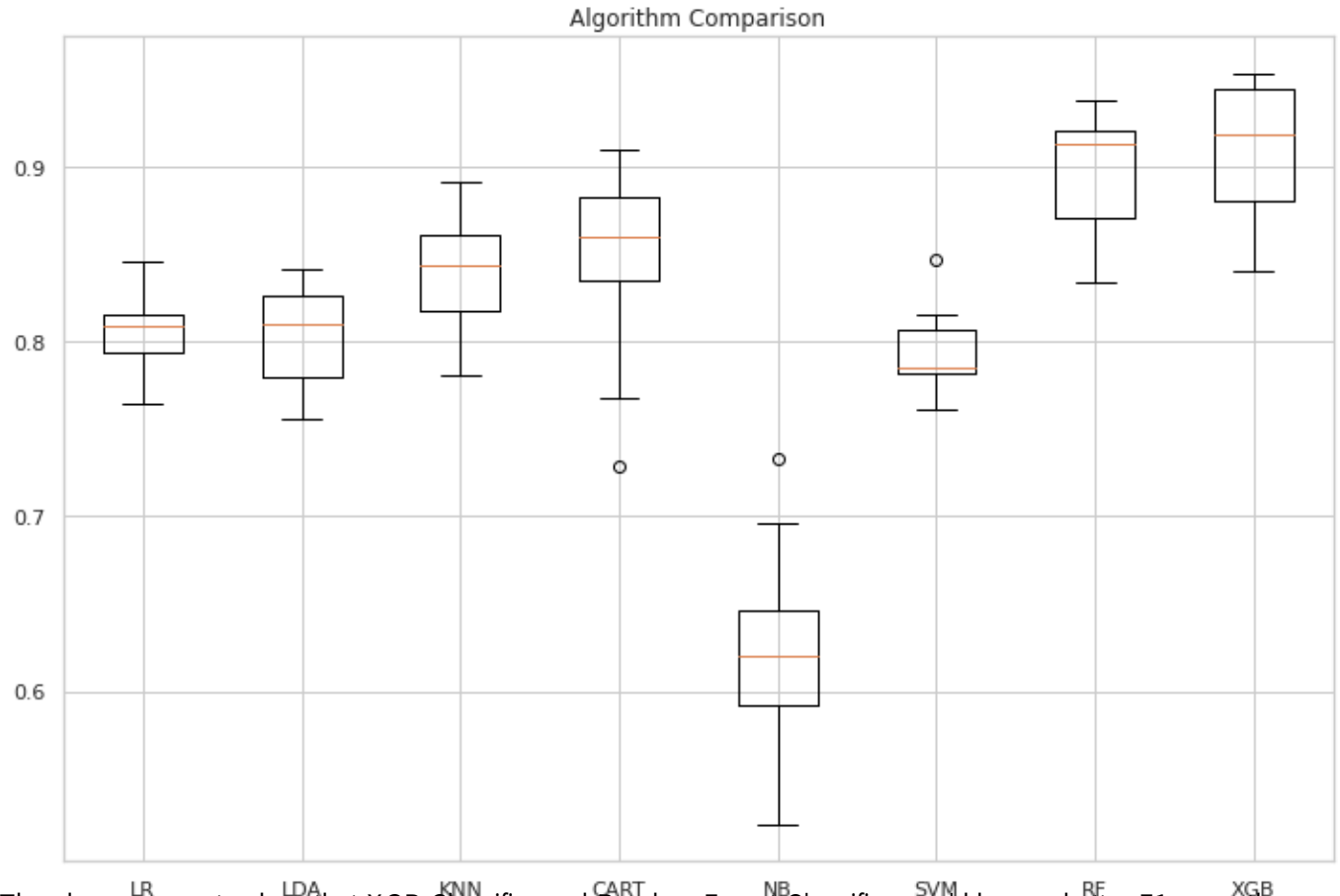
```
# compare algorithms
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
#Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

```
# List Algorithms together
models = []
models.append(('LR', <strong>LogisticRegression</strong>(solver='liblinear', multi_cl
ass='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', <strong>KNeighborsClassifier</strong>()))
models.append(('CART', <strong>DecisionTreeClassifier</strong>()))
models.append(('NB', <strong>GaussianNB</strong>()))
models.append(('SVM', <strong>SVC</strong>(gamma='auto')))
models.append(('RF', <strong>RandomForestClassifier</strong>(n_estimators=100)))
models.append(('XGB', <strong>XGBClassifier</strong>())) #clf = XGBClassifier()
```

```
# evaluate each model in turn
results = []
```

```
names = []
for name, model in models:
    kfold = StratifiedKFold(n_splits=10, random_state=1)
    cv_results = cross_val_score(model, X_train_res, y_train_res, cv=kfold, scoring='
f1')  ## accuracy, precision,recall
    results.append(cv_results)
    names.append(name)
    print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))



# Compare all model's performance. Question - would like to see a Integrated item on
it?
pyplot.figure(4, figsize=(12, 8))
pyplot.boxplot(results, labels=names)
pyplot.title('Algorithm Comparison')
pyplot.show()
```

LR: 0.805390 (0.021905) **LDA**: 0.803804 (0.027671) **KNN**
: 0.841824 (0.032945) **CART**: 0.845596 (0.053828)
NB: 0.622540 (0.060390) **SVM**: 0.793754 (0.023050) **RF**
: 0.896222 (0.033732) **XGB**: 0.907529 (0.040693)



The above seems to show that XGB Classifier and Random Forest Classifier would have a better F1 score than other models.

Let's compare their actual test results on the same set of normalised test data as well:

```
import time
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC



for name, model in models:
    print(name + ':\n\r')
    start = time.clock()
    model.fit(X_train_res, y_train_res)
    print("Train time for ", model, " ", time.clock() - start)
    predictions = model.predict(X3_test) #(X_validation)
    # Evaluate predictions
    print(accuracy_score(y3_test, predictions))  # Y_validation
    print(confusion_matrix(y3_test, predictions))
    print(classification_report(y3_test, predictions))




LR:
Train time for  LogisticRegression(multi_class='ovr', solver='liblinear')   0.0281449
9999999498
0.8444444444444444
[[145  20]
 [ 15  45]]
              precision    recall  f1-score   support
           0       0.91      0.88      0.89       165
           1       0.69      0.75      0.72        60
    accuracy                           0.84       225
   macro avg       0.80      0.81      0.81       225
weighted avg       0.85      0.84      0.85       225

LDA:
Train time for  LinearDiscriminantAnalysis()   0.2280070000000194
0.8488888888888889
[[147  18]
 [ 16  44]]
              precision    recall  f1-score   support
           0       0.90      0.89      0.90       165
           1       0.71      0.73      0.72        60
    accuracy                           0.85       225
   macro avg       0.81      0.81      0.81       225
weighted avg       0.85      0.85      0.85       225

KNN:
Train time for  KNeighborsClassifier()   0.13023699999999394
0.8355555555555556
[[145  20]
 [ 17  43]]
              precision    recall  f1-score   support
           0       0.90      0.88      0.89       165
           1       0.68      0.72      0.70        60
    accuracy                           0.84       225
   macro avg       0.79      0.80      0.79       225
```

```
weighted avg       0.84      0.84      0.84       225

CART:
Train time for  DecisionTreeClassifier()   0.32616000000001577
0.8266666666666667
[[147  18]
 [ 21  39]]
              precision    recall  f1-score   support
           0       0.88      0.89      0.88       165
           1       0.68      0.65      0.67        60
    accuracy                           0.83       225
   macro avg       0.78      0.77      0.77       225
weighted avg       0.82      0.83      0.83       225

NB:
Train time for  GaussianNB()   0.0034229999999979555
0.8355555555555556
[[154  11]
 [ 26  34]]
              precision    recall  f1-score   support
           0       0.86      0.93      0.89       165
           1       0.76      0.57      0.65        60
    accuracy                           0.84       225
   macro avg       0.81      0.75      0.77       225
weighted avg       0.83      0.84      0.83       225

SVM:
Train time for  SVC(gamma='auto')   0.3596520000000112
0.8977777777777778
[[157   8]
 [ 15  45]]
              precision    recall  f1-score   support
           0       0.91      0.95      0.93       165
           1       0.85      0.75      0.80        60
    accuracy                           0.90       225
   macro avg       0.88      0.85      0.86       225
weighted avg       0.90      0.90      0.90       225

RF:
Train time for  RandomForestClassifier()   0.50123099999999
0.9066666666666666
[[158   7]
 [ 14  46]]
              precision    recall  f1-score   support
           0       0.92      0.96      0.94       165
           1       0.87      0.77      0.81        60
    accuracy                           0.91       225
   macro avg       0.89      0.86      0.88       225
weighted avg       0.91      0.91      0.90       225

XGB:
Train time for  XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)   1.6495209
```

```
99999993
0.8844444444444445
[[155  10]
 [ 16  44]]
              precision    recall  f1-score   support

           0       0.91      0.94      0.92       165
           1       0.81      0.73      0.77        60

    accuracy                           0.88       225
   macro avg       0.86      0.84      0.85       225
weighted avg       0.88      0.88      0.88       225
```

The result appears that RF is actually better than XGB. It could imply that XGB might have a bit more over-fitting somehow. RFC result is a slight improvement over LR too.

Run Selected Model by further "Parameter Tuning via Grid Search"

Now let's assume we settled down Random Forest Classifier being our choice of model. We can execute a further Grid Search on this model to see whether we can further drive out a tiny bit more performance of the results.

Remember our objective is still to optimise the Recall in this case, by minimising the False Negative of possible ICU risks for the patient encounter, so we will 'recallscore' to re-fit the grid search below. Again it will use 10-fold cross-validation as usual, since our test set above was always set to about 12% of those 2915 records.

```
from sklearn.model_selection import GridSearchCV
# Create the parameter grid based on the results of random search

param_grid = {'bootstrap': [True],
 'ccp_alpha': [0.0],
 'class_weight': [None],
 'criterion': ['gini', 'entropy'],
 'max_depth': [None],
 'max_features': ['auto', 'log2'],
 'max_leaf_nodes': [None],
 'max_samples': [None],
 'min_impurity_decrease': [0.0],
 'min_impurity_split': [None],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 4],
 'min_weight_fraction_leaf': [0.0],
 'n_estimators': [100, 125],
 #'n_jobs': [None],
 'oob_score': [False],
 'random_state': [None],
 #'verbose': 0,
 'warm_start': [False]
}



#Fine-tune by confusion matrix
from sklearn.metrics import roc_curve, precision_recall_curve, auc, make_scorer, reca
ll_score, accuracy_score, precision_score, confusion_matrix
scorers = {
    'recall_score': make_scorer(recall_score),
    'precision_score': make_scorer(precision_score),
```

```
    'accuracy_score': make_scorer(accuracy_score)
}



# Create a based model
rfc = RandomForestClassifier()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rfc, param_grid = param_grid,
                           scoring=scorers, refit='recall_score',
                           cv = 10, n_jobs = -1, verbose = 2)



train_features = X_train_res



grid_search.fit(train_features, train_labels)



rf_best_grid = grid_search.best_estimator_



rf_best_grid.fit(train_features, train_labels)
rf_predictions = rf_best_grid.predict(X3_test)
print(accuracy_score(y3_test, rf_predictions))
print(confusion_matrix(y3_test, rf_predictions))
print(classification_report(y3_test, rf_predictions))




0.92
[[ 46  14]
 [  4 161]]
              precision    recall  f1-score   support
           0       0.92      0.77      0.84        60
           1       0.92      0.98      0.95       165
    accuracy                           0.92       225
   macro avg       0.92      0.87      0.89       225
weighted avg       0.92      0.92      0.92       225
```

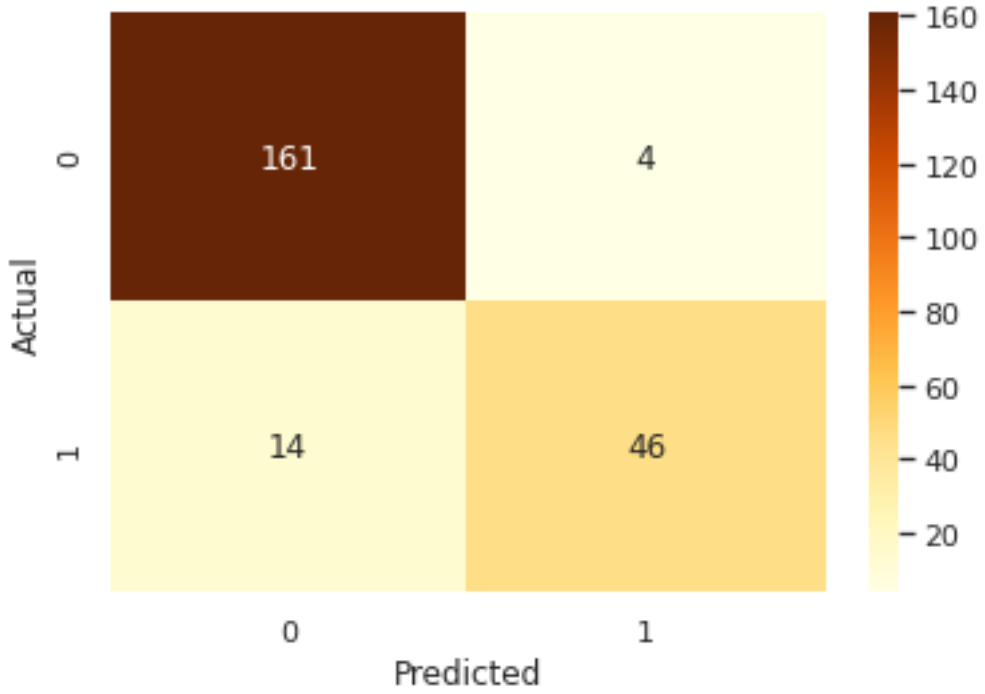
The result showed that a Grid Search did manage to drive up overall accuracy a bit, while keeping FN the same.

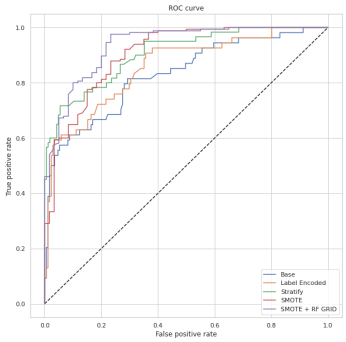
Let's plot the AUC comparisons as well:

```
confusion_matrix4 = pd.crosstab(y3_test, rf_predictions, rownames=['Actual'], colname
s=['Predicted'])
sns.heatmap(confusion_matrix4, annot=True, fmt = 'g', cmap="YlOrBr")


print("LABEL ENCODING + STRATIFY")
print(classification_report(y3_test, 1-y3_hat))
print("AUC = ",roc_auc_score(y3_test, 1-y3_hat),'\n\n')
```

```
print("SMOTE")
print(classification_report(y3_test, 1-y_res_hat))
print("AUC = ",roc_auc_score(y3_test, 1-y_res_hat), '\n\n')



print("SMOTE + LBG Selected Weights + RF Grid Search")
print(classification_report(y3_test, rf_predictions))
print("AUC = ",roc_auc_score(y3_test, rf_predictions), '\n\n\n')



y_res_hat_probs = LR.predict_proba(X3_test)
y_res_hat_probs = y_res_hat_probs[:, 1]



predictions_rf_probs = rf_best_grid.predict_proba(X3_test) #(X_validation)
predictions_rf_probs = predictions_rf_probs[:, 1]



fpr_res, tpr_res, _ = roc_curve(y3_test, 1-y_res_hat_probs)
fpr_rf_res, tpr_rf_res, _ = roc_curve(y3_test, predictions_rf_probs)



plt.figure(figsize=(10,10))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr, label="Base")
plt.plot(fpr2,tpr2,label="Label Encoded")
plt.plot(fpr3,tpr3,label="Stratify")
plt.plot(fpr_res,tpr_res,label="SMOTE")
plt.plot(fpr_rf_res,tpr_rf_res,label="SMOTE + RF GRID")
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc="best")
plt.show()



LABEL ENCODING + STRATIFY
              precision    recall  f1-score   support
           0       0.95      0.58      0.72        60
           1       0.87      0.99      0.92       165
    accuracy                           0.88       225
   macro avg       0.91      0.79      0.82       225
weighted avg       0.89      0.88      0.87       225
AUC =  0.7856060606060606

SMOTE
              precision    recall  f1-score   support
           0       0.69      0.75      0.72        60
           1       0.91      0.88      0.89       165
    accuracy                           0.84       225
   macro avg       0.80      0.81      0.81       225
weighted avg       0.85      0.84      0.85       225
```

```
AUC =  0.8143939393939394

SMOTE + LBG Selected Weights + RF Grid Search
              precision    recall  f1-score   support
           0       0.92      0.77      0.84        60
           1       0.92      0.98      0.95       165
    accuracy                           0.92       225
   macro avg       0.92      0.87      0.89       225
weighted avg       0.92      0.92      0.92       225
AUC =  0.8712121212121211
```





The result showed that after algorithm comparisons and further grid search we managed to drive up the AUC from 78% to 87%, with an overall accuracy of 92% and a recall 77%.

Recap of "traditional ML" approach

So, how is this result really? It's ok for a basic manual process with traditional ML algorithms. How would it appear in Kaggle competition tables? Well, it would not be in leader board. I ran the raw dataset through DataRobot's current AutoML service, the best result would claim an equivalent AUC of 90+% (to be further confirmed with like-alike data) with the model "XGB Trees Classifier with Unsupervised Learning Features", out of a comparison of top 43 models. This might be the kind of bottom-line model we would scope in if really want to be competitive on Kaggle. I will attach its top list of best results vs models into the github too. Eventually, for real-world cases specific to care sites, my feeling is we also need to scope in some degree of customised deep learning approaches, as mentioned in the "Data & Task" section of this post. Sure, in real-world cases, where to collect some quality data columns could be an upfront issue too.

IntegratedML Approach?

The above walked through so called traditional ML process, which normally includes the data EDA, feature engineering, feature selection, model selection, and performance optimisation by grid search etc. That's the simplest approach I could think of so far for this task, and we haven't even touched on the model deployment and service management life-cycles yet - we will touch that in next post, by looking into how we could leverage Flask/FastAPI/IRIS and deploy this basic ML model into a Covid-19 X-Ray demo service stack.

IRIS has IntegratedML now, which is a sleek SQL wrapper of a powerful options of AutoMLs. In Part II we can look into how to accomplish the above task in a much simplified process, so we don't have to bother with feature selection, model selection, and performance optimisation etc etc anymore, while possibly driving out equivalent ML results for business benefit.

Up to here this post might be too long for a 10-minute note to squeeze in a quick run of integratedML with the same data, so I move it to next post Part II.

#Artificial Intelligence (AI) #IntegratedML #Machine Learning (ML) #SQL #InterSystems IRIS
Check the related application on InterSystems Open Exchange